Remember the Build

Robert McNulty
Experian Marketing Services
robert.mcnulty@experian.com
http://www.experian.com/marketing-services

June 18, 2014

#jenkinsconf

Agenda

- Introduction
- Things As They Are
- Things As They Should Be
- Demonstration
- Q & A

About Me

- 20+ years in software development and database design
- Lead Technical Developer and Architect at Experian Marketing Services
- Lead and mentor a global team of developers
  - United States
  - Costa Rica
  - Kuala Lumpur, Malaysia
  - Melbourne, Australia

Problem Statement

- In an out-of-the-box Jenkins build scenario, parameterized build input such as notes, Git branches, and bug tracking issues are lost once a build has completed.
- The only historical reference is in a report.
- This data cannot easily be reused for future builds.
- Using Jenkins, Groovy and MySQL, the data can be persisted for the life of a project.

Standard Success Email

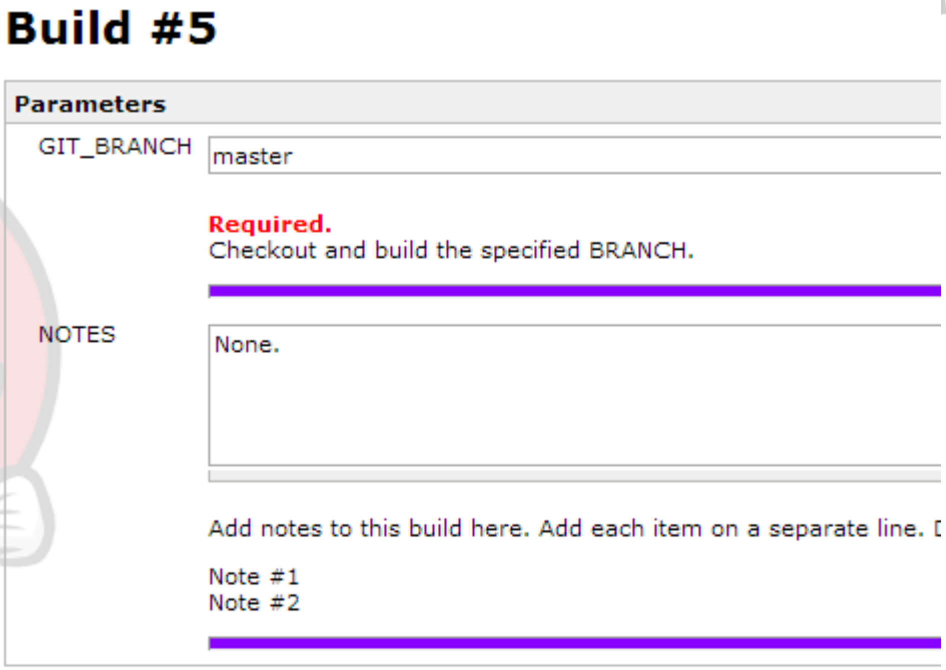
Sample body content of unmodified email-ext plugin email

– juc-hello-world-1 - Build # 5 - Successful: Check console output at http://localhost:8180/jenkins/job/juc-hello-world-1/5/ to view the results.

- This is what is contained in the body of the email message sent by an unmodified project.
- Not very user-friendly.

Standard Success Email – cont.

Clicking on the link, the information that the user entered for this build can be viewed:



Build #5

Parameters

GIT_BRANCH: master

Required.
Checkout and build the specified BRANCH.

NOTES: None.

Add notes to this build here. Add each item on a separate line. [

Note #1
Note #2

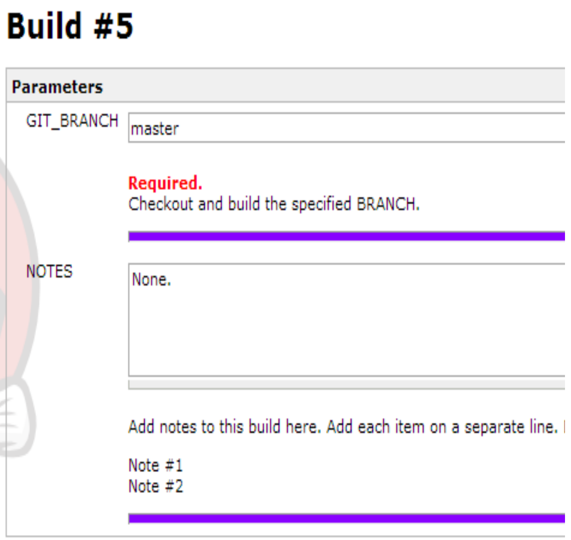
Easy enough, but not very helpful.

Standard Success Email – Multiple Builds

- The information is not so easily available to subsequent builds. The user must open each build individually to see what information had been entered.
- For example, a user has run three builds and included notes for each build. If the email has just been received for build #7, clicking on the link will take the user to the build page where the user notes can be viewed.

Standard Success Email – Multiple Builds

- However, if the user wants to see the notes of all the builds to date, each build must be opened one at a time to view the notes for that build. What a hassle!



Build #5

Parameters

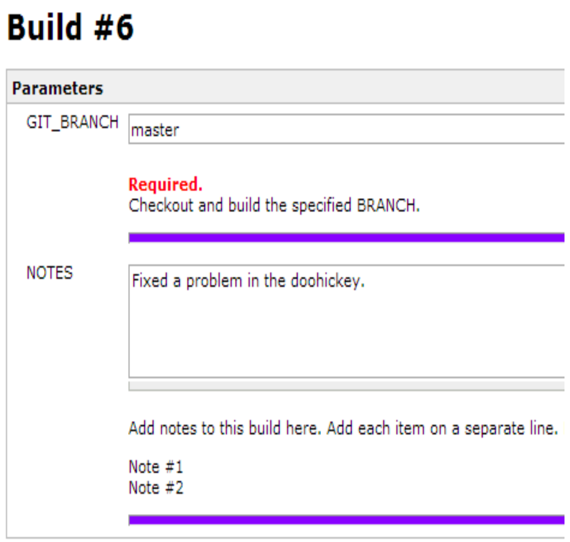
GIT_BRANCH master

Required.
Checkout and build the specified BRANCH.

NOTES None.

Add notes to this build here. Add each item on a separate line.

Note #1
Note #2



Build #6

Parameters

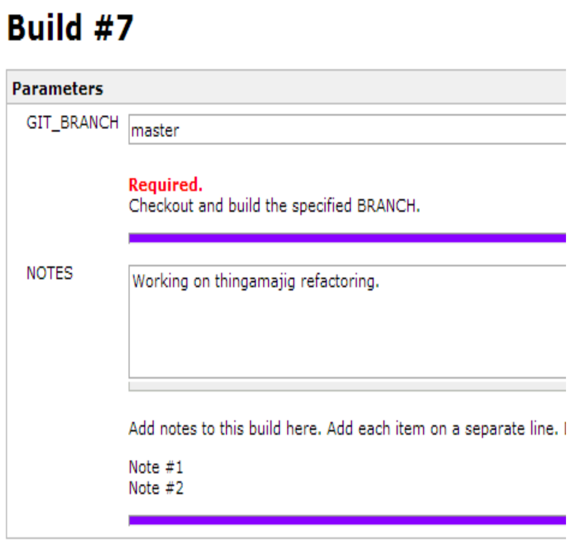
GIT_BRANCH master

Required.
Checkout and build the specified BRANCH.

NOTES Fixed a problem in the doohickey.

Add notes to this build here. Add each item on a separate line.

Note #1
Note #2



Build #7

Parameters

GIT_BRANCH master

Required.
Checkout and build the specified BRANCH.

NOTES Working on thingamajig refactoring.

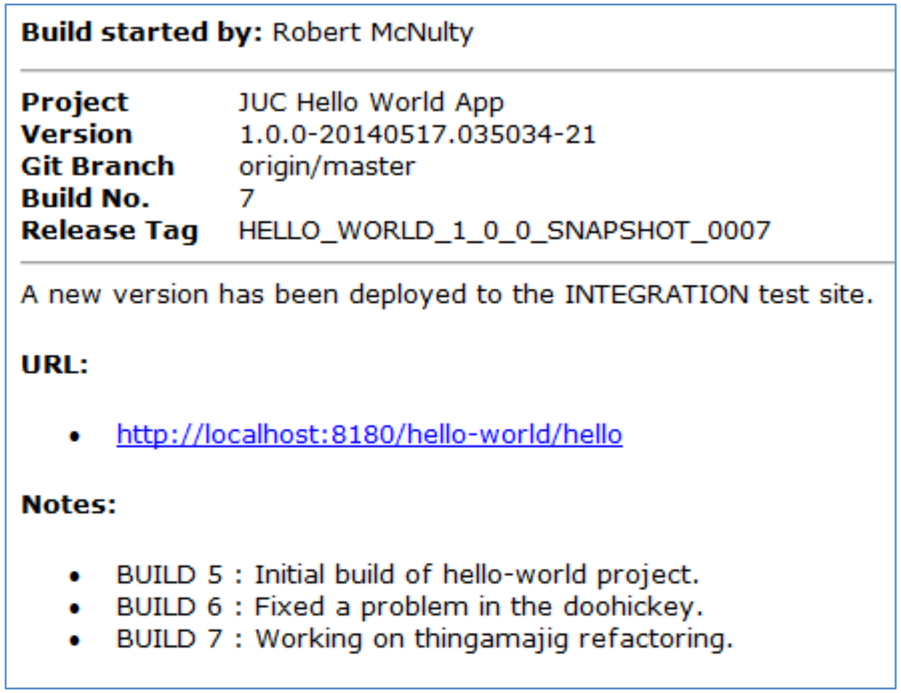
Add notes to this build here. Add each item on a separate line.

