**MODULAR DEPLOYMENT**

**-RapidDeploy plugin**

\*RapidDeploy project deploy: invoke a deployment task in RapidDeploy server as either a Build step or Post build step. Multiple deployments can be executed sequentially or parallel. You can check the result and the process logs in the Jenkins build output.

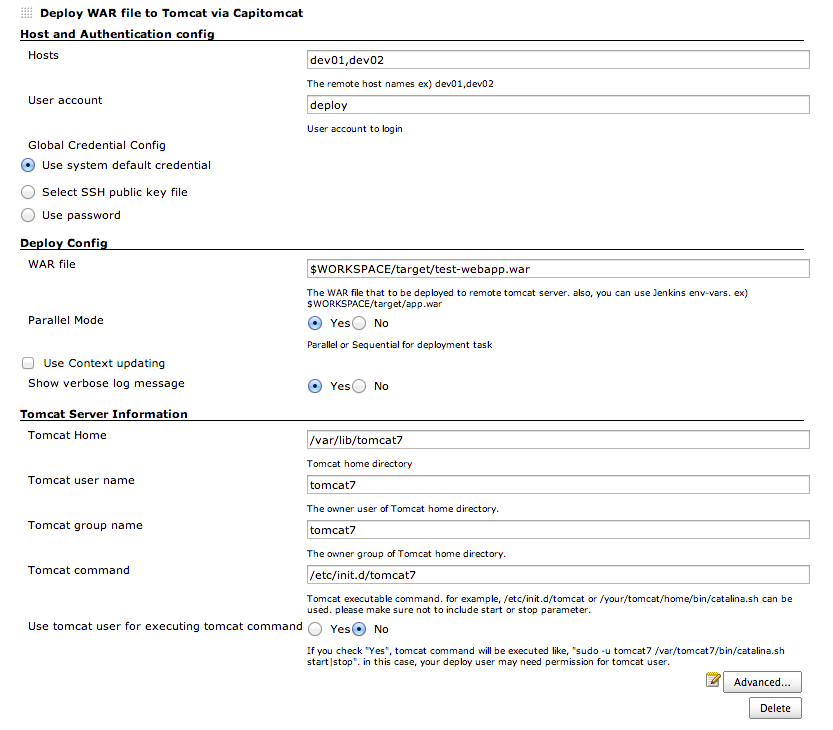
**-RapidDeploy**

\*RapidDeploy is a powerful application release and deployment automation tool. Built for DevOps teams and enterprise continuous delivery, RapidDeploy delivers consistent high volume deployments so your teams can focus on developing. Release and deploy applications at pace without error or risk.

\* For more information you can visit http://www.midvision.com/product

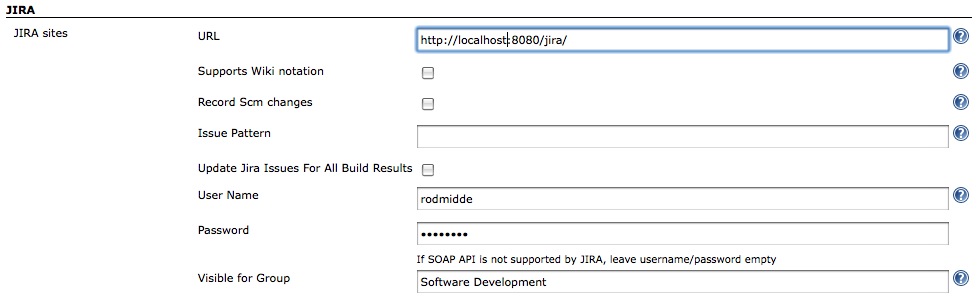
**-Capitomcat Plugin**

\*Capitomcat plugin deploy the WAR file to multiple remote Tomcat servers using Capitomcat which is Capistrano 3 recipe and library for Tomcat deployment.



# -[JIRA Plugin](https://wiki.jenkins-ci.org/display/JENKINS/JIRA+Plugin)

* This plugin has an optional feature to update JIRA issues with a back pointer to Jenkins build pages. This allows the submitter and watchers to quickly find out which build they need to pick up to get the fix.



