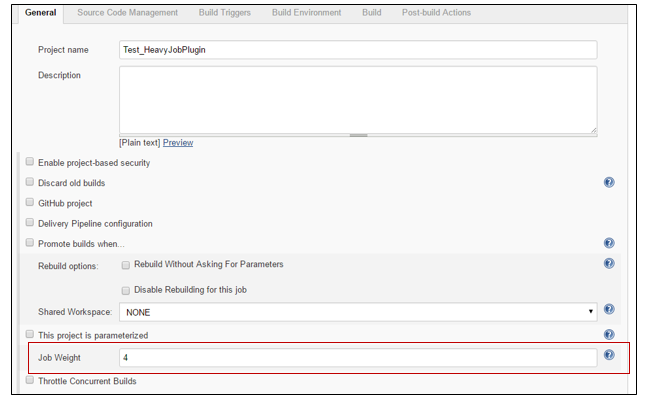
**Jenkin Plugins:**

1. **Heavy Job Plugin :**

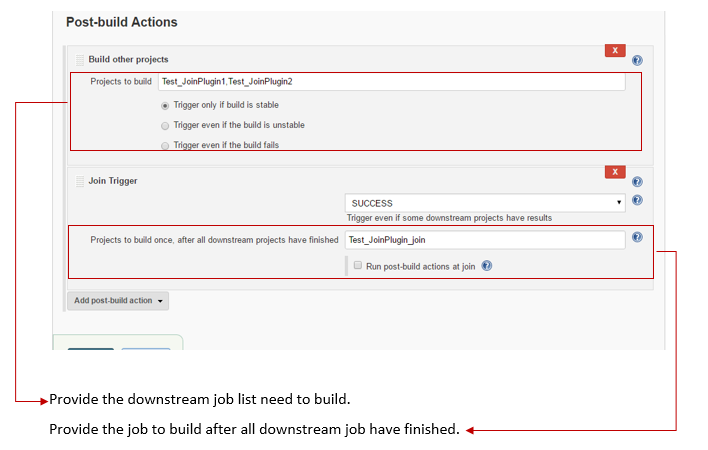
This plugin allows you to define "weight" on each job, and making each job consume that many executors (instead of just one.)

Need to specify the number of executors that this job needs to occupy. Setting this to a bigger value is useful if your job is known to consume multiple threads, so that the scheduling decision is done accordingly.



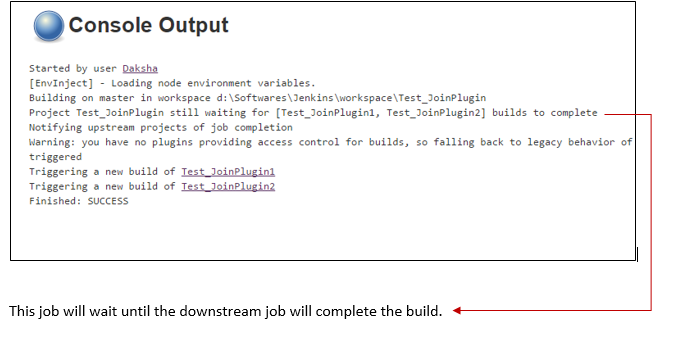
1. **Join Plugin**

This plugin allows a job to be run after all the immediate downstream jobs have completed. In this way, the execution can branch out and perform many steps in parallel, and then run a final aggregation step just once after all the parallel work is finished.   
The plugin is useful for creating a 'diamond' shape project dependency.   This means there is a single parent job that starts several downstream jobs.   Once those jobs are finished, a single aggregation job runs.   More complex interactions are not possible with this plugin.



E.g.

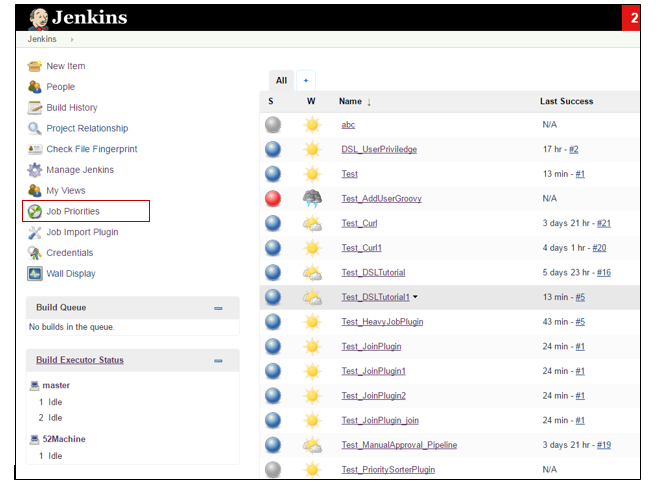
Test\_JoinPlugin is main job in which the downstream job are configured.



