Version 3.2
3/15/21

Important Prerequisites and Setup

- Have VirtualBox installed and ready

https://www.virtualbox.org/

- Download VM from

https://www.dropbox.com/s/a76njwon38db6js/BDPJ2v3.ova?dl=0

or

https://s3.console.aws.amazon.com/s3/object/bclconf?region=us-west-2&prefix=BDPJ2v3.ova

- See instructions at

https://github.com/brentlaster/safaridocs/blob/master/bdpj2-setup.pdf

- See labs at

https://github.com/brentlaster/safaridocs/blob/master/bdpj2-class-labs.pdf

Building a Deployment Pipeline with Jenkins 2

Brent Laster

About me

- R&D Director
- Global trainer – training (Git, Jenkins, Gradle, CI/CD, pipelines, Kubernetes, Helm, ArgoCD, operators)
- Author -
  - OpenSource.com
  - Professional Git book
  - Jenkins 2 – Up and Running book
  - Continuous Integration vs. Continuous Delivery vs. Continuous Deployment mini-book on Safari
- https://www.linkedin.com/in/brentlaster
- @BrentCLaster
- GitHub: brentlaster

Book – Professional Git

- Extensive Git reference, explanations, ar examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

Amazon Customer

I can't recommend this book more highly

February 12, 2017

Format: Kindle Edition

Brent Laster's book is in a different league from the many print and video sources that I've looked at in my attempt to learn Git. The book is extremely well organised and very clearly written. His decision to focus on Git as a local application for the first several chapters, and to defer discussion about it as a remote application until later in the book, works extremely well.

Laster has also succeeded in writing a book that should work for both beginners and people with a fair bit of experience with Git. He accomplishes this by offering, in each chapter, a core discussion followed by more advanced material and practical exercises.

I can't recommend this book more highly.

Ideal for hands-on reading and experimentation

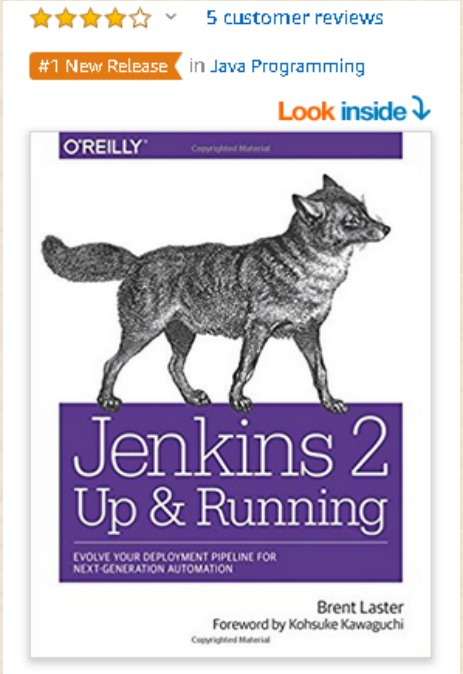
February 23, 2017

Format: Paperback | Verified Purchase

I just finished reading Professional Git, which is well organized and clearly presented. It works as both a tutorial for newcomers and a reference book for those more experienced. I found it ideal for hands-on reading and experimentation with things you may not understand at first glance. I was already familiar with Git for everyday use, but I've always stuck with a convenient subset. It was great to be able to finally get a much deeper understanding. I highly recommend the book.

Jenkins 2 Book

- Jenkins 2 – Up and Running
- "It's an ideal book for those who are new to CI/CD, as well as those who have been using Jenkins for many years. This book will help you discover and rediscover Jenkins." ***By Kohsuke Kawaguchi, Creator of Jenkins***



This is highly recommended reading for anyone looking to use Jenkins 2 to ...
By Leila on June 2, 2018
Format: Paperback

Brent really knows his stuff. I'm already a few chapters in, and I'm finding the content incredibly engaging. This is highly recommended reading for anyone looking to use Jenkins 2 to implement CD pipelines in their code.

A great resource
By Brian on June 2, 2018
Format: Paperback

I have to admit that most of the information I get usually comes through the usual outlets: stack overflow, Reddit, and others. But I've realized that having a comprehensive resource is far better than hunting and pecking for scattered answers across the web. I'm so glad I got this book!

O'Reilly Training

LIVE ONLINE TRAINING

Containers A-Z

An overview of containers, Docker, Kubernetes, Istio, Helm, Kubernetes Operators, and GitOps

Topic: System Administration

LIVE ONLINE TRAINING

Helm Fundamentals

Deploying, upgrading, and rolling back applications in Kubernetes

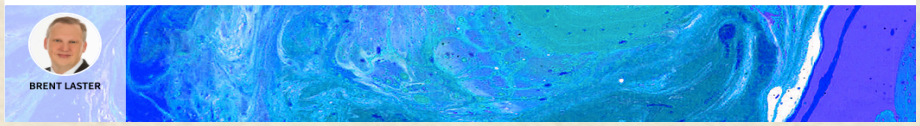
Topic: System Administration

LIVE ONLINE TRAINING

Building a Kubernetes Operator: Extending Kubernetes to Fit Your Applications

Extending Kubernetes to Fit Your Applications

Topic: System Administration

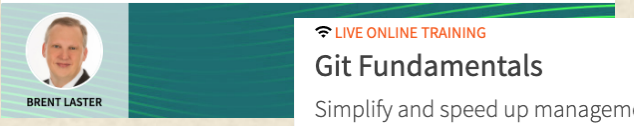


LIVE ONLINE TRAINING

Continuous Delivery in Kubernetes with ArgoCD

Automating deployment and lifecycle management

Topic: System Administration



LIVE ONLINE TRAINING

Git Fundamentals

Simplify and speed up management of your source code

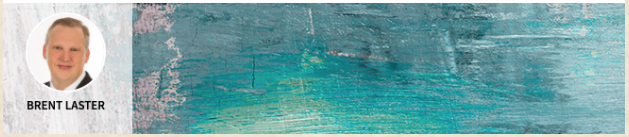
Topic: Software Development

LIVE ONLINE TRAINING

Getting started with continuous delivery (CD)

Move beyond CI to build, manage, and deploy a working pipeline

Topic: System Administration

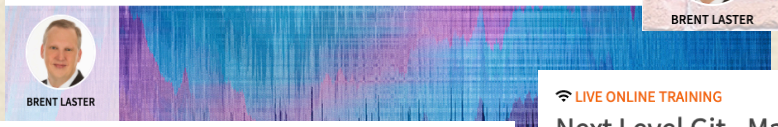


LIVE ONLINE TRAINING

Next Level Git - Master your content

Use powerful tools in Git to simplify merges, rewrite history, and perform autor

Topic: Software Development

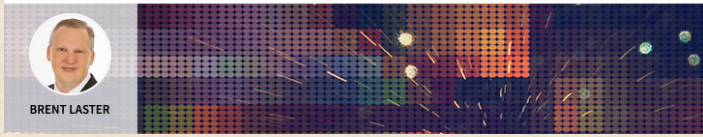


LIVE ONLINE TRAINING

Next Level Git - Master your workflow

Use Git to find problems, simplify working with multiple branches and repositories, and customize behavior with hooks

Topic: Software Development



LIVE ONLINE TRAINING

Building a deployment pipeline with Jenkins 2

Manage continuous integration and continuous delivery to release software

Topic: System Administration

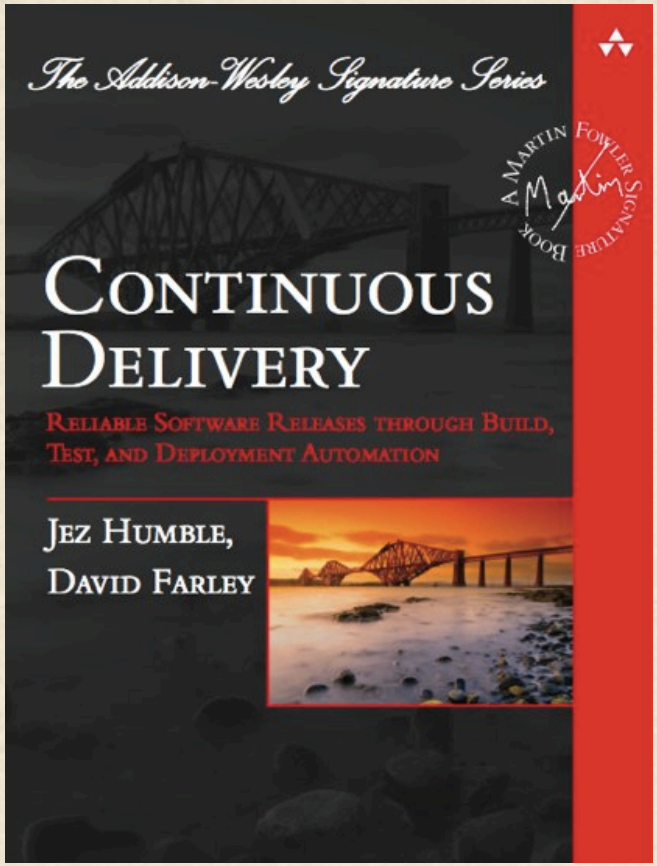
LIVE ONLINE TRAINING

Git Troubleshooting

How to solve practically any problem that comes your way

Topic: Software Development

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

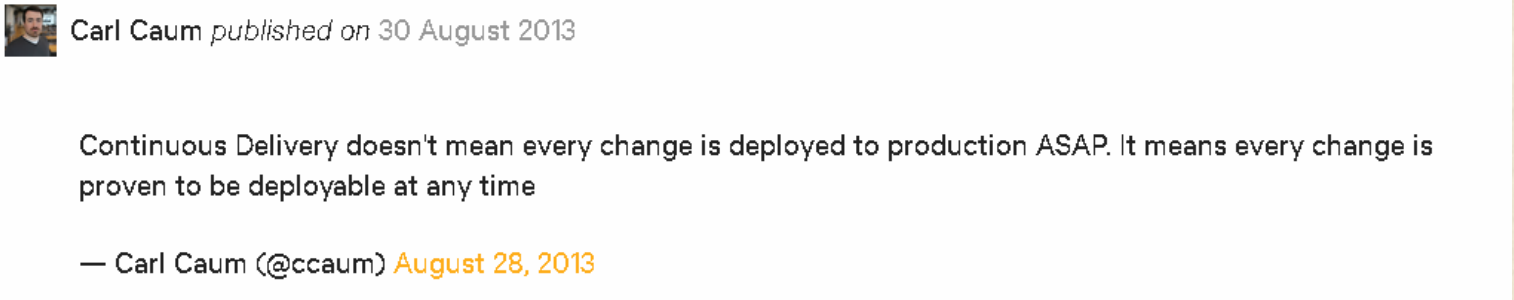
Benefits

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

Continuous Delivery

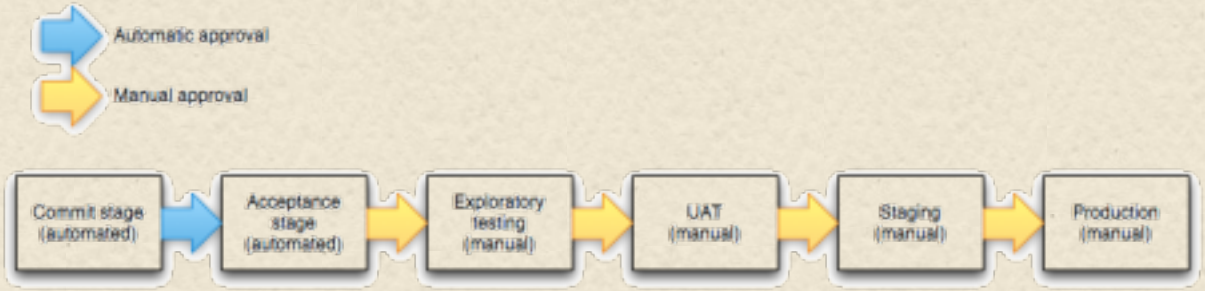
Practice

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



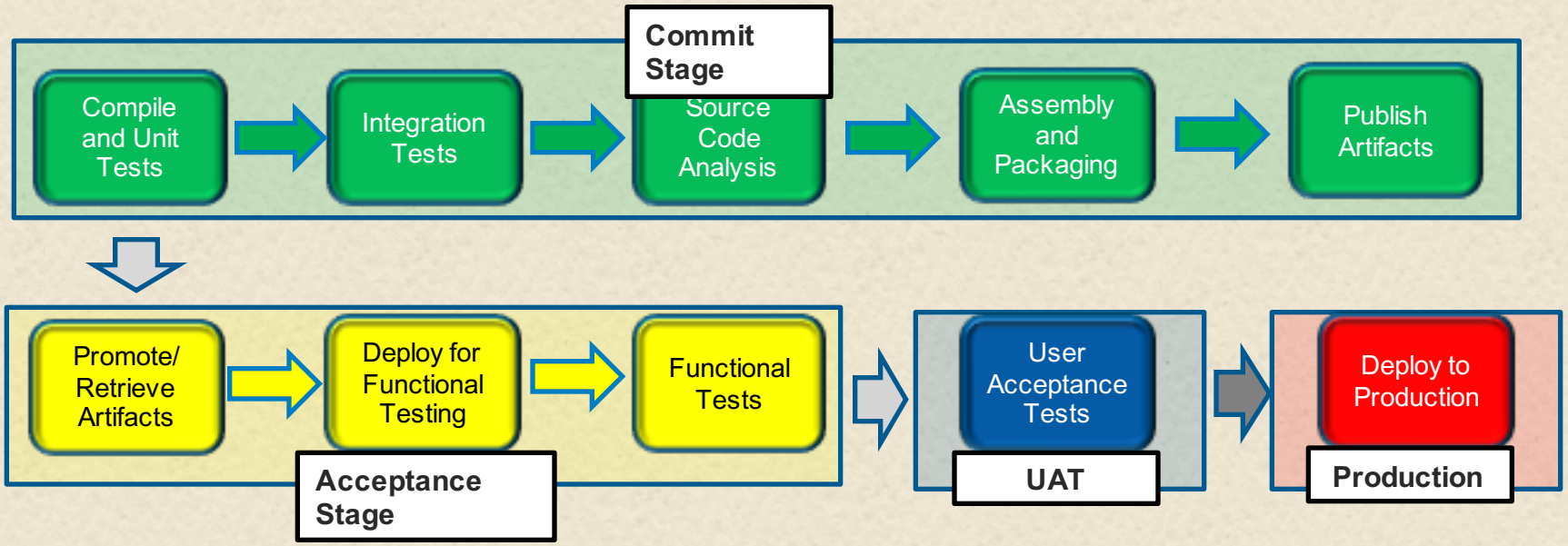
Deployment Pipeline

- "… 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

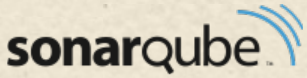


- **Approach**
  - **Model steps in pipeline as stages in Jenkins pipeline script**
  - **Some Linux-specific implementations**
  - **Using Jenkins wherever possible**
  - **Using free versions of products**
  - **Light treatment of each area**
  - **Selected set of technologies**

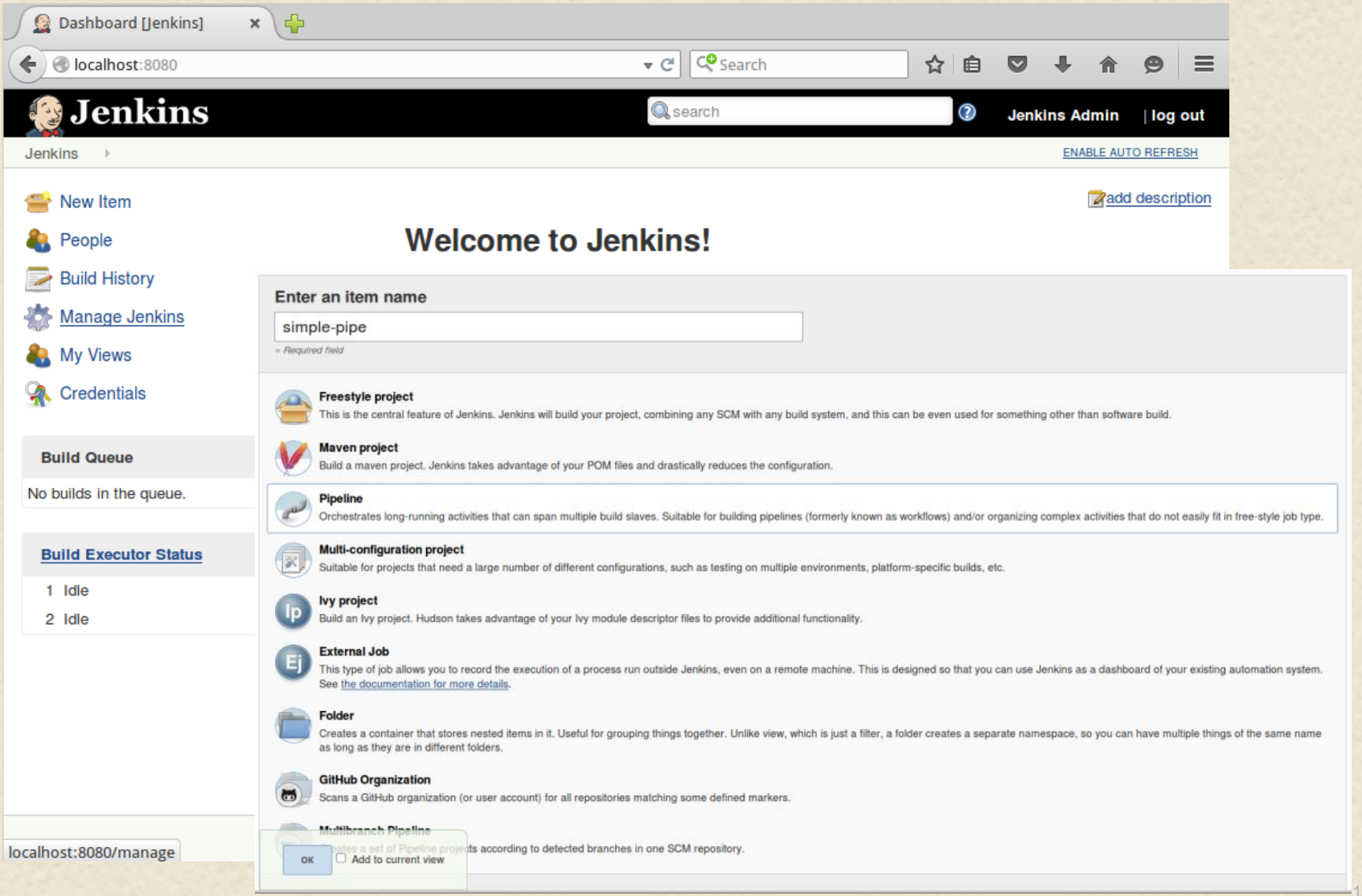
What is Jenkins 2 (2.0+)?