Note #1
Note #2

There Is A Better Way!!

Enhanced Success Email

Wouldn't it be nice to receive this email instead?



Build started by: Robert McNulty

Project	JUC Hello World App
Version	1.0.0-20140517.035034-21
Git Branch	origin/master
Build No.	7
Release Tag	HELLO_WORLD_1_0_0_SNAPSHOT_0007

A new version has been deployed to the INTEGRATION test site.

URL:

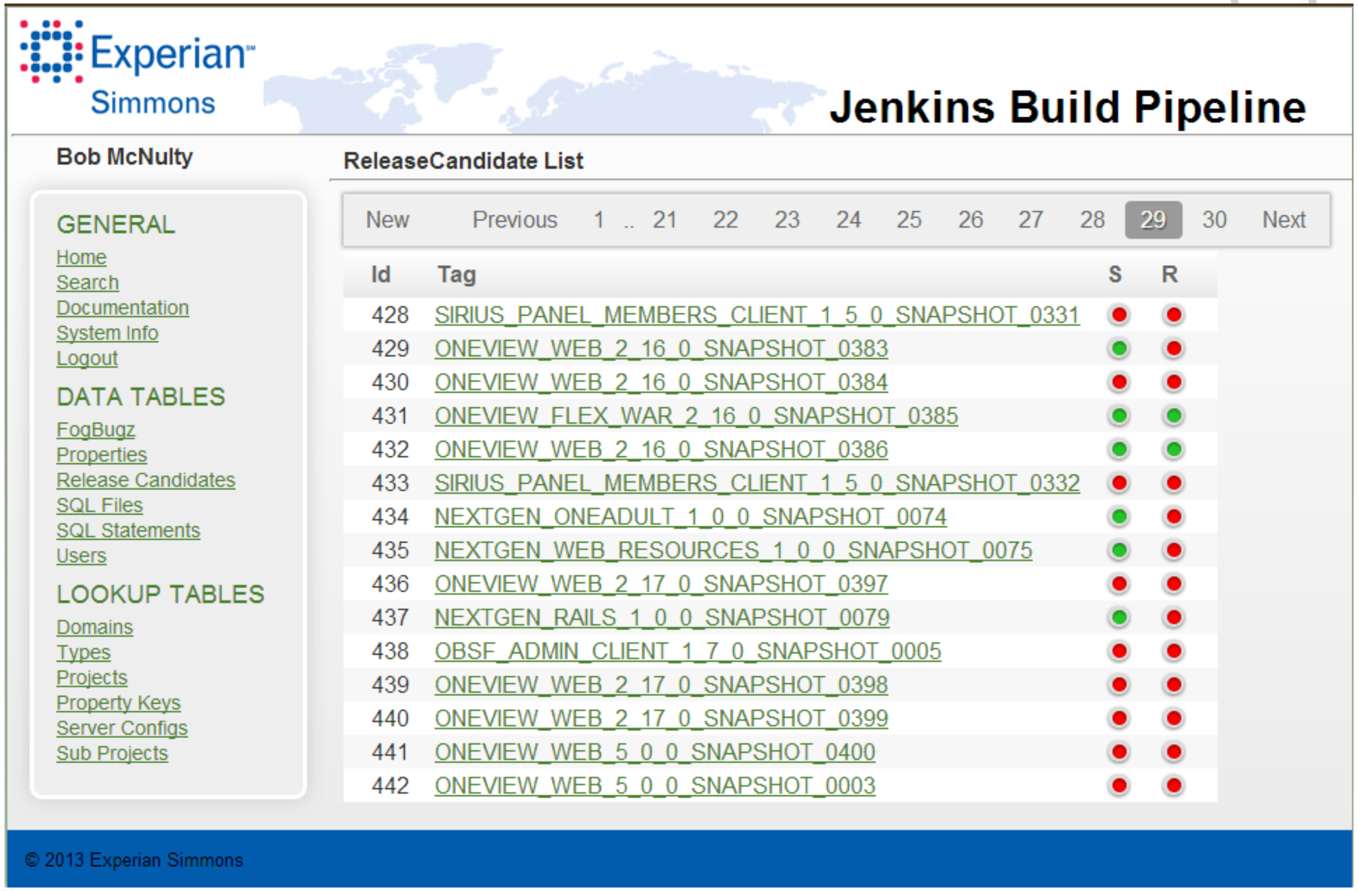
- http://localhost:8180/hello-world/hello

Notes:

- BUILD 5 : Initial build of hello-world project.
- BUILD 6 : Fixed a problem in the doohickey.
- BUILD 7 : Working on thingamajig refactoring.

User Interface Possibilities

Wouldn't it be nice to save, review, reconstruct and print reports from any and all builds via a spiffy user interface?



Experian Simmons

Jenkins Build Pipeline

Bob McNulty

ReleaseCandidate List

GENERAL
- Home
- Search
- Documentation
- System Info
- Logout

DATA TABLES
- FogBugz
- Properties
- Release Candidates
- SQL Files
- SQL Statements
- Users

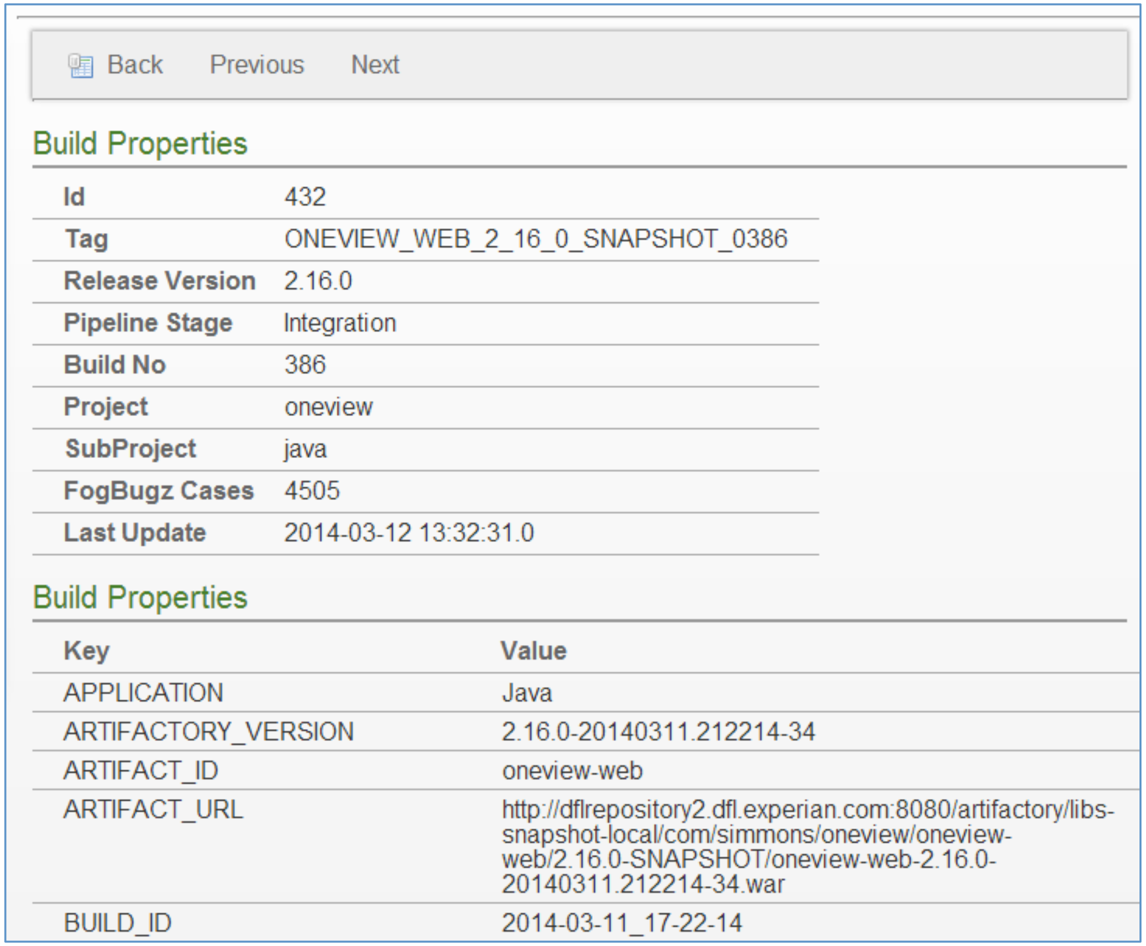
LOOKUP TABLES
- Domains
- Types
- Projects
- Property Keys
- Server Configs
- Sub Projects

