**Some useful Jenkins Groovy scripts**

**21 Sep 2014**

Recently I needed to automate the deployment and configuration of a [Jenkins](http://jenkins-ci.org/) server. My configuration management tool of choice is [Chef](https://www.getchef.com/) and my starting point is the [Opscode Jenkins cookbook](https://github.com/opscode-cookbooks/jenkins). At this time the Jenkins cookbook only provides resources for basic Jenkins configuration, eg. jenkins\_user, however it also exposes the jenkins\_script resource for running arbitrary [Groovy](http://groovy.codehaus.org/) scripts on the server.

Using Groovy it's possible to configure pretty much every part of a Jenkins server and even its plugins. However, finding documentation on how is not so trivial - the best place to start is the [Jenkins Javadocs](http://javadoc.jenkins-ci.org/).

All of the following scripts were run with the jenkins\_script resource as follows

jenkins\_script 'resource name' do

command <<-EOH.gsub(/^ {4}/, '')

[Groovy Script]

EOH

end

It should also be noted that although it might be possible to write idempotent Groovy scripts the following are not exactly idempotent (although they can mostly be run as many times as you like without messing up the configuration)

**Setting permissions**

The first thing I needed to do was configure user permisssions so that only a user called admin could access anything. Note that the following script assumes that an admin user has been added already. Also in the context of a Chef run, immediately after running this, Chef will no longer be able to run scripts on the server unless it uses a private key that has been associated with the admin user (the solution to this problem is given in the [appendix](http://pghalliday.com/jenkins/groovy/sonar/chef/configuration/management/2014/09/21/some-useful-jenkins-groovy-scripts.html#appendix)).

import jenkins.model.\*

import hudson.security.\*

def instance = Jenkins.getInstance()

def hudsonRealm = new HudsonPrivateSecurityRealm(false)

instance.setSecurityRealm(hudsonRealm)

def strategy = new GlobalMatrixAuthorizationStrategy()

strategy.add(Jenkins.ADMINISTER, "admin")

instance.setAuthorizationStrategy(strategy)

instance.save()

**Set the slave agent port**

By default the jenkins slave agent port is randomized. However in my case I needed to configure my cluster using a firewall on the Jenkins master and thus wanted to open a single port to use for build slaves to communicate with the master using JNLP.

import jenkins.model.\*

def instance = Jenkins.getInstance()

instance.setSlaveAgentPort([the fixed port number])

instance.save()

**Set the administrator email address**

This is the System Admin e-mail address set in the Jenkins configuration

import jenkins.model.\*

def jenkinsLocationConfiguration = JenkinsLocationConfiguration.get()

jenkinsLocationConfiguration.setAdminAddress("[your admin name] <[your admin email address]>")

jenkinsLocationConfiguration.save()

**Set the mail server configuration**

I need the Jenkins server to mail notifications so I need to configure an SMTP server for it to use

import jenkins.model.\*

def inst = Jenkins.getInstance()

def desc = inst.getDescriptor("hudson.tasks.Mailer")

desc.setSmtpAuth("[SMTP user]", "[SMTP password]")

desc.setReplyToAddress("[reply to email address]")

desc.setSmtpHost("[SMTP host]")

desc.setUseSsl([true or false to use SLL])

desc.setSmtpPort("[SMTP port]")

desc.setCharset("[character set]")

desc.save()

**Set the Git plugin configuration**

The git client used by Jenkins should have a user name and email set

import jenkins.model.\*

def inst = Jenkins.getInstance()

def desc = inst.getDescriptor("hudson.plugins.git.GitSCM")

desc.setGlobalConfigName("[name to use with git commits]")

desc.setGlobalConfigEmail("[email to use with git commits]")

desc.save()

**Sonar plugin configuration**

We use Sonar to run static analysis on code and record code coverage, etc. As such I needed to configure both a Sonar installation and a default Sonar runner for Jenkins to use. Javadocs for the Sonar plugin were not so easy to track down and when I did they were not so accurate so some of this was arrived at through trial and error. The best reference however, is the [Sonar plugin source](https://github.com/SonarSource/jenkins-sonar-plugin) itself.

**Add the Sonar installation**

import jenkins.model.\*

import hudson.plugins.sonar.\*

import hudson.plugins.sonar.model.\*

def inst = Jenkins.getInstance()

def desc = inst.getDescriptor("hudson.plugins.sonar.SonarPublisher")

def sinst = new SonarInstallation(

"[name of the sonar installation - I use the host name]",

[true or false to disable the sonar installation],

"[sonar server url]",

"[sonar database url]",

"[sonar database driver]",

"[sonar database user]",

"[sonar database password]",

"[version of sonar maven plugin - I don't use maven so leave this blank]",

"[additional properties to pass to maven - again I leave this blank]",

new TriggersConfig(),

"[sonar user]",

"[sonar password]"

)

desc.setInstallations(sinst)

desc.save()

