Important Prerequisites and Setup

- Have VirtualBox installed and ready

https://www.virtualbox.org/

- Download VM from

https://www.dropbox.com/s/11saxbkarayrs3d/cdv3.ova?dl=0

- See instructions at

https://github.com/brentlaster/safaridocs/raw/master/cd-setup.pdf

Getting Started with Continuous Delivery

Brent Laster

About me

- Director, R&D
- Part-time trainer
- Git, Gerrit, Gradle, Jenkins
- Author - Professional Git, Jenkins 2 – Up and Running, opensource.com
- https://www.linkedin.com/in/brentlaster
- @BrentCLaster

Book – Professional Git

- Extensive Git reference, explanations, and 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

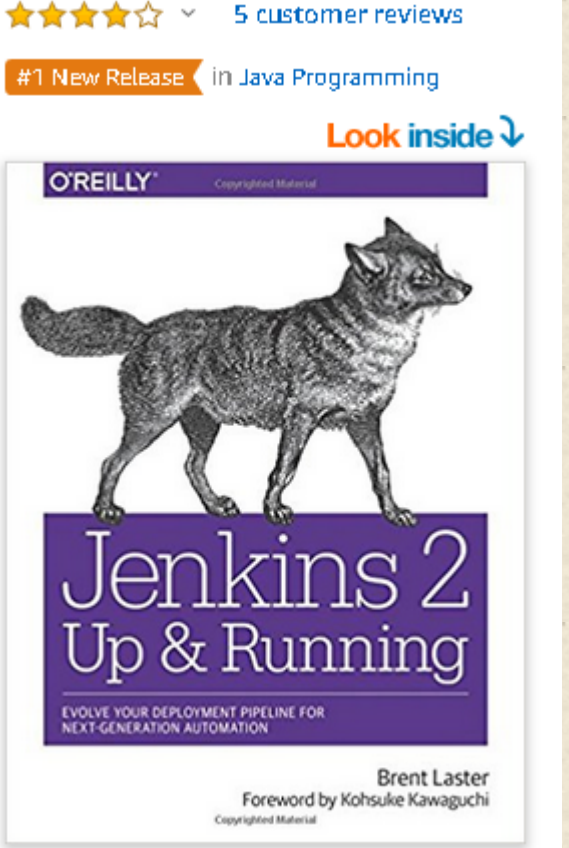
February 23, 2017

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.

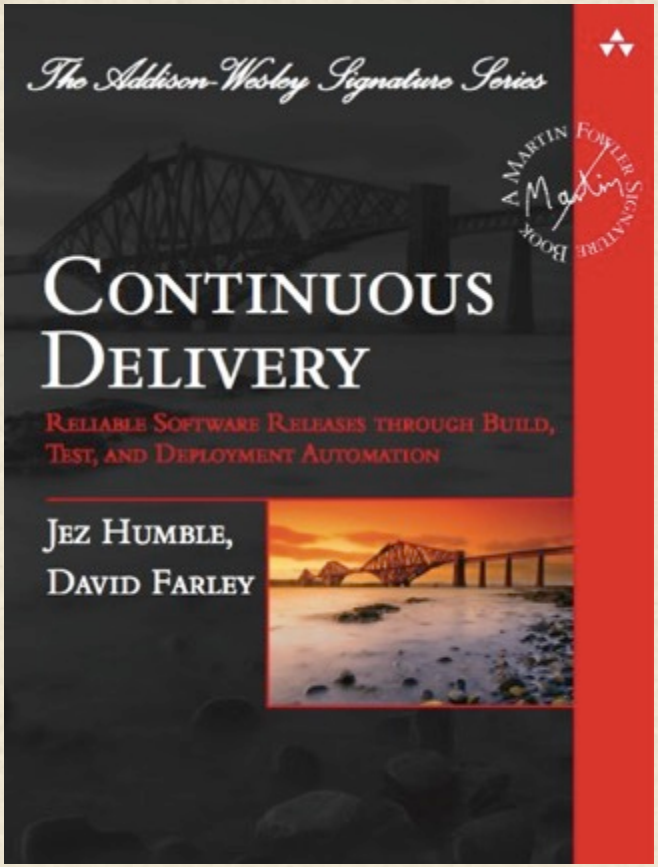
★★★★★ **A great resource**

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!

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."

The Big Picture

- An assembly line in a factory produces consumer goods from raw materials in a fast, automated, reproducible manner.
- Similarly, a software delivery pipeline produces releases from source code in a fast, automated, and reproducible manner.
- The overall design for how this is done is called "continuous delivery."
- The process that kicks off the assembly line is referred to as "continuous integration."
- The process that ensures quality is called "continuous testing“.
- The process that makes the end product available to users is called "continuous deployment."
- The overall efficiency experts that make everything run smoothly and simply for everyone are known as "DevOps" practitioners

Goals and Benefits

- 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

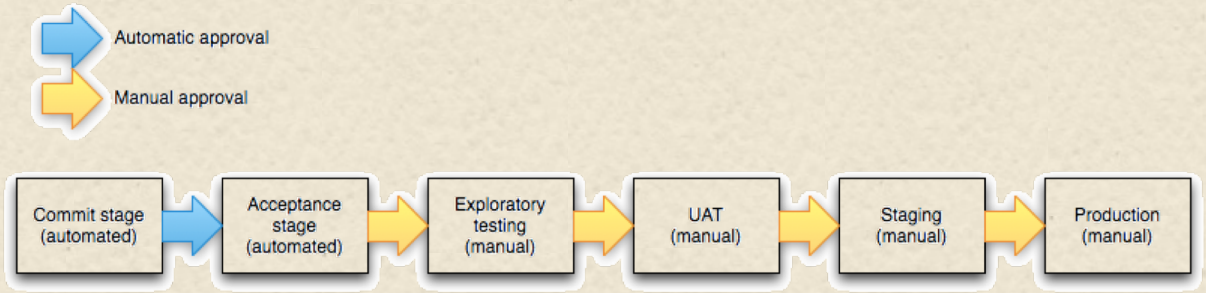
- Less risk
- Less stress
- Quicker ROI
- Improved response to business needs/challenges
- Improved quality
- Fail-fast
- Higher confidence

What does “continuous” mean?

- Continuous is used to describe many different processes that follow a typical set of practices we describe here. It doesn't mean "always running." It does mean "always ready to run." In the context of creating software, it also includes several core concepts/best practices:
  - **Frequent releases**
  - **Automated processes**
  - **Repeatable**
  - **Fast processing**
- **Usually used in the context of a Continuous Delivery Pipeline (aka Deployment Pipeline)**

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.

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
- May also include other areas, such as metrics.

What is Continuous Integration?

- Process of automatically detecting, pulling, building, and (in most cases) doing unit testing as source code is changed for a product.
- Activity that starts the pipeline (although certain pre-validations—often called "pre-flight checks"—are sometimes incorporated ahead of CI).
- Goal of CI is to quickly make sure a new change from a developer is "good" and suitable for further use in the code base – or “fail fast”.

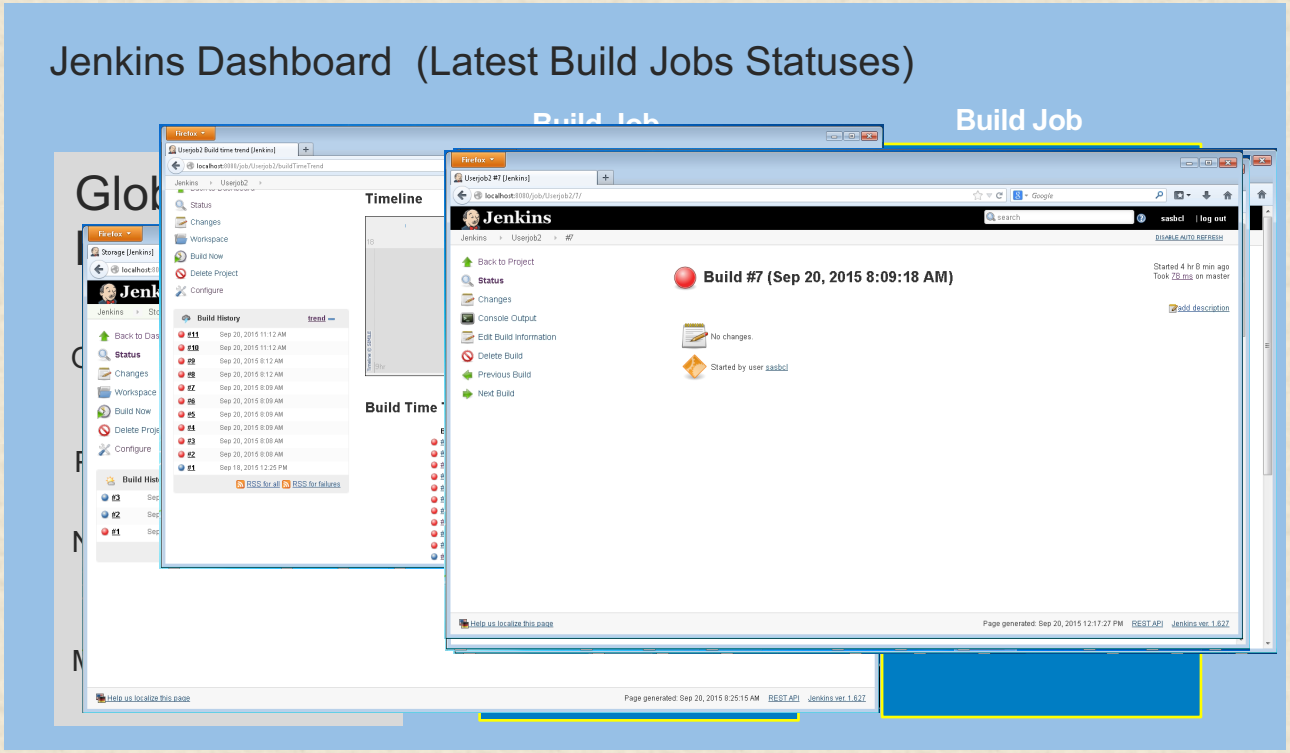
How does Continuous Integration work?