# -[Git Parameter Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Git+Parameter+Plugin)

* This plugin allows you to assign git tag or revision number as parameter in Parametrized builds.  
  There is no need to set up anything special, this plugin will read your default configuration from Git Plugin

# -[Deployment Sphere Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Deployment+Sphere+Plugin)

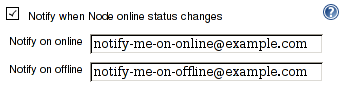
# Features:

* Collects application versions
* Collects application versions deployments
* Configures environments
* Configures applications
* Provides dashboard of deployed applications versions across environments

# -Mail watcher plugin

This plugin notifies configured email recipients on various events.

https://wiki.jenkins-ci.org/download/attachments/63144688/mwp-job.png?version=1&modificationDate=1375268476000



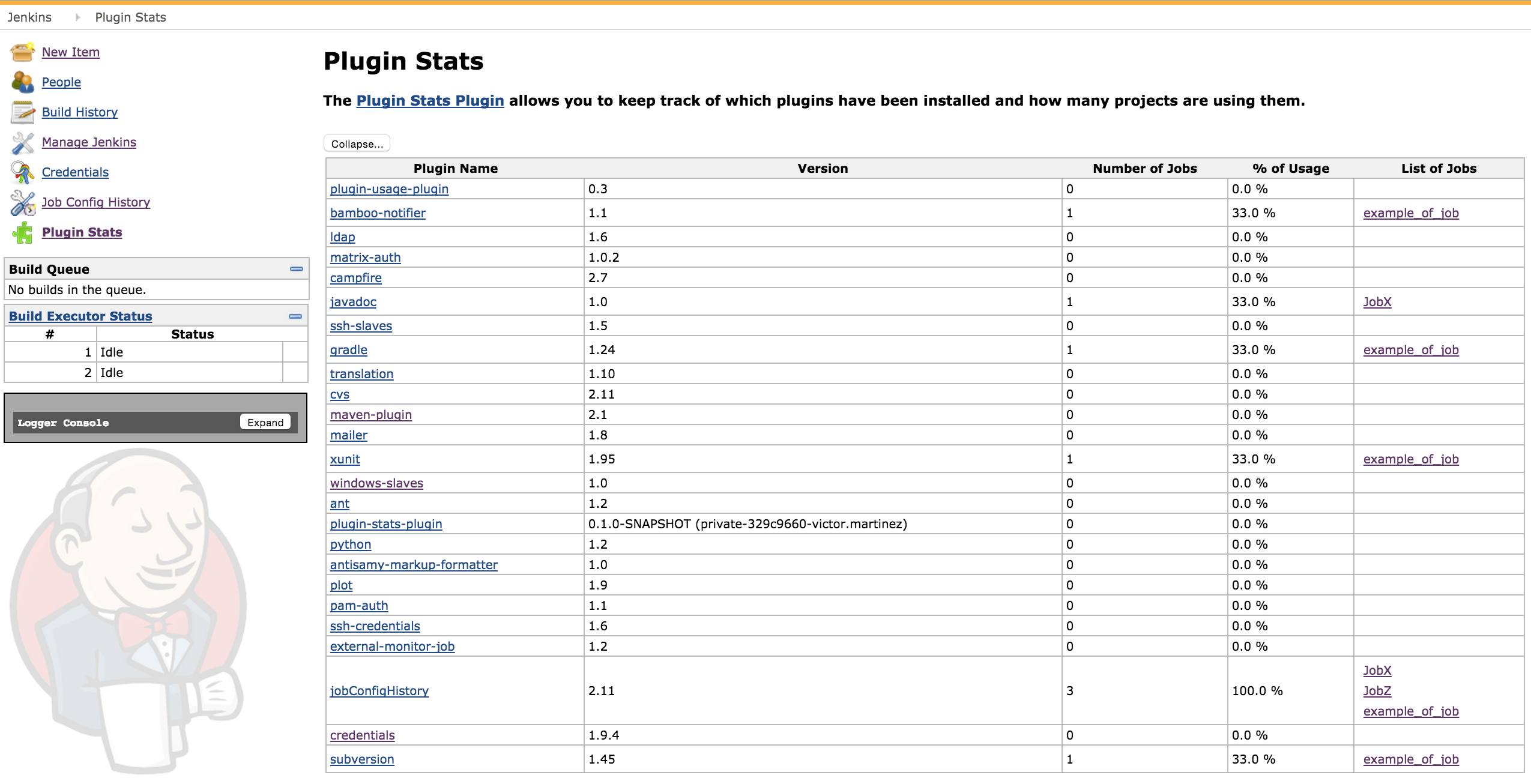
* Notify when Job configuration changes.
* Recipients are configured at job configuration page.
* Recipients are notified when one of these events occurs

# -[Plugin Stats Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Plugin+Stats+Plugin)

* This plugin allows you to keep track of which plugins have been installed and how many projects are using them

## Features:

* List Installed plugins and how many Jobs are using them.
* Percentage of usage.
* List Jenkins jobs with those installed plugins.
* Support Multi-Branch Project Plugin
* Support Maven Project Plugin