- 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

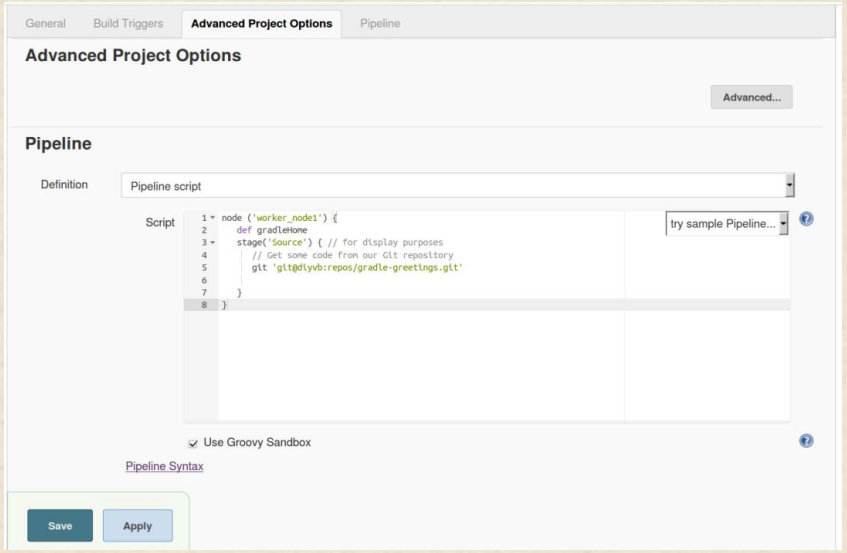
- Jenkins - workflow management  Jenkins
- Git - source management 
- Gradle - build engine and automation 
- SonarQube - code analysis and metrics 
- Jacoco - code coverage 
- Artifactory - binary artifact storage & management 
- Docker - container and image creation 

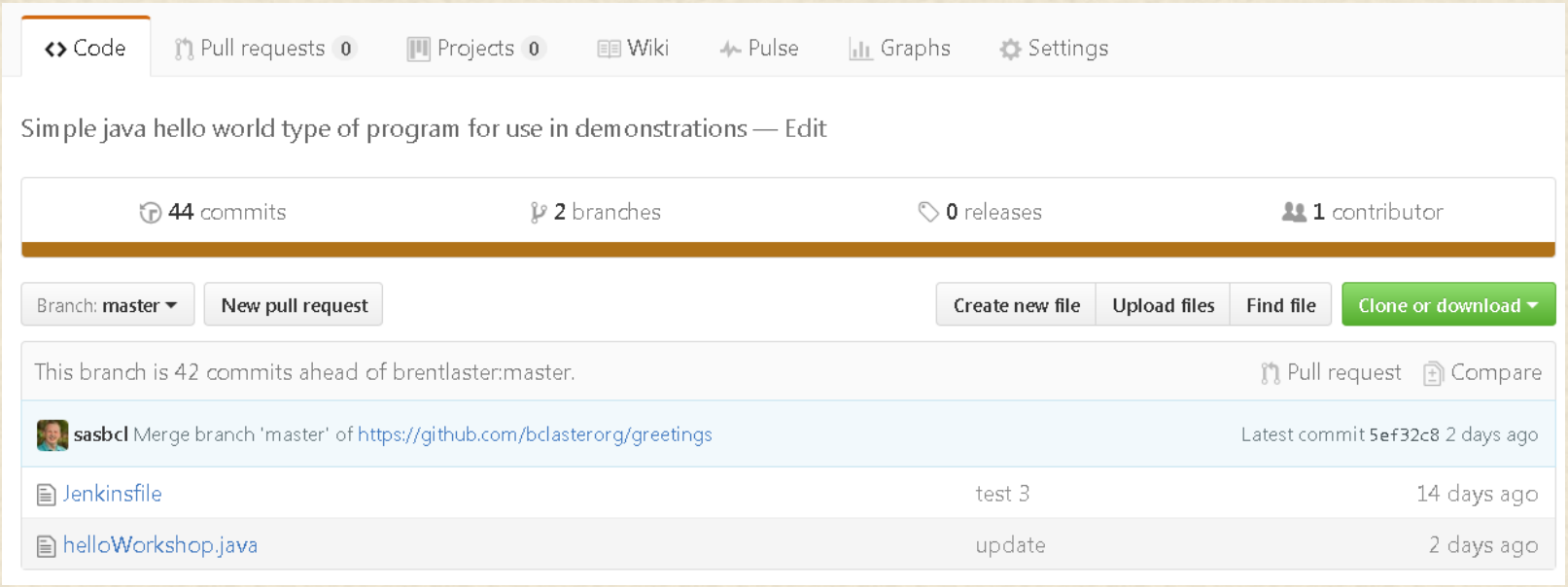
Home Screen -> Types of Projects



Jenkins Domain Specific Language (DSL)

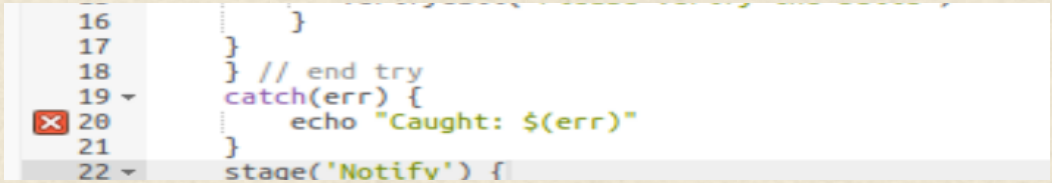
- Groovy-based
- Allows for orchestrating process steps
- Can be written and stored as pipeline script in job itself or as an external Jenkinsfile stored in the repository



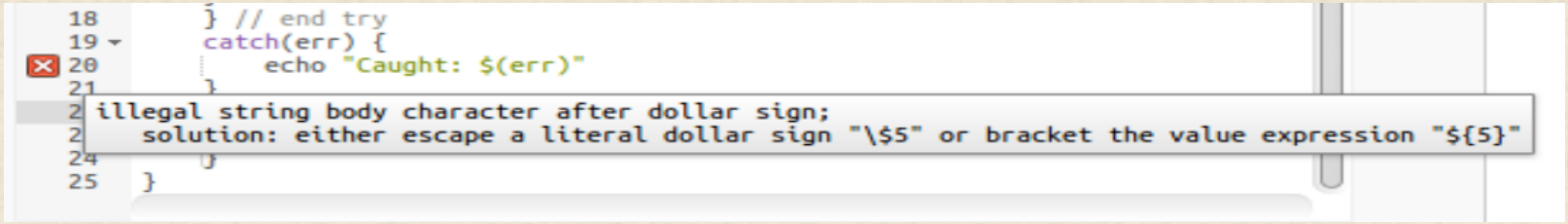


Working with the pipeline script editor

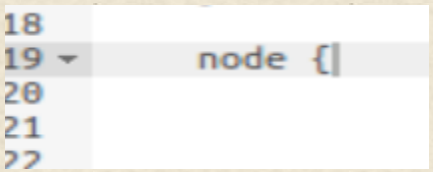
- Syntax error highlighting - not always valid

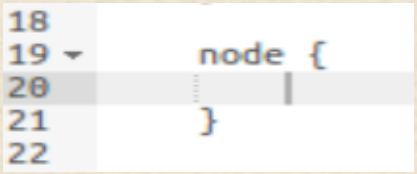


- Hover over X to see explanation



- 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





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') {...}
7      stage('Compile') {
8          // Run gradle to execute comp
9          gbuild3 'clean compileJava -x
```

```
1  @Library('Utilities2') _
2  node('worker_node1') {...}
13
```

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

```groovy
// Scripted Pipeline
node('worker') {
    stage('Source') { // Get code
        // Get code from our git repository
        git 'git@diyvb2:/home/git/repositories/workshop.git '
    }
    stage('Compile') {  // Compile and do unit testing
        // Run gradle to execute compile and unit testing
        sh "gradle clean compileJava test"
    }
}
```

```groovy
pipeline {   // declarative pipeline
    agent label:'worker'
    stages {
        stage('Source') { // Get code
            steps {
                // Get code from our git repository
                git 'git@diyvb2:/home/git/repositories/workshop.git '
            }
        }
        stage('Compile') {  // Compile and do unit testing
            steps {
                // Run gradle to execute compile and unit testing
                sh "gradle clean compileJava test"
            }
        }
    }
}
```

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'){
          ...
        }},
    )
```

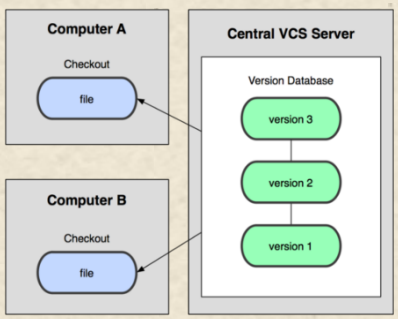
Lab 1 – Node setup

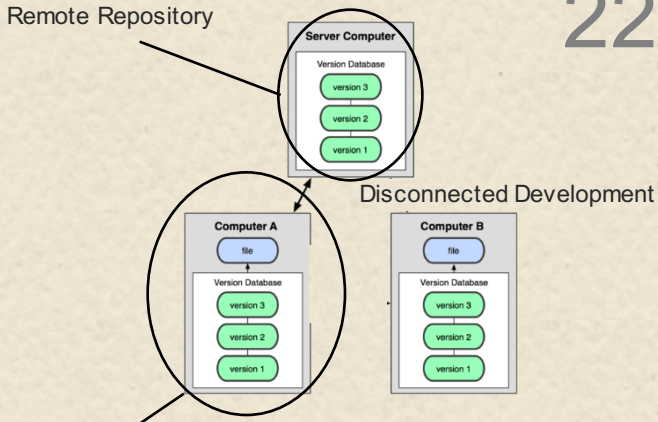
Git

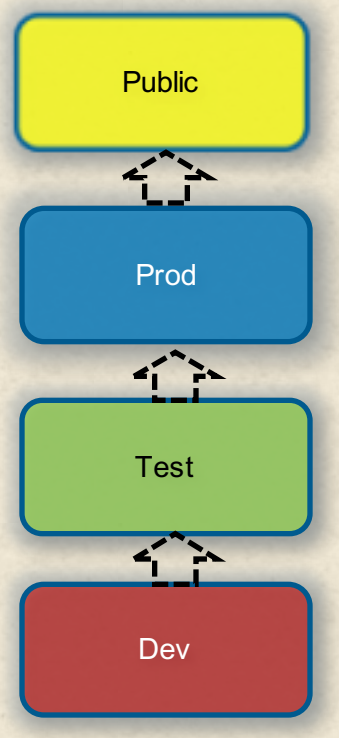
- Distributed Version Control System
- Open source
- Available for multiple platforms
- Roots in source control efforts for Linux Kernel (2005)
- Primary developer – Linus Torvalds

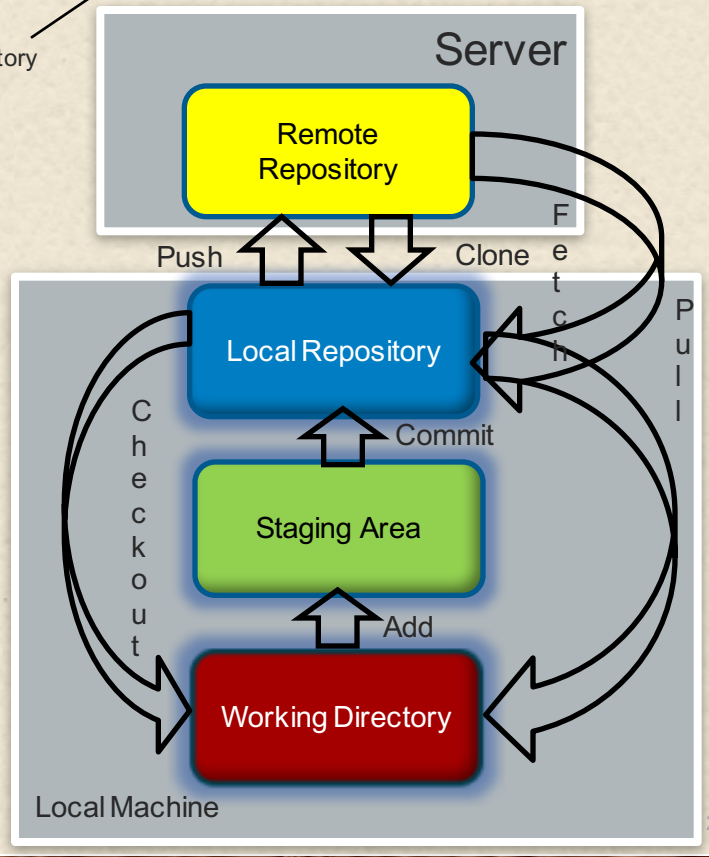
workflow

```
git init or git clone
<create or edit content>
git add .
git commit –m “<msg>”
git push origin master
```







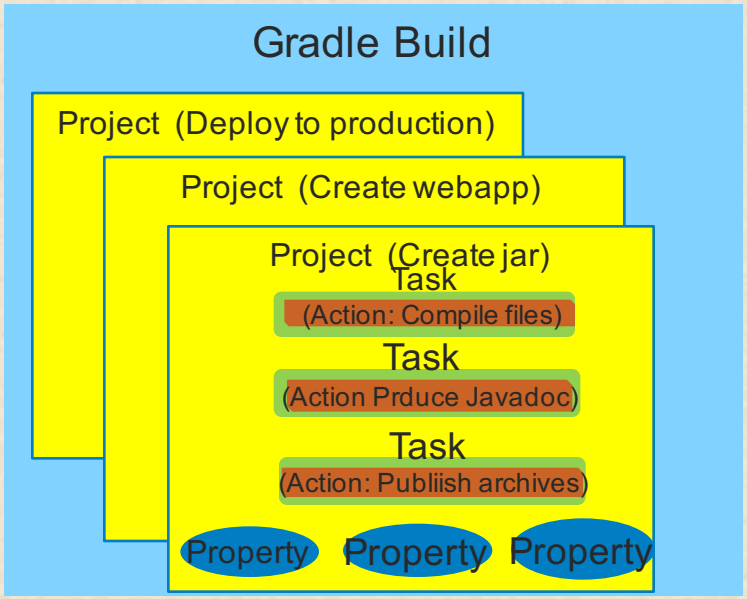


Gradle

- 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

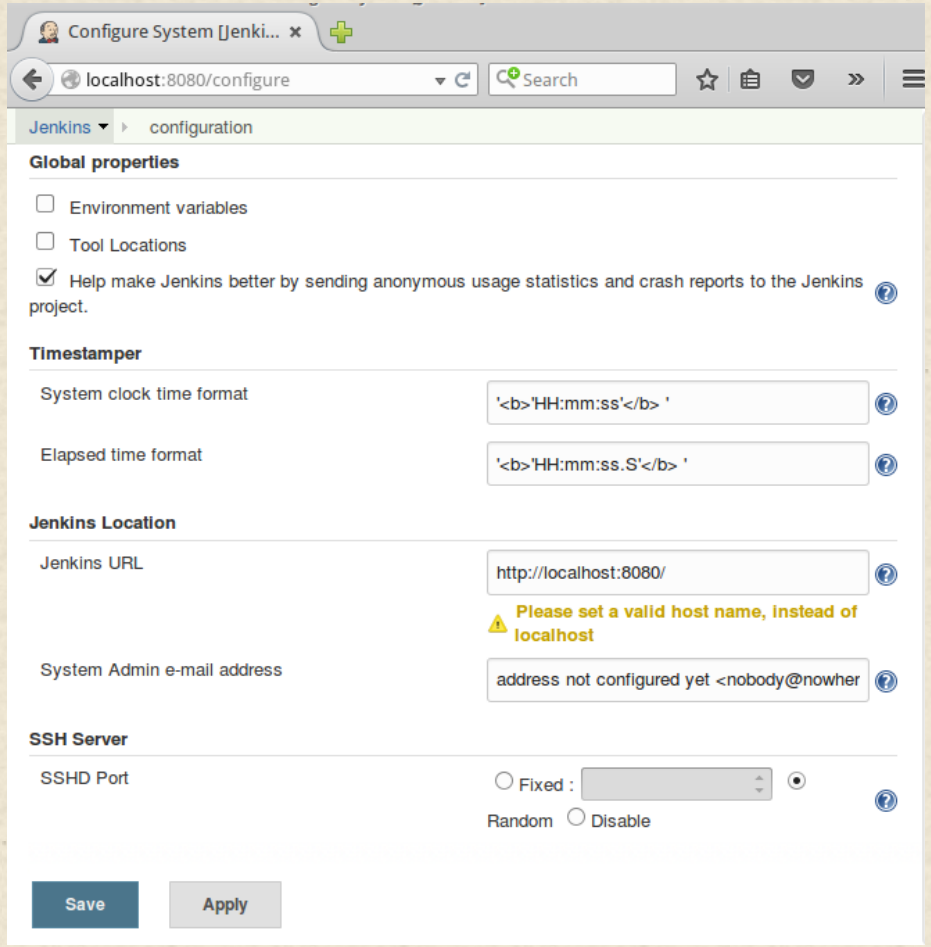
- Two basic concepts – projects and tasks
- Project
  - Every Gradle build is made up of one or more projects
  - A project is some component of software that can be built (examples: library JAR, web app, zip assembled from jars of other projects)
  - May represent something to be done, rather than something 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