New Previous 1 .. 21 22 23 24 25 26 27 28 29 30 Next

Id	Tag	S	R
428	SIRIUS_PANEL_MEMBERS_CLIENT_1_5_0_SNAPSHOT_0331	🔴	🔴
429	ONEVIEW_WEB_2_16_0_SNAPSHOT_0383	🟢	🔴
430	ONEVIEW_WEB_2_16_0_SNAPSHOT_0384	🔴	🔴
431	ONEVIEW_FLEX_WAR_2_16_0_SNAPSHOT_0385	🟢	🟢
432	ONEVIEW_WEB_2_16_0_SNAPSHOT_0386	🟢	🟢
433	SIRIUS_PANEL_MEMBERS_CLIENT_1_5_0_SNAPSHOT_0332	🔴	🔴
434	NEXTGEN_ONEADULT_1_0_0_SNAPSHOT_0074	🟢	🔴
435	NEXTGEN_WEB_RESOURCES_1_0_0_SNAPSHOT_0075	🟢	🔴
436	ONEVIEW_WEB_2_17_0_SNAPSHOT_0397	🔴	🔴
437	NEXTGEN_RAILS_1_0_0_SNAPSHOT_0079	🟢	🔴
438	OBSF_ADMIN_CLIENT_1_7_0_SNAPSHOT_0005	🔴	🔴
439	ONEVIEW_WEB_2_17_0_SNAPSHOT_0398	🔴	🔴
440	ONEVIEW_WEB_2_17_0_SNAPSHOT_0399	🔴	🔴
441	ONEVIEW_WEB_5_0_0_SNAPSHOT_0400	🔴	🔴
442	ONEVIEW_WEB_5_0_0_SNAPSHOT_0003	🔴	🔴

© 2013 Experian Simmons

User Interface Possibilities



Build Properties

Id	432
Tag	ONEVIEW_WEB_2_16_0_SNAPSHOT_0386
Release Version	2.16.0
Pipeline Stage	Integration
Build No	386
Project	oneview
SubProject	java
FogBugz Cases	4505
Last Update	2014-03-12 13:32:31.0

Build Properties

Key	Value
APPLICATION	Java
ARTIFACTORY_VERSION	2.16.0-20140311.212214-34
ARTIFACT_ID	oneview-web
ARTIFACT_URL	http://dflrepository2.dfl.experian.com:8080/artifactory/libs-snapshot-local/com/simmons/oneview/oneview-web/2.16.0-SNAPSHOT/oneview-web-2.16.0-20140311.212214-34.war
BUILD_ID	2014-03-11_17-22-14

And this…

is how… it's done!

Components Used for this Jenkins Job

Main Components	Jenkins Plugins
• Git	• Git plugin
• Groovy	• Groovy plugin
• Gradle	• Gradle plugin
• Artifactory	• Artifactory plugin
• Database (MySQL)	• email-ext plugin
	• Credentials plugin
	• Build User Vars plugin
	• EnvInject plugin
	• Workspace Cleanup plugin
	• Export Parameters plugin
	• Job Exporter plugin
	• Deploy plugin

Database Schema

Table Name	Function
release_candidates	Main table containing the build tags and version numbers. Only the release version is saved. SNAPSHOTs are logged under the release version number. Also indicates if this build was promoted to staging and production.

```
+-------------------+-------------+
| Field             | Type        |
+-------------------+-------------+
| id                |unsigned     |
| rc_project_id     | smallint(2) |
| rc_sub_project_id | smallint(3) |
| rc_version        | varchar(10) |
| tag               | varchar(80) |
| staging           | tinyint(1)  |
| production        | tinyint(1)  |
+-------------------+-------------+
```

Example:

```
               id: 3
    rc_project_id: 1
rc_sub_project_id: 1
       rc_version: 1.0.0
              tag:
HELLO_WORLD_1_0_0_SNAPSHOT_0007
          staging: 0
       production: 0
```

Database Schema – cont.

Table Name	Function
rc_sub_projects	Contains the data necessary to build and deploy the project.

```
+-----------------+-----------------------+
| Field           | Type                  |
+-----------------+-----------------------+
| id              | smallint(3) unsigned  |
| rc_project_id   | smallint(2) unsigned  |
| value           | varchar(40)           |
| title           | varchar(80)           |
| git_repo_name   | varchar(80)           |
| project_dir     | varchar(80)           |
| artifact_dir    | varchar(80)           |
| javadoc_dir     | varchar(80)           |
| artifact_id     | varchar(80)           |
| artifact_type   | varchar(8)            |
| build_profiles  | varchar(80)           |
| build_tasks     | varchar(80)           |
| domain_id       | smallint(2) unsigned  |
| context_path    | varchar(80)           |
| build_dir       | varchar(80)           |
+-----------------+-----------------------+
```

Example:

```
           id: 1
 rc_project_id: 1
        value: hello-world
        title: JUC Hello World App
 git_repo_name: juc-hello-world
   project_dir: /
  artifact_dir: build/lib
   javadoc_dir:
   artifact_id: hello-world
 artifact_type: war
build_profiles:
   build_tasks:
     domain_id: 1
  context_path: hello-world/hello
     build_dir
```

Database Schema – cont.

Table Name	Function
rc_server_configs	Contains the server data that the artifact will be deployed to

```
+----------------+-----------------------+
| Field          | Type                  |
+----------------+-----------------------+
| id             | int(10) unsigned      |
| rc_project_id  | smallint(2) unsigned  |
| rc_type_id     | smallint(2) unsigned  |
| host           | varchar(80)           |
| domain_id      | smallint(2) unsigned  |
| http_port      | smallint(5) unsigned  |
+----------------+-----------------------+
```
	Example:
	```
	           id: 1
	rc_project_id: 1
	   rc_type_id: 1
	         host: localhost
	    domain_id: 1
	    http_port: 8180
	```

Database Schema – cont.

Table Name	Function
rc_types	The type of server this build is destined for.

```
+-------------+----------------------+
| Field       | Type                 |
+-------------+----------------------+
| id          | smallint(2) unsigned |
| value       | varchar(24)          |
+-------------+----------------------+
```

```
+----+-------------+
| id | value       |
+----+-------------+
|  1 | Integration |
|  2 | Staging     |
|  3 | Production  |
|  4 | Test        |
+----+-------------+
```

Database Schema – cont.

Table Name	Function
rc_properties	Contains the miscellaneous property values that will be saved for every build.

```
+-------------+----------------------+
| Field       | Type                 |
+-------------+----------------------+
| id          | int(10) unsigned     |
| rc_id       | int(10) unsigned     |
| rc_type_id  | smallint(2) unsigned |
| build_no    | smallint(5) unsigned |
| prop_key_id | smallint(3) unsigned |
| value       | varchar(1024)        |
+-------------+----------------------+
```

Example:

```
juc_build_pipeline
hello-world
HELLO_WORLD_1_0_0_SNAPSHOT_0007
origin/master
war
hello-world
Robert McNulty
1.0.0-20140517.035034-21
hello-world/hello
/dev/data/jenkins/jobs/juc-hello-world
JUC Hello World App
Working on thingamajig refactoring.
1.0.0-SNAPSHOT
```

Groovy SQL

Groovy makes it easy to work with SQL

- From the Groovy script, just import the SQL class

```
import groovy.sql.Sql
```

- Get a connection to the database

```
def conn = Sql.newInstance('jdbc:mysql://hostName:3306/dbname',
'user', 'pwd', 'com.mysql.jdbc.Driver')
```

- This is a standard JDBC connection string.

Groovy SQL

- Run a simple, single-row query

```
def sql = """
    SELECT *
    FROM release_candidates
    WHERE tag = $tag
"""
def row = conn.firstRow(sql)
```

- Use the returned data

```
def id      = row.id          // 7
def version = row.rc_version  // 1.0.0
def tag     = row.tag         // HELLO_WORLD_1_0_0_SNAPSHOT_0007
```

Groovy SQL – cont.

