Using Jenkins As Continuous Build Software

This is a quote from http://jenkins-ci.org/content/about-jenkins-ci:

> *"In a nutshell Jenkins CI is the leading open-source continuous integration server. Built with Java, it provides 800 plugins to support building and testing virtually any project."*

In This Chapter

Overview

Jenkins is a free open-source continuous build software package. It is managed by the Jenkins CI community. It is well suited for Java based projects, and some of the more notable features include:

- Plugins for many popular builders such as Ant, Maven, ...; source control such as CVS, SVN, Git, Mercurial, ...; integration with GitHub through various plugins; notification via email and and other means; authentication via default container authentication/OpenID/Custom users/...
- Other 3rd part extensions for lots of other nice features.
- Multiple jobs can be made to run at the same time, even on multiple master/slave Jenkins nodes and, using some plugins this can be configured around a number of available external 'resources'.
- Groovy scripts can be run directly from the web interface against a Jenkins API.
- Open source so extensible and customizable.
- Web application - the entire build/job configuration can be done via a web browser.

Installing Jenkins

Download and install Jenkins native package from http://jenkins-ci.org/. Jenkins can also be deployed as a .war file in containers such as Tomcat, but the configuration would be identical.

For the purpose of this tutorial we will use the native Jenkins stand-alone installer.

You should now be able to access Jenkins at `http://localhost:8080`. This tutorial is based on a native Windows service installation, but it should be similar on other OSes as well.

Configuration Tips

As Jenkins and Servoy are both Java driven applications, they share some Java technologies, and by default, share port configurations as well. If you plan to run Jenkins and Servoy Application Server on the same machine, port changes will need to be made one application or the other in order for the applications to work properly.

Only one or the other needs to be modified, not both.

Changing ports for a Servoy Application Server installation is detailed here.

To change port configurations for Jenkins instead, modify the `jenkins.xml` file found in the Jenkins main directory. See the section below:

jenkins.xml

```
  <arguments>-Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar "%BASE%\jenkins.war"
--httpPort=8080</arguments>
```

Note that '--httpPort=8080' is in conflict with Servoy Application Server default port. Change this port to any other available port on the machine (for example 8085) then restart the Jenkins service. The new URL to access Jenkins will be `http://localhost:8085`. (If you plan to install Jenkins as a .war inside another container such as tomcat, then that container's configuration determines the port and the URL will be something like http://localhost:8085/jenkins)

Jenkins is fully configurable from the browser. It has many useful plugins. If you think you need something, look through the available plugins. What you need is probably already available.

In this tutorial we will use two plugins (to install them go - in the browser - to **Manage Jenkins** > **Manage plugins**):

- openID plugin (allows authentication based on google account (or company Google account) or any other account that provides OpenID authentication)
- build timeout plugin (makes sure jobs will automatically be stopped after a timeout - so your build fails if it take too long for some reason, instead of just never ending)

Some plugins that we will use such as SVN plugin are pre-installed.

If you experience strange errors like '500 Internal Server Error' when configuring a Jenkins job to connect to a SVN location that requires authentication, check for updates to the Subversion Jenkins Plugin. Before plugin version 2.0 there were problems connecting to some SVN servers (not all) due to an old SVNKit version used by this plugin.

At each SVN location defined in a job you will be able to choose a separate 'Credential'. Those credentials can be managed from **Manage Jenkins** > **Manage Credentials**.

Configure Security Using openID (skip this step if you are not interested in authentication on the Jenkins server)

- go to **Manage Jenkins** > **Configure global security**
- check **Enable security**
- choose **OpenID SSO** as **Security Realm** and in the Provider URL field put: https://www.google.com/accounts/o8/id
- choose **Matrix-based** security
- type your google username in **User/group to add** and click on "**Add** button
- grant yourself all rights by checking all checkboxes in your user's row
- uncheck all checkboxes in the **Anonymous** row, except for **Job - Discover** (this one allows people to get prompted with the login when try to access a job's URL). Please do this only after you granted yourself all rights, otherwise noone will be able to access Jenkins.
- click on **Save**

Now you should be the only one that can access the Jenkins server using your google credentials.

You can add any other people and configure their rights as needed now.

Note: Jenkins access can be configured in other ways as well, if you don't have a google/other openID account or you just want to use something else (LDAP/custom users).