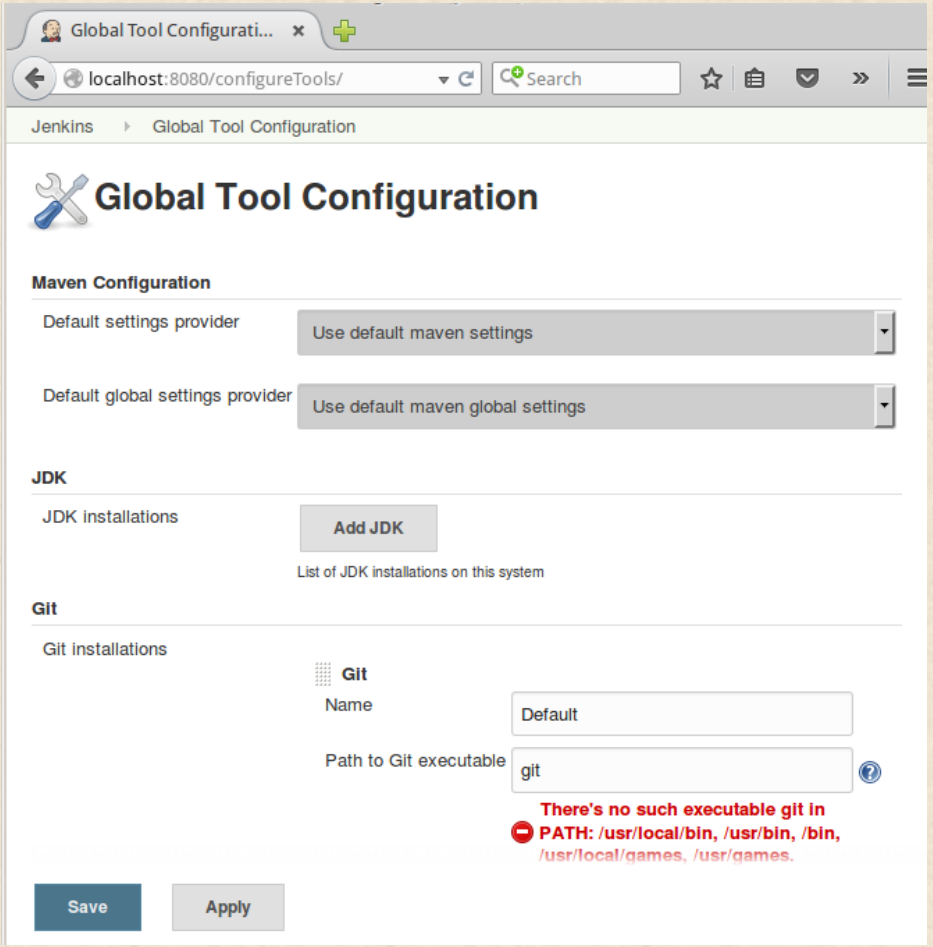



```
def destDirs = new File('target3/results')

task createTarget(type: DefaultTask) {
    doLast {
        destDirs.mkdirs()
    }
}
```

KEYWORD
TASK NAME
TYPE SPECIFICATION
API
ACTION

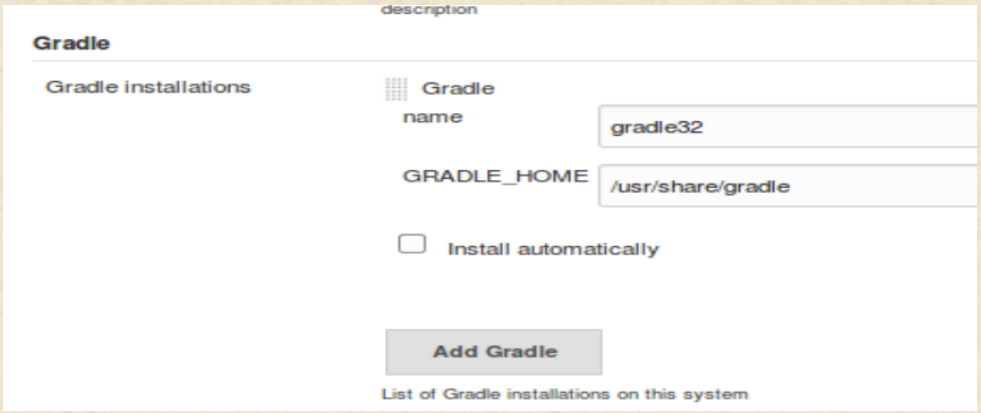
Configure System vs. Global Tool Config





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  }
```

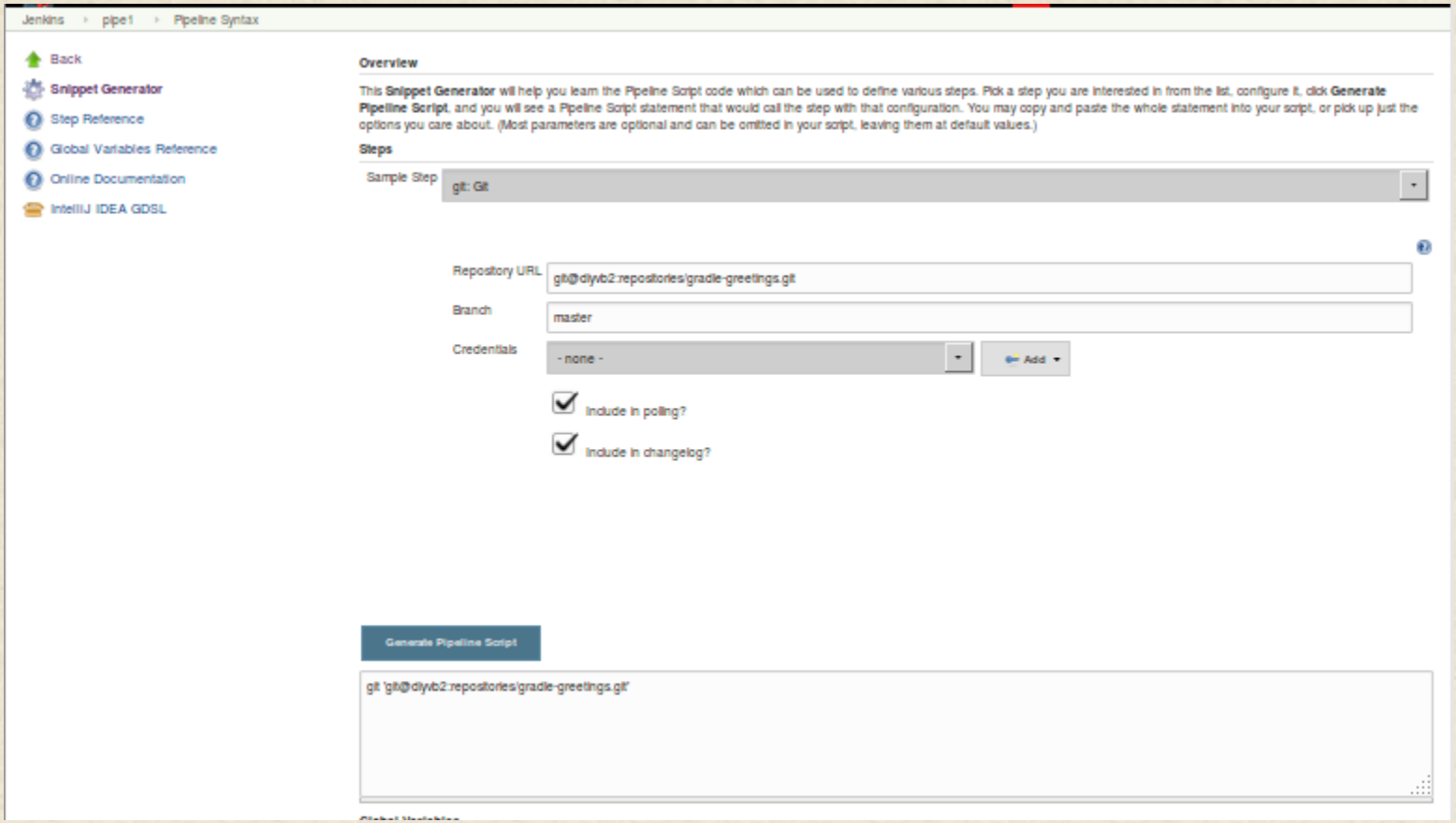
```
[Pipeline] sh
[test-script] Running shell script
+ /usr/share/gradle/bin/gradle clean build
Starting a Gradle Daemon (subsequent builds will be faster)
:clean UP-TO-DATE
:compileJava
:processResources UP-TO-DATE
:classes
```

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'**

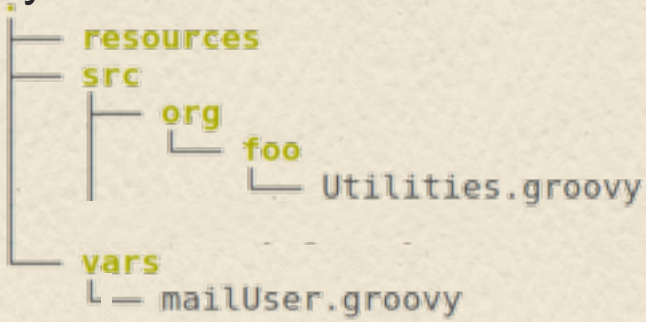
Snippet Generator

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



Shared Pipeline Libraries

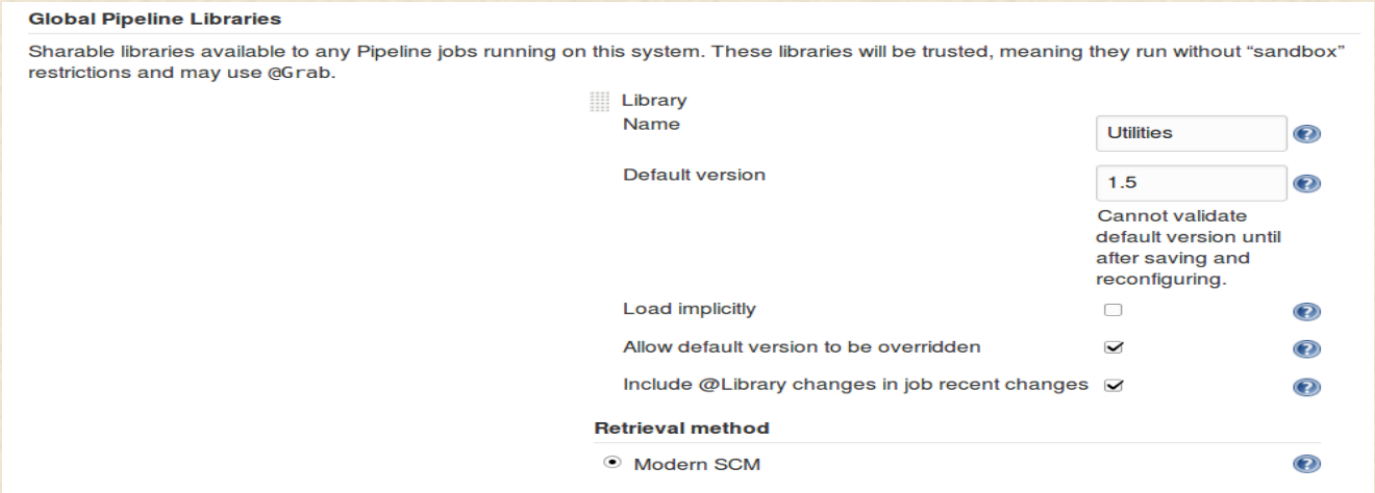
- Allows you to create a shared SCM repository of pipeline script code and functions.
- General mode: Any SCM repository (“external library”)
- Repository Structure



  - 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

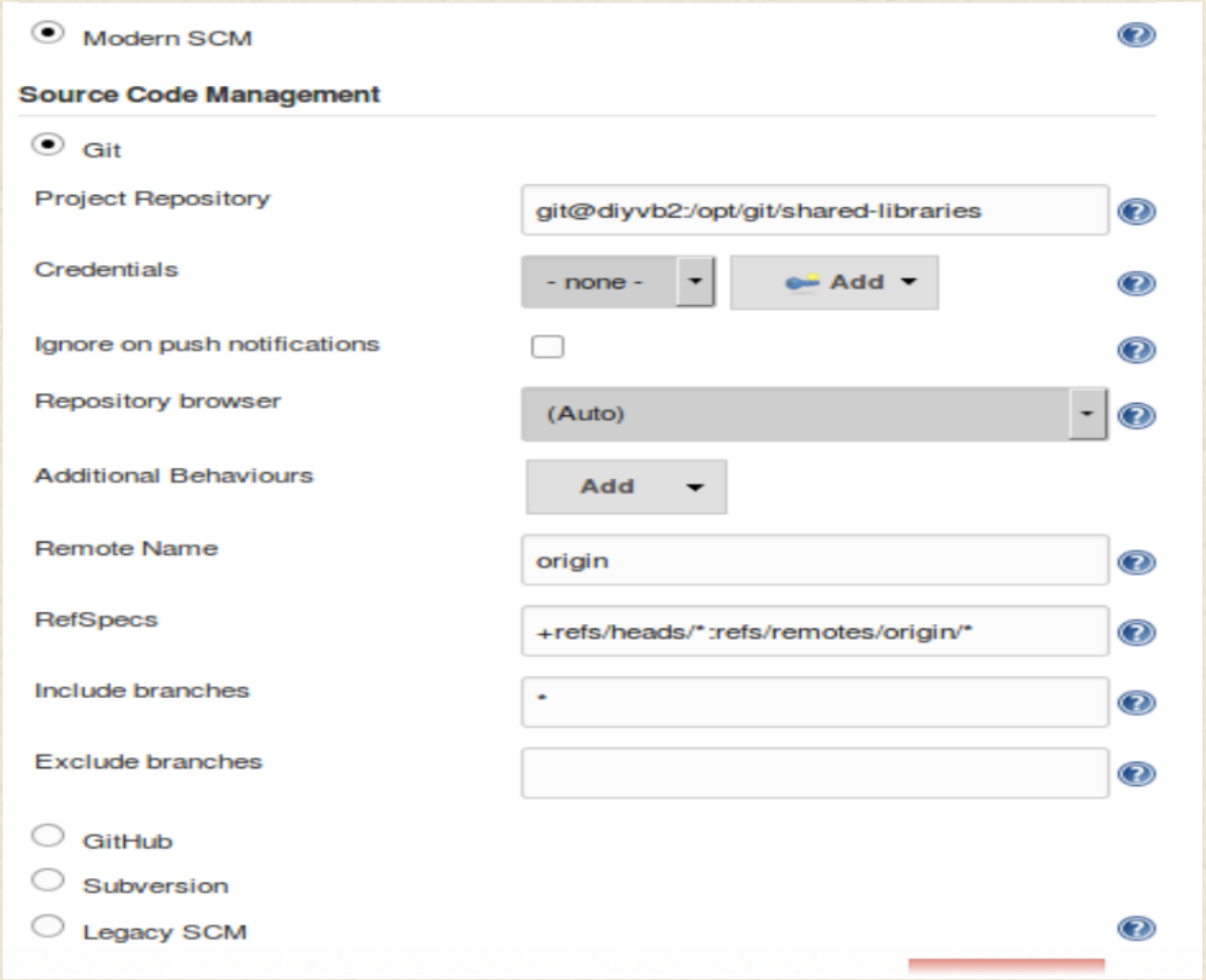
Defining Global Shared External Pipeline Libraries in Jenkins

- Defined in Manage Jenkins->Configure System



- 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

Global Shared External Pipeline Libraries SCM Configuration Example - Git



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

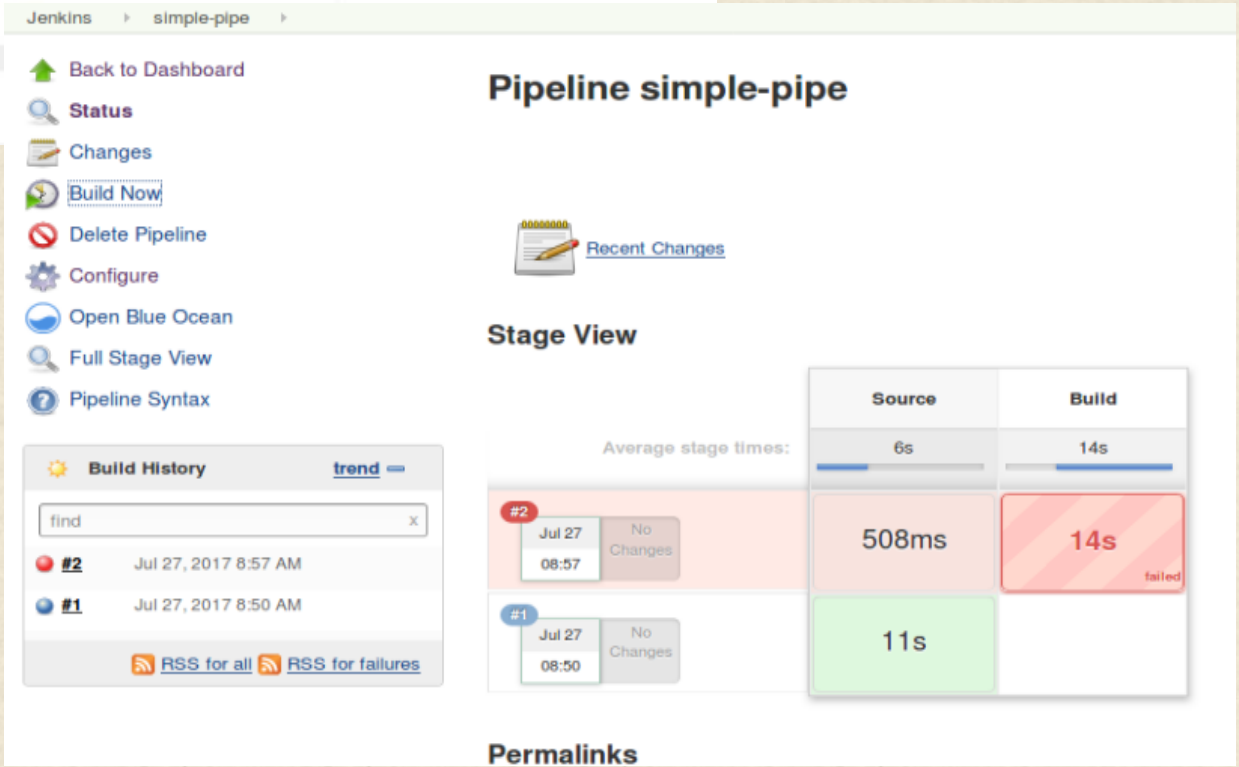
- Aggregates build steps into sections
- Stages are inside of a node block
- Stages take a name (usually corresponding to the function)