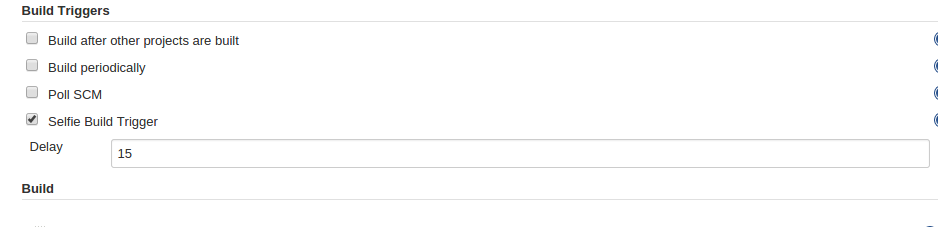


### Planned upcoming features:

* Support Properties plugins.
* Disable plugins.
* Classify what kind of plugins are based on: Builders, Wrappers, Properties, Actions, Publishers, SCM and the above plugins.

# -[Selfie Trigger Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Selfie+Trigger+Plugin)

* Selfie trigger plugin enables build to trigger itself after a configured delay.
* The delay (in seconds) can be set while configuring the build as shown in the image below



**-Shell Script Security**

\*Shell scripts running as CGI scripts on a web server take input from the network.

\*Shell scripts that read files and take actions based on their contents may take input from untrusted files.

\*Shell scripts that perform web queries (with curl, for example) or other network requests may take input from untrusted servers or clients.

* **Environment Attacks**

Environment variable attacks are the most common way to manipulate script behavior. By manipulating the environment of a script, you can change its behavior if the script depends on the values of those environment variables.

**The attack:**

Create an executable binary or script that does something harmful and name it “ls”. Then do this:

|  |
| --- |
| export PATH=/path/to/malicious/binary:$PATH |
| /path/to/above/script |

Because the path to the malicious binary is first in the search path, the malicious ls command gets executed instead of the real one.

**Mitigation:**

Always specify absolute or relative paths when executing binaries or other scripts. If your script runs other scripts or binaries that do not use absolute or relative paths internally, you should explicitly set the value of the PATH environment variable in your scripts to prevent problems.

### Temporary File Attack

The simplest example of this attack is a tool storing secret information into a temporary file.

**The attack:**

Create a tool that watches for the file /tmp/mysecretdata to appear. (Although this can be done with a shell script, it probably won’t be fast enough to work very often. Use the File System Events API in C instead.)

Upon detecting the existence of the path, do this:

|  |
| --- |
| FILE \*fp=fopen("/tmp/mysecretdata", "r"); |

If the attacker manages to open the file before the script executes the chmod command, it can continue to read data from the file for as long as it keeps the file open.

**Mitigation:**

There are two things you must do to fix this:

* Always use the [umask](https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/umask.1.html" \l "//apple_ref/doc/man/1/umask" \t "_self) command to specify initial permissions on the file when you create it.
* Always create temporary files with the [mktemp](https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/mktemp.1.html" \l "//apple_ref/doc/man/1/mktemp" \t "_self) command. This creates a new file with the specified template, ensuring that a file or symbolic link with that name does not already exist.

For example:

|  |
| --- |
| #!/bin/sh |
|  |
| SECRETDATA="My password is 12345." |
| umask 0177 |
| FILENAME="$(mktemp /tmp/mytempfile.XXXXXX)" |
| echo "$SECRETDATA" >> "$FILENAME" |

## Authentication Attacks

In general, you should not rely on a script to determine whether a user does or does not have permission to do something. It is clumsy and error-prone. It is possible to do so, however, and there are right and wrong ways to do it.

The wrong way:

|  |
| --- |
| if [ $UID = 100 -a $USER = "myusername" ] ; then |
| cd $HOME |
| fi |

This code has three security bugs, and they’re all caused by using variables in ways that are unsafe. For historical compatibility, the OS provides the UID, USER, and HOME environment variables. They are quite useful as long as you aren’t using them for security reasons.

**The attack:**

|  |
| --- |
| $ tcsh |
| % setenv UID 100 |
| % setenv USER myusername |
| % setenv HOME $HOME/.ssh |
| % /path/to/script.sh |

Even though most modern Bourne shells protect against modifying UID, the USER variable is unprotected, and not all shells protect the UID variable, either.

Fortunately, the script just changed into a directory.

**Mitigation:**

To obtain the user ID:

|  |
| --- |
| # Effective UID |
| MYEUID="$(/usr/bin/id -u)" |

To obtain the username:

|  |
| --- |
| MYUID="$(/usr/bin/id -u -n)" |

To obtain the actual home directory:

|  |
| --- |
| HOMEDIR="$(dscl . -read /Users/dg NFSHomeDirectory | sed 's/^NFSHomeDirectory: //')" |

**Queries:**

1. Rollback issue