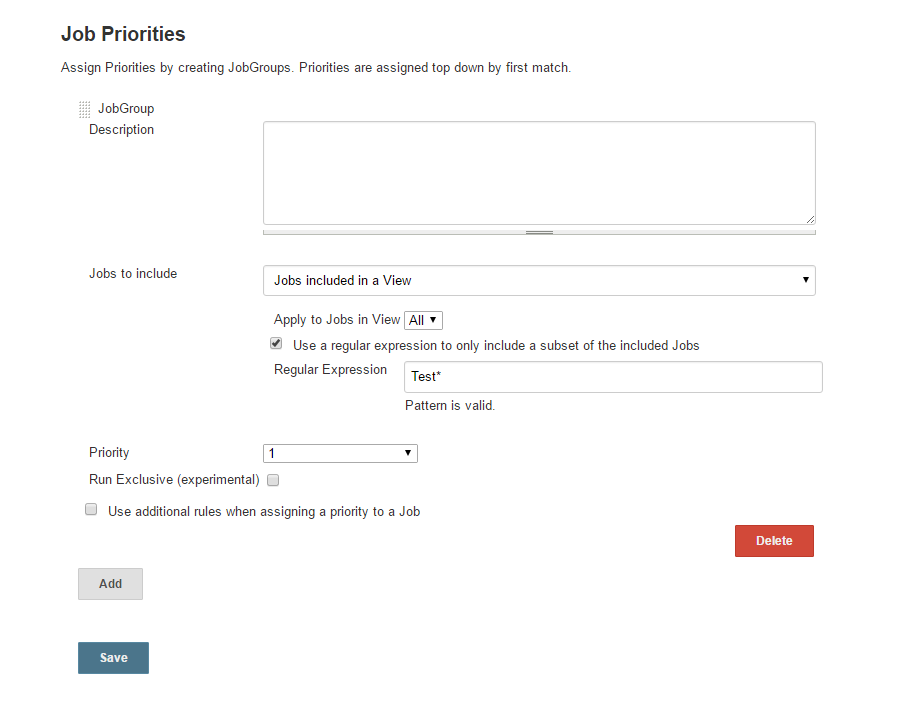
1. **Priority Sorter Plugin**

This plugin adds the ability to assign different priorities to Jobs, the lower priority the job has the sooner the Job will run.

This can be very helpful when one wants to add low priority jobs but wants to have higher-priority jobs run first when hardware is limited or when there are different groups of Jobs that should share resources (equally).

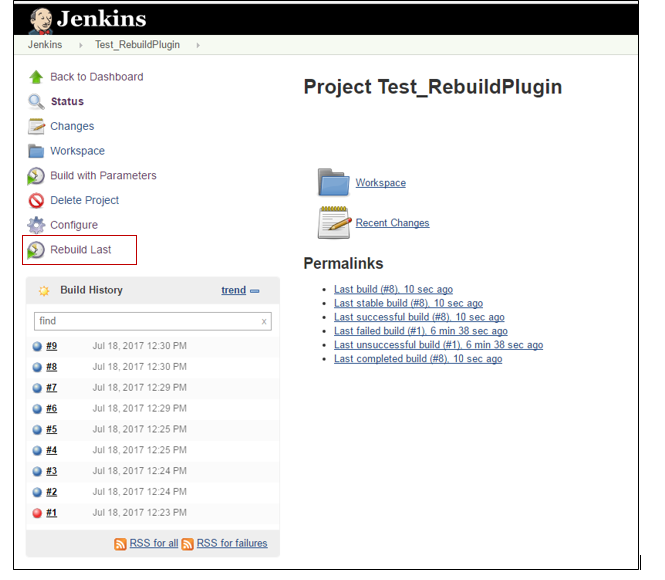


Click on Job Priorities link on main Jenkins page, configure job Priorities.