```
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'
   }
```

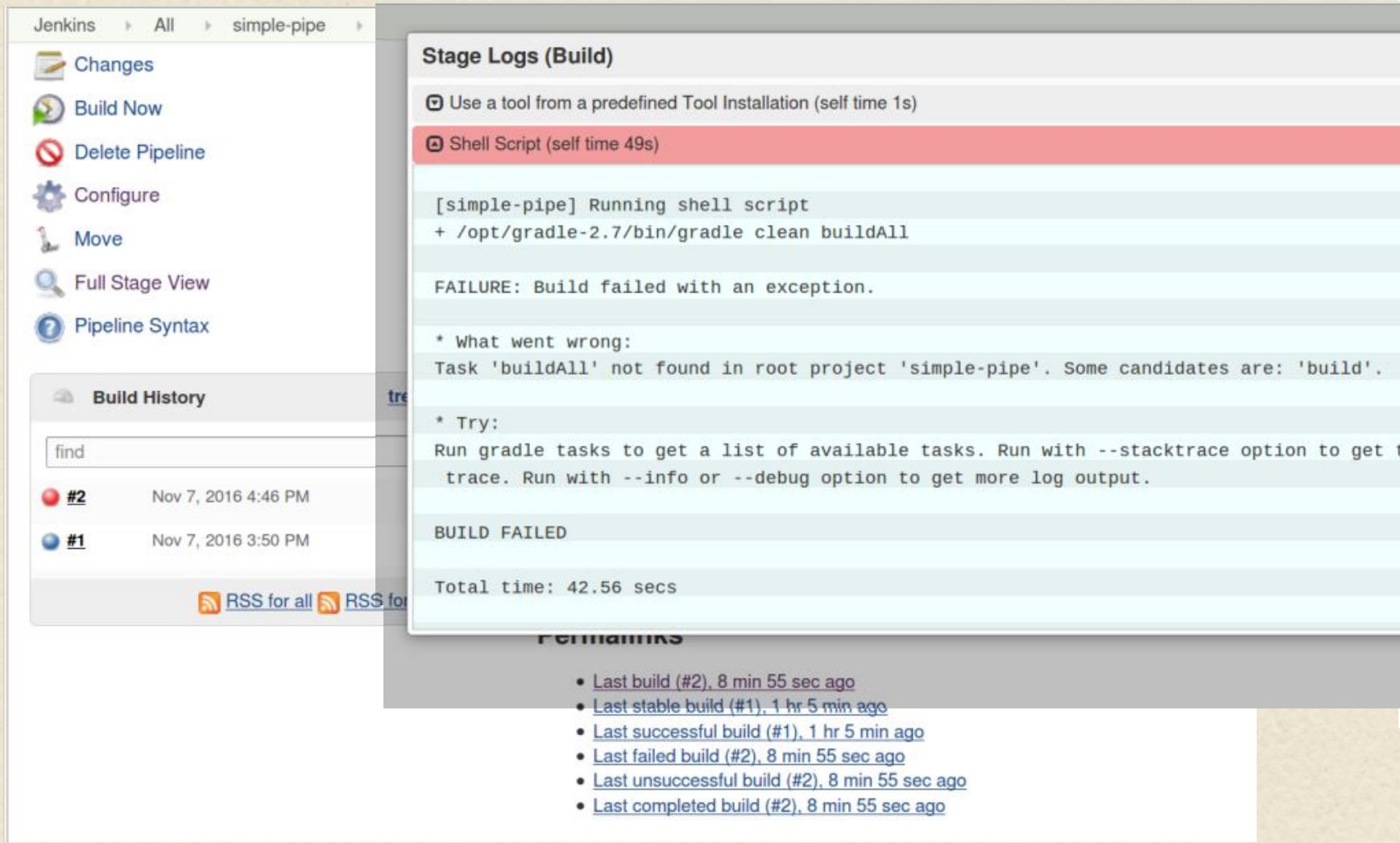
```
   stage('Results') {
      junit '**/target/surefire-reports/TEST-*.xml'
      archive 'target/*.jar'
   }
```

Stage View


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

   }
   stage('Build') {
      // Run the gradle build
      gradleHome = tool 'gradle27'
      sh "'${gradleHome}/bin/gradle' clean buildAll"
   }

}
```

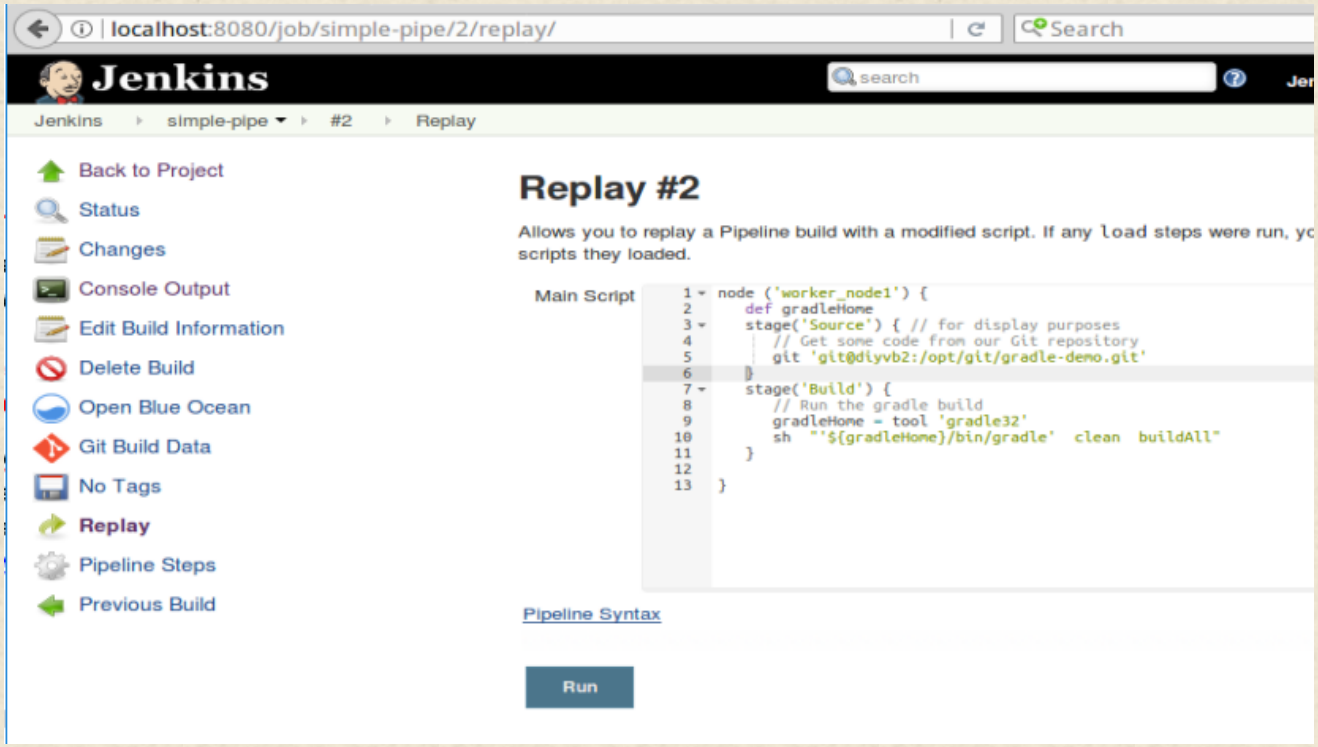


Stage View and Logs



Replay - try out a change to a previous run

- 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



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 specificed, 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/worksp
[Pipeline] [worker2] node
[Pipeline] [master] {
[worker2] Running on worker_node2 in /home/jenkins/worker_node2/workspac
[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'.
```

Alternate Parallel Syntax – Declarative Pipeline 1.2

- Elevates the parallel step to a separate construct within a stage
- Branches can be defined as stages instead of map elements
- More consistent with declarative syntax

```
stage('Unit Test') {
          parallel{
              stage ('Util unit tests') {
                 agent { label 'worker_node2' }
                 steps {
                    cleanWs()
                    unstash 'ws-src'
                    gbuild4 ':util:test'
                 }

              }
              stage ('API unit tests set 1') {
                 agent { label 'worker_node3'}
                 steps {
                    // always run with a new workspace
                    cleanWs()
                    unstash 'ws-src'
                    gbuild4 '-D test.single=TestExample1* :api: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


```
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([[$class: 'UsernamePasswordMultiBinding', credentialsId: 'gitcreds',
13                        usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PASS']]) {
14
15         sh "git tag -a ${env.BUILD_TAG} -m 'demostrate push of tags'"
16         sh "git push http://${env.GIT_USER}:${env.GIT_PASS}@${urlRepo} --tags"
17      }
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

Pipeline script

Script

```
1  node {
2      def sshRepodef = "git@diyvb:repos/shared_libraries.git"
3
4      stage ("Get Source") {
5          git url: sshRepodef
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 'demostrate push of tags'"
15             sh "git push ${sshRepodef} --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)

```
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 {
                        compileClasspat
                        runtimeClasspat
                        srcDir file('sr
                }
                resources.srcDir file('
        }
}

configurations {
        integrationTestCompile.extendsF
        integrationTestRuntime.extendsF
}

configurations {
        functionalTestCompile.extendsFr
        functionalTestRuntime.extendsFr
}
```

```
tasks.withType(Test) {
        reports.html.destination = file("${reporting.baseDir}/${name}")
        outputs.upToDateWhen { false }
}

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

}
```