Other Things to Configure

Here are some settings you should/could tweak in **Manage Jenkins** > **Configure System**:

- if you don't have Ant installed, in **Ant** section add a new installation, give it a name and check **Install automatically**.
- download and unzip ant-contrib (for example I downloaded and unzipped ant-contrib-0.6-bin.zip) from http://sourceforge.net/projects/ant-contrib/files/ (project page is http://ant-contrib.sourceforge.net/). Check **Environment variables** in Jenkins configuration page and set a key-value pair with key 'ANT_CONTRIB_JAR' (drop the quotation marks) and value containing the full path to where the downloaded `ant-contrib*.jar` is located in the unzipped contents - including file name.
- '# of executors': set it to 1 for now; this limits the concurrent executing jobs to 1. In the future if you want to set-up a more advanced configuration you can configure multiple jobs to run simultaneously (for example for multiple solutions) - just make sure they use separate resources so that they don't affect each other (for example separate Servoy Installations).
- 'Jenkins URL': to the real URL that will be used by users to access the Jenkins server. (for example this will appear as part of notification emails)
- 'System Admin e-mail address': use your email address.
- 'E-mail Notification': configure this as needed. By default it will try to use the localhost SMTP server which will already be working probably on some linux machines. You can also test the configuration by sending a test email.
- hit the **Save** button.

Step-by-step Configuration for Using It with Servoy

In the SVN repository we will have Servoy solutions that need to be automatically tested using import-smart-client and a solution to be automatically tested in mobile-test-client. The ANT scripts used by Jenkins to automatically export and test Servoy solutions will be in a separate project 'antRunner' in the same SVN repository.

This is what will happen when a Servoy - Jenkins job (defined later in this chapter) runs:

1. changes are fetched from SVN (solution projects, resources project(s), antRunner project);
2. root solutions are exported as .servoy files or .war mobile files. (because the .servoy workspace exporter uses the '-dbi' option in the sample ant scripts, it will not need databases to be online, but servoy.properties must still contain the required server definitions; for the sample workspace servoy.properties is already configured correctly);
3. test client starts a database repository - based on the repository database that must be defined in servoy.properties (if it is not defined to be an in-memory server / HSQL, then the database must be online; for the sample workspace it will use an in-memory server, so no database needs to be started);
4. solutions from (2) are imported into that repository (.servoy files) or in Servoy's embedded Tomcat (for mobile .war files) (databases needed by the solutions must be online if not defined as in-memory; for the sample workspace no database server needs to be started);
5. tests run on those solutions in a test client (which is either a slightly modified smart client - to not block with modal dialogs when errors occur for example, or a slightly modified version of mobile client);
6. results are published and emails are sent if necessary.

Step 1 -- Sample Workspace and SVN Content

Sample Workspace

First we have to set-up sample SVN contents. This can be done on any computer with access to the SVN server, not necessarily on the same computer as Jenkins.

Here is the sample Servoy workspace that we will use: SoftwareFactory_Jenkins_Workspace.zip.

Unzip. Start Servoy Developer and switch workspace to the unzip location (**File** > **Switch Workspace**).

Go to **File** > **Import** > **General** > **Existing Projects into Workspace**. At 'Select root directory' click the **Browse** button then enter (it should select the workspace directory). Check all projects - you should see all projects that you previously unzipped. Click **Finish**.

You can now have a look at the sample solutions and run tests on them from developer.

Commit to Your SVN Repository

Install Subclipse plugin in Servoy Developer (**Help** > **Install New Software**) if you haven't already done so.

Open Navigator view (**Window** > **Show view** > **Other** > **General** > **Navigator**). Select all projects and **right click** > **Team** > **Share Project** > **SVN**.

For each project click on **Share project** button and add it to your SVN repository (you can use an existing SVN repository or just make a SVN test server with SVN server software such as VisualSVN). I added it to something like `http://{myHost}:{svnServerPort}/svn/jenkinsSolutions/trunk/{projectName}`. For the first project you will have to choose 'create repository (connection) > `http://{myHost}:{svnServerPort}/svn/jenkinsSolutions` and directory 'trunk/demo_ipAddressValidator'. For the rest you will choose an existing SVN repository (the connection you just created).

