# Automating your CI/CD Stack with Java and Groovy Hands on Lab

Automating your CI/CD Stack with Java and Groovy Hands on Lab 1

Step 0: Install tools 3

Docker 3

Docker images 3

Groovy 4

Disk space 5

The ability to run curl 5

Step 1: Start Nexus and Jenkins 6

Create network bridge 6

Install Nexus 6

Install Jenkins 6

Unlock Jenkins 6

Step 2: Basic Jenkins Scripting 7

Run your first Groovy script in Jenkins 7

Setup Jenkins so can run jobs with Groovy steps 7

Run your first job containing Groovy scripting 8

Writing your own script in Jenkins 9

Step 3: Basic Nexus Scripting 10

Run your first Groovy script in Nexus 10

Looking in the task log on the command line 10

Writing your own script in Nexus 11

log vs println 11

Step 4: Jenkins System Scripting 11

Tools for finding methods 12

Creating a user 12

Setting up the authorization strategy 13

Step 5: Nexus System Scripting 14

Learn what objects are available 14

Create global access and Jenkins access 14

Setup Nexus Access 14

Step 6 – Create a Jenkins pipeline 15

Setup Nexus Credentials in Jenkins 15

Create a job to setup jobs with proper configuration 15

Testing Access and Running Jobs 16

Editing a pipeline job 16

Step 7 – Switch services 16

Stop Nexus and Jenkins to free up some resources 16

Install SonarQube 16

Step 8 – Creating a custom SonarQube rule 16

Deploying a custom rule 17

Add this rule to the default quality profile 17

Run a scan with this custom rule 17

Step 9 – Extra credit in Jenkins 18

Final step: Cleanup 18

Stop containers 19

If want to reclaim disk space 19

Uh oh. I managed to make it so the admin user doesn’t have read access to Jenkins 19

Faster way – install your favorite text editor 19

Slower way – delete your docker container/image and start over 20

References 20

## Step 0: Install tools

This lab requires the following:

1. Docker
2. Docker images
3. Groovy
4. The ability to run curl

### Docker

To install Docker, follow the instructions at: <https://docs.docker.com/install/>

Validation:

At the command line, run **docker –version**. (Small version differences are ok in the output)

$ docker --version

Docker version 18.06.0-ce, build 0ffa825

### Docker images

These images take up just under 2GB combined. So as not to tax the conference network, please pull them in advance. This will also let you deal with any corporate internet proxies while you are still at work and can ask for help.

At the command line run:

**docker pull sonatype/nexus3:3.13.0**

**docker pull jenkins/jenkins:2.140**

**docker pull sonarqube:7.1**

**Note: “latest” will probably work. These version numbers are the ones we tested with.**

$ docker pull sonatype/nexus3:3.13.0

3.13.0: Pulling from sonatype/nexus3

256b176beaff: Pull complete

18d124afa1e9: Pull complete

9bb412307f82: Pull complete

Digest: sha256:19d186d5bc8be1ea4f7bae72756baa830e79bf20aae0e9e7b1a0c7d3ce7ac136

Status: Downloaded newer image for sonatype/nexus3:3.13.0

$ docker pull jenkins/jenkins:2.140

2.140: Pulling from jenkins/jenkins

55cbf04beb70: Pull complete

1607093a898c: Pull complete

9a8ea045c926: Pull complete

d4eee24d4dac: Pull complete

c58988e753d7: Pull complete

794a04897db9: Pull complete

70fcfa476f73: Pull complete

806029475e0c: Pull complete

67959b355155: Pull complete

4d217ccd3d4c: Pull complete

0261bb88a4a5: Pull complete

96f2a3ae5539: Pull complete

f6bf99db32d5: Pull complete

bb47d4bbb0e1: Pull complete

4b48ec5d60cf: Pull complete

7280a8dfb767: Pull complete

91091f8d44ca: Pull complete

8ca02cad320f: Pull complete

46009bfec329: Pull complete

f9860b79812e: Pull complete

89ac8103ea67: Pull complete

Digest: sha256:161cb25fbb23a1c5ac5fdd0feebd713edd62c235e199e68b34d1a78205a42da7

Status: Downloaded newer image for jenkins/jenkins:2.140

$ docker pull sonarqube:7.1

7.1: Pulling from library/sonarqube

55cbf04beb70: Already exists

1607093a898c: Already exists

9a8ea045c926: Already exists

d4eee24d4dac: Already exists

c58988e753d7: Already exists

794a04897db9: Already exists

70fcfa476f73: Already exists

806029475e0c: Already exists

67959b355155: Already exists

1e6b3af7f55a: Pull complete

e0b67c57c8e1: Pull complete

ce12e009fbe7: Pull complete

3edf8e47f9c4: Pull complete

Digest: sha256:4438a37735caa24d80da31ee29e72d686abdaa8f5009746ec60e0d43519e1a57

Status: Downloaded newer image for sonarqube:7.1

### Groovy

Follow the instructions at: <http://groovy-lang.org/install.html>

Note: Please download Groovy 2.X rather than 3.X. Version 3.X is in alpha at the time of writing this lab.

Validation:

At the command line, run **groovy –version**.(Small version differences are ok in the output)

$ groovy -version

Groovy Version: 2.5.2 JVM: 1.8.0\_45 Vendor: Oracle Corporation OS: Mac OS X

### Disk space

This lab uses about 1.5 GB of disk space (not counting Docker itself.) The last step of the lab explains how to recover disk space.

### The ability to run curl

If you are running Mac, Linux or Windows 10, you don’t need to do anything. If you are running an older version of Windows, you can install <https://www.cygwin.com>.

Validation:

At the command line, run

**curl https://www.oracle.com/code-one/index.html**

and ensure the output isn’t an error message.

## Step 1: Start Nexus and Jenkins

In this lab, we use basic Docker to make setup simple. We do not set up a volume to persist the data since the emphasis is Groovy setup. (We will start Sonar at the end after stopping Nexus and Jenkins to limit the laptop resources needed by this lab)

### Create network bridge

**docker network create hol-network**

### Install Nexus

At the command line, run:

**docker run -d -p 8081:8081 --name nexus --network hol-network sonatype/nexus3:3.13.0**

*Note: If you’ve run it in the past, type* ***docker start nexus*** *instead*

Validation

In a browser, go to <http://localhost:8081>. If you do not see Nexus, wait 15 seconds and try again. (It could take up to a minute)

### Install Jenkins

At the command line, run

**docker run -d -p 8080:8080 --name jenkins --network hol-network jenkins/jenkins:2.140**

*Note: If you’ve run it in the past, type* ***docker start jenkins*** *instead*

### Unlock Jenkins

1. At the command line, run:

**docker exec -it jenkins bash**

1. Print the password:

**cat /var/jenkins\_home/secrets/initialAdminPassword**

1. In a browser, go to <http://localhost:8080>



1. Paste in the password and click continue
2. Allow Jenkins to install the suggested plugins
3. In the command line window, type “exit” to get out of bash.
4. Pick a username/password for your admin test user. Enter all the fields and click “Save and Continue”
5. Click to Finish
6. Click “Start using Jenkins”



## Step 2: Basic Jenkins Scripting

### Run your first Groovy script in Jenkins

Let’s confirm what version of Groovy is being used in Jenkins.

1. On the left navigation, click “Manage Jenkins”
2. Click “Script Console”
3. Write:

**println 'Hello Oracle Code One!** **'**

**println GroovySystem.version**

1. The result is right below the text area:



### Setup Jenkins so can run jobs with Groovy steps

We need the Groovy plugin to create jobs with Groovy steps.

1. In the left navigation, click “Manage Jenkins”
2. Click “Manage Plugins”
3. Click the “Available” tab
4. Select “Groovy” (note the plugins are not in alphabetical order). Also, install the Role-based Authorization Strategy plugin. We’ll use that one later.



1. Click the button “Download now and install after restart”
2. Check to restart



1. Refresh the page after a few seconds (or click “Enable Auto Refresh” on top)
2. Login as admin
3. Click “Manage Jenkins” again
4. Click “Global Tool Configuration”
5. Click “Add Groovy”



1. Set the name to “Groovy 2.X”. Leave the version as the default. (This gives us a slightly different of Groovy, but this is fine as long as we stay on the 2.X series.)