- Inserting data is just as easy

```
def sql = """
   INSERT INTO release_candidates
    (rc_project_id, rc_sub_project_id, rc_version, tag)
   VALUES ($pid, $sid, $ver, $tag)
"""“


// Get the id of the inserted record
def id = conn.executeInsert(sql)[0][0]
```

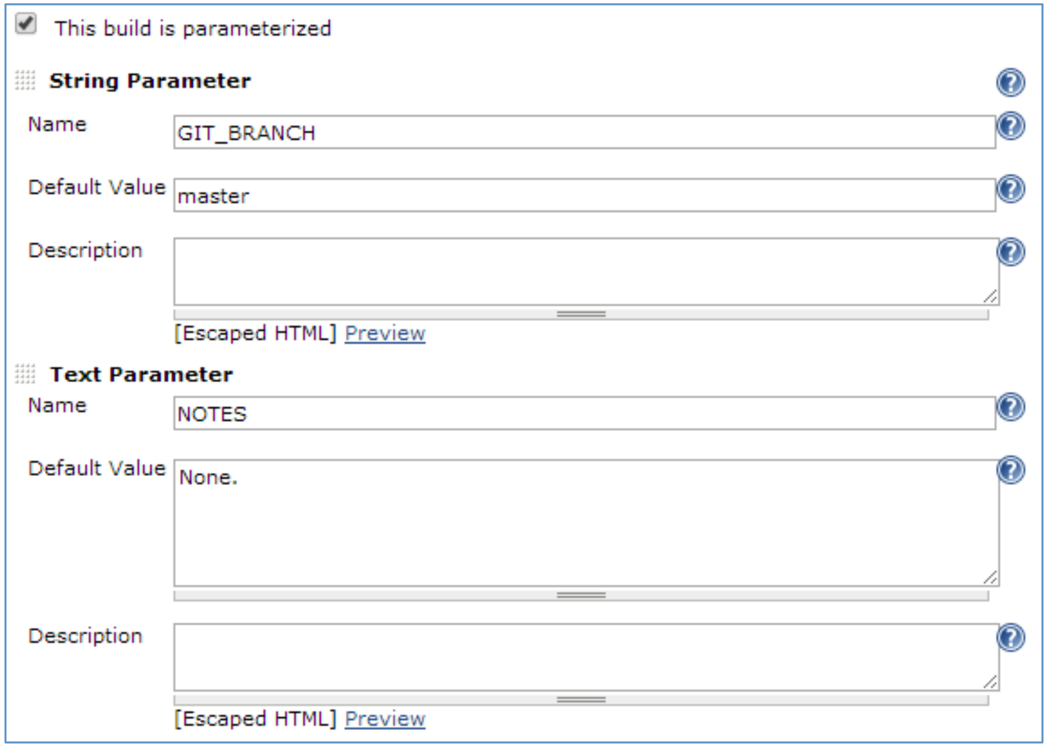
Putting it All Together

Configuring the Build

Parameterized Build

Add build parameters

- This example will only define 2 parameters
  - GIT_BRANCH
  - NOTES



[✓] This build is parameterized

String Parameter

Name	GIT_BRANCH
Default Value	master
Description	

[Escaped HTML] Preview

Text Parameter

Name	NOTES
Default Value	None.
Description	

[Escaped HTML] Preview

Configuring the Build
Git Setup

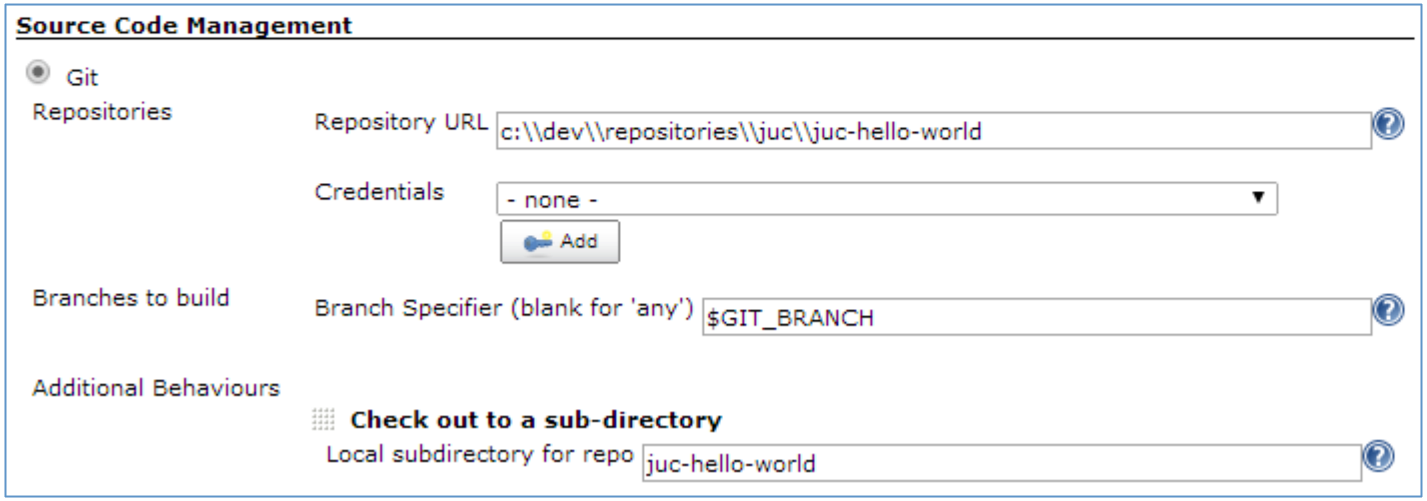
Source Code Management

- In ‘Branches to build’, enter the parameter
  - `$GIT_BRANCH`
- The variable will be replaced by the value entered by the user, defaulting to ‘master’. This is the branch that will be checked out and built.

Configuring the Build

Git Setup – cont.

- Check out to a sub-directory
  - Since we will also be checking out the Groovy code from a different repository, it is important to keep the projects separated.



Source Code Management

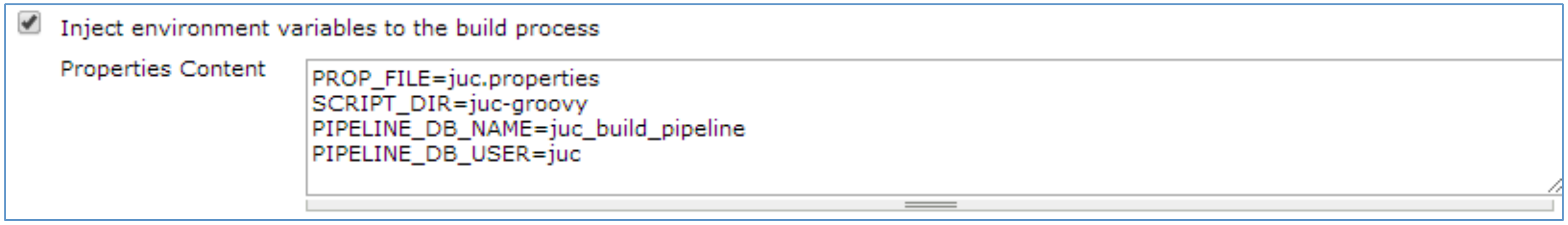
● Git

Repositories	Repository URL	c:\\dev\\repositories\\juc\\juc-hello-world
	Credentials	- none -
Branches to build	Branch Specifier (blank for 'any')	$GIT_BRANCH
Additional Behaviours	**Check out to a sub-directory**	
	Local subdirectory for repo	juc-hello-world

Configuring the Build
Environment Variables

Inject environment variables to the build process

- Define global properties, used in both the Jenkins job and the Groovy script, in the 'Property content' section.



☑ Inject environment variables to the build process

Properties Content

```
PROP_FILE=juc.properties
SCRIPT_DIR=juc-groovy
PIPELINE_DB_NAME=juc_build_pipeline
PIPELINE_DB_USER=juc
```

Configuring the Build

Environment Variables

Inject environment variables to the build process

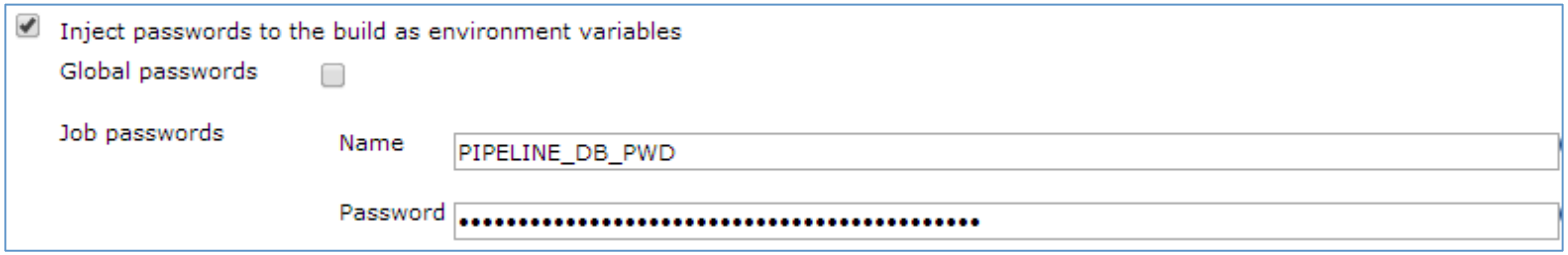
Property	Value	Description
PROP_FILE	juc.properties	The file containing all of the properties used in the build. An easy method of transferring data between Jenkins and Groovy script.
SCRIPT_DIR	juc-groovy	The location of the Groovy files. The files are also under source control, so this is the cloned location.
PIPELINE_DB_NAME	juc_build_pipeline	Name of the database to use. It is convenient to define this per job so that a test database can be easily swapped in.
PIPELINE_DB_USER	juc	Name of the database user.

Configuring the Build

Inject Passwords

Inject passwords into the build as environment variables

- Passwords are not included in the regular environment variable section because they would then be visible to end-users via to logs.
- Here, we are defining the database password that will be passed to the Groovy script.