After all projects are shared, select all from Navigator view, **right click** > **Team** > **Synchronize**. In the **Synchronize** view select all and **right click** > **Commit**.

Step 2 -- Install Separate Servoy Developer

We need an installation of Servoy Developer on the machine running Jenkins for exporting test solutions and running tests.

Download and install Servoy Developer. It's best to use a clean installation (no extra eclipse plugins) to avoid potential problems generated by other eclipse plugins (for example installing Subclipse in this developer can cause a hang as that plugin is trying to access the UI while blocking a resource - in the headless solution export environment).

In case of Servoy 7.x: if for some reason you intend to custom-sign any jars in that Servoy installation you must sign all the jars from both application server (relevant here is *application_server/lib/j2db.jar*) and developer, as well as jars contained in *developer/plugins/com.servoy.eclipse.jsunit_*.jar* (relevant here is *j2db_test.jar*). If these jars are not identically signed then you will get an IllegalStateException when trying to start the (smart) unit test client. But normally you would use just a clean install without custom-signing any jars.

Step 3 -- Configure Smart Client Jenkins Job

Your Jenkins server should be able to access the SVN server you just committed to - in case you didn't use the same computer. Navigate to the Jenkins server page in your browser.

Click on **New Item** > **Build a free-style software project**, and set the Job name to 'smart_client_tests' ('_' is used to avoid problems with spaces in paths where used as command line arguments - extra care is needed otherwise).

Under 'Advanced Project Options' click the button then in 'Display Name' type Smart Client Tests.

Under 'Source Code Management' select Subversion. We will add the projects needed for smart client tests one by one:

1. fill Repository URL with `http://{myHost}:{svnServerPort}/svn/jenkinsSolutions/trunk/solution1`. When you tab out of the field, if your SVN repository requires authentication - and you didn't already provide it, you will get a red error message 'Unable to access ... '. Click on the **enter credential** link and add the SVN credentials that this test server is meant to use.

2. remove the dot in 'Local module directory (optional)'
3. click on **Add more locations** and repeat the two steps above 4 times - for the other needed projects. Use the URLs for 'solution2', 'moduleof2', 'resources' and 'antRunner'.

Note: you can also checkout the whole branch at once if all your projects are needed for smart-client tests. Then you will have to leave the '.' in 'Local module directory (optional)'.

Under 'Build Triggers' select 'Poll SCM' and in 'Schedule' field use H/3 * * * * (every 3 minutes, let Jenkins choose the minute when it checks for SVN changes to avoid overcrowding in the future). You can increase that to decrease requests on the SVN server. If changes are found Jenkins will trigger the build. If you want a check every minute use * * * * *.

Under 'Build Environment' check 'Abort the build if it's stuck', then select 'Absolute' - 7 minutes. This will make sure that the build ends with failure after 7 minutes if something gets stuck. This is optional and of course you need to tweak this depending on how long your tests usually take. This option is contributed to Jenkins by the 'build timeout plugin'.

Under 'Build' click on **Add build step**. Choose **Invoke Ant**. Select the version you configured to auto-install at 'Other things to configure' above. Click on **Advanced**. In 'Targets' type 'main_smart'. In 'Build file' type 'antRunner/jenkins_build.xml'. In 'Properties', expand the field (arrow down button) then type:

```
work.servoy.install.dir = [path_to_your_separate_Servoy_developer_installation]
smart_test_solutions    = solution1,solution2
WORKSPACE               = ${WORKSPACE}
ANT_CONTRIB_JAR         = ${ANT_CONTRIB_JAR}
```

If you are on windows use '/' or '\\' instead of '\' in that path. For example `work.servoy.install.dir=f:/ServoyInstalls/Servoy_7.3`.

Under 'Post-build Actions':

- add 'Archive the artifacts' with this in 'Files to archive': `antRunner/jsunit_results/junit-noframes.html`, `antRunner/jsunit_results/smart_import_test_client_log.txt`, `antRunner/jsunit_results/workspace_exporter_app_server_log.txt`, `antRunner/jsunit_results/workspace_exporter_workspace_log.txt`
- add 'Publish JUnit test result report' with 'Test report XMLs' set to: `antRunner/jsunit_results/TESTS-TestSuites.xml`
- add 'E-mail notification' and type your email address in 'Recipients' field.