1. Click “Save”

### Run your first job containing Groovy scripting

1. In the left navigation, click “New item”
2. Enter “my-first-script” as the job name and choose “Freestyle Project”



1. Scroll down and click “Ok”
2. Add a Groovy build step



1. Choose “Groovy 2.X” to run
2. In the textbox, enter the code

**println 'Hello Oracle Code One!** **'**

**println GroovySystem.version**

****

1. Click “Save”
2. Click “Build Now”
3. Click your job number and view the console output.

Tip: You can click the circle to go directly to the console.



1. Note that Jenkins installed Groovy the first time it was needed.



### Writing your own script in Jenkins

This time we are going to write a script where you fill in the blank to print out all the strings that end with “day”.

Tip: If you’ve never used Groovy before, see the solution guide for how to do this. (also note that Groovy uses grep where Java uses filter)

def list = ['monday', 'wednesday', 'chocolate', 'friday']

println \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Use the Jenkins admin script console to test your script until you are happy with it. Then create a job named “days” that runs the same Groovy code.

## Step 3: Basic Nexus Scripting

### Run your first Groovy script in Nexus

Let’s confirm what version of Groovy is being used.

1. Login using admin/admin123.
2. Click the gear icon to get the administration options



1. In the left navigation, scroll down and choose the last line “Tasks”
2. Click “Create Task”
3. Choose “Admin - Execute Script”
4. Enter a task name.
5. For the script source, enter:

**log.info 'Hello Oracle Code One!** **'**

**log.info GroovySystem.version**

1. For task frequency, choose “Manual”
2. Click “Create Task”
3. Click the task name
4. Click Run



1. Click “Yes” to confirm
2. Click Summary tab. Depending on how fast you are, it will either say Running or Waiting. Refresh if it is still running to confirm it completes.



1. In the left navigation go to Support > Logging > Log Viewer
2. Notice your logging is there:



1. Note that this is not the same version as Jenkins used; but it is close enough to not cause us difficulties.

### Looking in the task log on the command line

1. In a new command line tab/window, run:

**docker exec -it nexus bash**

1. Print the log file:

**cd /nexus-data/log/tasks**

**ls**

**cat \*.log**

1. Confirm you see 8 lines.
2. Now find the main nexus.log and run

**tail –f nexus.log**

1. In your browser, re-run your Groovy task. How many lines do you see added to the tail output? It’s not 8 lines. Can you figure out the difference between the nexus log and task log?

### Writing your own script in Nexus

Now try to write your own Groovy script that prints out a countdown. The output includes the arrows and should be:

--> 10

--> 9

--> 8

--> 7

--> 6

--> 5

--> 4

--> 3

--> 2

--> 1

Blast off!

Tip for Groovy developers: the log.info method takes a Java String and not a GString.

Tip for Java developers: you can use groovy on your machine or the Jenkins scripting console to interactively nail down the syntax. Or look at the solution guide.

Check the output in both a browser (nexus log) and command line (task log).

**Challenge**: can you write the Groovy code in two lines of no more than 40 characters each? You can use <https://www.lettercount.com> to check the number of characters.

### log vs println

1. In your prior example, change **log.info** to **println**.
2. Run the task
3. Check the output in both the browser and command line task log. Where do you see it?

## Step 4: Jenkins System Scripting

We are going to setup Jenkins to have different teams and permissions for each.

* The Open Ospreys allow all Jenkins users to see their projects. Olivia and Owen are on this team.
* The Secret Sea Lions only allow their team members to see their projects. Sam and Sophia are on this team.

It’s easiest to start in the scripting console and create our script interactively. Once we know the syntax and APIs, we’ll delete that configuration. Then we will run the Groovy code through a job to be repeatable.

Note: If you are using LDAP/AD and/or Jenkins Enterprise, you’ll have more powerful access control options available to you.

### Tools for finding methods

Note that Jenkins was an offshoot of Hudson. The source code betrays this fact and still has some Hudson class/package names.