[✓] Inject passwords to the build as environment variables

Global passwords [ ]

Job passwords

Name: PIPELINE_DB_PWD

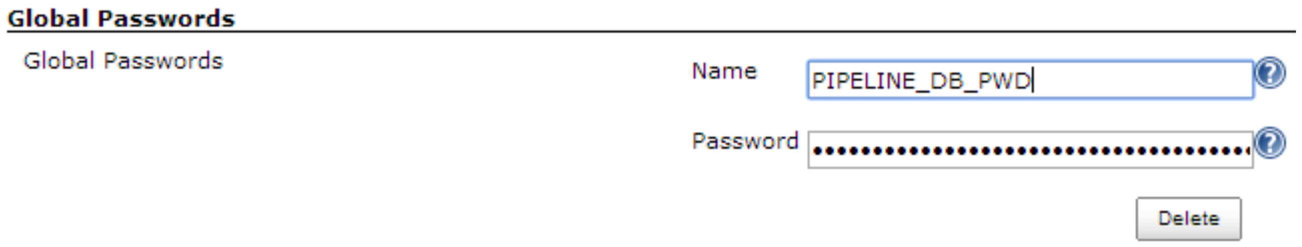
Password: ••

Configuring the Build

Inject Passwords – cont.

Inject passwords into the build as environment variables

- Passwords can also be defined at the global level so they are available to ALL Jenkins jobs. Simply check the 'Global passwords' box.
- Globally, they are defined on the `Manage Jenkins -> Configure System` page.



Global Passwords

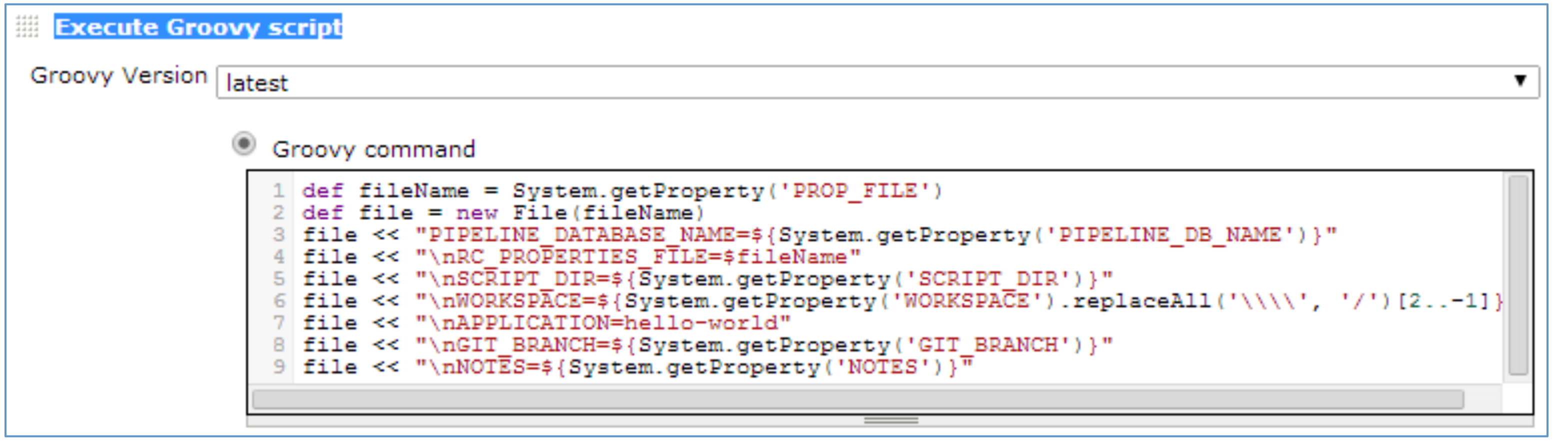
Global Passwords	Name	PIPELINE_DB_PWD
	Password	••••••••••••••••••••••••••••••••••••••

Configuring the Build

Build Steps

Execute Groovy script

- The first step is to execute inline Groovy commands to save the build properties to an external file.
- This makes it easier to pass properties back and forth between the Jenkins job and the Groovy script.
- Especially useful when there are many properties



Execute Groovy script

Groovy Version: latest

(●) Groovy command

```
def fileName = System.getProperty('PROP_FILE')
def file = new File(fileName)
file << "PIPELINE_DATABASE_NAME=${System.getProperty('PIPELINE_DB_NAME')}"
file << "\nRC_PROPERTIES_FILE=$fileName"
file << "\nSCRIPT_DIR=${System.getProperty('SCRIPT_DIR')}"
file << "\nWORKSPACE=${System.getProperty('WORKSPACE').replaceAll('\\\\', '/')[2..-1]}
file << "\nAPPLICATION=hello-world"
file << "\nGIT_BRANCH=${System.getProperty('GIT_BRANCH')}"
file << "\nNOTES=${System.getProperty('NOTES')}"
```

Configuring the Build

Build Steps – cont.

export job runtime parameters

- This build step exports various build job and hudson parameters into a property file named ***hudsonBuild.properties*** in the project workspace.
- The external Groovy script will read this file and load several of the properties into the master properties file

Property	Value
build.jobName	Name of running job
build.number	Number of running job
build.result	Job result until this build step
build.gitBranch	GIT branch, if configured
build.user.id	ID of user that triggered this job
build.user.name	User that triggered this job
build.user.fullName	Full name of user that triggered this job
build.user.emailAddress	Email address of user that triggered this job

Configuring the Build

Build Steps – cont.

Execute Windows batch or Shell command

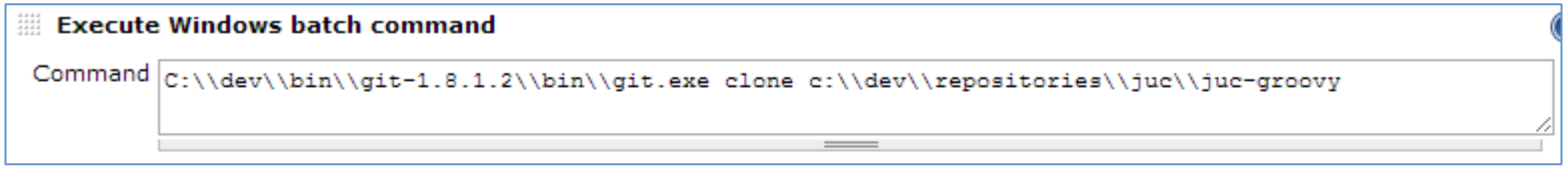
- The Jenkins Git plugin does not currently allow multiple repositories to be cloned into distinct locations
- We have used the Git plugin to check out the project code, that is, the code the developer has checked in
- We will also be checking out the Groovy code project from a different repository
- It is important to keep the projects separated
- A sort of project namespacing

Configuring the Build

Build Steps – cont.

Execute Windows batch or Shell command

- This command will clone the Git project containing the Groovy script that will create new properties, access the database, etc.
- Windows

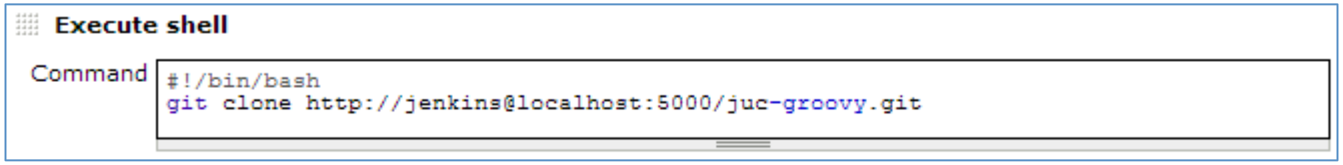


Execute Windows batch command

Command

```
C:\\dev\\bin\\git-1.8.1.2\\bin\\git.exe clone c:\\dev\\repositories\\juc\\juc-groovy
```

- Linux



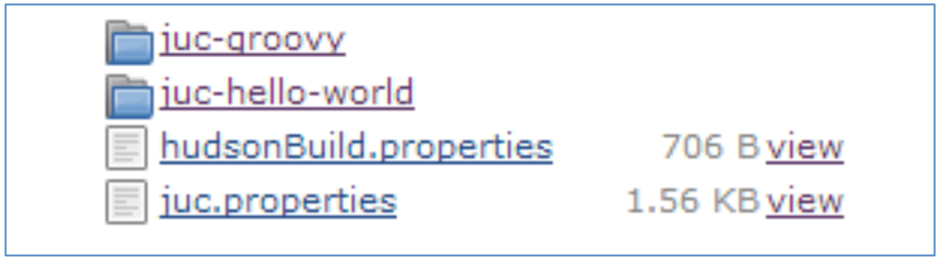
Execute shell

Command

```
#!/bin/bash
git clone http://jenkins@localhost:5000/juc-groovy.git
```

Configuring the Build
Workspace

- View of the job workspace after the Git repositories have been cloned and the property files created



```
juc-groovy
juc-hello-world
hudsonBuild.properties      706 B view
juc.properties             1.56 KB view
```