Click on **Save** and go back to Jenkins dashboard. The new job you just created 'Smart Client Tests' should automatically start in under 3 minute (or you can start it manually using the **play** icon next to it). It should pass.

If anything goes wrong you can click on your job's name then on the left side you see a list of builds. Click on the build that failed and you can see it's details (time/status/tests/console/...).

I also like to have 'AUTO REFRESH' enabled as well (a link in Jenkins dashboard) - just to make sure the page is showing the latest status of Jenkins.

After a build finishes you should have 4 files in your `build/Status/Build Artifacts` - one showing the test results and 3 Servoy log files showing what happened during export/import/test.

Step 4 -- Configure Mobile Client Jenkins Job

For mobile testing we will need Chrome browser installed (other browsers, even mobile device simulations/emulations can be used, check the help page of Servoy mobile test runner for details), a copy of SeleniumServer jar downloaded (http://docs.seleniumhq.org/download/) and a copy of Chrome selenium driver downloaded/extracted (http://code.google.com/p/chromedriver/downloads/list). The mobile test solution will automatically be deployed in Servoy Developer's bundled Tomcat server (the one at [work.servoy.install.dir]) and started using Selenium in the Chrome browser to run tests - by the mobile servoy test suite. If the tests pass, the actual mobile solution will be exported and copied to/deployed in [deploy.servoy.install.dir]'s bundled Tomcat.

Click on **New Item** > **Build a free-style software project**, and set the Job name to 'mobile_client_tests'.

Under 'Advanced Project Options' click the button then in 'Display Name' type 'Mobile Client Tests'.

Under 'Source Code Management' select Subversion. We will add the projects needed for mobile client tests one by one:

1. fill Repository URL with `http://{myHost}:{svnServerPort}/svn/jenkinsSolutions/trunk/demo_ipAddressValidatorTester`. When you tab out of the field, if your SVN repository requires authentication - and you didn't already provide it, you will get a red error message 'Unable to access ... '. Click on the "enter credential" link and add the SVN credentials that this test server is meant to use
2. remove the dot in 'Local module directory (optional)'
3. click on 'Add more locations' and repeat the two steps above 5 times - for the other needed projects. Use the URLs for 'demo_ipAddressValidatorTester_service', 'demo_ipAddressValidator', 'demo_ipAddressValidator_service', 'resources' and 'antRunner'.

Under 'Build Triggers' select 'Poll SCM' and in 'Schedule' field use H/3 * * * * (every 3 minutes, let Jenkins choose the minute when it checks for SVN changes to avoid overcrowding in the future). You can increase that to decrease requests on the SVN server. If changes are found Jenkins will trigger the build. If you want a check every minute use * * * * *.

Under 'Build Environment' check 'Abort the build if it's stuck', then select 'Absolute' - 10 minutes. This will make sure that the build ends with failure after 10 minutes if something gets stuck. This is optional and of course you need to tweak this depending on how long your tests usually take. This option is contributed to Jenkins by the 'build timeout plugin'.

Under 'Build' click on 'Add build step'. Choose 'Invoke Ant'. Select the version you configured to auto-install at 'Other things to configure' above. Click on 'Advanced'. In 'Targets' type 'main_mobile'. In 'Build file' type 'antRunner/jenkins_build.xml'. In 'Properties', expand the field (arrow down button) then type:

If you are on windows use '/' or '\\' instead of '\' in that path.

```
work.servoy.install.dir   = [path_to_your_separate_Servoy_developer_installation]
deploy.servoy.install.dir = [path_to_your_deployment_Servoy_developer_installation]
webdriver.chrome.driver   = [path_to_chromedriver.exe_that_was_downloaded_above]
selenium.server.jar       = [path_to_downloaded_selenium_server_jar]
mobile_test_solutions     = demo_ipAddressValidatorTester
mobile_service_solutions  = demo_ipAddressValidatorTester_service
mobile_to_deploy          = demo_ipAddressValidator
mobile_to_deploy_services = demo_ipAddressValidator_service
WORKSPACE                 = ${WORKSPACE}
ANT_CONTRIB_JAR           = ${ANT_CONTRIB_JAR}
```

For example:

```
work.servoy.install.dir   = f:/ServoyInstalls/Servoy_7.3
deploy.servoy.install.dir = f:/ServoyInstalls/Servoy_7.3_deployment
webdriver.chrome.driver   = f:/Selenium/Drivers/Chrome/chromedriver.exe
selenium.server.jar       = f:/Selenium/ServerJar/selenium-server-standalone-2.39.0.jar
(...)
```

The deployment installation of Servoy can be the one that 'Install/Start Servoy Server Importer' step will use.