- Basic idea is having an automated process “watch” one or more source code repositories for changes.
- When a change is pushed to the repositories, the change is detected.
- The automated process then pulls down a copy, builds it, and runs any associated unit tests.
- These days, the automated process is usually an application like Jenkins that also orchestrates all (or most) of the processes running in the pipeline and monitors for changes as one of its functions.
- The watching application can monitor for changes in several different ways. These include:
  - Polling
  - Periodic
  - Push

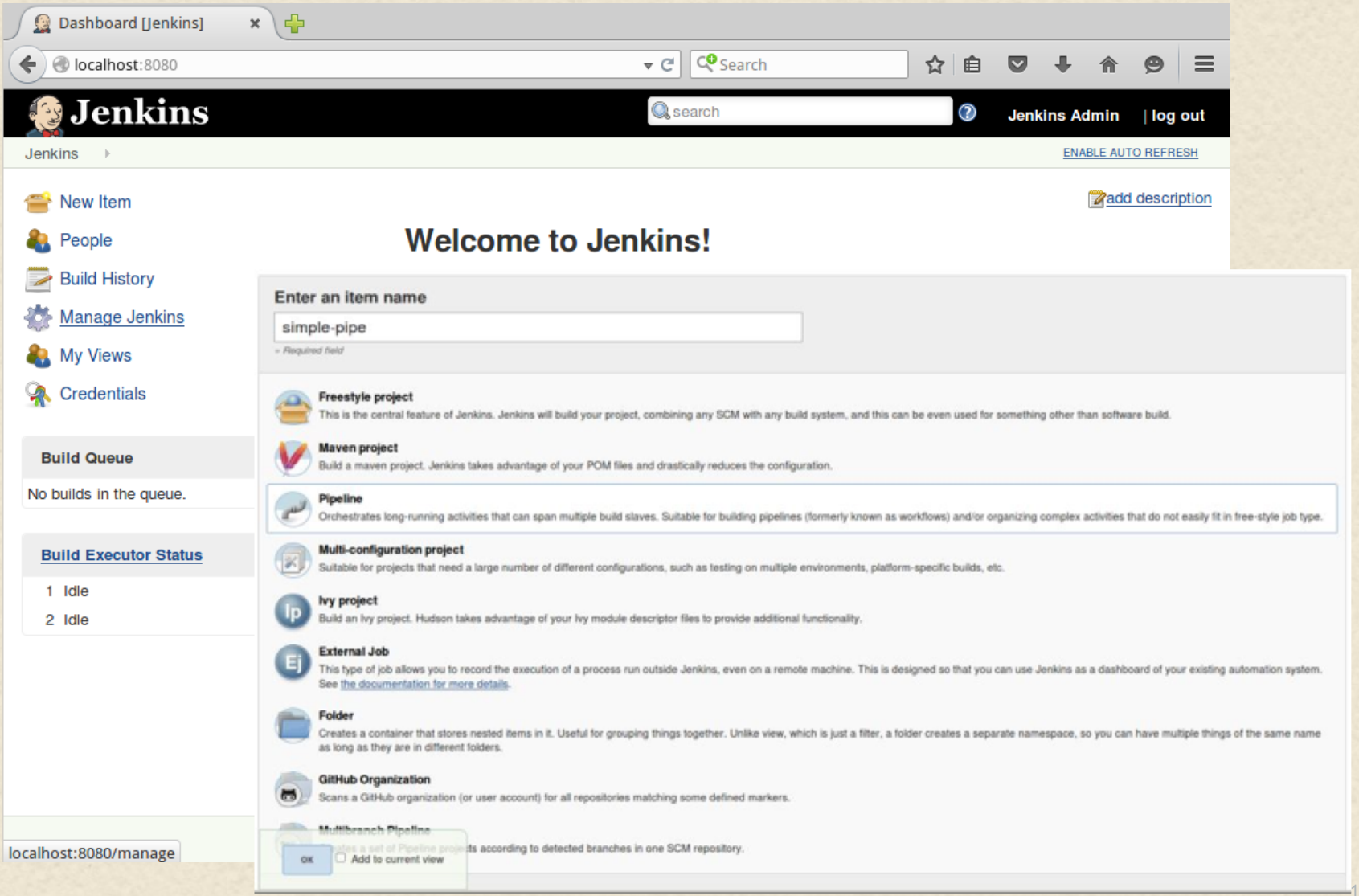
Jenkins – What is it?

- Open-source framework for CI, CD, creating pipelines
- Monitors execution of repeated jobs, provides notifications, records results, distributes workloads across systems
- Generally used for:
  - **Creating Continuous Delivery Pipelines**
  - **Building/testing software projects continuously**
  - **Monitoring executions of externally-run jobs, such as cron jobs**
- Created by Kohsuke Kawaguchi
- Over 1000 plug-ins
- Maintained by Jenkins Community
- Enterprise support by CloudBees

The Jenkins “Object Model”

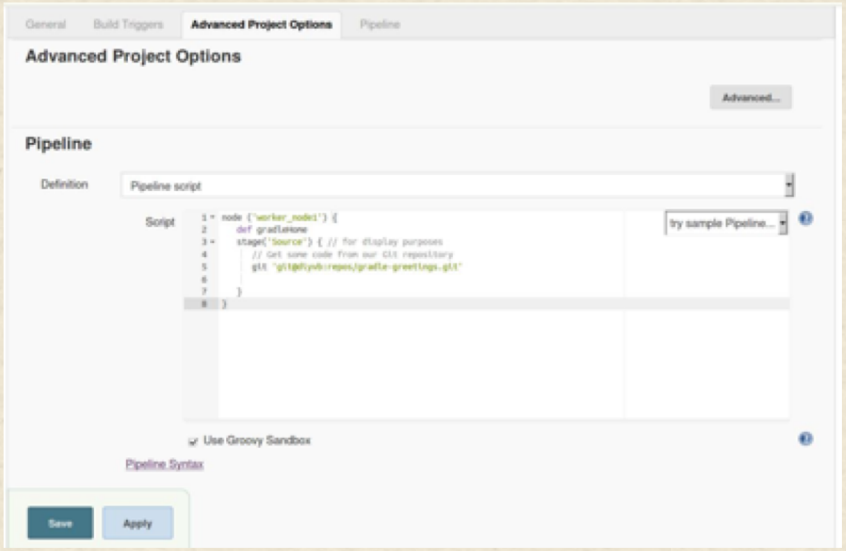


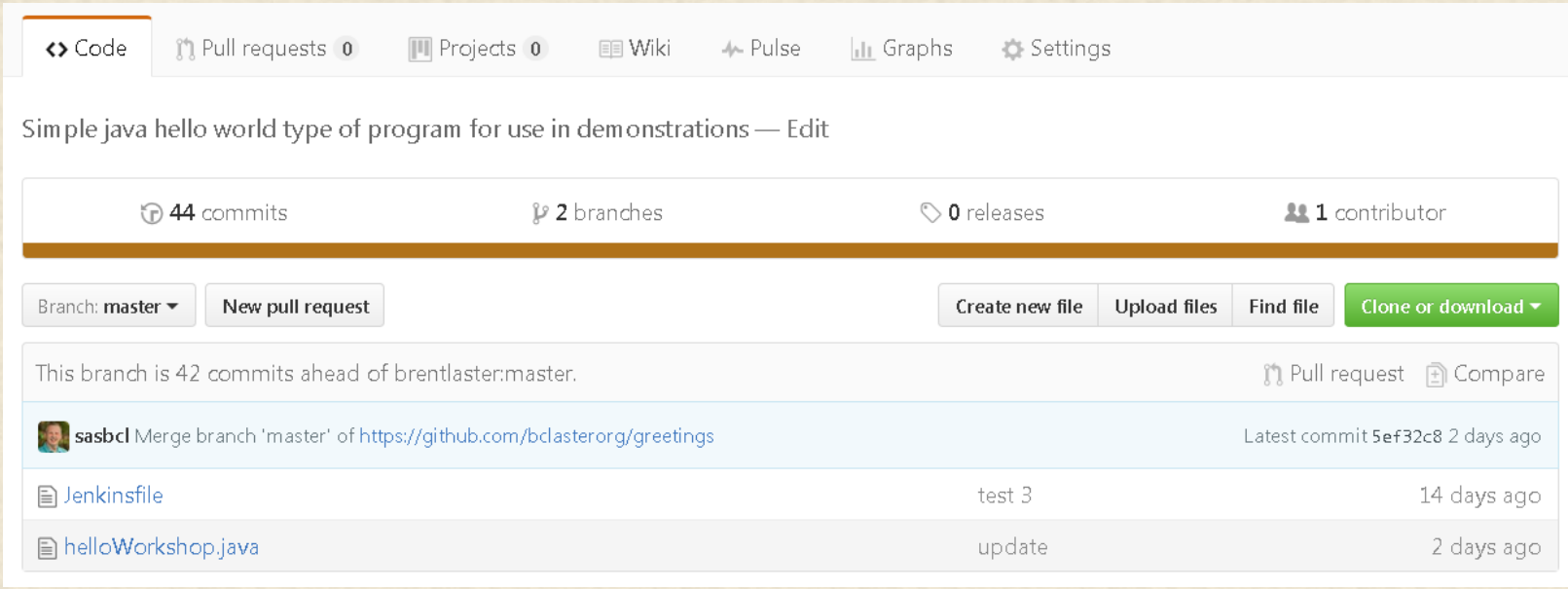
Home Screen -> Types of Projects



Jenkins 2 Pipelines

- Can write “programs” for CD
- Programming steps provided by plugins
- Can be written and stored as pipeline script in job itself or as an external Jenkinsfile stored in the repository





Terminology - Node and Stage

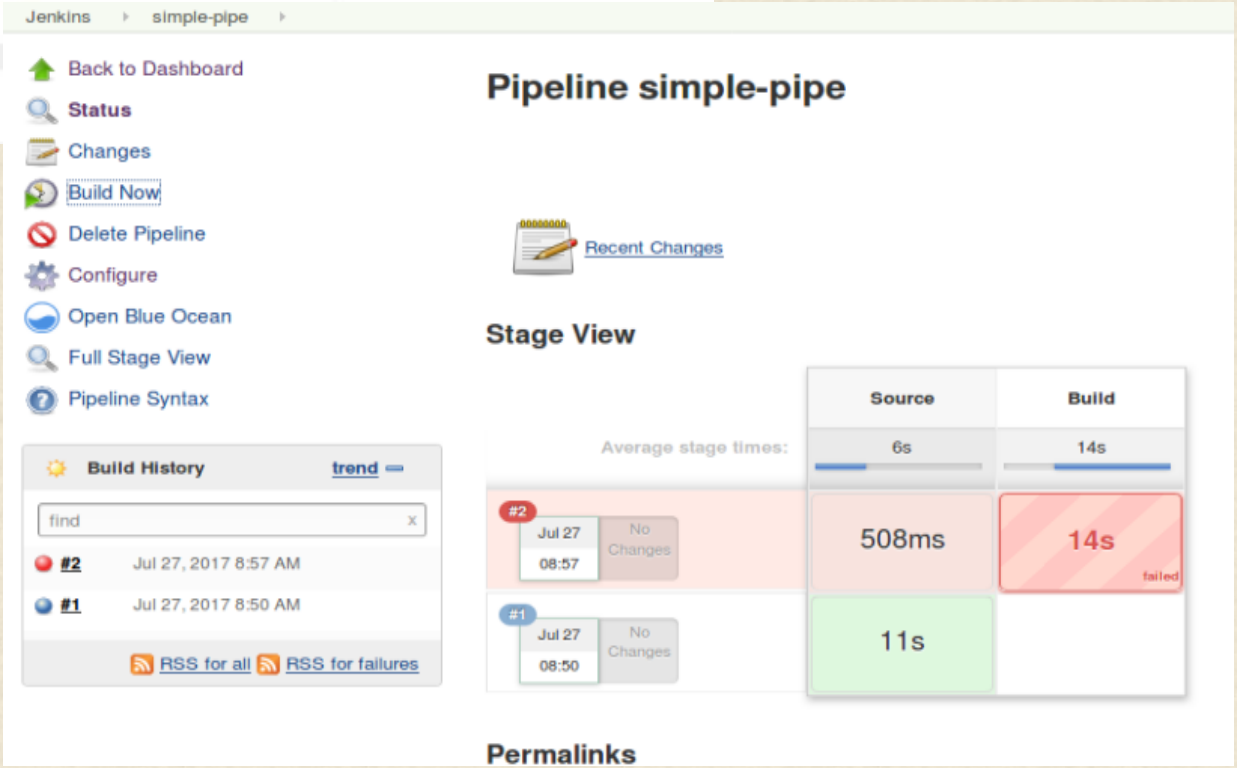
- Node is system where pipeline runs
- Stage aggregates build steps into sections
- Stages are inside of a node block
- Stages take a name (usually corresponding to the function)

```
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'
   }
```