Tool 1 - JavaDoc

1. In the Jenkins scripting console, write the following code

**def instance = Jenkins.getInstance()**

**println instance.class**

1. This prints out

class hudson.model.Hudson

1. Knowing the class name allows you to look at the JavaDoc for Jenkins core along with popular plugins

<https://javadoc.jenkins.io>

<https://javadoc.jenkins.io/plugin>

Tool 2 - Groovy

1. Now write the following code

**def instance = Jenkins.getInstance()**

**println instance.metaClass.methods\*.name.sort().unique()**

1. The metaClass methods approach is a quick way to find out what methods are available on an object you currently have.
2. You can even see the method signatures:

**def instance = Jenkins.getInstance()**

**println instance.metaClass.methods**

Tool 3 – Google

Seriously. People have posted many Groovy scripts online that you can either use directly or make minor edits to.

### Creating a user

To help us keep track of who is who, we are going to give our users the same username and password. Obviously, don’t do this on your real Jenkins!

1. Users are created in security realms. Run this code to create a user:

**def instance = Jenkins.getInstance()**

**def realm = Jenkins.getInstance().securityRealm**

**realm.createAccount('olivia', 'olivia')**

**instance.save()**

1. In the left navigation, click “People” and note Olivia is there now.
2. Click her id and then click “Delete” from the left navigation.
3. Create a new freestyle job
4. This time add a System Groovy build step



1. Add three more lines of code so Owen, Sam and Sophia get accounts.

Note: Since we are now running in a job, we need to explicitly include the package for Jenkins. The console assumed this for you. (**import jenkins.model.Jenkins**)

1. Run the job.
2. Go to “People” in the left navigation and confirm the accounts were added.

### Setting up the authorization strategy

Jenkins has a number of authorization strategies. Since the Ospreys and Sea Lions have different requirements, we are using the project level authorization strategy. We set up the admin and general user settings. We will also set up roles so the right users can see each project.

1. Create another Jenkins job.
2. In the Groovy console section, paste the code from <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/blob/master/jenkinsScripts/auth.groovy>
3. Run the job
4. Look at <http://localhost:8080/role-strategy/manage-roles>
5. Note that the ospreys now have team permissions.
6. Update the groovy code to set up the sea lions.

Their team project prefix is sea-lion.\*

1. Update the groovy code to add Sophia and Sam to this new team. (remember that saving must be the last statement)
2. Re run the job and confirm:
   1. The assigned roles table has a column for the sea-lions. The rows for Sam and Sophia should have one checkbox selected on each.
   2. The items role table has rows for Sam and Sophia. Their rows should have two checkboxes selected on each.

Note: Another option is to have Jenkins pull the project from GitHub and run the Groovy code as a script. However, this triggers the script security plugin and requires an extra step. Since this is a conference and not a system we are maintaining, we optimized. The effort to setup script security properly is more than the effort to copy/paste this config. My rant about the script security plugin is here: <https://www.selikoff.net/2018/09/02/a-rant-about-jenkins-script-security/>

## Step 5: Nexus System Scripting

### Learn what objects are available

Nexus exposes some variables to Groovy for ease in scripting:

* core
* repository
* blobStore
* security
* container

For this to be useful, you need to know the class name of each. This allows you to use JavaDoc (or autocomplete) to determine what methods are available.

Write your own Groovy task to output the classes in use here. (See the solution guide if you get stuck)

Note: Looking at the source code for these classes in the IDE, it is possible to find the interfaces as well. The technique of using a Groovy task to learn more about a class remains useful.

### Create global access and Jenkins access

Jenkins needs access to Nexus. While it can download open source jars without credentials, it does need to provide credentials to upload the artifacts from Osprey and Sea Lion projects. We don’t want to use the admin id for this! We also need to create an upload role and the repositories.

We used a combination of IDE autocomplete and examples online to create the Groovy script.

1. Copy the code from <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/blob/master/nexus-scripts/src/main/groovy/nexusBasicSetup.groovy>
2. Run it in a Groovy task in Nexus.
3. In the left navigation, click on Roles and see that there is now a jenkins-role.

### Setup Nexus Access

Now we are going to create content selectors, privileges and roles to configure access. The Sea Lion team wants their jars only available to their team. The Osprey team wants their jars available to all logged in users, but not anonymous users.