Configuring the Build

Groovy Script

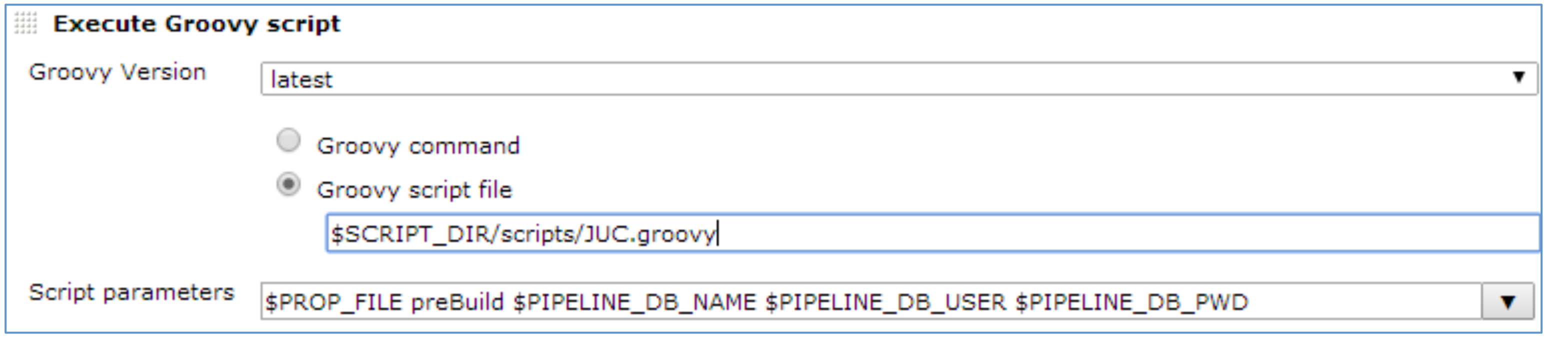
Execute Groovy Script

- The Groovy script is called twice, first , using the ‘preBuild’ action and finally using the ‘postBuild’
- ‘preBuild’ is executed prior to running the Gradle build script
- ‘postBuild’ is executed after the successful completion of the Gradle build

Configuring the Build

Groovy Script – preBuild

Execute Groovy Script - preBuild



Execute Groovy script

Groovy Version: latest

Groovy command [✓ Groovy script file]

$SCRIPT_DIR/scripts/JUC.groovy

Script parameters: $PROP_FILE preBuild $PIPELINE_DB_NAME $PIPELINE_DB_USER $PIPELINE_DB_PWD

This script will:

```
1. Load the properties files ($PROP_FILE)
2. Query the database for the project and server information
3. Save database values to properties
4. Append new properties back to properties file
```

Some of these properties will be used by the Gradle build.

Configuring the Build

Groovy Script – juc.properties

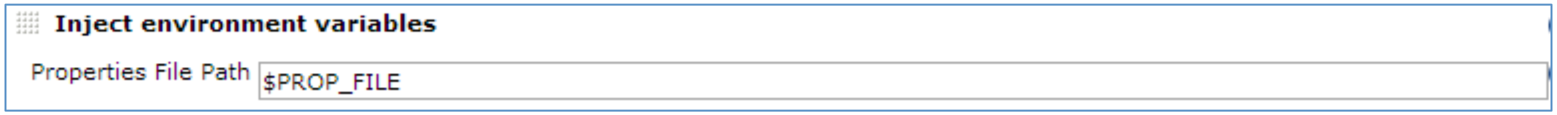
```
PIPELINE_DATABASE_NAME=juc_build_pipeline
RC_PROPERTIES_FILE=juc.properties
SCRIPT_DIR=juc-groovy
WORKSPACE=/dev/data/jenkins/jobs/juc-hello-world-2/workspace
APPLICATION=hello-world
GIT_BRANCH=origin/master
ARTIFACT_DIR=/dev/data/jenkins/jobs/juc-hello-world-2/workspace/juc-hello-world//build/lib
ARTIFACT_ID=hello-world
ARTIFACT_TYPE=war
BUILD_DIR=/dev/data/jenkins/jobs/juc-hello-world-2/workspace/juc-hello-world
BUILD_NUMBER=90
BUILD_USER_FULLNAME=Robert McNulty
CONTEXT_PATH=hello-world/hello
EMAIL_SUBJECT=JUC Hello World App
GIT_REPOSITORY_DIR=/dev/data/jenkins/jobs/juc-hello-world-2/workspace/juc-hello-world
GIT_REPO_NAME=juc-hello-world
PROJECT_DIR=/dev/data/jenkins/jobs/juc-hello-world-2/workspace/juc-hello-world/
SERVER_HOST=localhost
SERVER_PORT=8180
SERVER_URL=http://localhost:8180/
APP_LINKS_HTML=<li>http://localhost:8180/hello-world/hello</li>
ARTIFACTORY_VERSION=1.0.0-20140519.034403-8
POM_GROUP_ID=juc
POM_VERSION=1.0.0-SNAPSHOT
RC_TAG=HELLO_WORLD_1_0_0_SNAPSHOT_0090
RELEASE_VERSION=1.0.0
USER_NOTES_HTML=<li>BUILD 5 : Initial build of hello-world project.</li><li>BUILD 6 : Fixed a
problem in the doohickey.</li><li>BUILD 7 : Working on thingamajig refactoring.</li>
WAR_PATH=/dev/data/jenkins/jobs/juc-hello-world-2/workspace/juc-hello-world//build/lib/hello-
world-1.0.0-SNAPSHOT.war
```

Configuring the Build

Build Steps – cont.

Inject environment variables

- After the properties have been written to the file, it is necessary to reload the file into Jenkins so the build can use the properties
- The '**Inject environment variables'** plugin accomplishes this task



Inject environment variables

Properties File Path $PROP_FILE

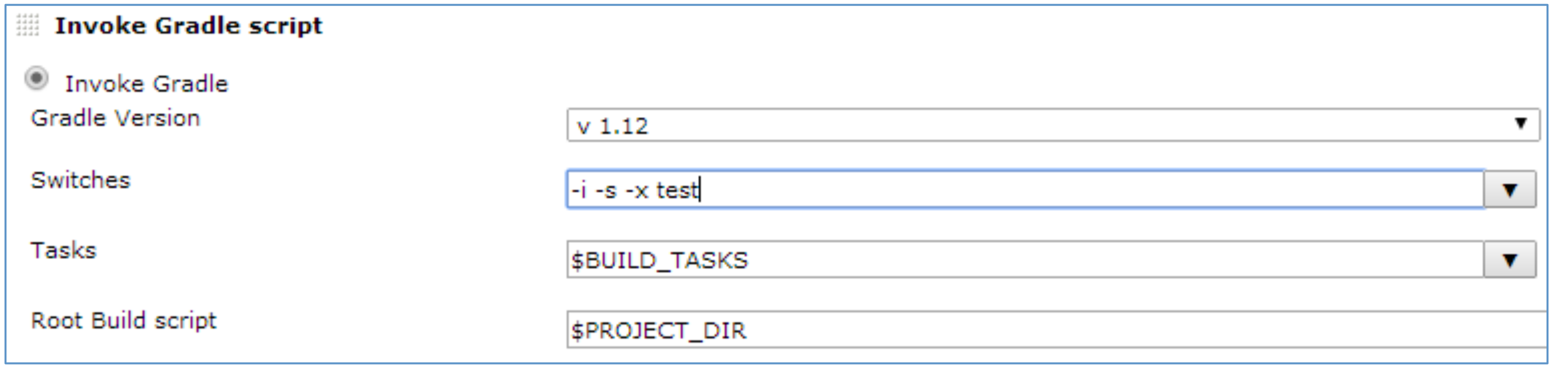
- The properties that were retrieved from the database are now available to the Jenkins job

Configuring the Build

Build Steps – Gradle

Run the Gradle Build

- Build the project using the Gradle plugin
  - **Note**: Maven can be used in place of Gradle



Invoke Gradle script

● Invoke Gradle

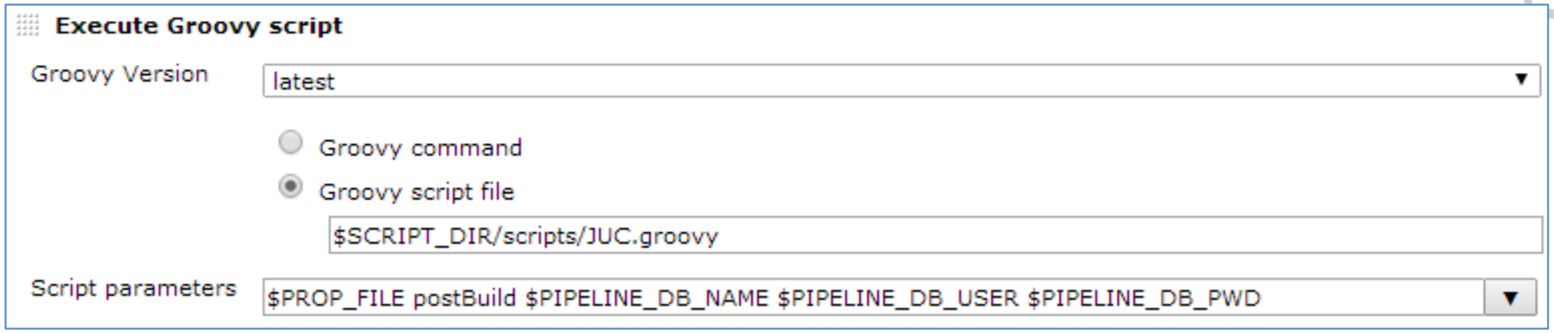
Gradle Version	v 1.12
Switches	-i -s -x test
Tasks	$BUILD_TASKS
Root Build script	$PROJECT_DIR