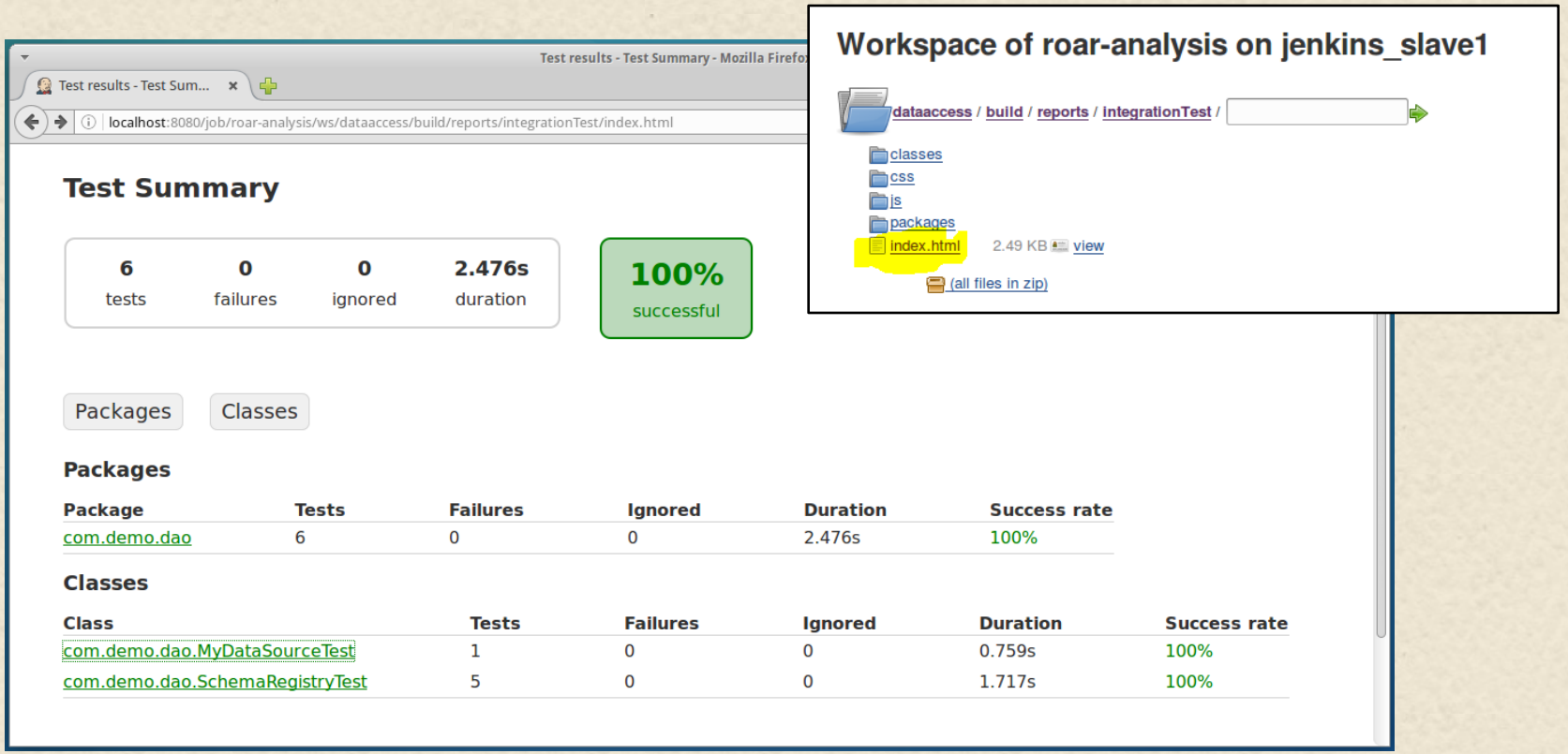
Controlling which database to use


```
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



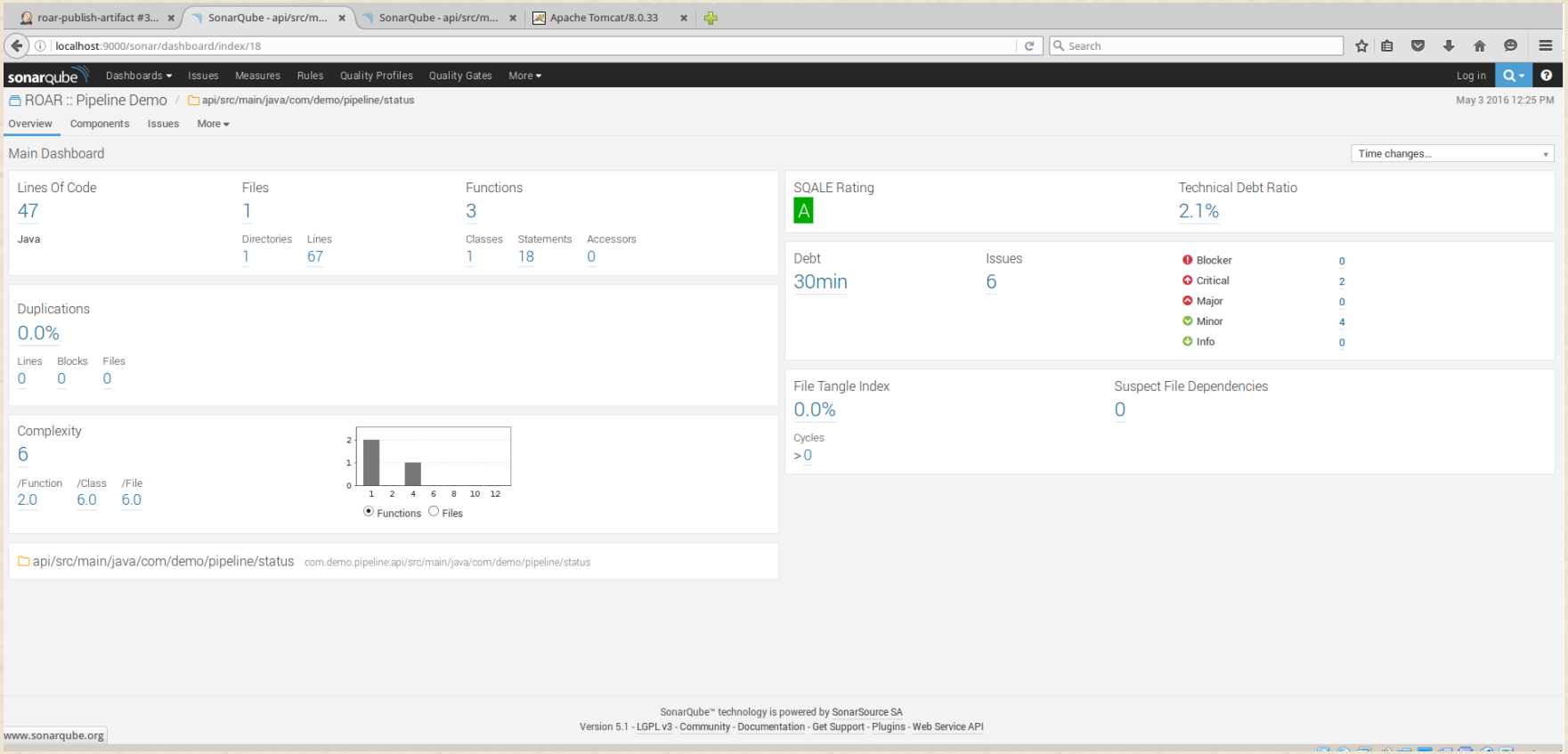
Lab 3 – Adding in testing

Technology - SonarQube (sonarsource.org)

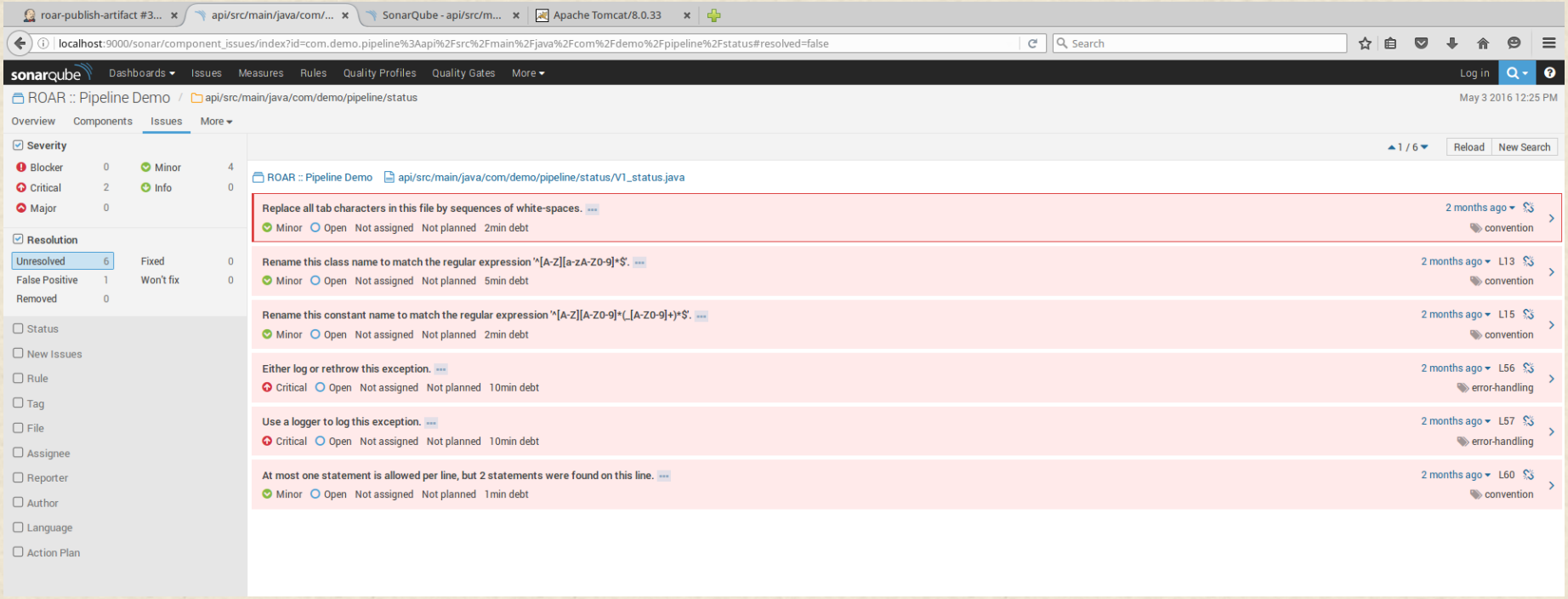
- Open platform for helping manage code quality in 7 areas:
  - Architecture and Design
  - Comments
  - Coding rules
  - Potential bugs
  - Duplications
  - Unit tests
  - Complexity
- Allows configuration of rules, alerts, thresholds, exclusions, etc.
- Extensible via plugins
- Example site: http://nemo.sonarqube.org

Sonar - drilling in

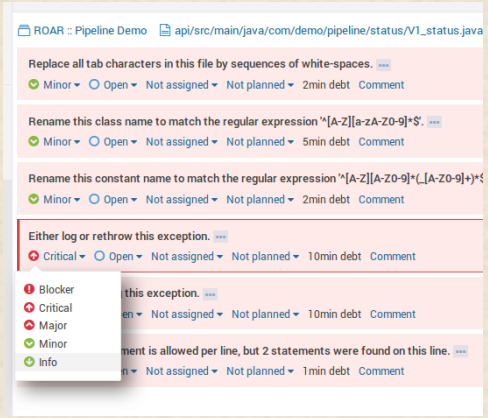
http://localhost:9000

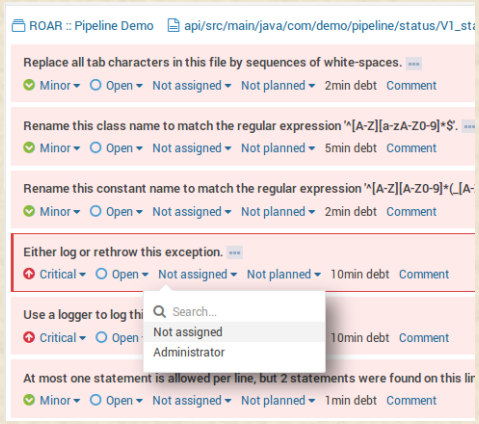


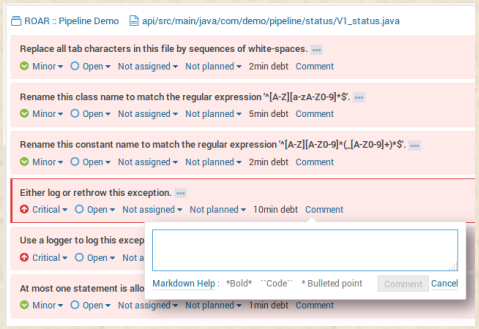
Sonar - looking at issues

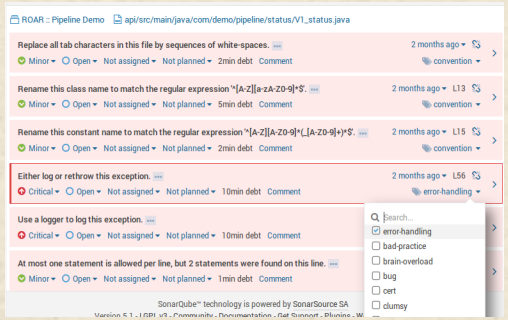


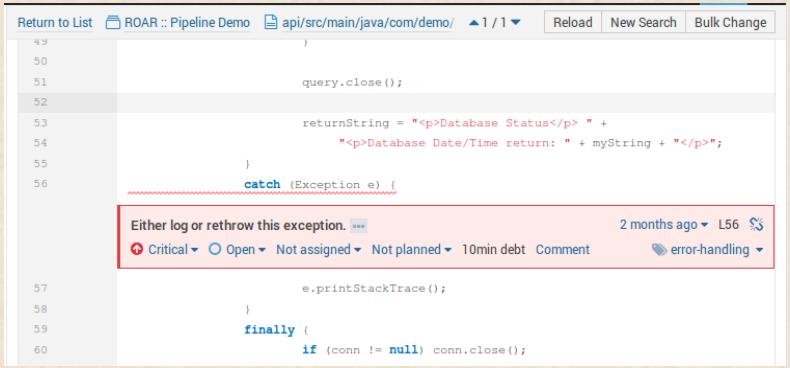
Sonar - responses



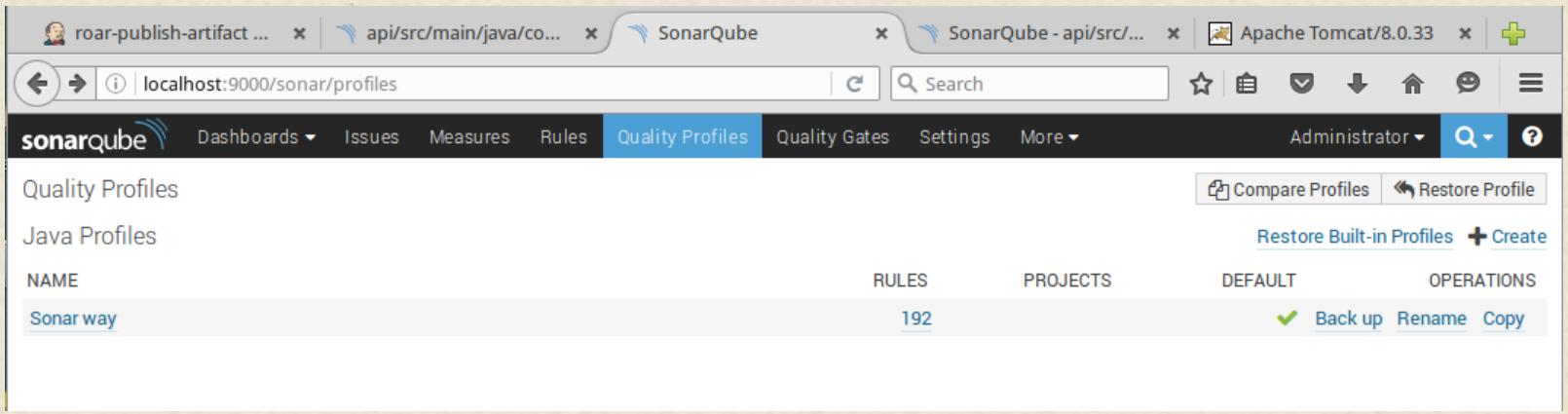




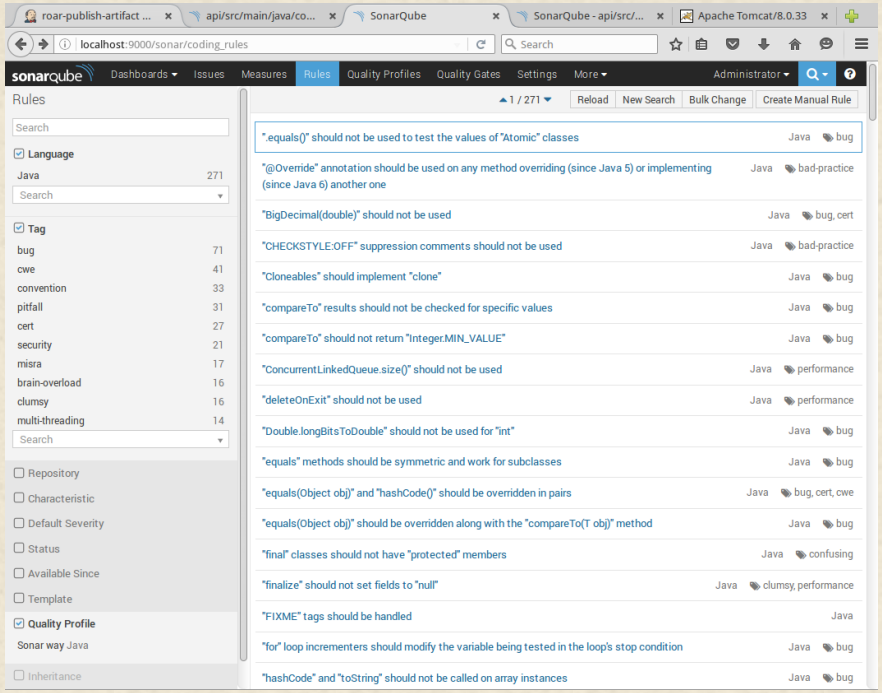


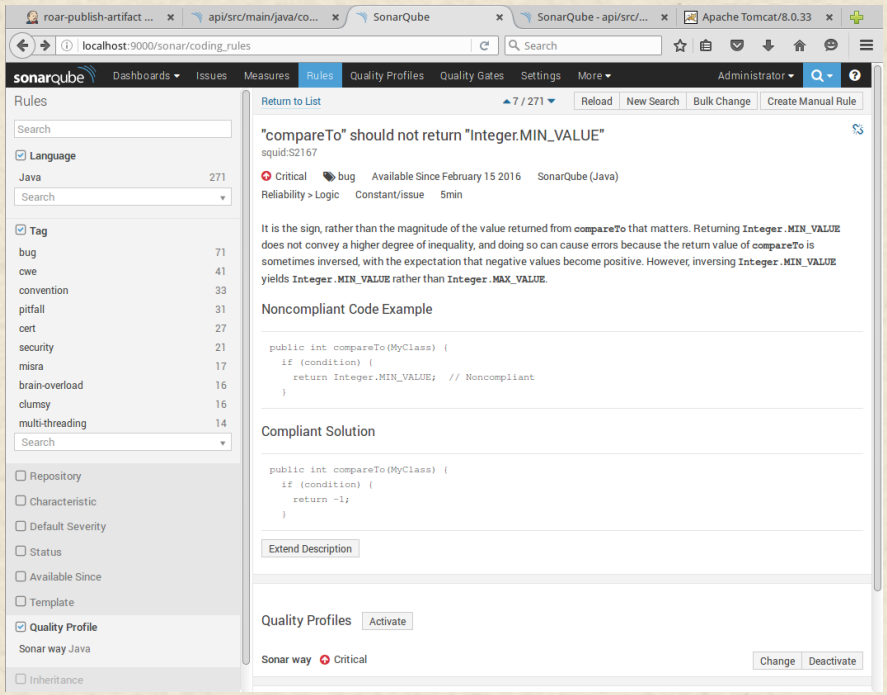


Sonar - Quality Profiles

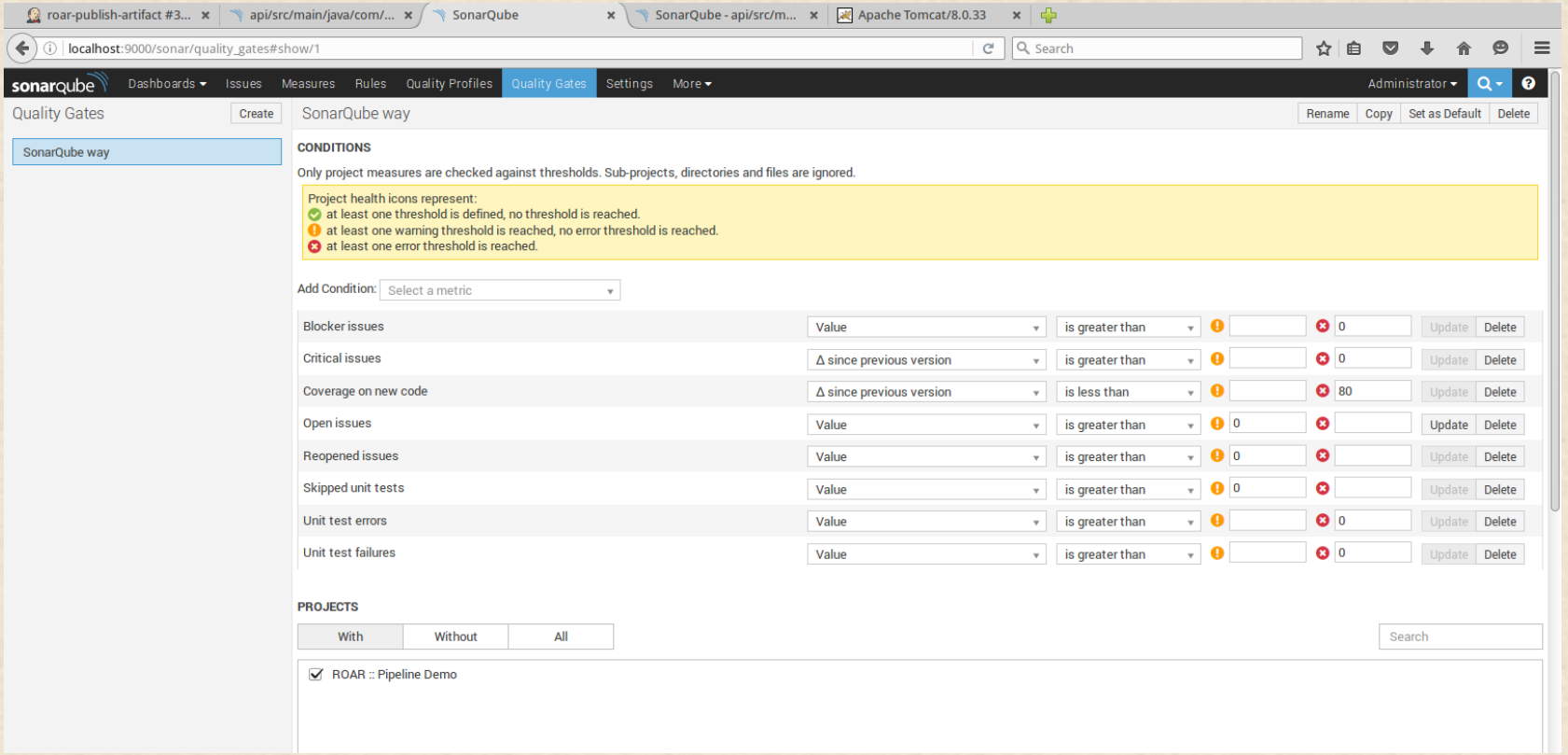


Sonar - rules





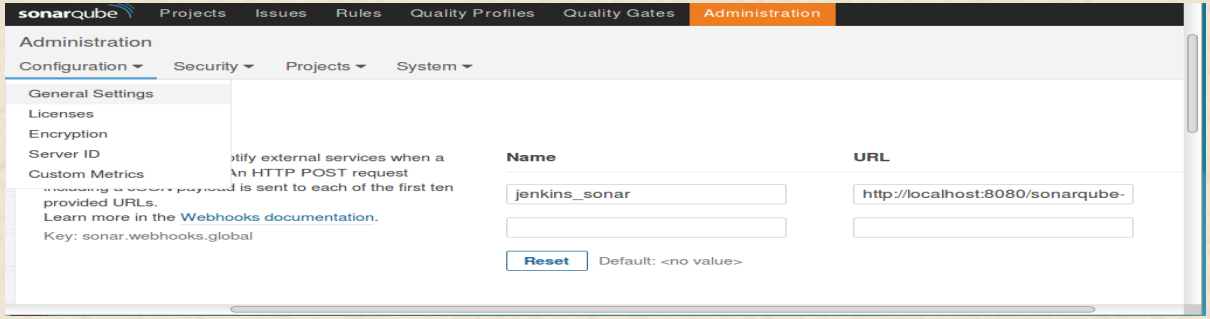
Sonar - quality gates



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