1. Copy the code from <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/blob/master/nexus-scripts/src/main/groovy/teamSetup.groovy>
2. Run it in a Groovy task in Nexus.
3. Now your turn. Try writing the Groovy code to set up the Sea Lion’s access. Remember that they get access to the Osprey code in addition to their own.

Note: You probably noticed it is a bit tedious to run the Nexus tasks. Luckily, Nexus provides REST APIs. However, much of this REST API is the Scripting API we have been using. So you both upload and run the script directly from your machine (or a build server).

## Step 6 – Create a Jenkins pipeline

## Setup Nexus Credentials in Jenkins

1. In the left navigation bar, click “Credentials”
2. Click “Jenkins”



1. Click “Global credentials (unrestricted)”
2. Click “Add Credentials”
3. Enter the username/password for Nexus (the one we set in the Groovy script). For id, make sure to name it “nexus”.

### Create a job to setup jobs with proper configuration

Teams at our imaginary company always start out with the same types of build – pull from github and run a gradle clean/install/publish. To save them some time, let’s create a meta-job that can generate Jenkins jobs with the build pipeline pre-configured.

1. Create a new job named “job-generator”
2. Select “freestyle project” as the type.
3. Select “this project is parameterized”
4. Add a string parameter with the name “gitUrl” and the description “URL in GitHub”
5. Add another string parameter with the name “gitFolder” and the description “subfolder name within GitHub project”
6. Add a build step to execute a system groovy script.
7. Paste in the script from <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/blob/master/jenkinsScripts/jobGenerator.groovy>
8. Save

### Testing Access and Running Jobs

Now that we have jobs and security, we can try everything out.

1. Logon as Sophia.
2. Confirm this user can see both the Osprey and Sea Lion job.
3. Run the Sea Lions job and ensure it passes.
4. Log out.
5. Logon as Olivia.
6. Confirm this user can see the Osprey job, but not the Sea Lion job.
7. Run the Osprey job and ensure it passes.

### Editing a pipeline job

The Sea Lions have decided they want to deploy to Nexus on every other build. For even numbered builds they want to just print a message to the console that they are skipping deploy. For odd numbered builds, they want the pipeline to behave normally.

Tip: This is the code to get the build number.

def buildNum = env['BUILD\_NUMBER'] as Integer

## Step 7 – Switch services

### Stop Nexus and Jenkins to free up some resources

Run these two commands to free up some CPU/RAM:

* docker stop jenkins
* docker stop nexus

Note: If you want to use Nexus and/or Jenkins again later, just run the command with “start” instead of “stop”

### Install SonarQube

At the command line, run:

**docker run -d -p 9000:9000 -p 9092:9092 --name sonarqube --network hol-network sonarqube:7.1**

Note: this uses the in memory database. For a real install, you’ll want to use an external database.

Validation:

In a browser, go to http://localhost:9000. It make take about a minute before the URL returns a webpage.

### Step 8 – Creating a custom SonarQube rule

The company is concerned about the Sea Lions committing trade secrets. In true Dilbert fashion, they decided the solution was to ban the word “secret” from method and variable names.

Conveniently, we had a developer (Jeanne) handy to write the rule. You can browse the source code at <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/tree/master/sonar-custom-rule>

(SecretNotAllowedRule.java is the most interesting class)

### Deploying a custom rule

1. Copy the snapshot jar of this rule to a directory on your machine. You can use Maven to build sonar-custom-rule or use the pre-built version from <https://github.com/boyarsky/OracleCodeOne2018-HOL-Automating-Stack-Groovy/blob/master/sonar-custom-rule-0.0.1-SNAPSHOT.jar>
2. cd to that directory
3. docker cp sonar-custom-rule-0.0.1-SNAPSHOT.jar sonarqube:/opt/sonarqube/extensions/plugins/sonar-custom-rule-0.0.1-SNAPSHOT.jar
4. docker restart sonarqube