Under 'Post-build Actions':

- add 'Archive the artifacts' with this in 'Files to archive': `antRunner/jsunit_results/junit-noframes.html`, `antRunner/jsunit_results/smart_import_test_client_log.txt`, `antRunner/jsunit_results/workspace_exporter_app_server_log.txt`, `antRunner/jsunit_results/workspace_exporter_workspace_log.txt`
- add 'Publish JUnit test result report' with 'Test report XMLs' set to: `antRunner/jsunit_results/TESTS-TestSuites.xml`
- add 'E-mail notification' and type your email address in 'Recipients' field.

Click on **Save** and go back to Jenkins dashboard. The new job should automatically start in under 3 minute (or you can start it manually using the **play** icon next to it). It should export/run tests in chrome/export normal solution/copy it to deployment Servoy installation - with 2 test failures. To make the tests pass fix the regular expression as hinted by the comment in solution demo_ipAddressValidator, in the globals scope, and then commit the change to SVN.

If anything goes wrong you can click on your job's name then on the left side you see a list of builds. Click on the build that failed and you can see it's details (time/status/tests/console/...).

After a build finishes you should have 4 files in your build/Status/Build Artifacts - one showing the test results and 3 Servoy log files showing what happened while exporting/testing.

Customize Jenkins Jobs to Suite Your Set of Test Solutions

Congratulations! Now you should have a working sample Mobile Client test job and a working sample Smart Client test job.

Next you will probably want to configure the same for your own test solutions. Here is how you can do that.

Basic Customization

All the steps for creating jobs for your own solutions are almost identical to the sample jobs (see 'Configure Smart Client Jenkins job' and 'Configure Mobile Client Jenkins job' above). What you should do differently:

- Under 'Source Code Management', instead of configuring sample SVN contents, configure the location of your own solutions to be checked out. Make sure the resulting job workspace contents are similar (after checkout the job workspace directory should contain directly the solution /resources project directories).
- Adjust 'Abort the build if it's stuck' value depending on how long you expect your test job to run.
- At **Build** > **Add build step** > **Invoke Ant** > **Advanced** > **Properties** > **expanded** - edit in that text-area the following properties to specifiy the names of the root solutions relevant for your test job:
    - in case of Smart Client job - replace sample tests root solution names below with your own; comma separated list:

      ```
      smart_test_solutions      = solution1,solution2
      ```

    - in case of Mobile Client job - replace sample root solution names below with your own; comma separated lists:

      ```
      mobile_test_solutions     = demo_ipAddressValidatorTester
      mobile_service_solutions  = demo_ipAddressValidatorTester_service
      mobile_to_deploy          = demo_ipAddressValidator
      mobile_to_deploy_services = demo_ipAddressValidator_service
      ```

- Change the `antRunner/servoy.properties` file as needed by your solutions (define servers and other needed changes; the repository server can be a Hslq (in memory) server to always import clean or you can change it to a persistent server as well).

Advanced Customization

The 'Basic customization' steps should be enough for most cases. If you have the need to control more aspects like using separate servoy.properties for exporter and import test clients, changing arguments used by exporter and import test client or modifying more timeouts that don't suit your solutions it can be done. Here are a few more things that you can do (of course you could do much more with what Jenkins or ant have to offer):