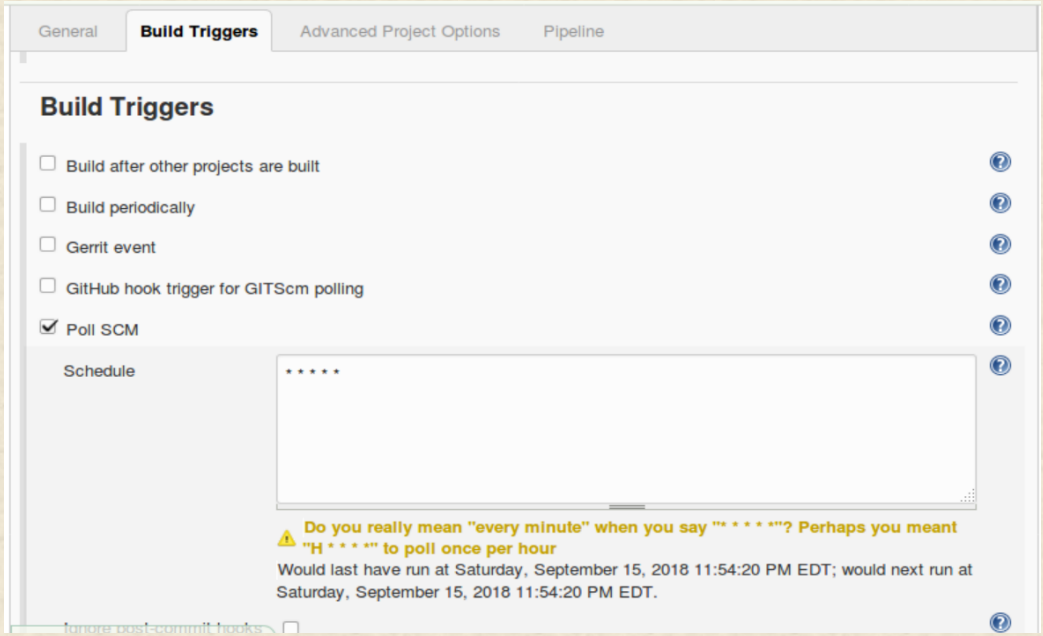
Stage View


```groovy
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"
    }

}
```

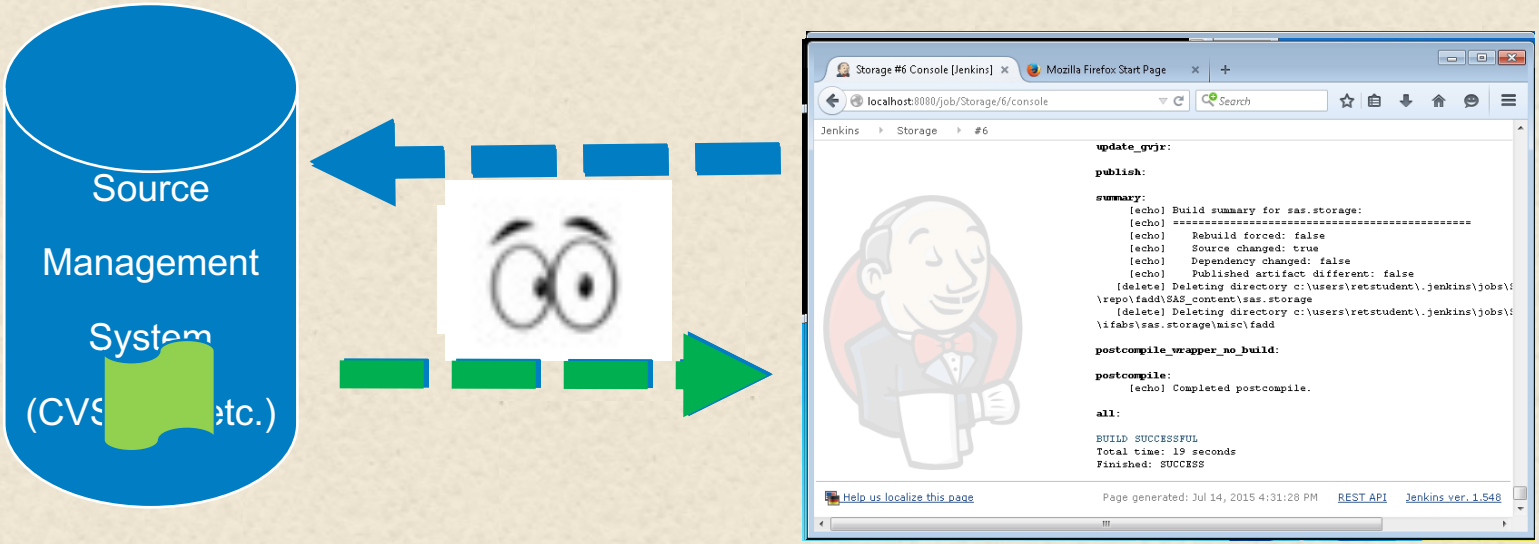


Build Triggers

- Events that cause job or pipeline code in Jenkins to start running
- Can be defined in pipeline code or in Jenkins job if pipeline is in application



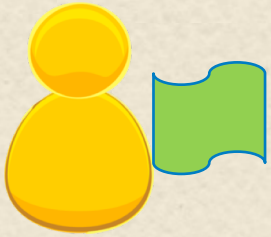
How Polling Works



Jenkins polls (watches) for changes in source control system.

User pushes a change.

Jenkins detects change and gets latest code.



Jenkins runs a new build

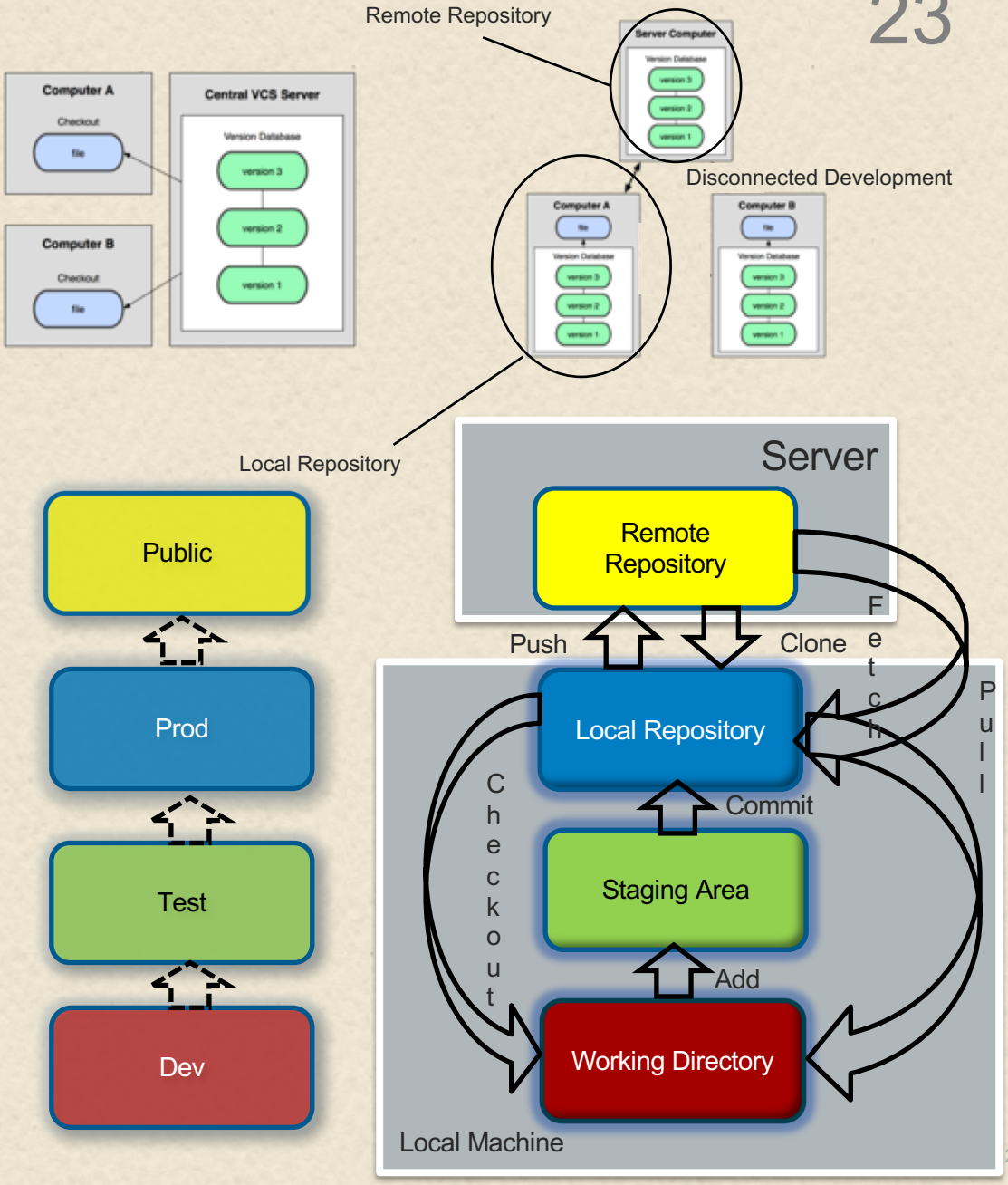
These days, the source control system is often Git.