### Add this rule to the default quality profile

1. Go to <http://localhost:9000>
2. Login (admin/admin)
3. Click “Quality Profiles” in the top navigation
4. Click “Create” on the upper right
5. Enter a name of your choosing and select “Java” from the pulldown
6. Click “Create”
7. Note your quality profile has zero rules
8. Click “Change parent” and choose the “Sonar way”
9. Note how many rules are in the quality profile now. (It was 299 when we tested)
10. Click “Activate More”
11. Search for “secret”



1. Click “activate”
2. Click “Quality Profiles” in the top navigation and observe one more rule is active.

### Run a scan with this custom rule

1. Download the CLI for your operating system from <https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>
2. In the conf directory of the downloaded file, uncomment

*sonar.host.url=http://localhost:9000*

1. cd into the directory of sea-lion-project
2. Run

*<absolute path of>/sonar-scanner*

1. Go back into Sonar in the browser and search for “sea lion”
2. Go to the project and note that now there is one vulnerability.



1. Click it to see the details. It’s our ‘secret’ rule!



### Step 9 – Extra credit in Jenkins

We have a security hole in the design here. Anyone can create any job they want. While solving this problem is out of scope for the lab, let’s deal with one piece of it. The task is to write Groovy code to ensure that only jobs with the prefix “osprey” build the osprey-project and only jobs with the prefix “sea-lion” build the sea-lion-project.

This one is extra credit so it comes with less instructions.

1. Stop the Sonar container
2. Restart the Jenkins container
3. In the Jenkins Groovy console, write a script to check this.
4. Create a Jenkins FreeStyle job that includes this a Groovy System build step with this code.
5. Add a build trigger to this job runs every 5 minutes.
6. Try renaming the sea-lion project to “osprey-invalid-permissions” and see if it fails.

While this doesn’t solve the actual security problem, it at least tells someone about it. (or it would, if we had build notifications set up). So you’ve made a toy example on your machine a bit safer ☺.

## Final step: Cleanup

### Stop containers

Remember you can enter the first couple characters of the hash rather than the whole thing.

**docker ps**

**docker stop nexus**

**docker stop jenkins**

**docker stop sonarqube**

### If want to reclaim disk space

Remove containers and images

**docker ps –a**

**docker rm sonarqube**

**docker rm jenkins**

**docker rm nexus**

**docker images**

**docker rmi <hash1>**

**docker rmi <hash2>**

**docker rmi <hash3>**

Delete configuration

**docker network rm hol-network**

## Uh oh. I managed to make it so the admin user doesn’t have read access to Jenkins

If you manage to change permissions so admin no longer has admin rights, you have two choices on how to fix it

### Faster way – install your favorite text editor

1. Launch bash as root

**docker exec -it --user root jenkins bash**

1. Install your favorite text editor. We choose vim

**apt-get update**

**apt-get install –y vim**

1. Open the config.xml

**vim /var/jenkins\_home/config.xml**

1. Update the authorization strategy and security realm to the following:

<authorizationStrategy class="hudson.security.FullControlOnceLoggedInAuthorizationStrategy">

<denyAnonymousReadAccess>true</denyAnonymousReadAccess>

</authorizationStrategy>

<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">

<disableSignup>true</disableSignup>

<enableCaptcha>false</enableCaptcha>

</securityRealm>

1. Exit your text editor
2. Exit bash
3. Restart Jenkins

**docker restart jenkins**

### Slower way – delete your docker container/image and start over

1. Follow the clean up section of this document
2. Follow the setup again

## Useful References

<https://hub.docker.com/r/jenkins/jenkins>

<https://github.com/jenkinsci/docker/blob/master/README.md>

<https://hub.docker.com/r/sonatype/nexus3>

<https://hub.docker.com/r/library/sonarqube/>

<https://support.sonatype.com/hc/en-us/articles/115015812727-Nexus-3-Groovy-Script-development-environment-setup>

<https://help.sonatype.com/repomanager3/rest-and-integration-api/script-api/writing-scripts#app>

<https://help.sonatype.com/repomanager3/rest-and-integration-api>

<https://github.com/SonarSource/sonar-custom-rules-examples/tree/master/java-custom-rules>

<https://docs.sonarqube.org/display/PLUG/Writing+Custom+Java+Rules+101>

<http://groovy-lang.org/style-guide.html>