- The **$BUILD_TASKS** and **$PROJECT_DIR** variables were loaded from the properties file

Configuring the Build

Build Steps – postBuild

Execute Groovy Script - postBuild



Execute Groovy script

Groovy Version: latest

( ) Groovy command
(●) Groovy script file

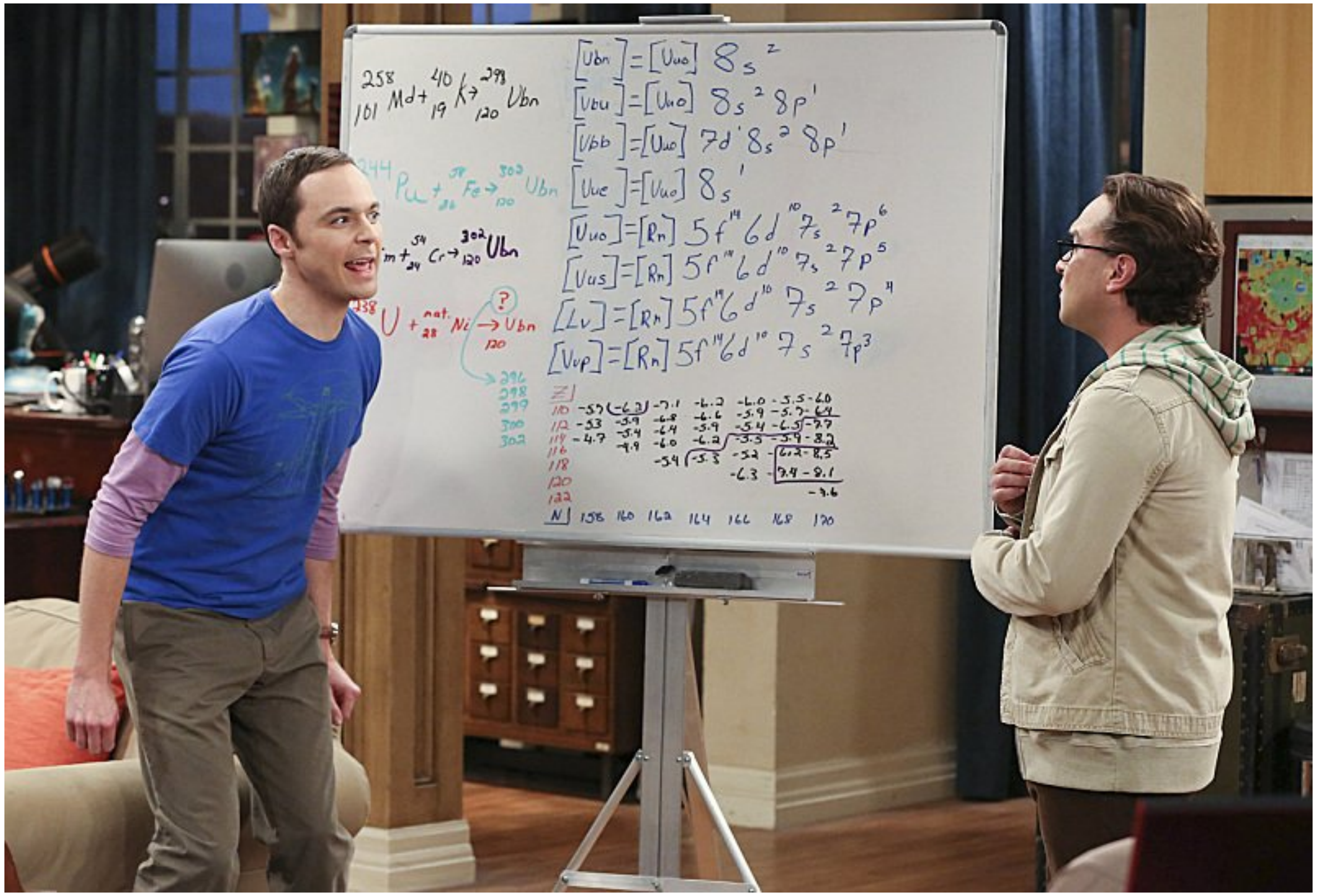
$SCRIPT_DIR/scripts/JUC.groovy

Script parameters: $PROP_FILE postBuild $PIPELINE_DB_NAME $PIPELINE_DB_USER $PIPELINE_DB_PWD

This script will:

```
1. Load the properties file ($PROP_FILE)
2. Get artifact information (groupId, version, Artifactory URL)
3. Create unique tag
4. Aggregate notes from previous builds
5. Create HTML elements to use in report
6. Create and push new branch to Git (using tag)
7. Append new properties back to properties file
```

Post-build Actions …and more!



Configuring the Build

Post-build Actions

Post-build actions

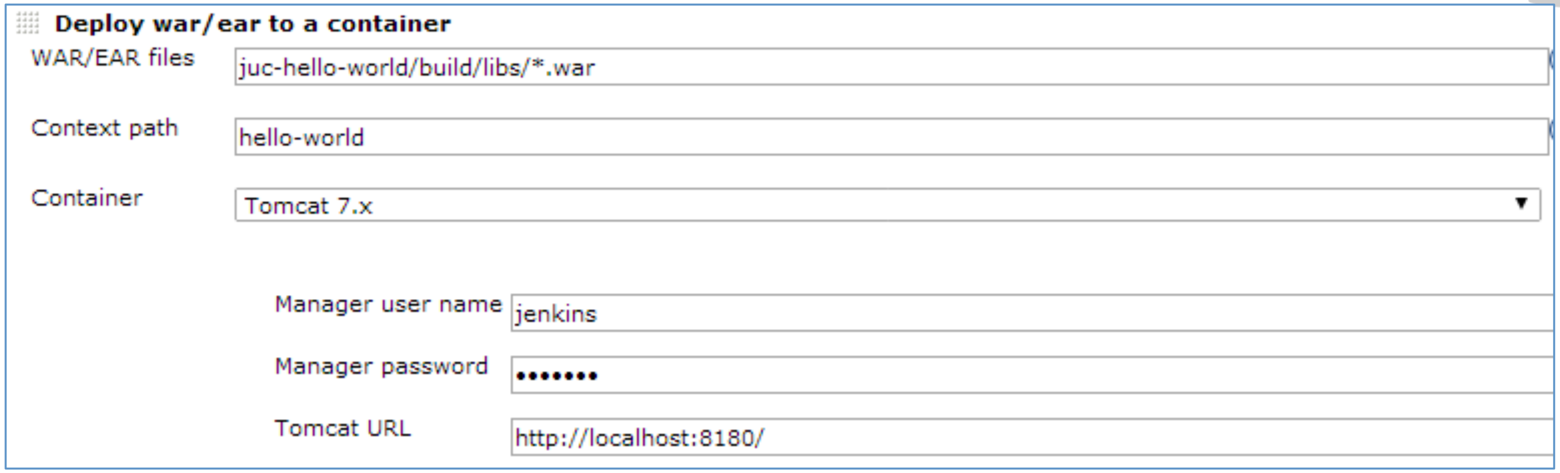
Once the Gradle build has completed, the artifacts have been deployed to Artifactory and the Groovy scripts have finished processing, there are still procedures that are required to successfully complete the Jenkins job

- Deploy artifact to Tomcat (if webapp)
- Send the Email

Configuring the Build

Deploy to Container

Deploy to Tomcat



Deploy war/ear to a container

WAR/EAR files	juc-hello-world/build/libs/*.war
Context path	hello-world
Container	Tomcat 7.x
Manager user name	jenkins
Manager password	•••••••
Tomcat URL	http://localhost:8180/

- For this demo, I used the 'Deploy Plugin' for simplicity.
- Unfortunately, this plugin does not recognize the environment variables imported into Jenkins.

Configuring the Build

Deploy to Container – cont.