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
```



What is Gradle?

- Build Application
- Used for building code, running tests, general build automation
- Basic unit of work is a task
- Build file with instructions is build.gradle by default
- Command line call is “gradle <task-name>”
- Integrated with Jenkins via Gradle plugin
- Can be called in a Jenkins pipeline script via “sh” (shell) command in script

Lab 1 – First CI Pipeline

What is Continuous Testing?

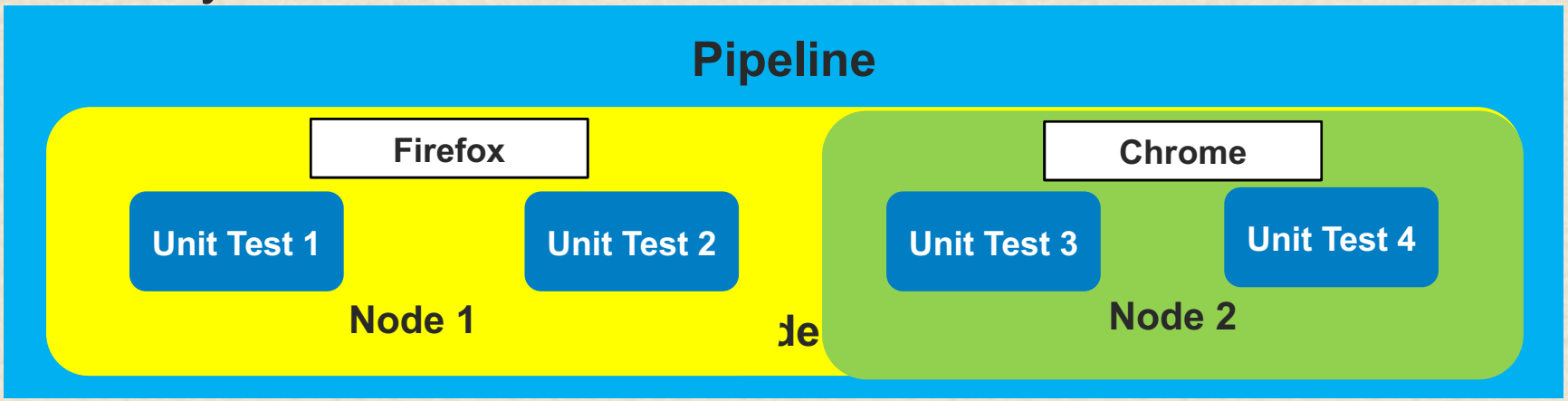
- Refers to the practice of running automated tests of broadening scope as code goes through the CD pipeline.
- **Unit testing** is typically integrated with the build processes as part of the CI stage and focused on testing code in isolation from other code interacting with it.
- **Integration testing** validates that groups of components and services all work together.
- **Functional testing** validates the result of executing functions in the product are as expected.
- **Acceptance testing** measures some characteristic of the system against acceptable criteria. Examples include performance, scalability, stress, and capacity.
- All may not be present in the automated pipeline, and the lines between some of the different types can be blurred.
- But the goal of continuous testing in a delivery pipeline is always the same: to prove by successive levels of testing that the code is of a quality that it can be used in the release that's in progress.
- Building on the continuous principle of being fast, a secondary goal is to find problems quickly and alert the development team. This is usually referred to as *fail fast*.

What are Unit Tests?

- Unit tests (also known as "commit tests") are small, focused tests written by developers to ensure new code works in isolation.
- "In isolation" - not depending on or making calls to other code that isn't directly accessible nor depending on external data sources or other modules.
- If such a dependency is required for the code to run, those resources can be represented by *mocks* *(*using a code stub that looks like the resource and can return values but doesn't implement any functionality).
- In most organizations, developers are responsible for creating unit tests to prove their code works.
- developer creates or updates the source in their local working environment
- unit tests ensure the newly developed function or method works.
- tests take the form of asserting that a given set of inputs to a function or method produces a given set of outputs
- generally test to ensure that error conditions are properly flagged and handled
- various unit-testing frameworks, such as JUnit for Java development, are available to assist.
- In pipelines, parallel processing is often used to quickly run unit tests

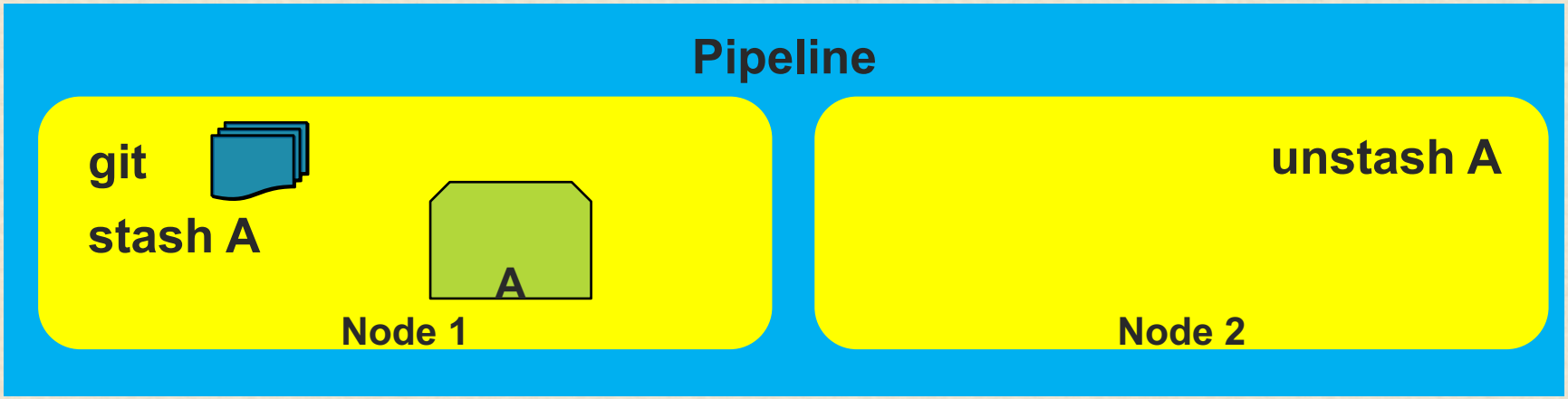
Running steps in Parallel

- DSL has a “parallel” operation
- Good for running multiple tests at the same time
- Parallel step takes a map (key – value) as an argument
- Value of map should be closure (pipeline steps defined in {})
- Use nodes in closures if available to get best parallelism
- May need to share files across nodes – can use “stash”



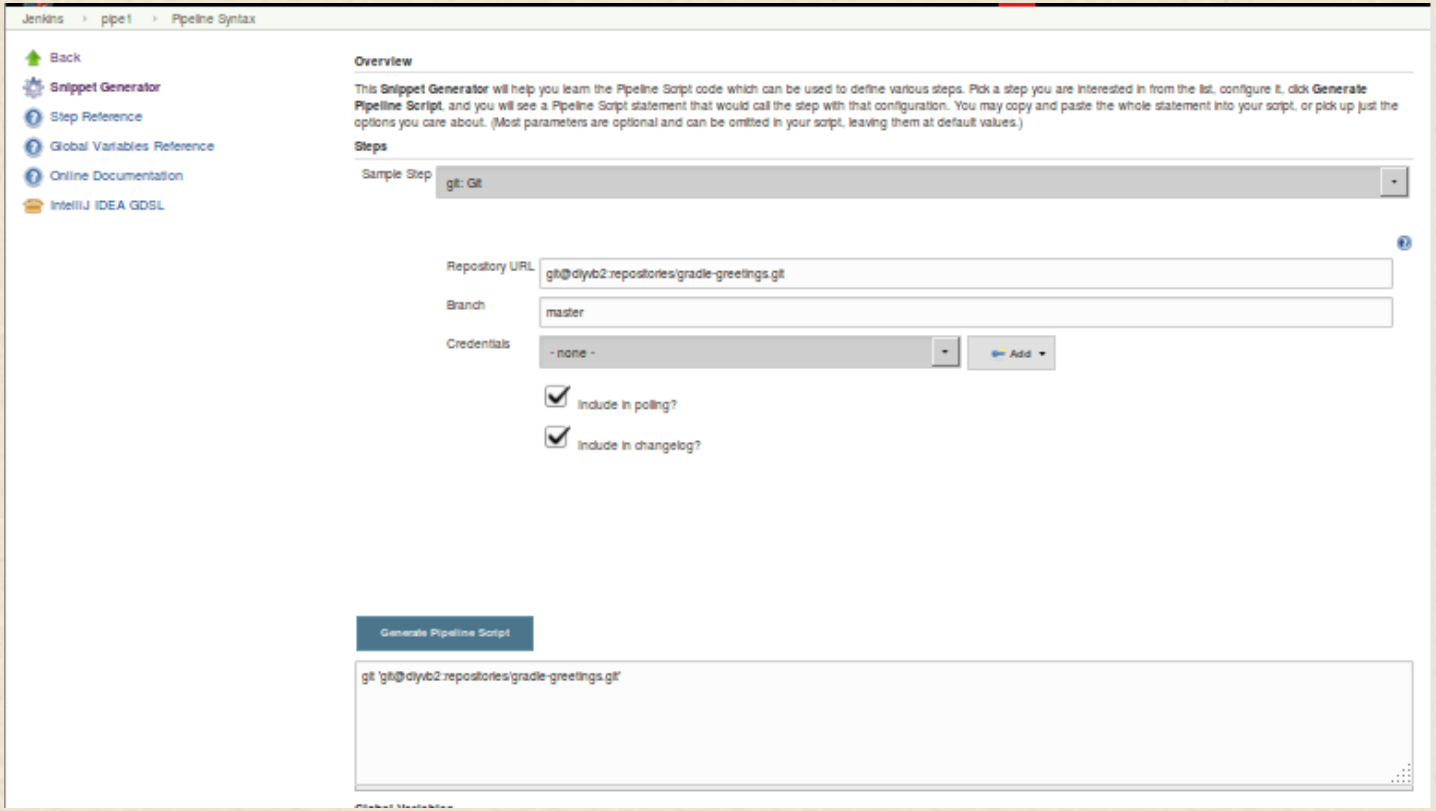
stash and unstash Functions

- Allows for saving (stashing) and retrieving (unstashing) files for different stages and/or nodes in a build
- 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>”
- Syntax can be created by “Snippet Generator” in Jenkins



Snippet Generator

- Guided interface (“wizard”) to select steps and arguments and generate code
- Facilitates generating Groovy code for typical actions
- Select the operation
- Fill in the arguments/parameters
- Push the button
- Copy and paste



Parallel Example (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'.
```

Integration Testing

- Gradle java plugin has built-in conventions for tests
- If certain directories exist, it treats them as test areas to process