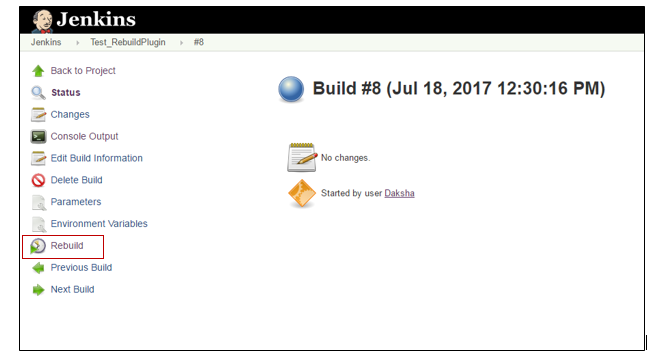


1. **Rebuild Plugin**

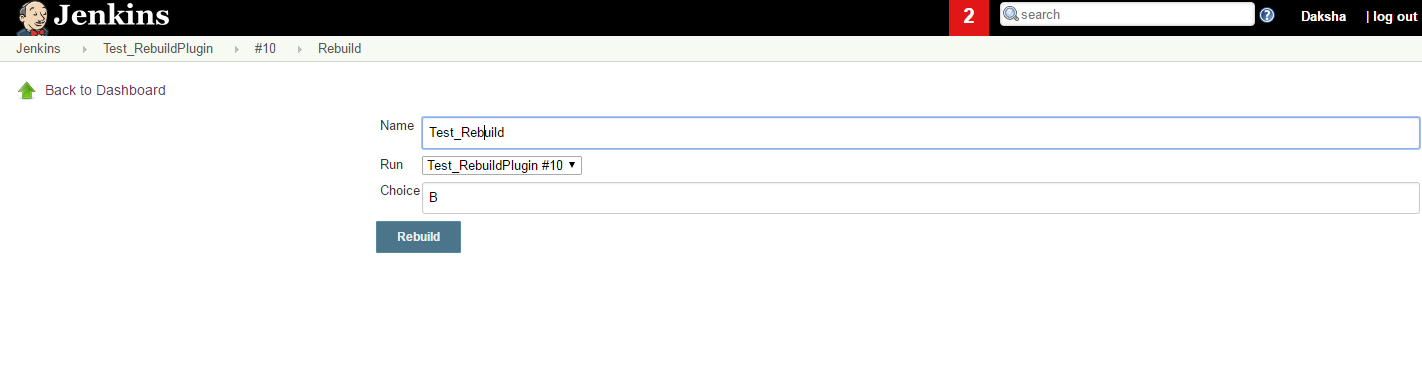
This plugin will rebuild the job.



Once you installed this plug-in whenever you execute a parametrized build you will get a rebuild button in the left pane of the run as shown in the image.



If you want to execute the build with same parameters, rebuild plug-in will allow you to build it through rebuild button. If you want to edit some of the existing parameters, the rebuild plug-in will allow you to edit the existing parameters and build it.



1. [**Shared workspace plugin**](https://wiki.jenkins.io/display/JENKINS/Shared+workspace+plugin)

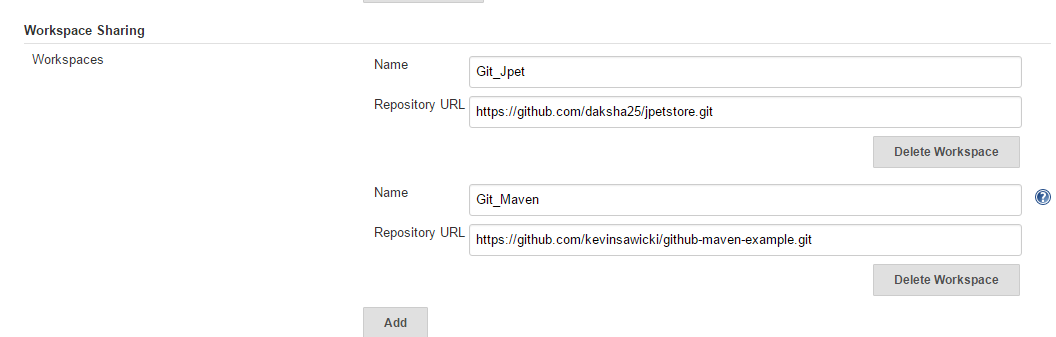
This plugin allows to share workspaces by Jenkins jobs with the same SCM repos.

It should save you some disk space, if you have different jobs with identical repos.

How to use it:

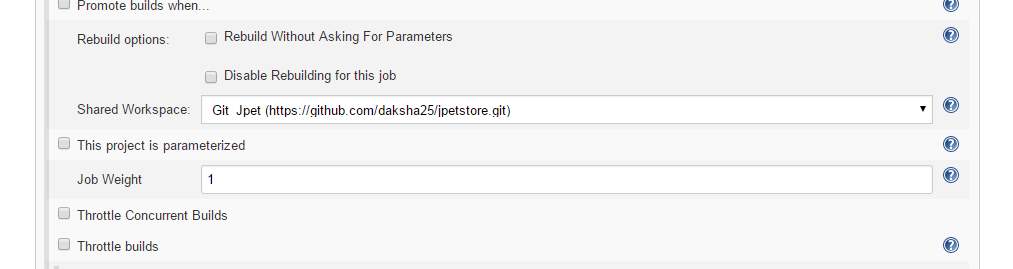
1) In "**Manage Jenkins**"->"**Configure System**" find new "**Workspace Sharing**" block.

2) Add some **Name + SCM Repo URL** pairs. Save.

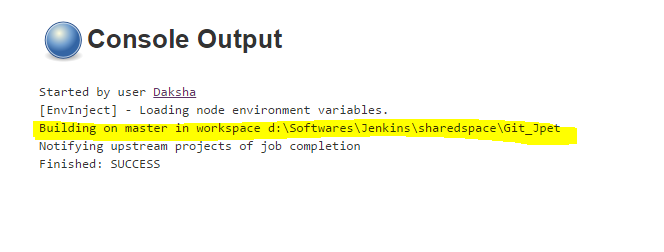


3) Go to the job configuration page, now you able to select "**Shared Workspace**" here.

4) Use **${SHAREDSPACE\_SCM\_URL}** variable in your SCM url field.