- 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)
- 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)
- 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 gernerate 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));
                        }
```

dataaccess > com.demo.dao > SchemaRegistry

SchemaRegistry

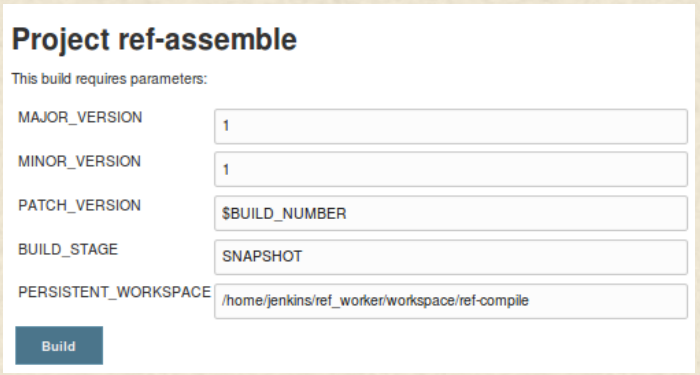
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
queryReturnSpeciesNameAgents(String, String)		54%		12%	4	5	6	19	0	1
queryReturnSpeciesAgents(String)		54%		12%	4	5	6	18	0	1
insertIntoAgents(String, String, String, String, String, String)		68%		17%	3	4	3	16	0	1
updateAgentLastServiceDate(String, String)		57%		17%	3	4	3	12	0	1
deleteFromAgents(String, String)		56%		17%	3	4	3	13	0	1
SchemaRegistry()		100%		n/a	0	1	0	2	0	1
Total	136 of 330	59%	29 of 34	15%	17	23	21	80	0	6

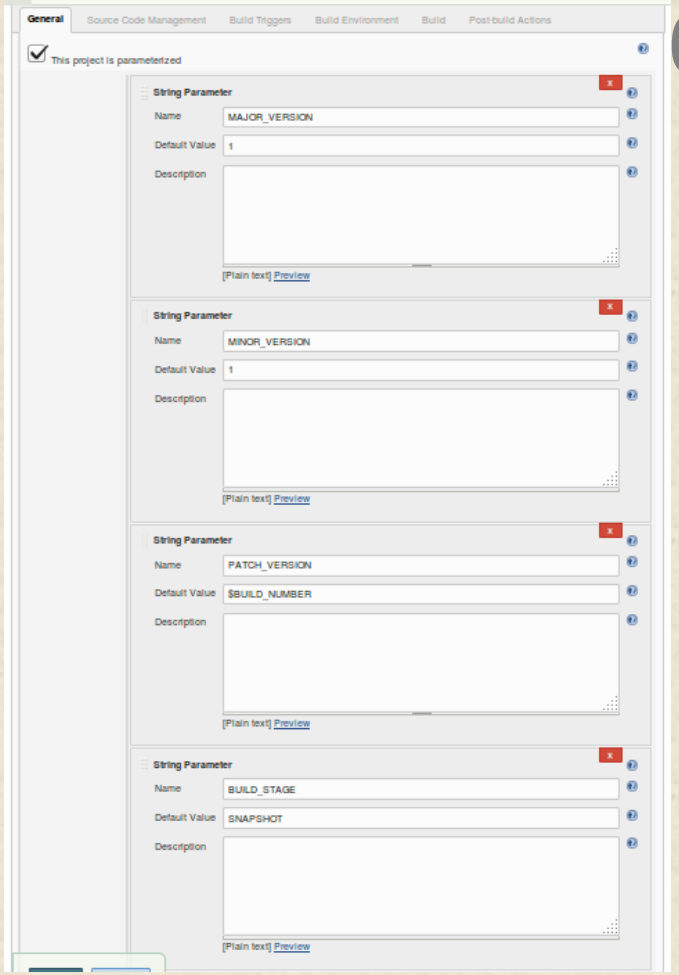
```
step([$class: 'JacocoPublisher',
        execPattern:'**/**.exec',
        classPattern:
'**/classes/main/com/demo/util,**/classes/main/com/demo/dao',
        sourcePattern:
'**/src/main/java/com/demo/util,**/src/main/java/com/demo/dao',
        exclusionPattern: '**/*Test*.class'])
```

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>”).