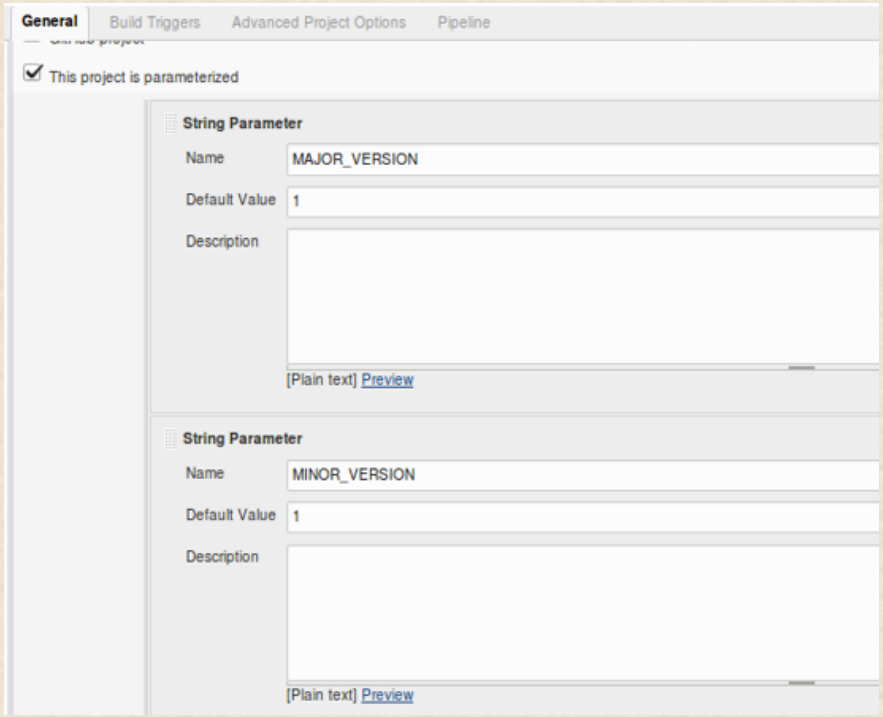
src/test/java	Test Java source
src/test/resources	Test resources

- The conventions above are for unit tests
- We can use a similar approach for integration tests (and functional tests)
- We just need to define new directory and compile/runtime processing paths for these types of tests in the build.gradle file
- We do that using Gradle sourcesets

Lab 2 – Adding in Testing

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
```

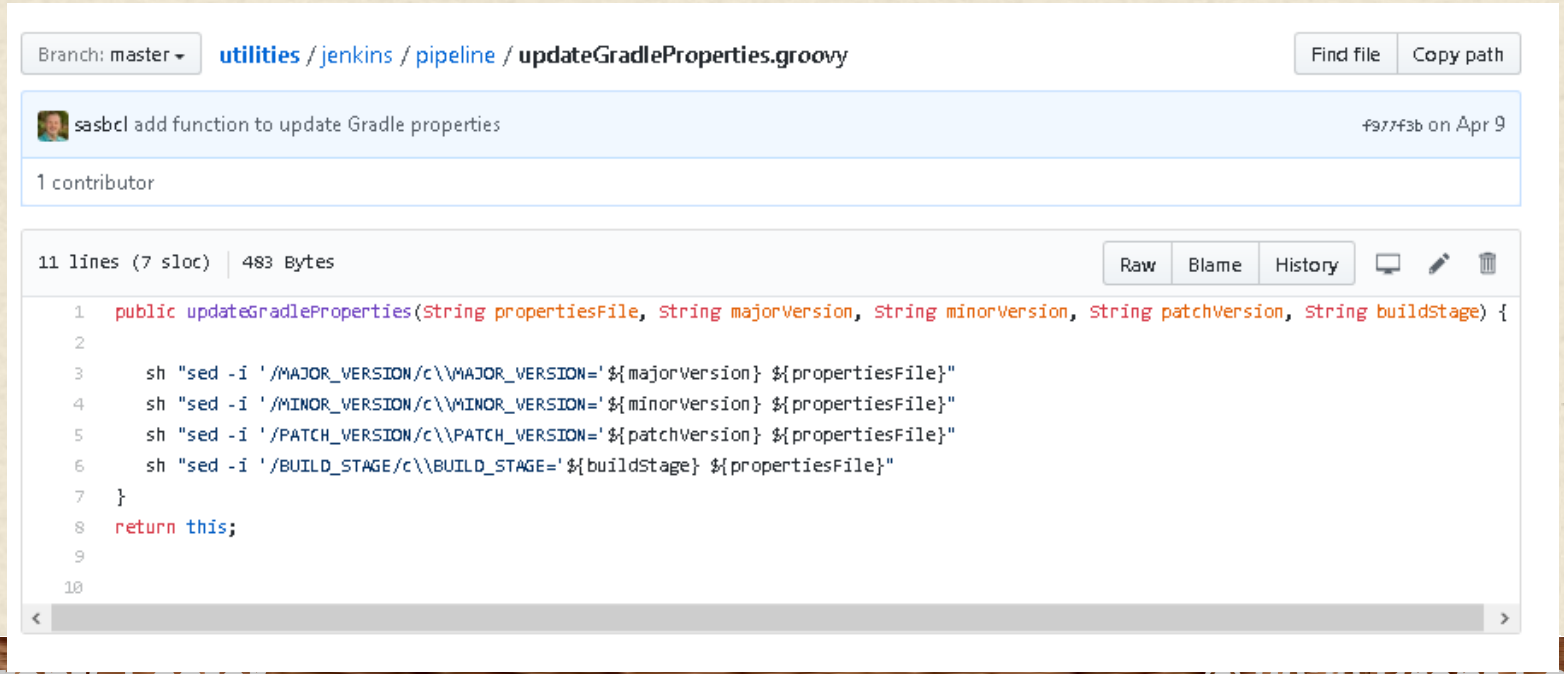
```
"${params.MAJOR_VERSION}",
"${params.MINOR_VERSION}",
"${params.PATCH_VERSION}",
"${params.BUILD_STAGE}"
```

Input and parameters in a Jenkinsfile


```
def userInput
  stage('Parameters') {

    timeout(time:1, unit:'MINUTES') {

        userInput = input message: 'Enter version changes (if any):',
        parameters: [string(defaultValue: '1', description: '', name: 'MAJOR_VERSION'),
                     string(defaultValue: '1', description: '', name: 'MINOR_VERSION'),
                     string(defaultValue: env.BUILD_NUMBER, description: '', name: 'PATCH_VERSION'),
                     string(defaultValue: 'SNAPSHOT', description: '', name: 'BUILD_STAGE')]
        major_version = userInput.MAJOR_VERSION
        minor_version = userInput.MINOR_VERSION
        patch_version = userInput.PATCH_VERSION
        build_stage = userInput.BUILD_STAGE

    }

  }

     def workspace = env.WORKSPACE
     def setPropertiesProc = fileLoader.fromGit('jenkins/pipeline/updateGradleProperties',
        'https://github.com/brentlaster/utilities.git', 'master', null, '')

     setPropertiesProc.updateGradleProperties("${workspace}/gradle.properties",
         "${userInput.MAJOR_VERSION}",
         "${userInput.MINOR_VERSION}",
         "${userInput.PATCH_VERSION}",
         "${userInput.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)



Assemble Job - assemble task (in gradle build file)


```
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')
          }
        }

}
```

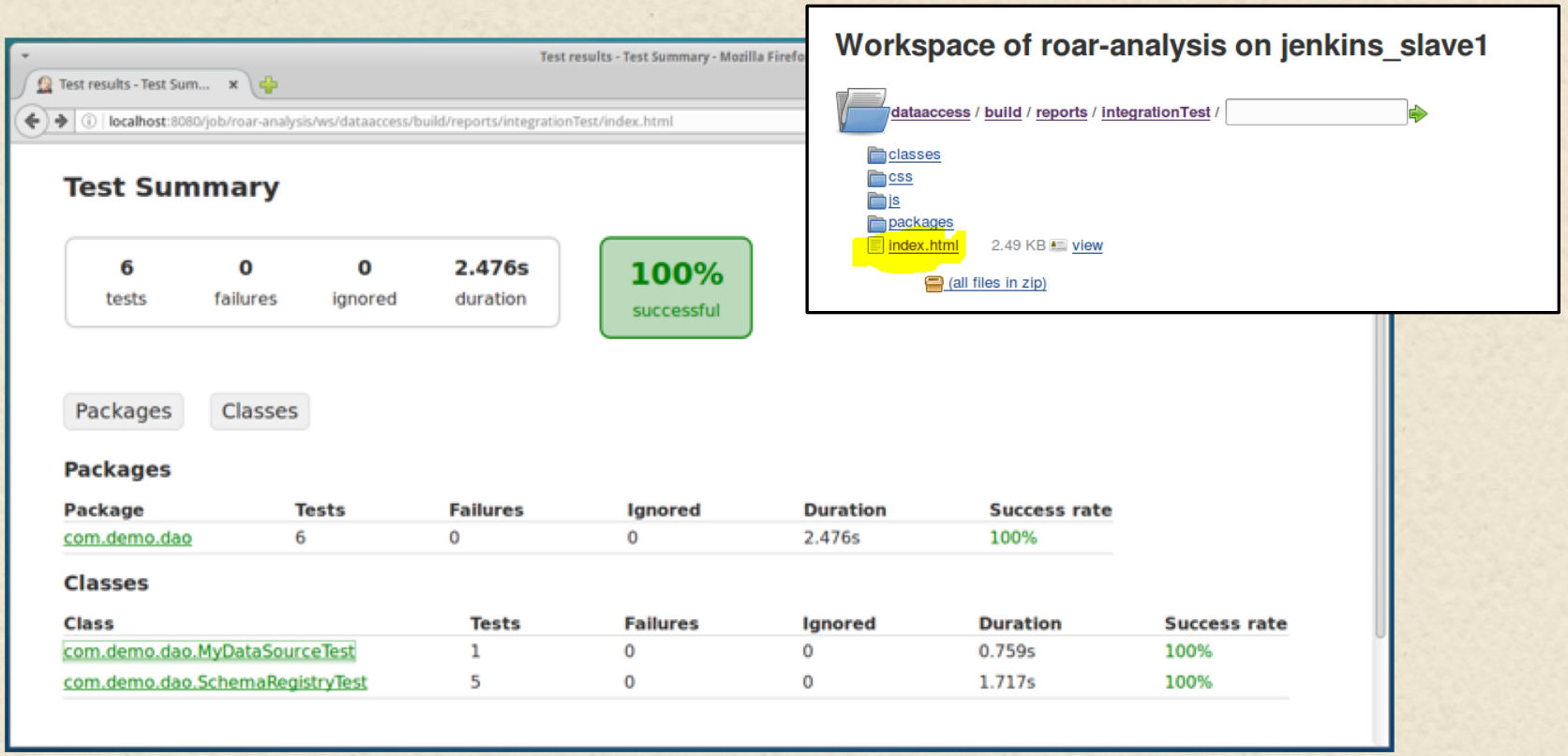
Source Sets for Integration and Functional Tests (build.gradle)

```
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

}
```

Example Integration Test Output



Lab 3 – Assembly

What is Artifactory?