**Add the Sonar runner**

This adds a runner that will be installed automatically from Maven central

import jenkins.model.\*

import hudson.plugins.sonar.\*

import hudson.tools.\*

def inst = Jenkins.getInstance()

def desc = inst.getDescriptor("hudson.plugins.sonar.SonarRunnerInstallation")

def installer = new SonarRunnerInstaller("[sonar runner version]")

def prop = new InstallSourceProperty([installer])

def sinst = new SonarRunnerInstallation("[name of the sonar runner - I called it Default]", "[home? - not sure how this is used and I left it blank]", [prop])

desc.setInstallations(sinst)

desc.save()

**Set the number of executors**

As I was building a Jenkins cluster I wanted all my builds to run on slaves and as such the master should have 0 executors. This was the only problem script as it requires a Jenkins restart to apply. I didn't want Jenkins to restart everytime the chef client ran (there's that idempotence problem) so had to wrap this with a flag to ensure it only ran on the first Chef run.

import jenkins.model.\*

def instance = Jenkins.getInstance()

instance.setNumExecutors(0)

instance.save()

and wrapped as follows

jenkins\_script 'master should have 0 executors' do

command <<-EOH.gsub(/^ {4}/, '')

import jenkins.model.\*

def instance = Jenkins.getInstance()

instance.setNumExecutors(0)

instance.save()

EOH

notifies :create, 'ruby\_block[set the executors\_set flag]', :immediately

notifies :restart, 'service[jenkins]', :delayed

not\_if { node.attribute?('executors\_set') }

end

ruby\_block 'set the executors\_set flag' do

block do

node.set['executors\_set'] = true

node.save

end

action :nothing

end

**Appendix**

Here's how to add a Jenkins user with Chef and the Jenkins cookbook such that once the user and key has been added, Chef then uses that key for future communication with the server.

# If security was enabled in a previous chef run then set the private key in the run\_state

# now as required by the Jenkins cookbook

ruby\_block 'set jenkins private key' do

block do

node.run\_state[:jenkins\_private\_key] = '[your private key]'

end

only\_if { node.attribute?('security\_enabled') }

end

# Add the admin user only if it has not been added already then notify the resource

# to configure the permissions for the admin user

jenkins\_user 'admin' do

password '[your admin password]'

public\_keys ['[your public key]']

not\_if { node.attribute?('security\_enabled') }

notifies :execute, 'jenkins\_script[configure permissions]', :immediately

end

# Configure the permissions so that login is required and the admin user is an administrator

# after this point the private key will be required to execute jenkins scripts (including querying

# if users exist) so we notify the `set the security\_enabled flag` resource to set this up.

# Also note that since Jenkins 1.556 the private key cannot be used until after the admin user

# has been added to the security realm

jenkins\_script 'configure permissions' do

command <<-EOH.gsub(/^ {4}/, '')

import jenkins.model.\*

import hudson.security.\*

def instance = Jenkins.getInstance()

def hudsonRealm = new HudsonPrivateSecurityRealm(false)

instance.setSecurityRealm(hudsonRealm)

def strategy = new GlobalMatrixAuthorizationStrategy()

strategy.add(Jenkins.ADMINISTER, "admin")

instance.setAuthorizationStrategy(strategy)

instance.save()

EOH

notifies :create, 'ruby\_block[set the security\_enabled flag]', :immediately

action :nothing

end

# Set the security enabled flag and set the run\_state to use the configured private key

ruby\_block 'set the security\_enabled flag' do

block do

node.run\_state[:jenkins\_private\_key] = '[your private key]'

node.set['security\_enabled'] = true

node.save

end

action :nothing

end

I successfully integrate testopia with jenkins with groovy script but failed to write a script to integrate tesBugzilla with jenkins. Could you check and help me to sort out the issue.

import jenkins.\*  
import jenkins.model.\*  
import hudson.\*  
import hudson.model.\*  
import hudson.plugins.bugzilla.\*  
import hudson.plugins.bugzilla.BugzillaSession;

def inst = Jenkins.getInstance()

def desc = inst.getDescriptor("hudson.plugins.bugzilla.BugzillaProjectProperty")

def sinst = new BugzillaSession(  
"http://10.10.100.100/bugzilla", "ankusha@abc.com","ankush123"  
)

desc.installations(sinst, "\b[0-9.]\*[0-9]\b" , true)

desc.save()

I added the following function (from [JavaWorld](http://www.javaworld.com/article/2073679/detecting-class-innards-in-groovy.html)) to inspect the desc object but see no way to set the properties using the API. The configure method sets things from a StaplerRequest but not sure if it's easy to fake that

def doInspector(obj) {

def inspector = new groovy.inspect.Inspector(obj)

def inspectorReport = new StringBuilder()

inspectorReport << "Object under inspection "

inspectorReport << (inspector.isGroovy() ? "IS" : "is NOT") << " Groovy!\n"

inspectorReport << "METHODS\n"

def methods = inspector.methods

methods.each {

inspectorReport << "\t" << it.toString() << "\n"

}

inspectorReport << "\nMETA METHODS\n"

def metaMethods = inspector.metaMethods

metaMethods.each {

inspectorReport << "\t" << it.toString() << "\n"

}

inspectorReport << "\nPROPERTY INFO\n"

def properties = inspector.propertyInfo

properties.each {

inspectorReport << "\t" << it.toString() << "\n"

}

println inspectorReport

}