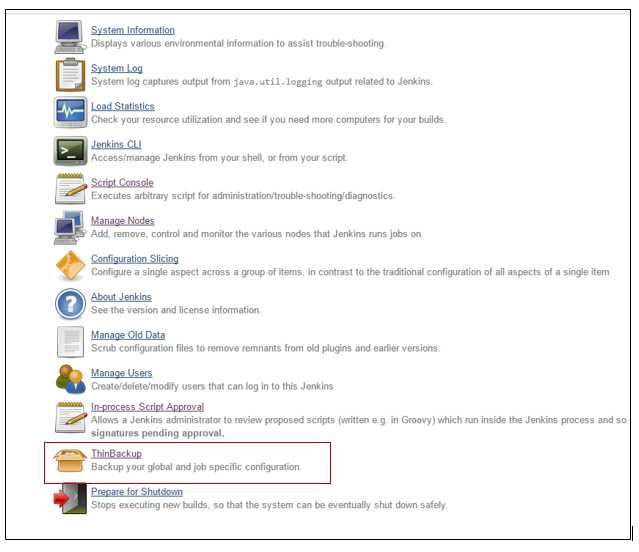


Workspaces will be created as **{node remote FS root}/sharedspace/{workspace name}**

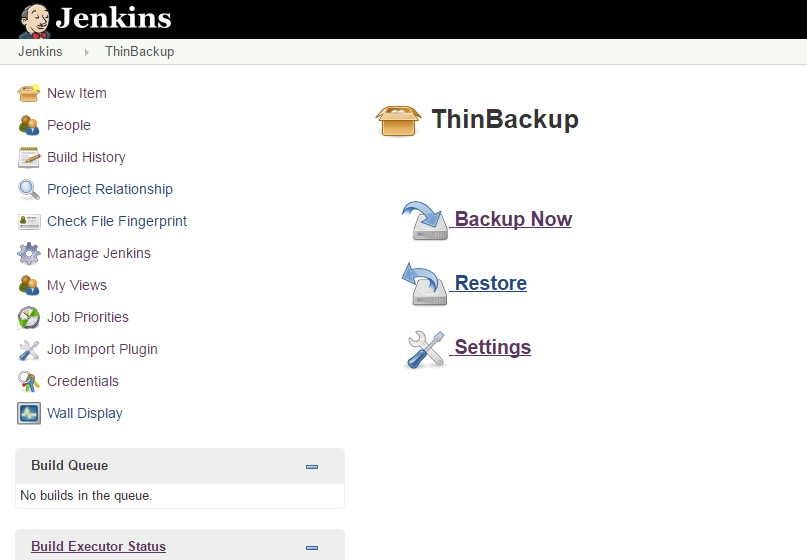


1. **Thin Backup**

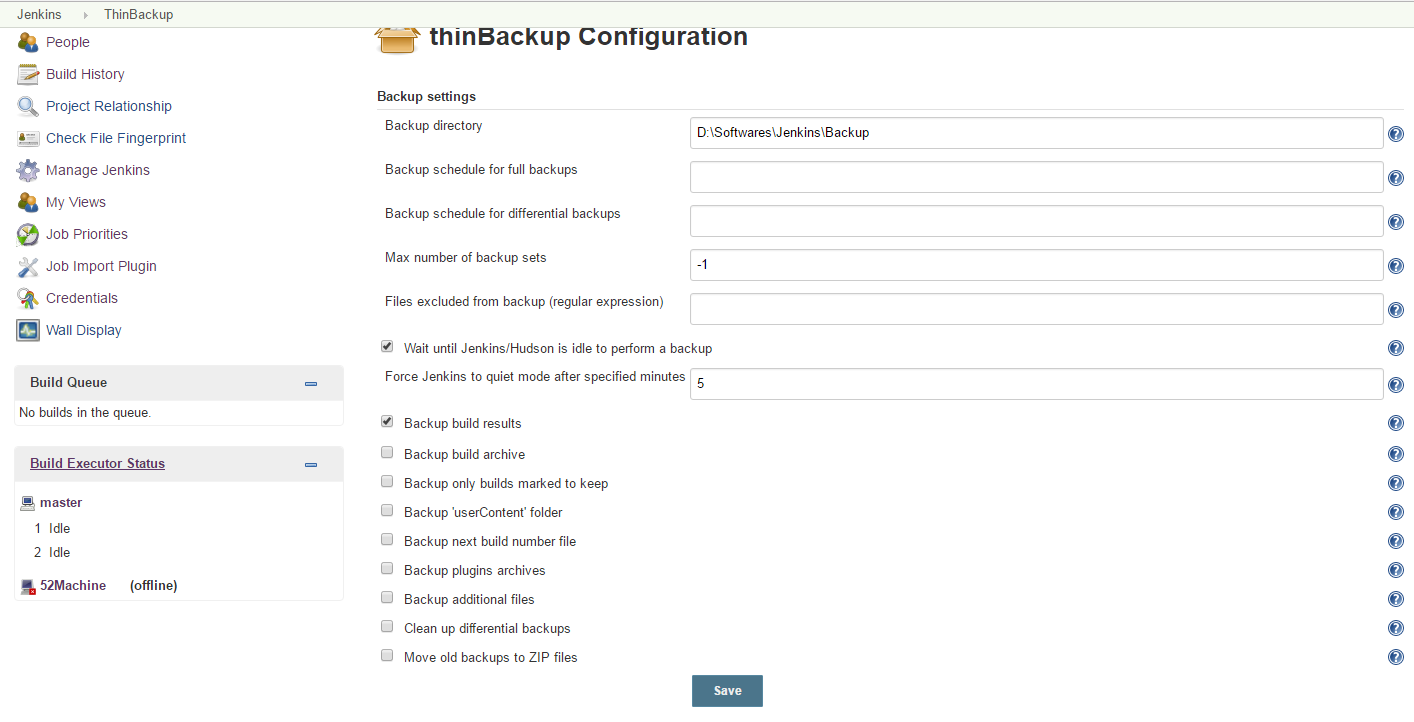
The Backup Plugin only does manual backups, and stores all data found in JENKINS\_HOME. thinBackups can be scheduled and only backs up the most vital configuration info.