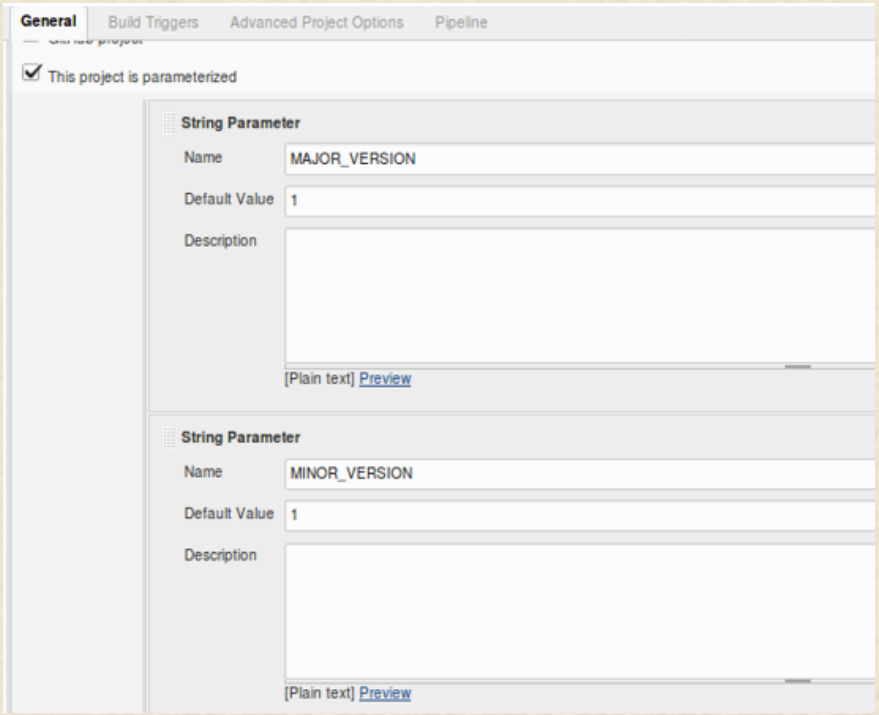





```
// if we have the semantic versions values set then use those

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




```
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
           baseName = "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’
                }
        }
}
```

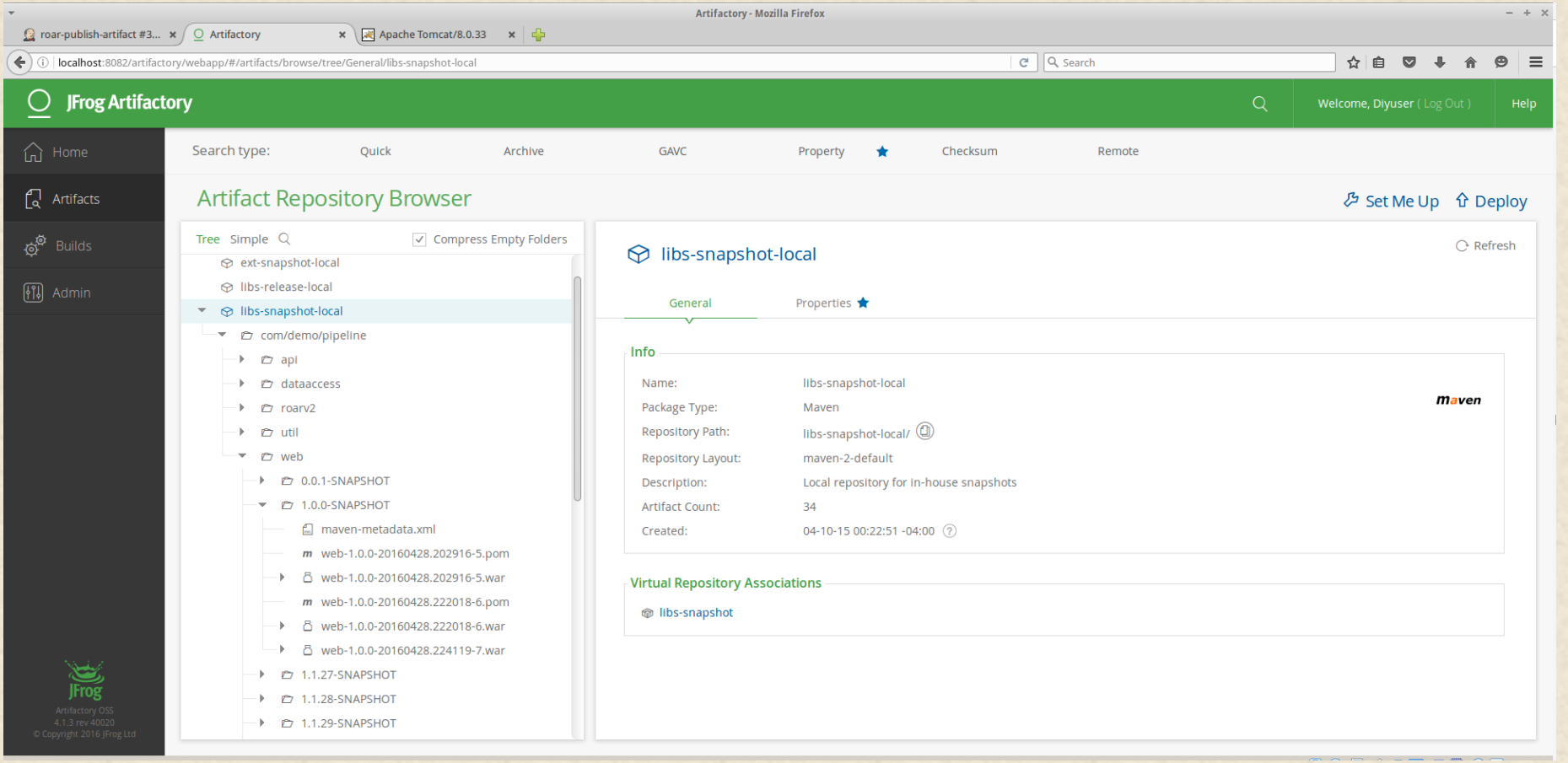
- Note: Can define an artifact using an archive task

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

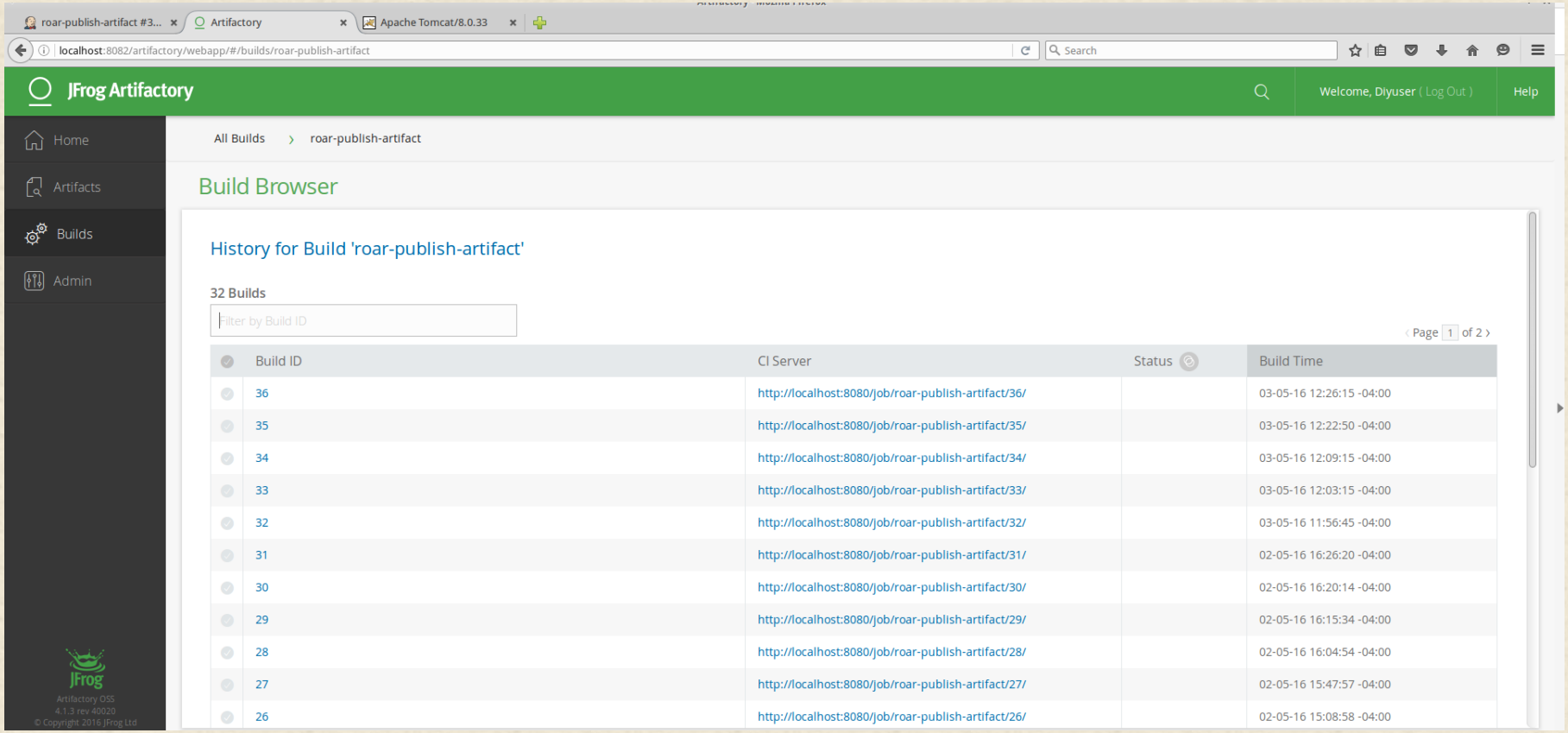
What is Artifactory?

- 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

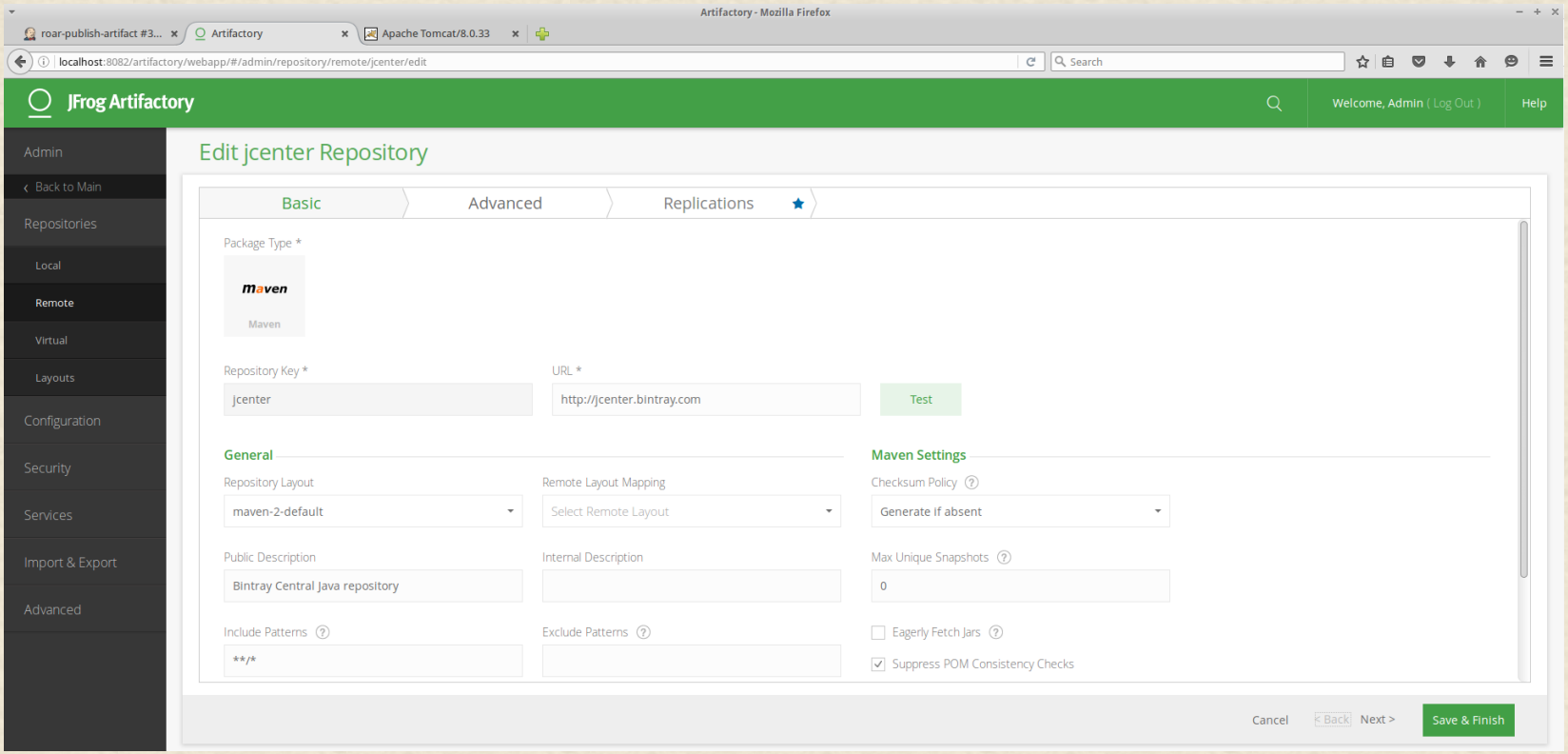
Artifactory - Artifacts



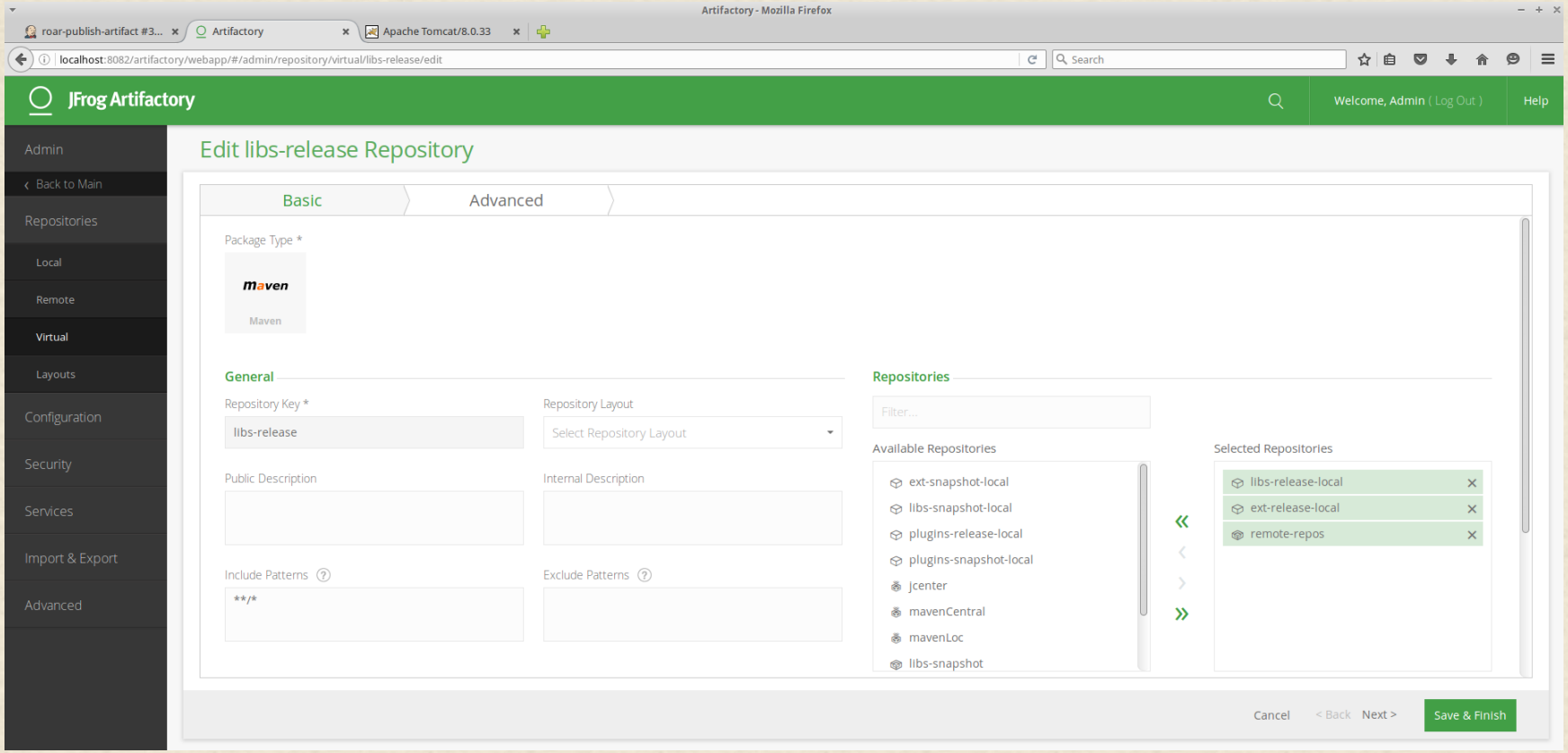
Artifactory - Builds



Artifactory - Editing/Defining Repositories



Artifactory - Virtual Repository

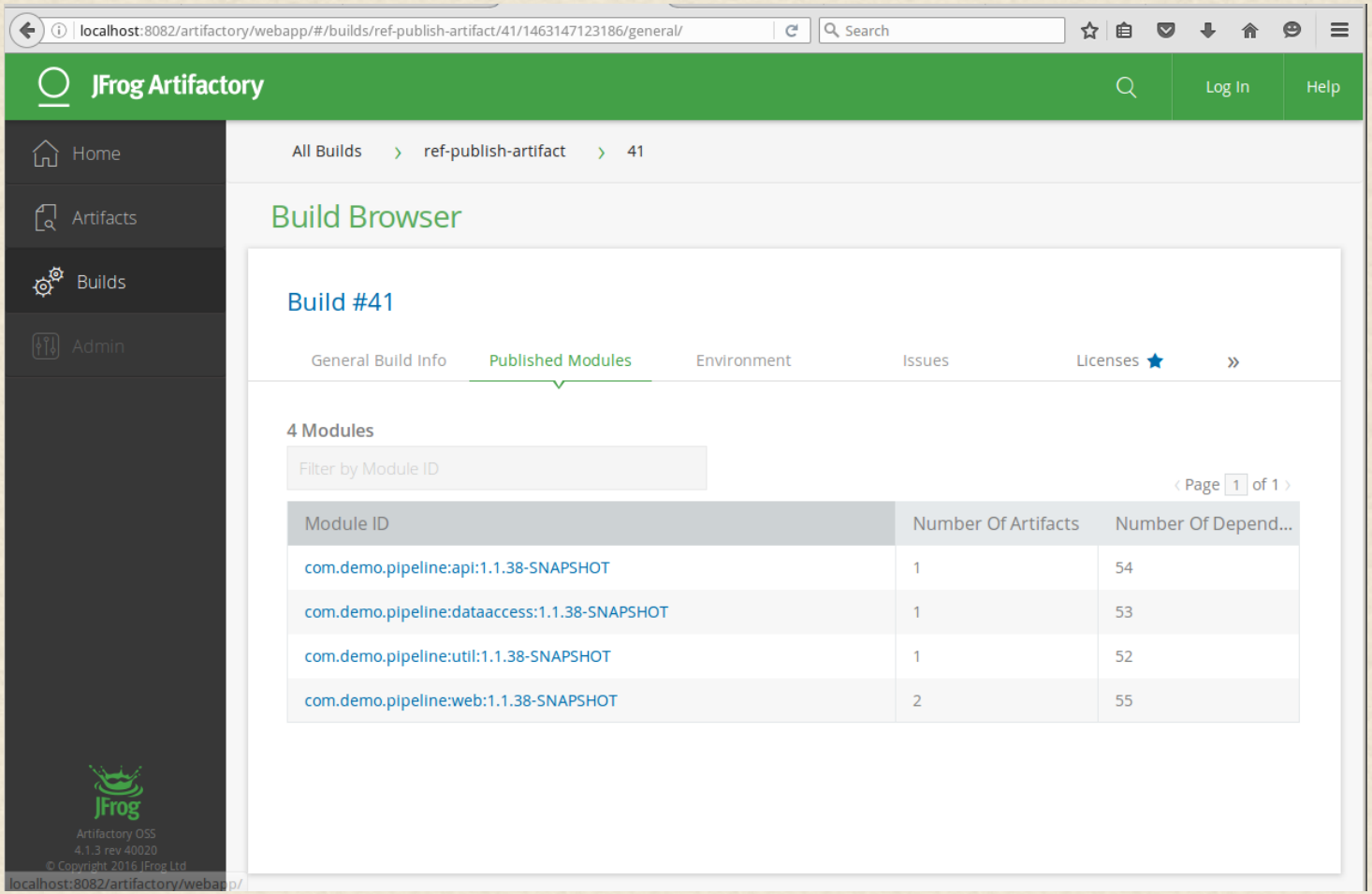


Artifactory Integration with Jenkins (and Gradle)

- **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

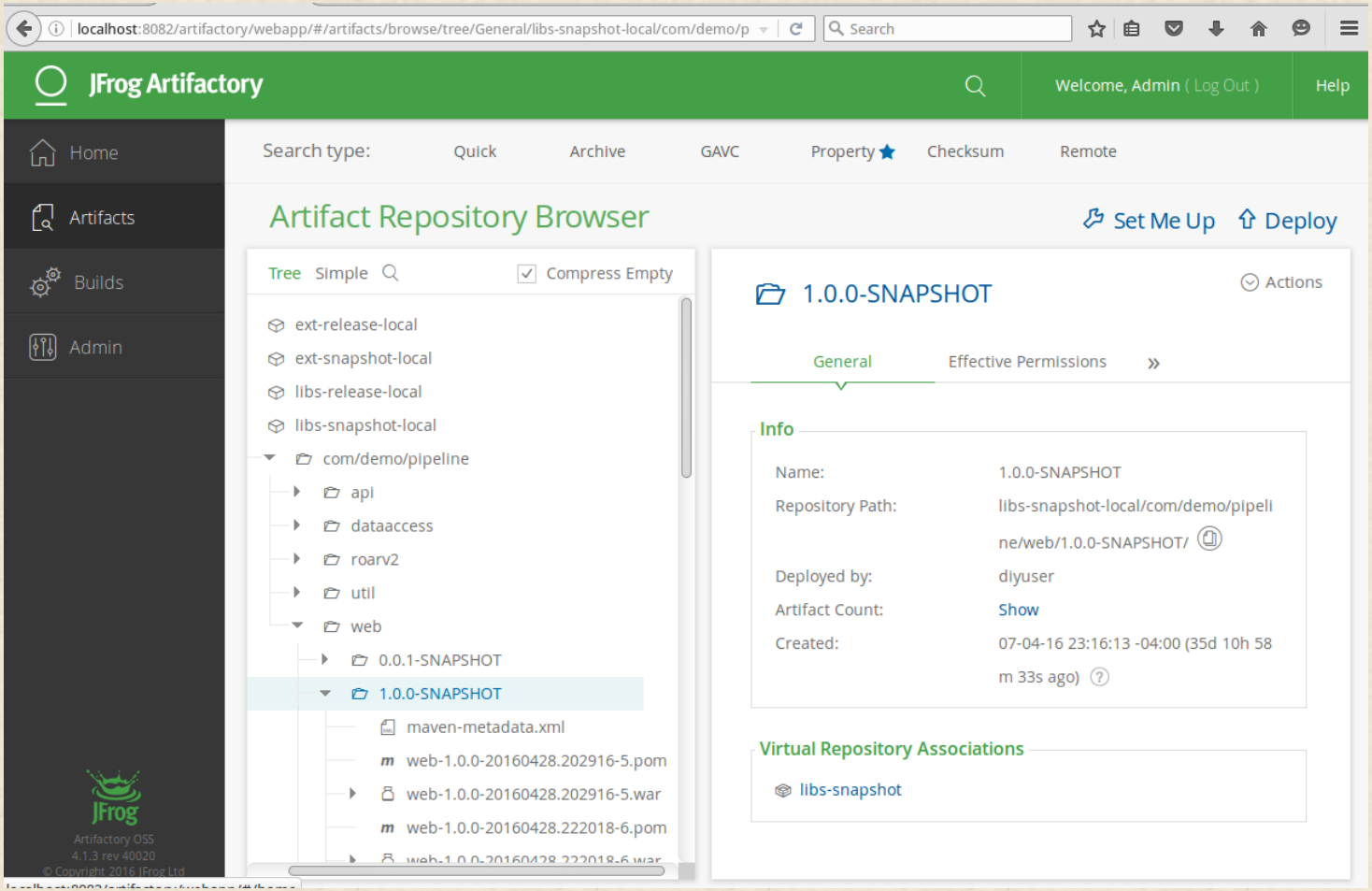
Build info in Artifactory