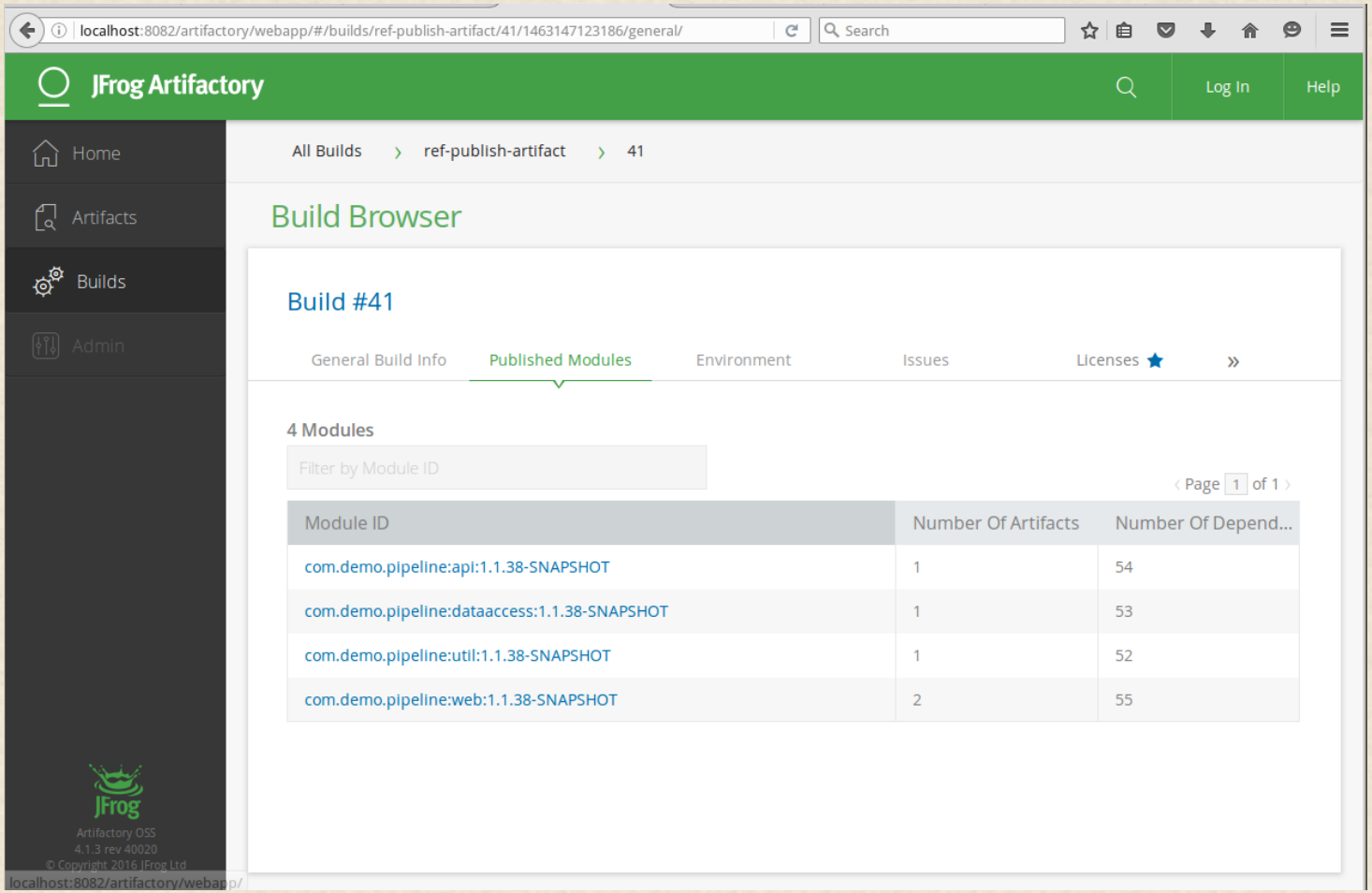
- Binary repository manager
  - We have source control for our source
  - Artifactory provides version control for binary artifacts (jars, wars, etc.)
- Why use it?
  - If you rebuild from source each time, that can introduce change and failure for consumers.
  - By having a versioned copy of a (tested) 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 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

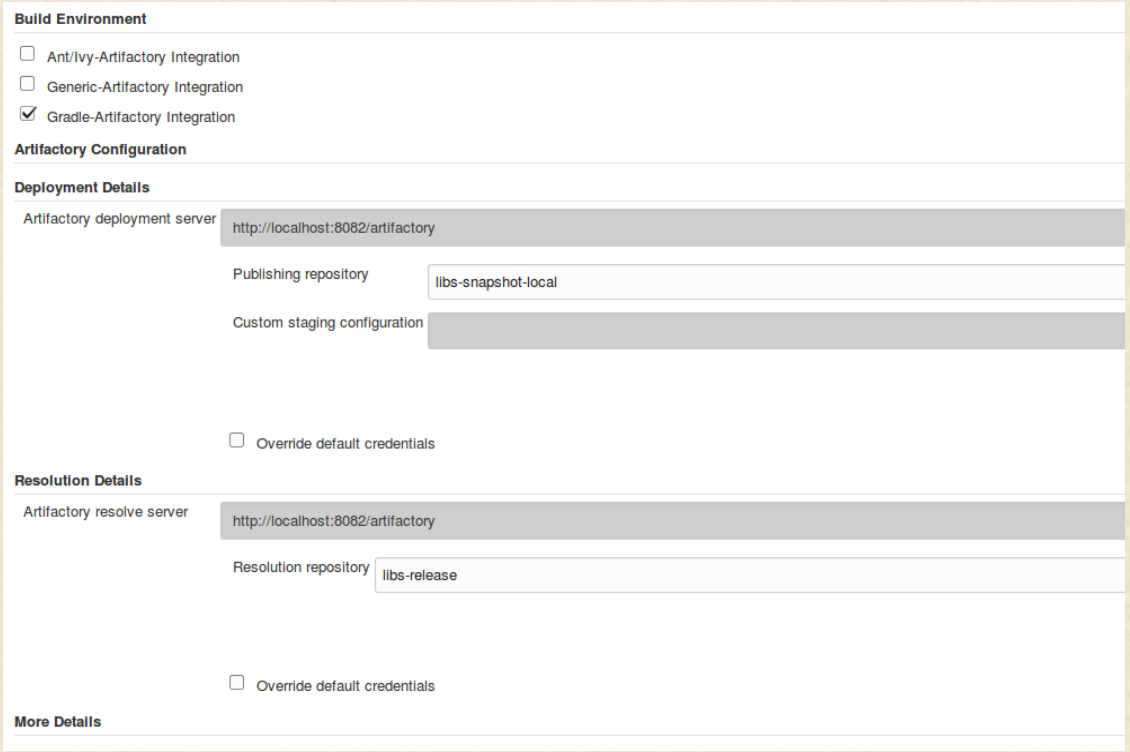
Build info in Artifactory

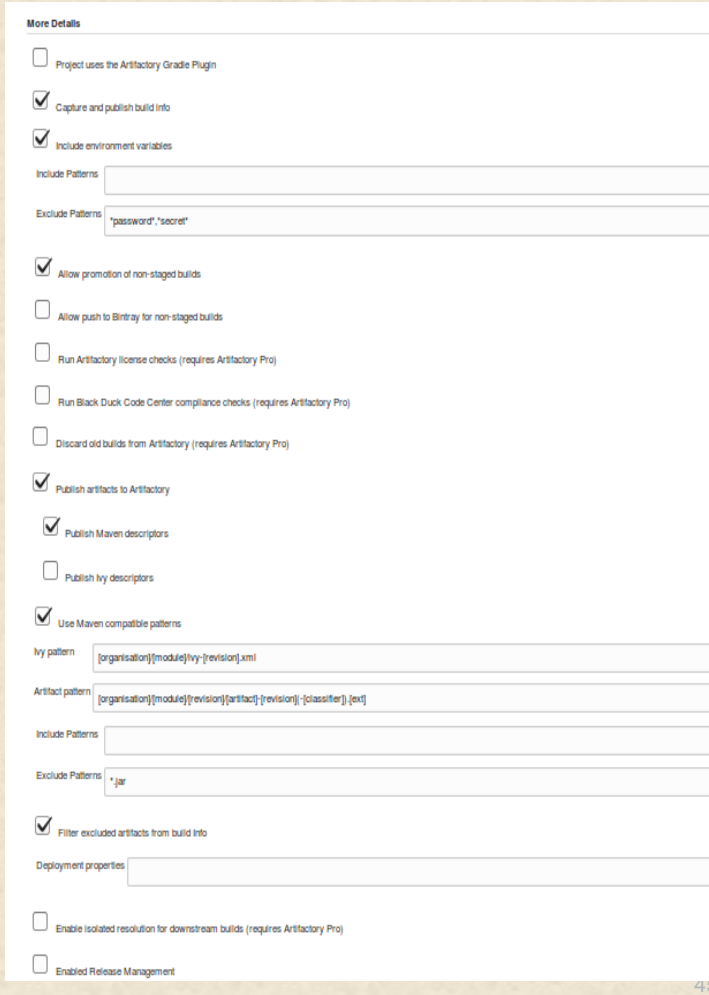

```
Deploying build descriptor to: http://localhost:8082/artifactory/api/build
Build successfully deployed. Browse it in Artifactory under http://localhost:8082/artifactory/webapp
/builds/ref-publish-artifact/41
```