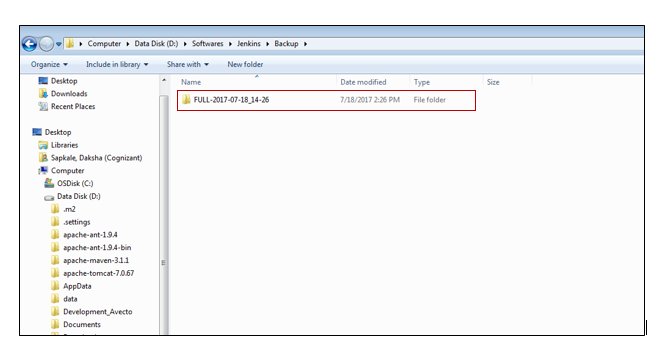
This plugin Provide Backup of Jenkins as well as restore.



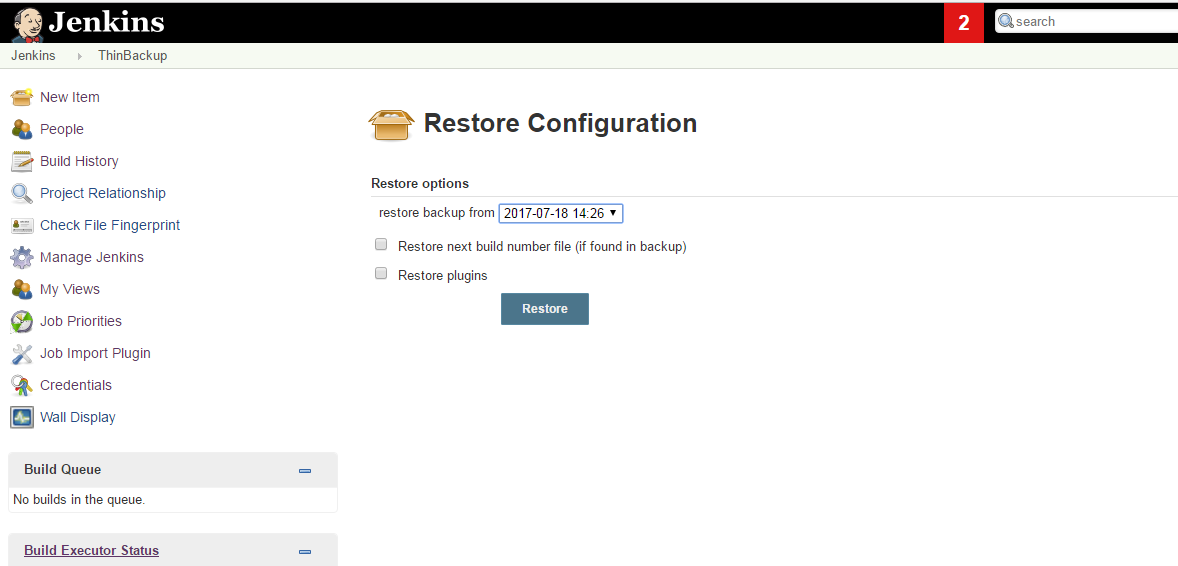
Configure the back\_up just by clicking on setting link -



By clicking on backup link it will take backup and stored it to specified location -