- customize the job ant build's behavior by defining more properties in **Build** > **Add build step** > **Invoke Ant** > **Advanced** > **Properties** > **expanded**. Customizable properties are listed in the first section of 'antRunner/export_and_test.properties' and are explained in their comments preceding each one:

```
# ------------------------------------------------------------------------------------------------
# Properties that you should set in Jenkins job configurations (ant advanced config; they will ---
# be available to ant and also have precedence over what is defined in this file               ---
# ------------------------------------------------------------------------------------------------

# Servoy developer installation that will be used for exporting/testing (a separate install
# is recommended for each Jenkins job in case multiple jobs are configured to run concurrently)

work.servoy.install.dir             = [your_jenkins_dedicated_servoy_installation_dir]
work.servoy.install.url             = http://localhost:8080

# Paths to other downloaded tools that we need

webdriver.chrome.driver             = [path_to_the_downloaded_selenium_chrome_driver_executable]
selenium.server.jar                 = [path_to_downloaded_selenium_server_jar]

# Solutions to be exported and then tested;
# multiple solutions can be specified via a comma separated list of solution names

smart_test_solutions                = solution1,solution2

mobile_test_solutions               = demo_ipAddressValidatorTester
mobile_service_solutions            = demo_ipAddressValidatorTester_service

mobile_to_deploy                    = demo_ipAddressValidator
mobile_to_deploy_services           = demo_ipAddressValidator_service

# If you are using a Servoy version >= 7.3 please uncomment or set the following (otherwise leave it
# commented out; setting it to "false" or "nope" is not the same); the old way should work on 7.3 or
# newer as well, but not the other way around; new way only only gets rid of some unneeded exporter
# initialization time

# work.servoy.install.7.3.or.higher = yep

# 1 hour time-out; edit this as needed - depending on how long it usually takes for your solution tests to
run (should probably be lower in most cases)

test.timeout                           = 3600000

# Uncomment the following to send '-pl' argument to workspace solution exporters: alternate project
locations;
# solution and resources projects will be searched for in direct subfolders of the given 'WORKSPACE'.
# Example: if the workspace needs to contain projects from different git repositories, those can be checked
# out in '[WORKSPACE]/repo1', '[WORKSPACE]/repo2' and so on

#alternate_project_locations.on=yep

# Uncomment the following line/set this if you want passed exports generated by previous build
# to be deleted at the beginning of each build cycle

#always_clean_passed_exports        = yep

# For Sauce Labs test cloud (testing mobile clients); also uncomment relevant lines in
# "run_mobile_client_tests_if_exported_ok" target of jenkins_build.xml; other changes might be needed as
well, but this is the main idea

#sauce.labs.user                    = [your Sauce Labs user here]
#sauce.labs.token                   = [access token goes here]
#sauce.labs.selenium.url            = http://${sauce.labs.user}:${sauce.labs.token}
@localhost:${SAUCE_ONDEMAND_PORT}/wd/hub
```

```
# define a timeout for sauce labs to not require additional Selenium commands for a solution (set to 1.5
min; should be updated for how long your test can possibly take to run, but not much higher)
#sauce.labs.selenium.idle-timeout   = 90

# If you want your mobile solutions deployed to a Servoy server after testing is done,
# uncomment / set values as in the following two lines and change the value of
# servoy.deploy.install.dir to point to it; passed .war files will be copied to deploy.webapps.dir
location.

#deploy.servoy.install.dir          = [your_mobile_war_deployment_servoy_installation]
#deploy.webapps.dir                 = ${deploy.servoy.install.dir}/application_server/server/webapps

# WORKSPACE property is normally already set by Jenkins in job build step
# ANT_CONTRIB_JAR is normally already set if you followed all the steps in "Building a Software Factory"
WIKI
# page; it's set as a system property in global Jenkins configuration - environment variables

#WORKSPACE                          = [if_not_set_by_Jenkins_you_can_configure_it_as_well]
#ANT_CONTRIB_JAR                    =
[if_not_set_in_Jenkins_config_as_environment_variable_you_can_configure_it_as_well]
```

- workspace exporter arguments (you should have available all the options present in UI Solution Export Wizard) can be customized by modifying an `tRunner/export_and_test.xml` macrodefs 'export-solution' and 'export-mobile-solution'. You can see a list of available command line arguments in a console window by executing `{servoyInstall}/developer/exporter/export.bat -help` and `{servoyInstall}/developer/exporter/mobile_export.bat -help`;
- smart test client and mobile test client arguments (which are actually system properties sent to a junit test suite) can be customized by modifying an `tRunner/export_and_test.xml` macrodefs 'run-smart-client-tests' and 'run-mobile-client-tests'. You can see a list of available arguments by commenting out the '`<!--arg value="-help" /-->`' lines in these marcrodefs and creating/executing an ant task that calls them from the command line (or alternatively just comment out those lines in a Jenkins job and check the job build's console - after it is executed).