Publishing Artifacts to Artifactory

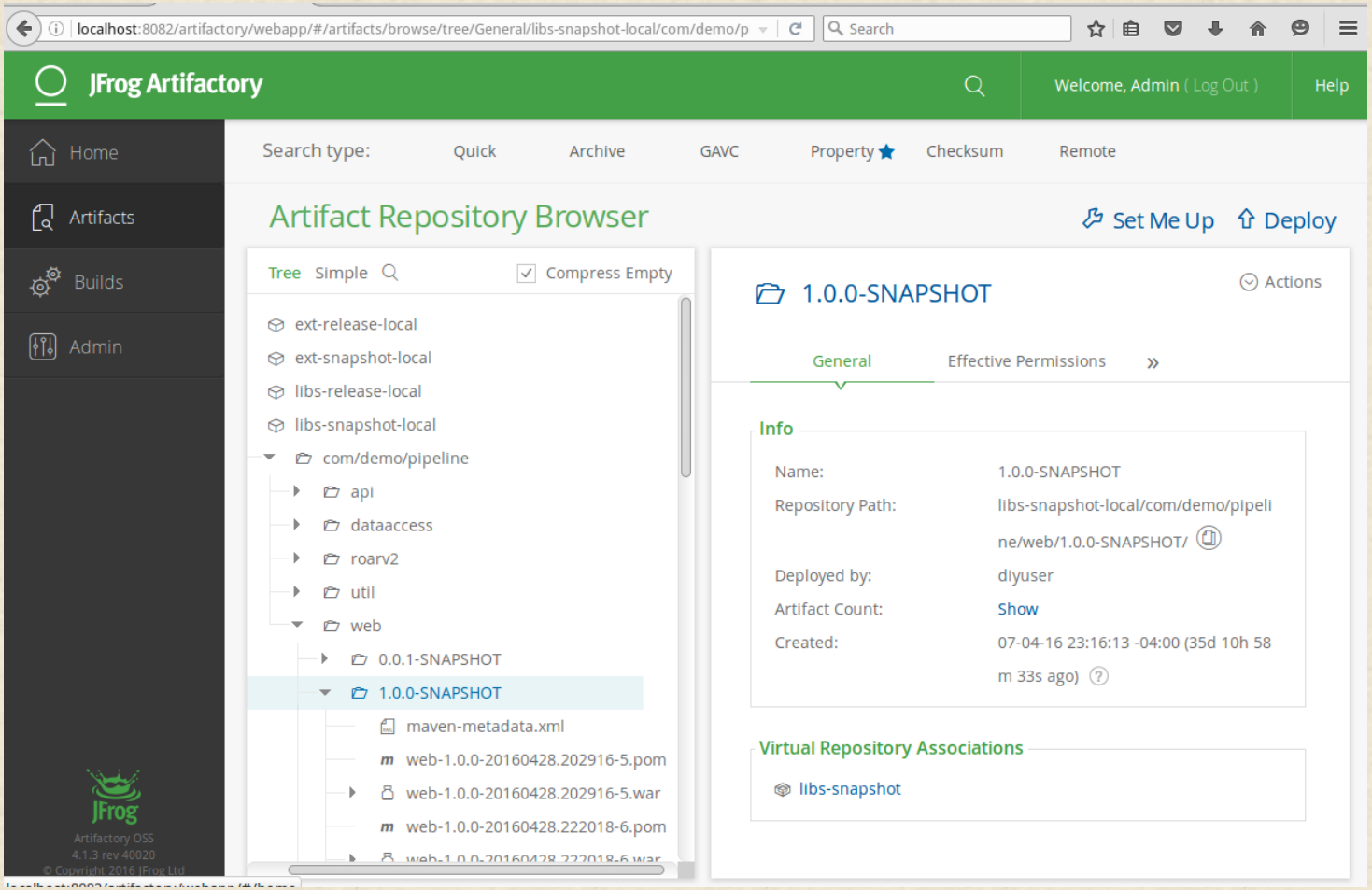
- Similar to how we used Artifactory before, but add a repository to publish to, and set switches to publish





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
```



Lab 4 – Storing and Tracking Artifacts

Shared Pipeline Libraries

- Allows you to create a shared SCM repository of pipeline script code and functions.

- 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

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")

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

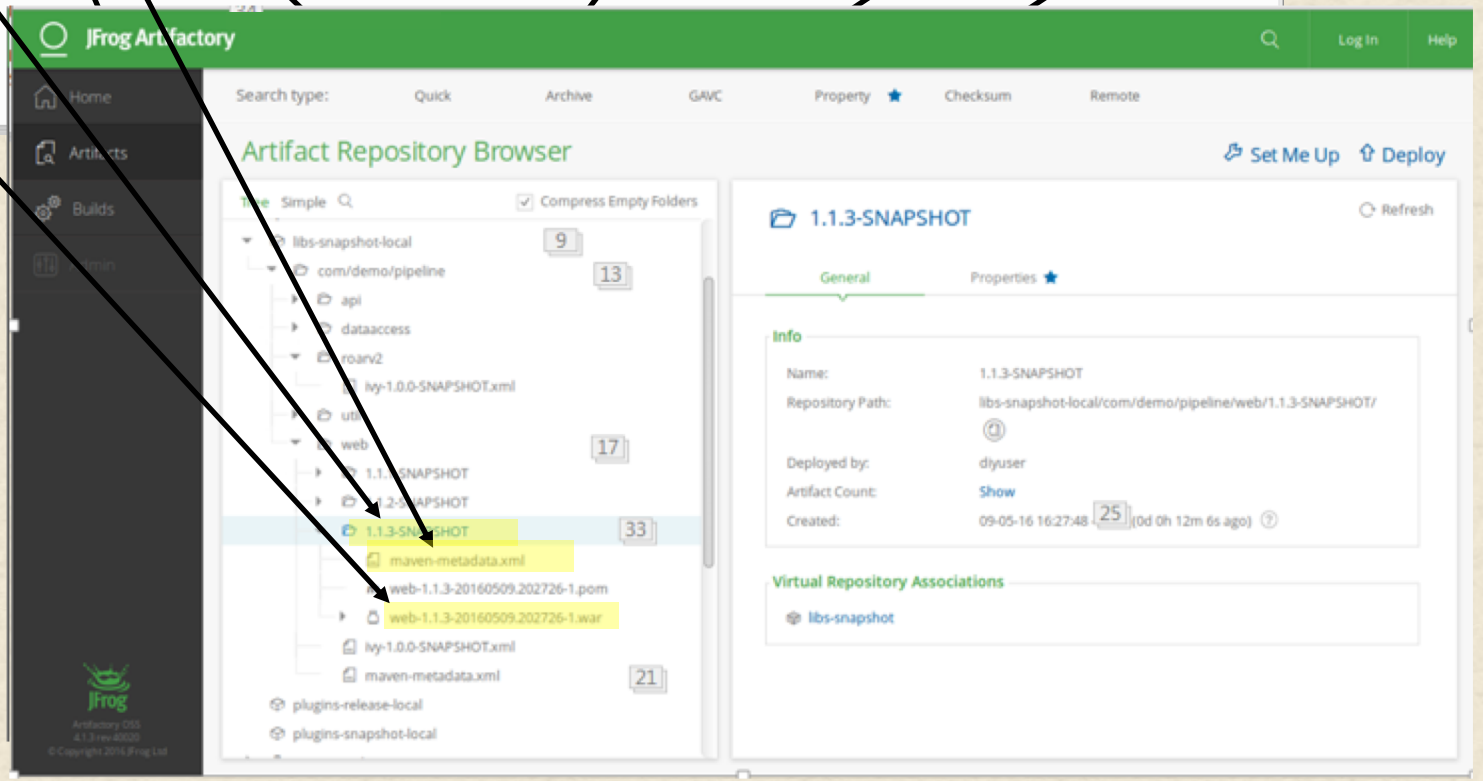
Jenkins Build Step

```
# 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 5 – Packaging – Part 1

52

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)

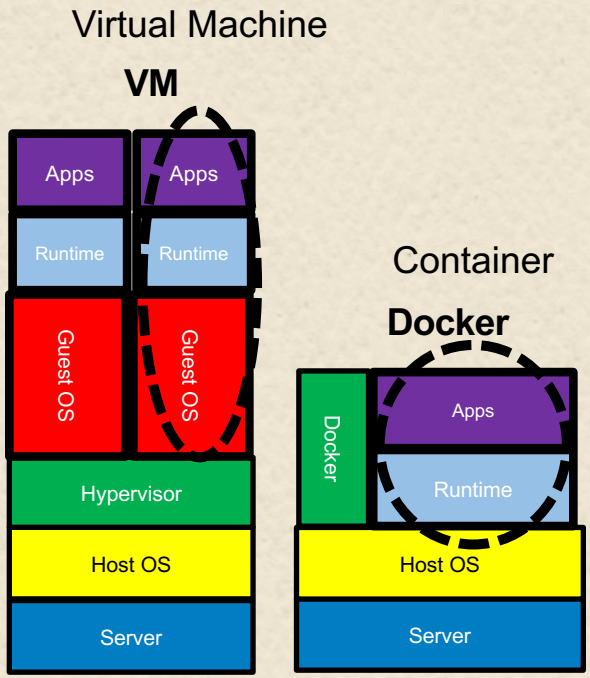
  Thin wrapper around Linux Container technology

  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

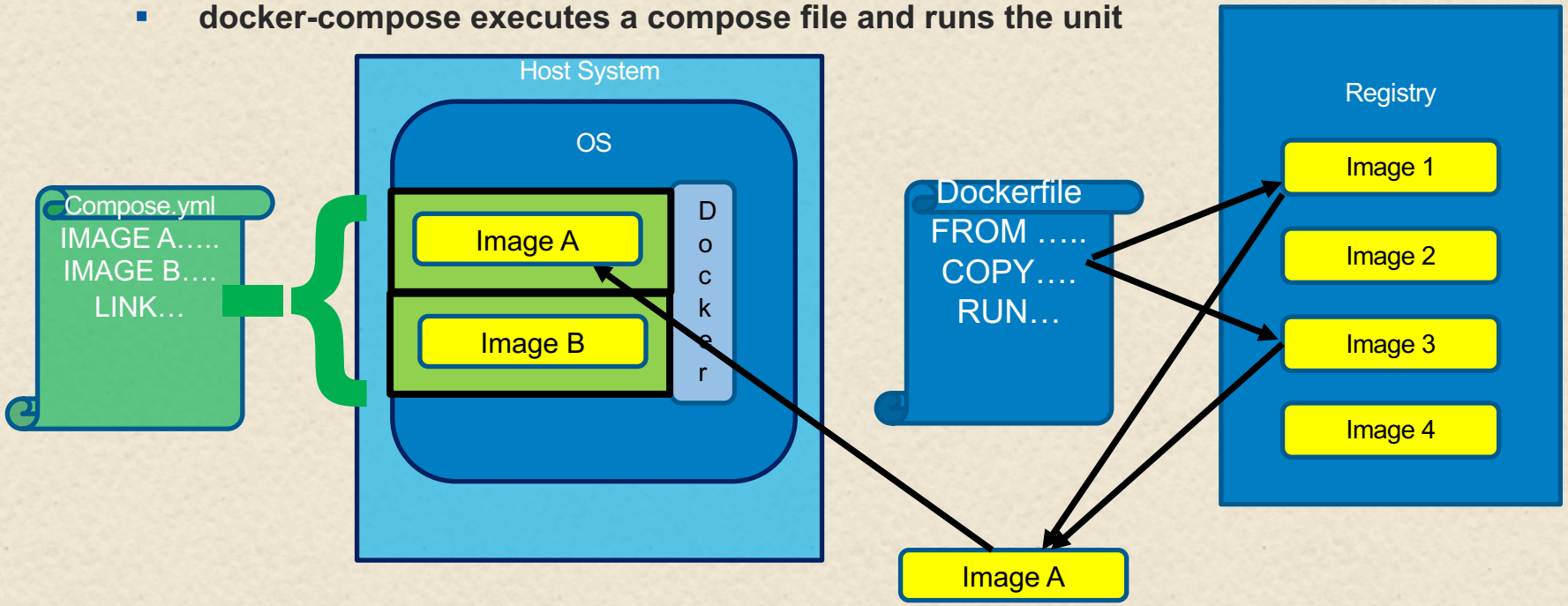
How Docker differs from a VM



- 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



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} ||:"
    }
```

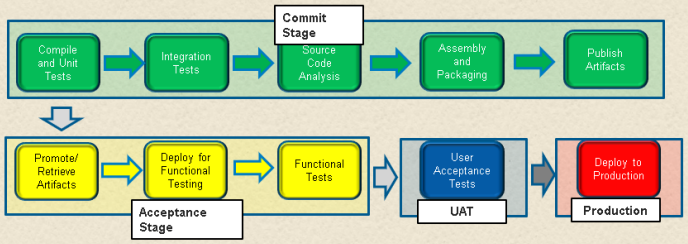
```
[Pipeline] stage
[Pipeline] { (Run Tests)
[Pipeline] sh
[workspace] Running shell script
+ docker run --privileged --rm -v /var/lib/jenkins/jobs/docker-test2/workspace:/var/lib/jenkins/jobs/docker-test2/workspace --name jenkins-
docker-test2-20 my-image:snapshot /bin/sh -c cd /var/lib/jenkins/jobs/docker-test2/workspace/gradle-greetings && 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 6 – Packaging – Part 2