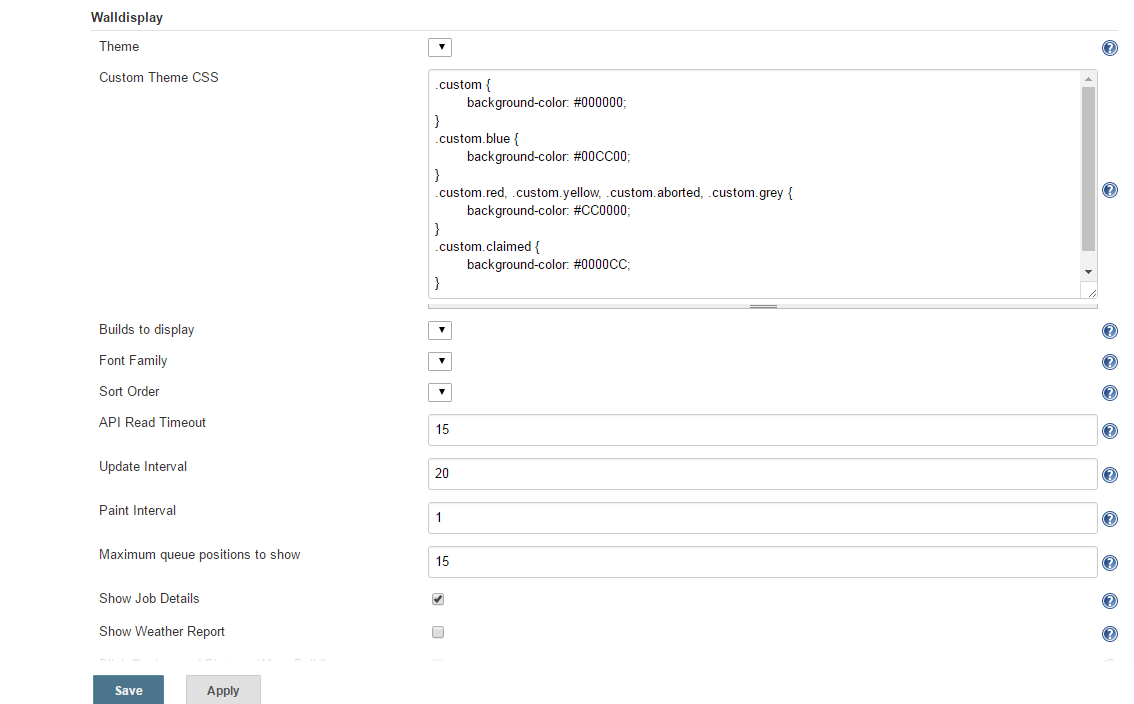


To configure restore click on restore link –

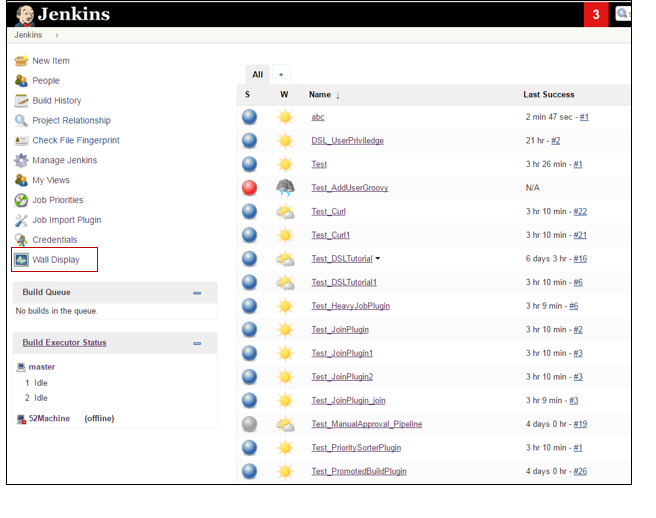


1. **Wall Display Plugin**

To do the Configuration go to Manage Jenkin 🡪 configure system



Simply click on the 'Wall Display' link in the sidebar of your project



# Configuration Slicing Plugin

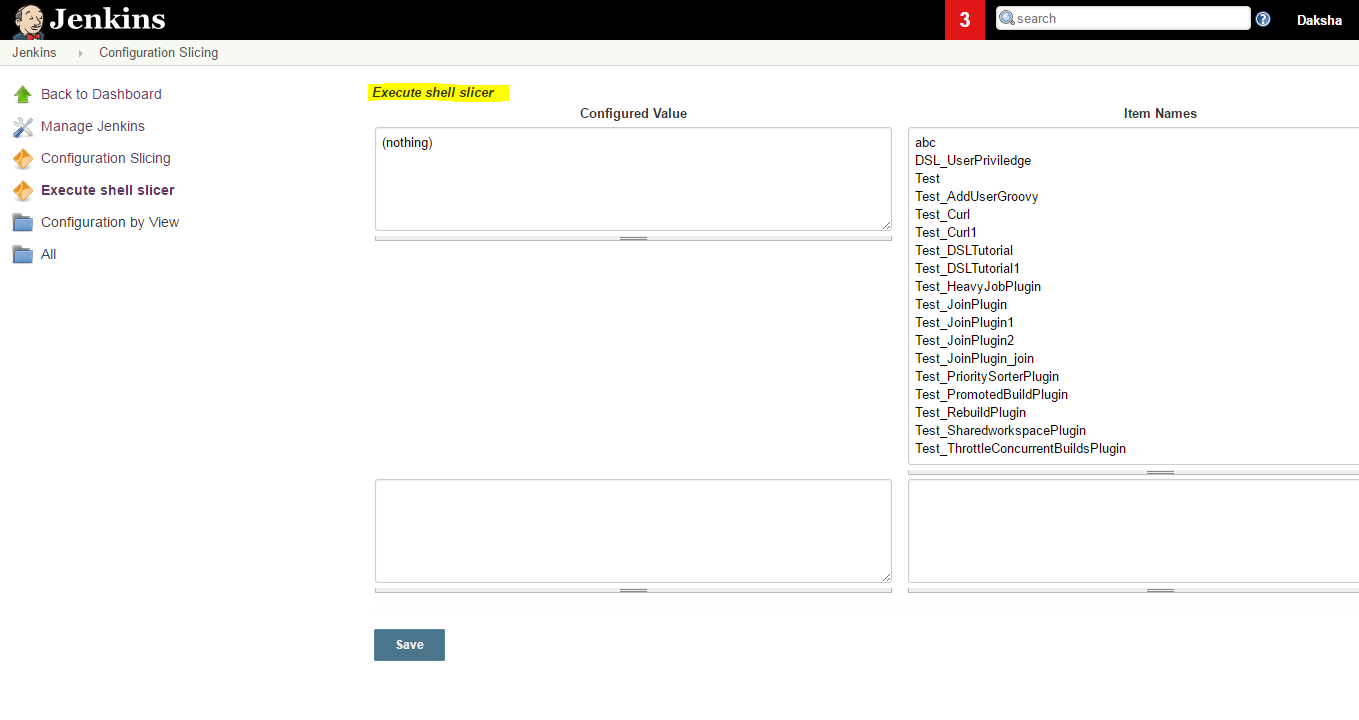
The main page of the configuration slicing plugin shows all the properties that can be sliced - select one and you are presented with a screen showing how that value is set across the entire Jenkins instance. Many properties on Jenkins projects are useful to set this way, but the configuration slicing plugin can handle properties on any collection, such as slaves, or builds of a project.