```
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
```



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
```



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

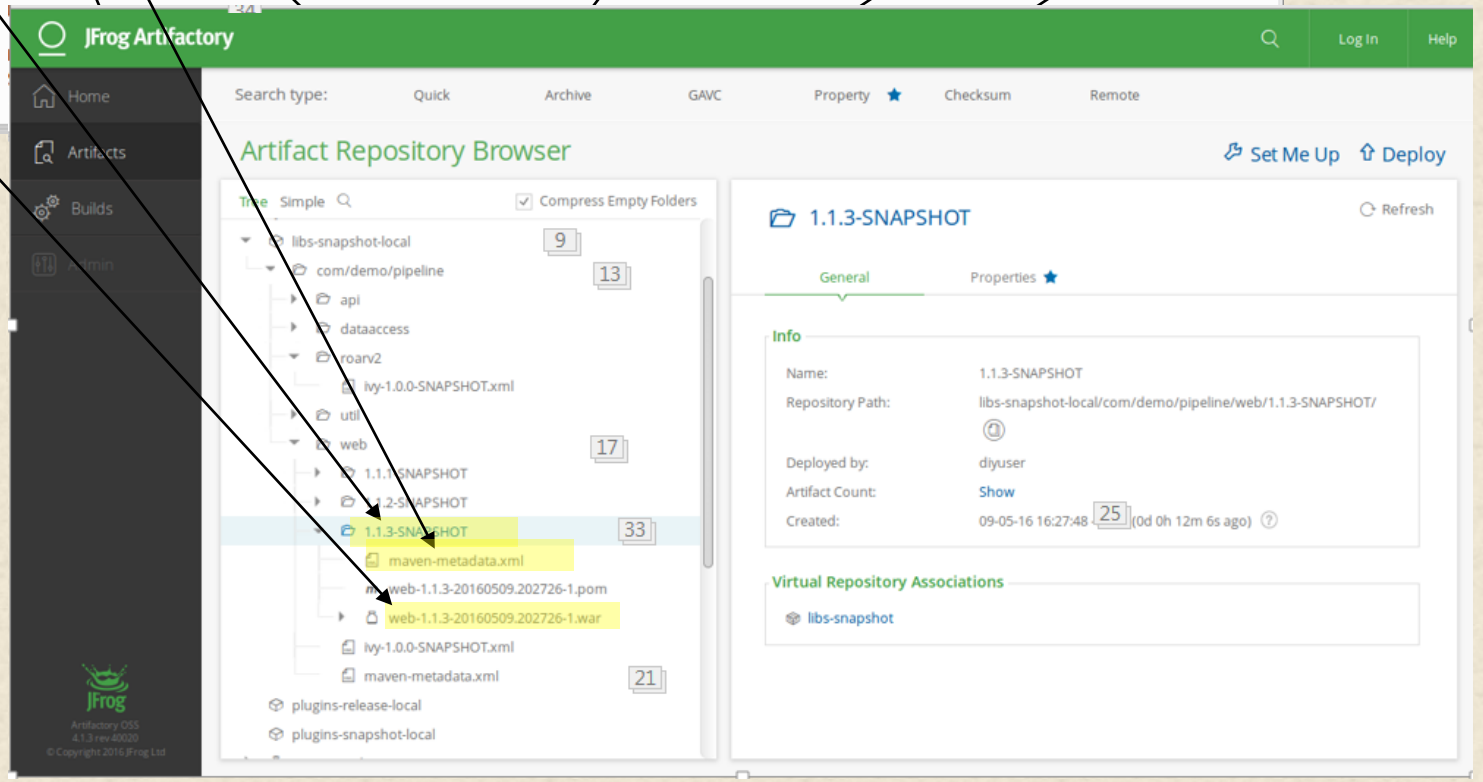
Jenkins Build Step

```
# 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>\([^<]*\)<\/latest>.*/\1/"`
build=`curl -s $path/$version/maven-metadata.xml | grep '<value>' | sort -t- -k2,2nr | head -1 | sed "s/.*<value>\([^<]*\)<\/value>.*/\1/"`
war=$name-$build.war
url=$path/$version/$war

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

cleanup

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

version = **1.1.3-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)

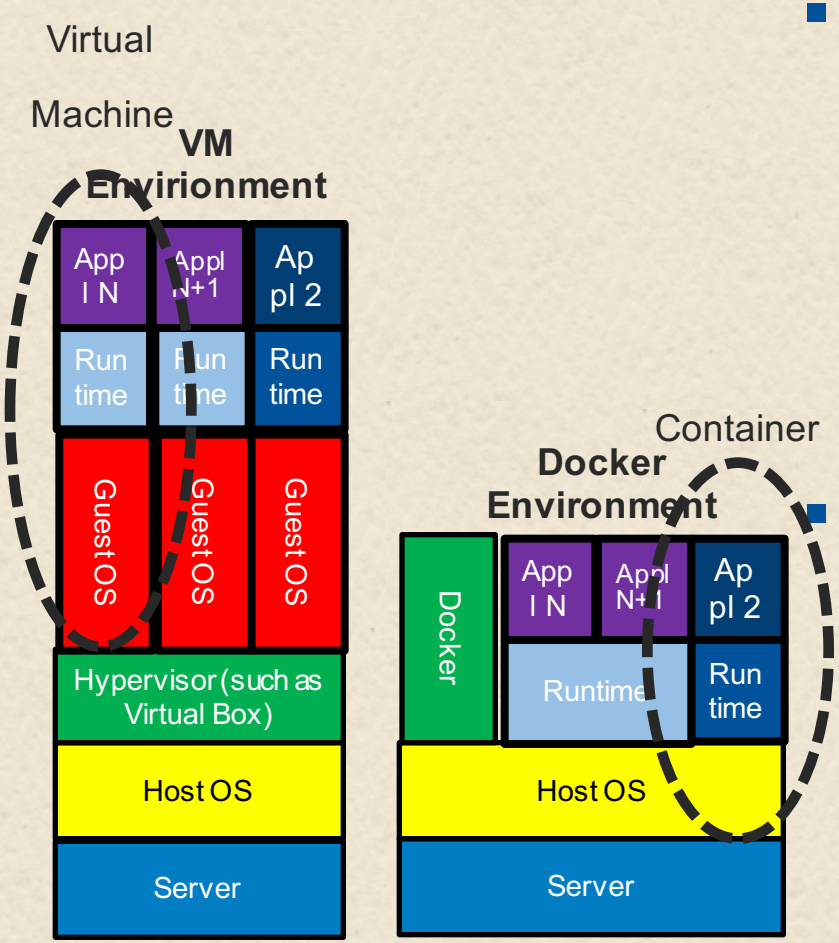
  Thin wrapper around Linux Container technology

  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

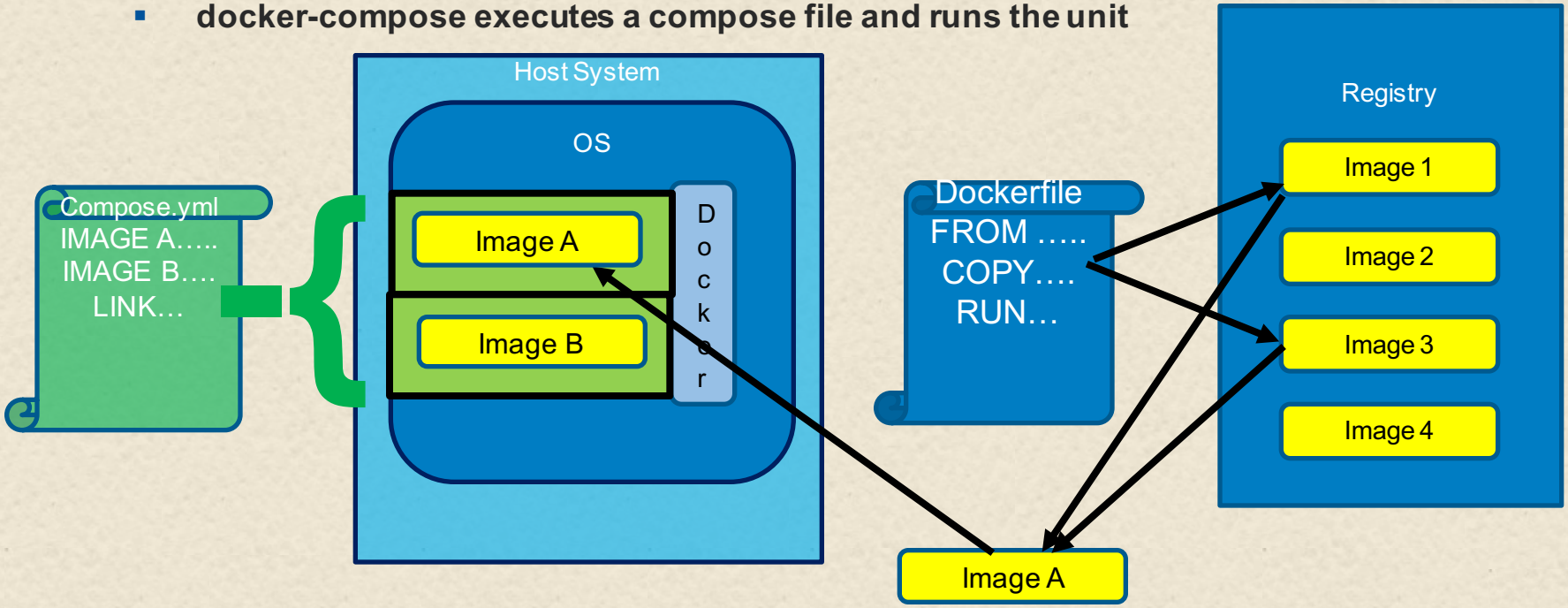
How Containers from Docker differ from VM



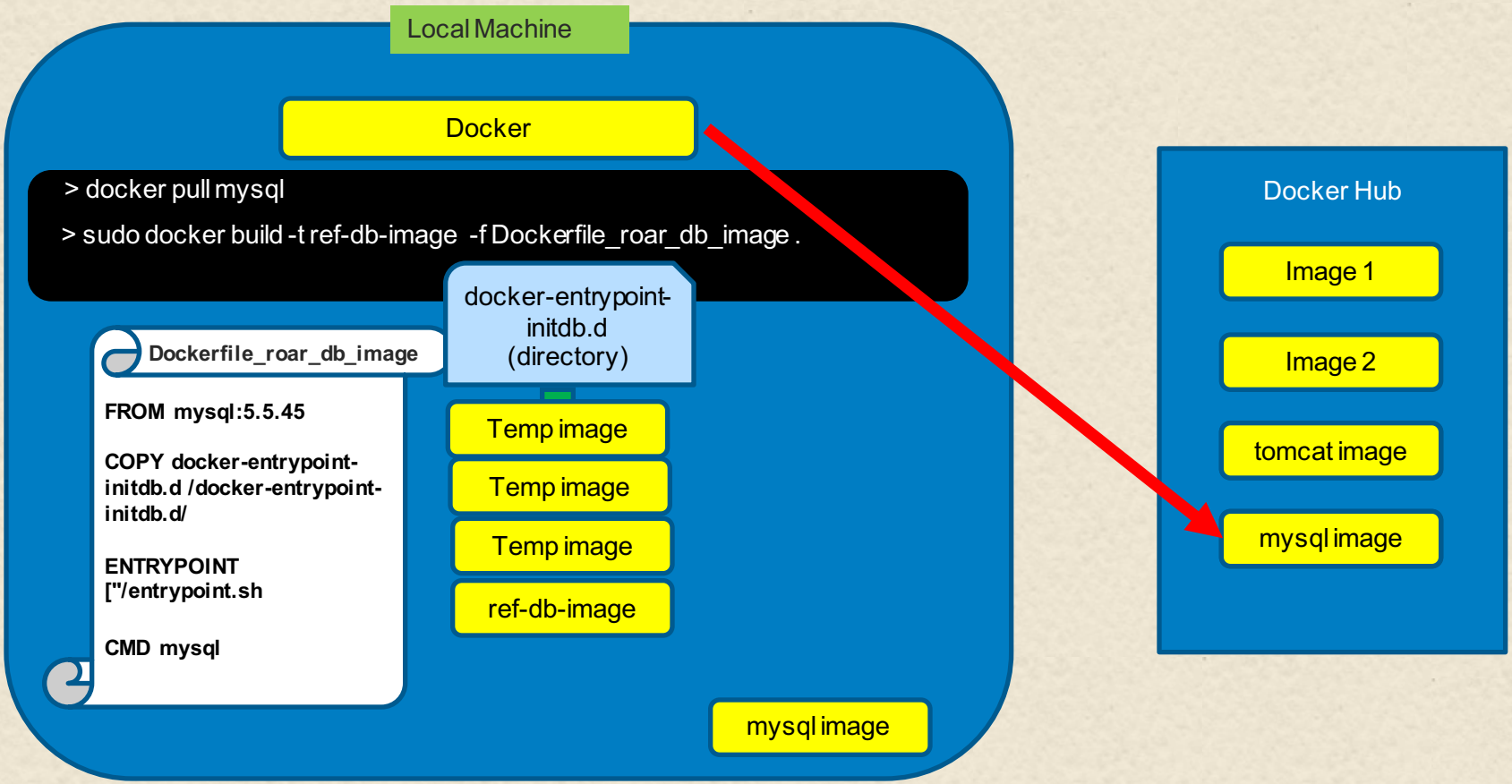
- A VM requires a Hypervisor and a Guest OS to create isolation; Docker uses Linux container technologies to run processes in separate spaces on the same OS
- Because it doesn't require the Hypervisor and a Guest OS, Docker is:
  - Faster to startup
  - More portable (can run an image unchanged in multiple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

Docker Flows

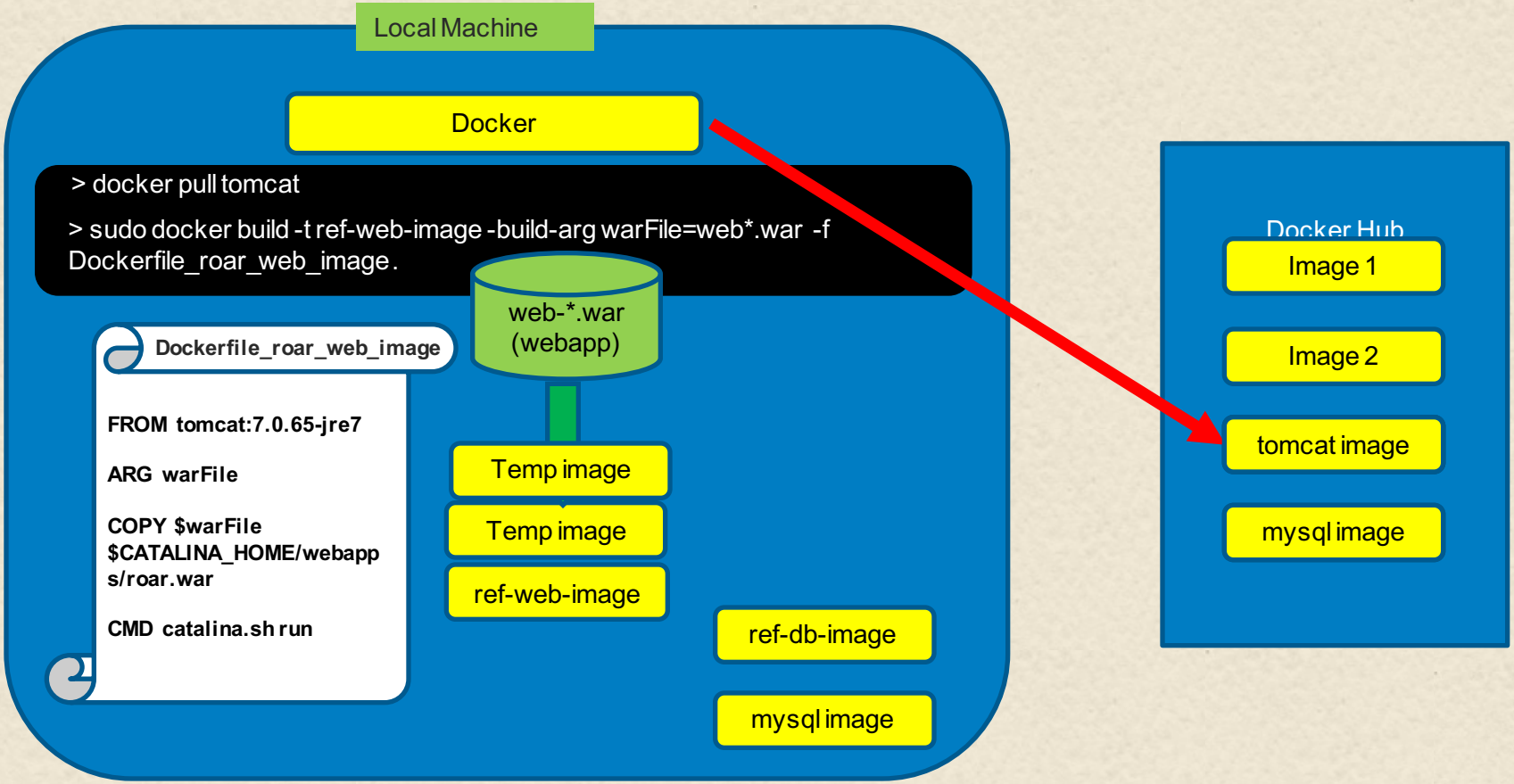
- Docker images are stored in a registry
- A Dockerfile describes how to create a new image
- docker build creates the new image
- docker run starts up a container with an image
- A docker compose file describes how to create containers and link multiple ones together
- docker-compose executes a compose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



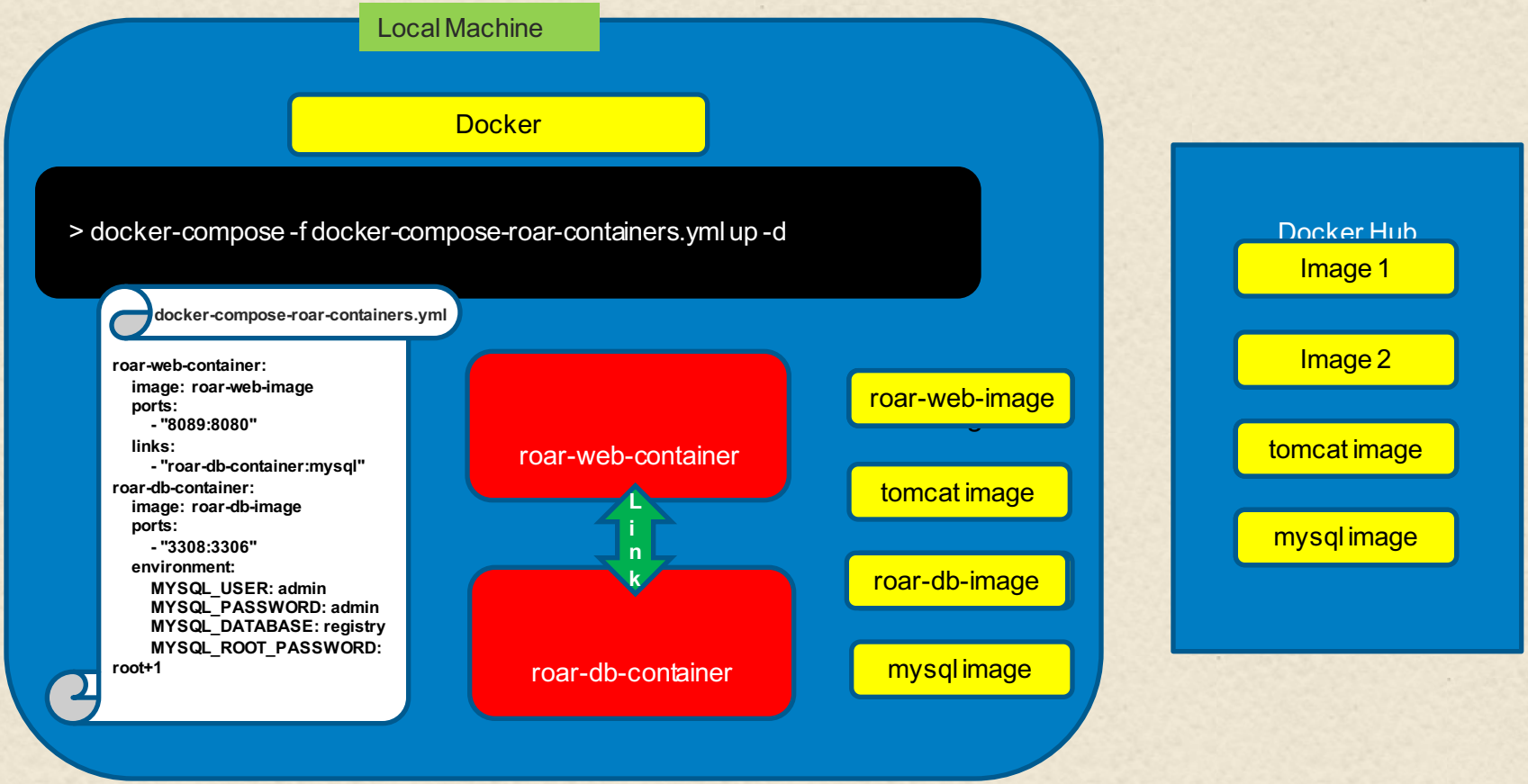
Creating the docker database image



Creating the web image

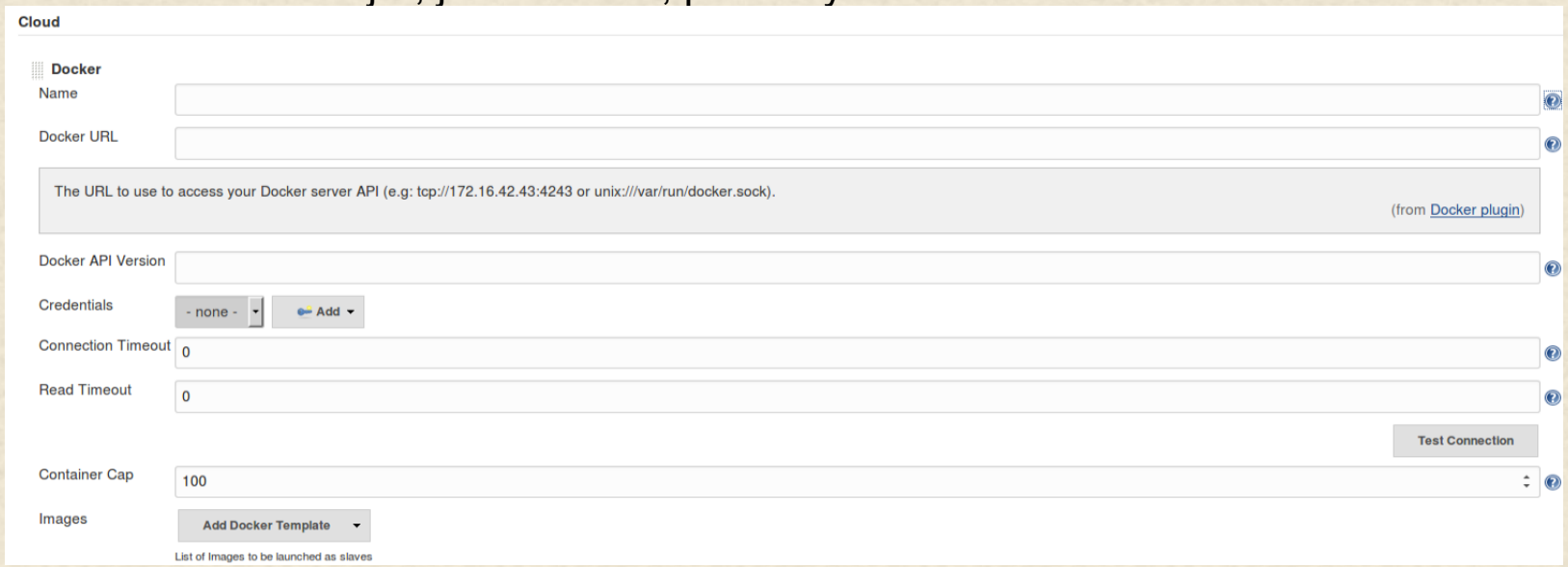


Composing the combined images and containers



Jenkins 2 and Docker

- 4 ways of using Docker via Jenkins
  - Configured as a “cloud” - standalone Jenkins agent
  - As declarative 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



Jenkins 2 and Docker

- 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/j
/jobs/docker-test2/workspace:rw -v /var/lib/jenkins/jobs/docker-test2/workspace@tmp:/var
******** -e ******** -e ******** -e ******** -e ******** -e ******** -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
```

Jenkins 2 and Docker - withDockerContainer

- 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

- **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 roo 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.

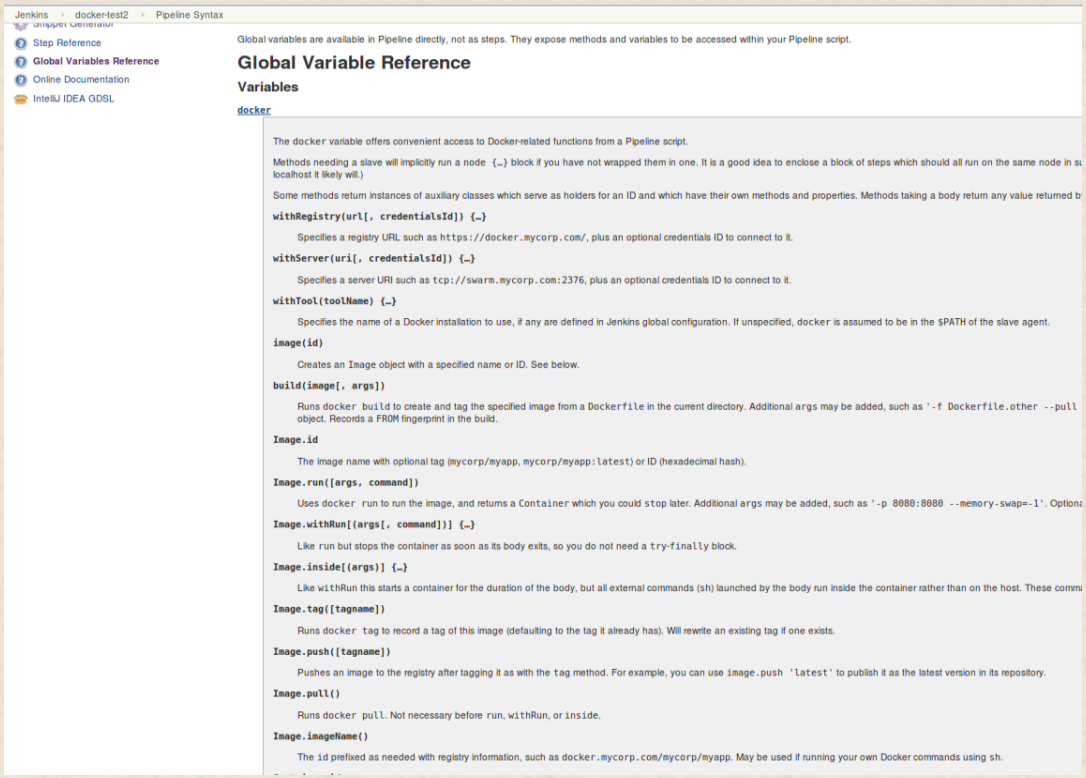
```
agent {
    docker {
        image "image-name"
        label "worker-node"
        args "-v /dir:dir"
    }
}
```

```
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




```
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/dockerResour
Fetching upstream changes from git@diyvb:repos/dockerResourc
 > git --version # timeout=10
 > git fetch --tags --progress git@diyvb:repos/dockerResourc
 > git rev-parse refs/remotes/origin/master^{commit} # timeo
 > git rev-parse refs/remotes/origin/origin/master^{commit}
Checking out Revision 742b984c53e96e7d1465d9442af6c6606757e8
 > 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 742b984c53e96e7d1465d9442af6c66067
 > git rev-list 742b984c53e96e7d1465d9442af6c6606757e845 # t
[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@nclasters.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/di
bin.zip"  && ln -s "/opt/gradle-${GRADLE_VERSION}/bin/gradle
 ---> Using cache
 ---> df50ff638f0d
Step 5 : ENV GRADLE_HOME /opt/gradle
```

Jenkins 2.0 and Docker - Running via shell


```
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
```

Lab 7 – Deploy to Docker

That's all - thanks!

Professional Git 1st Edition

by Brent Laster (Author)

3 customer reviews

Look inside