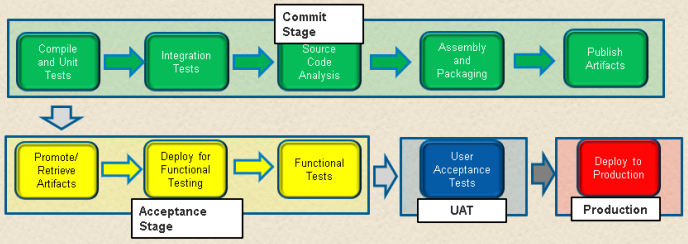
Blue/Green Deployments

- Two identical hosting environments are maintained — a *blue* one and a *green* one. (Colors are not significant and only serve as identifers.)
- At any given point, one is the *production* deployment and the other is the *candidate* deployment.
- In front of these instances is a router or other system that serves as the customer "gateway" to the product/application.
- By pointing the router to the desired blue or green instance, customer traffic can be directed to the desired deployment.
- Swapping out which deployment instance is pointed to (blue or green) is quick, easy, and transparent to the user.
- When new release is ready for testing, it can be deployed to the non-production environment. After tested and approved, the gateway can be changed to point the incoming production traffic to it (so it becomes the new production site).
- Now the hosting environment that was production is available for the next candidate.
- If a problem is found with the latest deployment and the previous production instance is still deployed in the other environment, a simple change can point the customer traffic back to the previous production instance
- Effectively takes the instance with the problem "offline" and rolling back to the previous version. The new deployment with the problem can then be fixed in the other area.

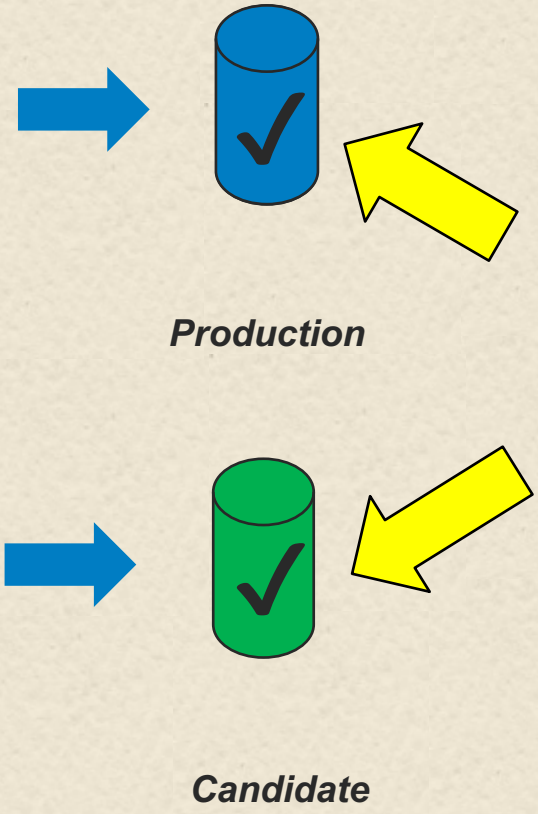
Blue/Green Deployment Model



BLUE



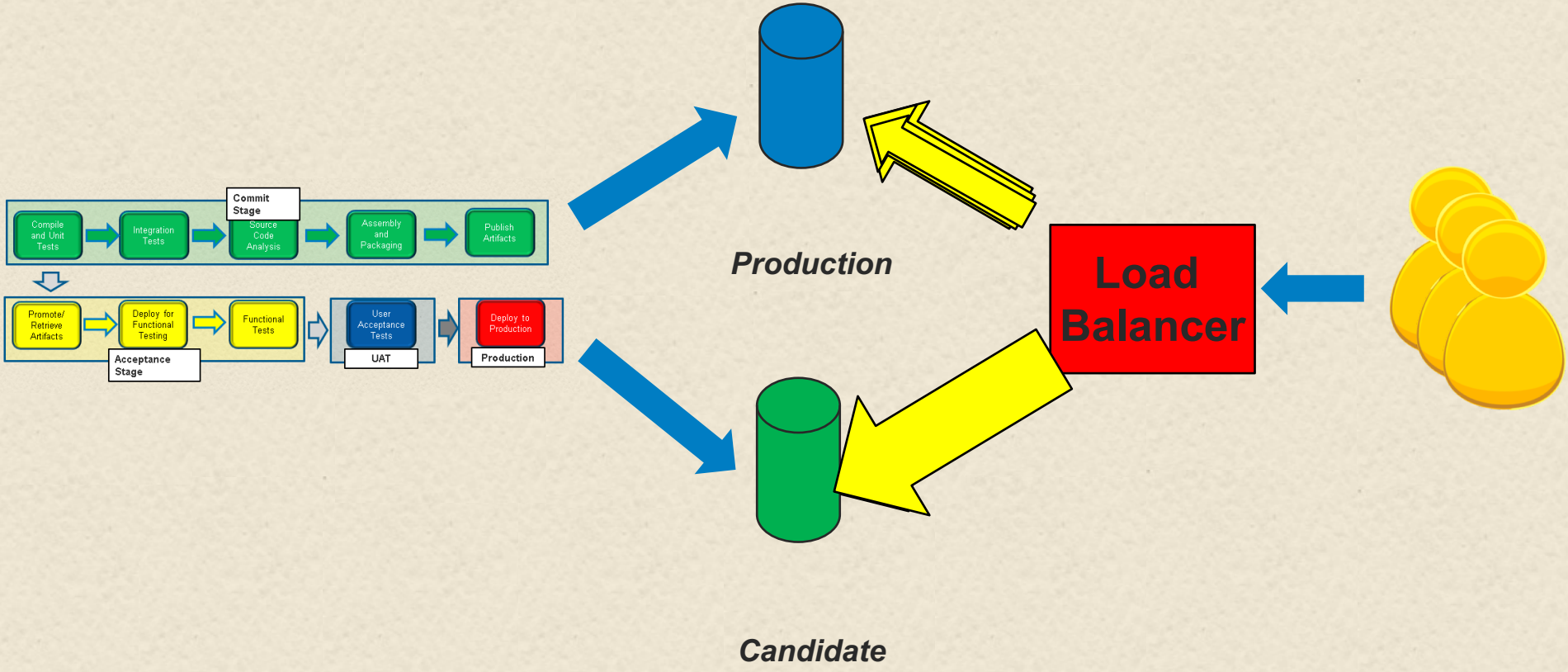
GREEN



Canary Deployment

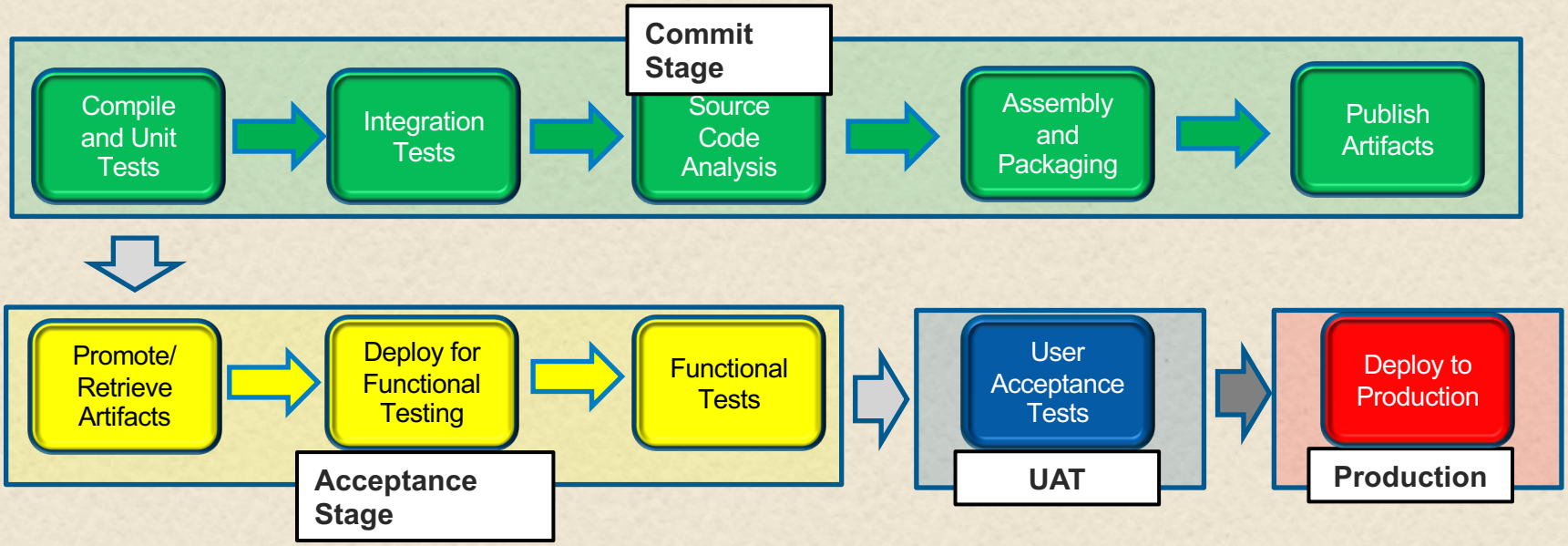
- In some cases, swapping out the entire deployment via a blue/green environment may not be workable or desired.
- Another approach is known as *canary* testing/deployment.
- In this model, a portion of customer traffic is rerouted to new pieces of the product.
- For example, a new version of a search service in a product may be deployed alongside the current production version of the service. Then, 10% of search queries may be routed to the new version to test it out in a production environment.
- If the new service handles the limited traffic with no problems, then more traffic may be routed to it over time.
- If no problems arise, then over time, the amount of traffic routed to the new service can be increased until 100% of the traffic is going to it.
- This effectively “retires” the previous version of the service and puts the new version into effect for all customers

Canary Deployment Model



Lab 7 - Deployment

Full CD Pipeline



- Make changes locally
- Build/test locally
- Push
- CI - Build and Unit Test
- Source code Analysis
- Assembly and Packaging
- Store and Retrieve Desired Artifacts
- Functional Testing
- Performance Testing
- Security Testing
- User Acceptance Testing
- Deploy to Production

Lab 8 – Full CD Example

That's all - thanks!

Professional Git 1st Edition

by Brent Laster (Author)

3 customer reviews

Look inside