# 

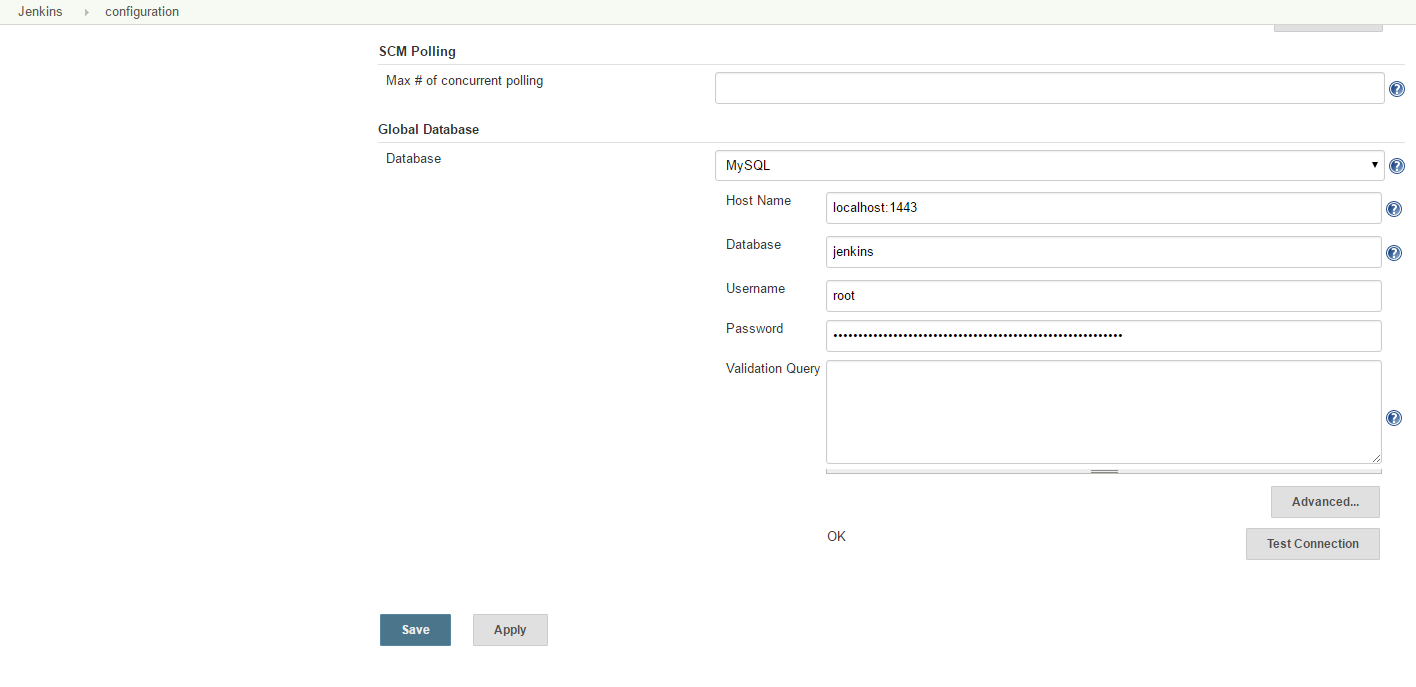
The following functions are supported -

# 

Just click on any function which need to configure in all jobs, and configure it :



Mysql Database plugin



# Monitor and Restart Offline Slaves

Tested the groovy script it will restart the offline slave, if it is not disconnected manually.