Deploy to Tomcat

- In our production instance of Jenkins, I deploy to Tomcat via Groovy script
- Since our Jenkins instance runs on Linux, it is a simple matter to use ‘curl’ to access the Tomcat manager and list, deploy or undeploy applications

```
static boolean deploy(context, path) {
   def curl = new StringBuilder().with {
      append "curl -X PUT -F file=@$path “
      append ‘http://jenkins@localhost:8180’
      append “/manager/text/deploy?path=$context"
   }.toString()
   def result = curl.execute().text
   return result.startsWith('OK')
}
```

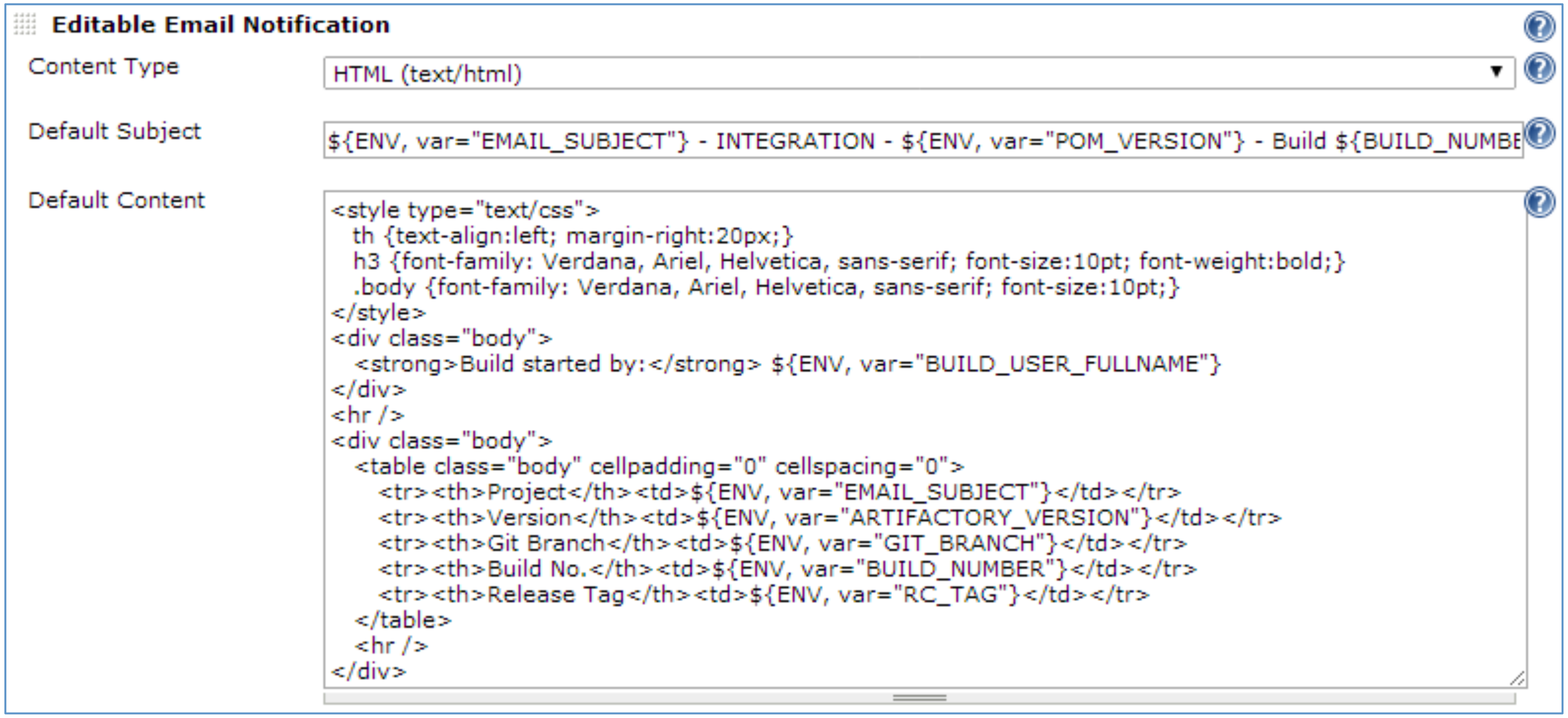
Configuring the Build
Email Report

Editable Email Notification

- The ‘piece de resistance’ is the email report delivered to a list of concerned recipients
- The ‘email-ext plugin’ does have the ability to use the imported environment variables
- Using the HTML Content Type enables the use of straight HTML to format the page
- The ‘dynamic’ portions of the email are populated using the properties generated by the Groovy script and imported into Jenkins

Configuring the Build
Email Report – cont.

Use HTML and CSS in the report template



Editable Email Notification

Content Type	HTML (text/html)
Default Subject	${ENV, var="EMAIL_SUBJECT"} - INTEGRATION - ${ENV, var="POM_VERSION"} - Build ${BUILD_NUMBE

Default Content

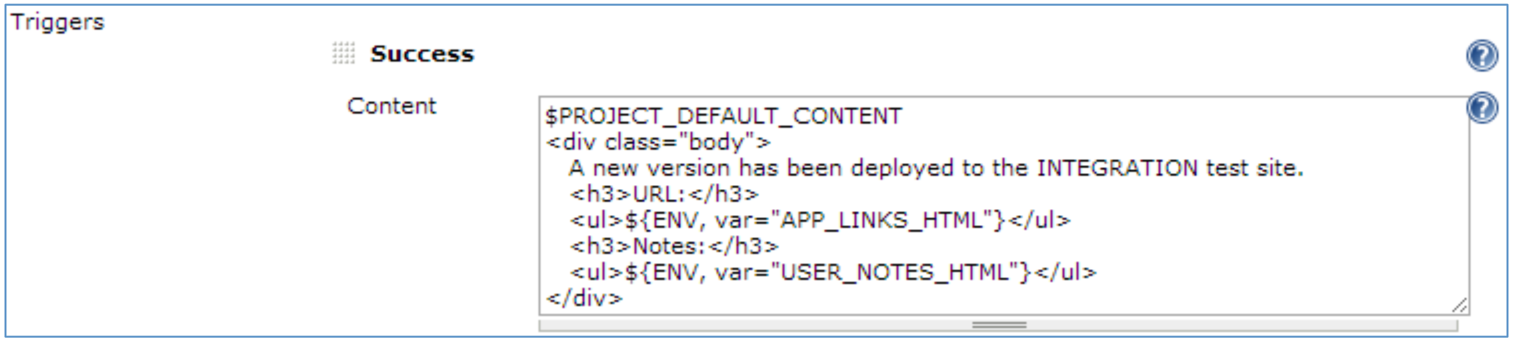
```
<style type="text/css">
  th {text-align:left; margin-right:20px;}
  h3 {font-family: Verdana, Ariel, Helvetica, sans-serif; font-size:10pt; font-weight:bold;}
  .body {font-family: Verdana, Ariel, Helvetica, sans-serif; font-size:10pt;}
</style>
<div class="body">
  <strong>Build started by:</strong> ${ENV, var="BUILD_USER_FULLNAME"}
</div>
<hr />
<div class="body">
  <table class="body" cellpadding="0" cellspacing="0">
    <tr><th>Project</th><td>${ENV, var="EMAIL_SUBJECT"}</td></tr>
    <tr><th>Version</th><td>${ENV, var="ARTIFACTORY_VERSION"}</td></tr>
    <tr><th>Git Branch</th><td>${ENV, var="GIT_BRANCH"}</td></tr>
    <tr><th>Build No.</th><td>${ENV, var="BUILD_NUMBER"}</td></tr>
    <tr><th>Release Tag</th><td>${ENV, var="RC_TAG"}</td></tr>
  </table>
  <hr />
</div>
```

Configuring the Build

Email Report – cont.

Use HTML and CSS in the report template

- If the build is successful, configure the 'Success' trigger



Triggers

Success

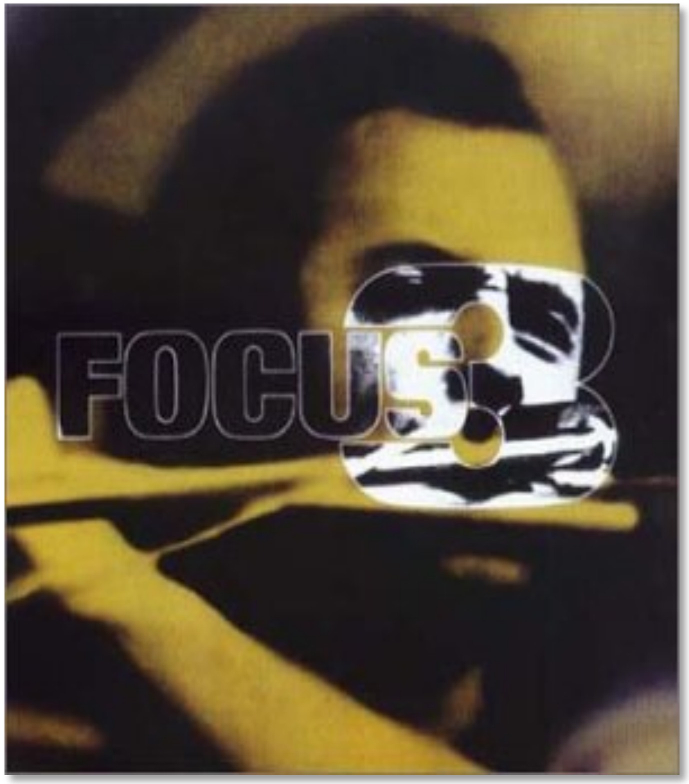
Content

```
$PROJECT_DEFAULT_CONTENT
<div class="body">
  A new version has been deployed to the INTEGRATION test site.
  <h3>URL:</h3>
  <ul>${ENV, var="APP_LINKS_HTML"}</ul>
  <h3>Notes:</h3>
  <ul>${ENV, var="USER_NOTES_HTML"}</ul>
</div>
```

Demonstration

Questions? Answers!

…Answers? Questions!



Thank You To Our Sponsors

Platinum

CloudBees

Gold

black diamond SOFTWARE

MidVision™ Release the innovation

choose digital

JFrog

XebiaLabs the deployment automation company

Silver

SOASTA Test Faster. Release Sooner.

ELIASSEN GROUP Agile Consulting Services

KENSHO

SERENA™

assembla