Groovy Script :

import hudson.model.\*

import hudson.node\_monitors.\*

import hudson.slaves.\*

import java.util.concurrent.\*

jenkins = Hudson.instance

def getEnviron(computer) {

def env

def thread = Thread.start("Getting env from ${computer.name}", { env = computer.environment })

thread.join(2000)

if (thread.isAlive()) thread.interrupt()

env

}

def slaveAccessible(computer) {

getEnviron(computer)?.get('PATH') != null

}

def numberOfflineNodes = 0

def numberNodes = 0

for (slave in jenkins.slaves) {

def computer = slave.computer

numberNodes ++

println ""

println "Checking computer ${computer.name}:"

def isOK = (slaveAccessible(computer) && !computer.offline)

if (isOK) {

println "\t\tOK, got PATH back from slave ${computer.name}."

println('\tcomputer.isOffline: ' + slave.getComputer().isOffline());

println('\tcomputer.isTemporarilyOffline: ' + slave.getComputer().isTemporarilyOffline());

println('\tcomputer.getOfflineCause: ' + slave.getComputer().getOfflineCause());

println('\tcomputer.offline: ' + computer.offline);

} else {

numberOfflineNodes ++

println " ERROR: can't get PATH from slave ${computer.name}."

println('\tcomputer.isOffline: ' + slave.getComputer().isOffline());

println('\tcomputer.isTemporarilyOffline: ' + slave.getComputer().isTemporarilyOffline());

println('\tcomputer.getOfflineCause: ' + slave.getComputer().getOfflineCause());

println('\tcomputer.offline: ' + computer.offline);

//sendMail(computer.name, slave.getComputer().getOfflineCause().toString())

if (slave.getComputer().isTemporarilyOffline()) {

if (!slave.getComputer().getOfflineCause().toString().contains("Disconnected by")) {

computer.setTemporarilyOffline(false, slave.getComputer().getOfflineCause())

}

} else {

computer.connect(true)

}

}}

With mail part :

import hudson.model.\*

import hudson.node\_monitors.\*

import hudson.slaves.\*

import java.util.concurrent.\*

jenkins = Hudson.instance

import javax.mail.internet.\*;

import javax.mail.\*

import javax.activation.\*

def sendMail (slave, cause) {

message = slave + " slave is down. Check http://10.219.193.108:8080/computer/" + slave + "\nBecause " + cause

subject = slave + " slave is offline"

toAddress = "daksha.sapkale@cognizant.com"

fromAddress = "sneha.gajare@cognizant.com"

host = " 173.194.219.27"

port = "25"

Properties mprops = new Properties();

mprops.setProperty("mail.transport.protocol","smtp");

mprops.setProperty("mail.host",host);

mprops.setProperty("mail.smtp.port",port);

Session lSession = Session.getDefaultInstance(mprops,null);

MimeMessage msg = new MimeMessage(lSession);

//tokenize out the recipients in case they came in as a list

StringTokenizer tok = new StringTokenizer(toAddress,";");

ArrayList emailTos = new ArrayList();

while(tok.hasMoreElements()){

emailTos.add(new InternetAddress(tok.nextElement().toString()));

}

InternetAddress[] to = new InternetAddress[emailTos.size()];

to = (InternetAddress[]) emailTos.toArray(to);

msg.setRecipients(MimeMessage.RecipientType.TO,to);

InternetAddress fromAddr = new InternetAddress(fromAddress);

msg.setFrom(fromAddr);

msg.setFrom(new InternetAddress(fromAddress));

msg.setSubject(subject);

msg.setText(message)

Transport transporter = lSession.getTransport("smtp");

transporter.connect();

transporter.send(msg);

}

def getEnviron(computer) {

def env

def thread = Thread.start("Getting env from ${computer.name}", { env = computer.environment })

thread.join(2000)

if (thread.isAlive()) thread.interrupt()

env

}

def slaveAccessible(computer) {

getEnviron(computer)?.get('PATH') != null

}

def numberOfflineNodes = 0

def numberNodes = 0

for (slave in jenkins.slaves) {

def computer = slave.computer

numberNodes ++

println ""

println "Checking computer ${computer.name}:"

def isOK = (slaveAccessible(computer) && !computer.offline)

if (isOK) {

println "\t\tOK, got PATH back from slave ${computer.name}."

println('\tcomputer.isOffline: ' + slave.getComputer().isOffline());

println('\tcomputer.isTemporarilyOffline: ' + slave.getComputer().isTemporarilyOffline());

println('\tcomputer.getOfflineCause: ' + slave.getComputer().getOfflineCause());

println('\tcomputer.offline: ' + computer.offline);

} else {

numberOfflineNodes ++

println " ERROR: can't get PATH from slave ${computer.name}."

println('\tcomputer.isOffline: ' + slave.getComputer().isOffline());

println('\tcomputer.isTemporarilyOffline: ' + slave.getComputer().isTemporarilyOffline());

println('\tcomputer.getOfflineCause: ' + slave.getComputer().getOfflineCause());

println('\tcomputer.offline: ' + computer.offline);

sendMail(computer.name, slave.getComputer().getOfflineCause().toString())

if (slave.getComputer().isTemporarilyOffline()) {

if (!slave.getComputer().getOfflineCause().toString().contains("Disconnected by")) {

computer.setTemporarilyOffline(false, slave.getComputer().getOfflineCause())

}

} else {

computer.connect(true)

}

}

}

println ("Number of Offline Nodes: " + numberOfflineNodes)

println ("Number of Nodes: " + numberNodes)