# buildout.jenkins Documentation

*Release 1.0*

**Timo Stollenwerk - Plone Foundation**

**Sep 28, 2017**

# Contents

buildout.jenkins allows you to easily set up a buildout for Plone projects that the Jenkins CI-Server can use to generate reports for tests, test-coverage and code-analysis.

CHAPTER 1

# Buildout Installation

Create a jenkins.cfg file in your buildout directory that extends your existing buildout.cfg. Set the 'jenkins-test-eggs' parameter for the eggs you want to be tested by jenkins:

```
[buildout]
extends =
    buildout.cfg
    https://raw.github.com/plone/buildout.jenkins/master/jenkins.cfg

jenkins-test-eggs = plone.app.collection [test]
```

If you want to run code analysis tools (e.g. PEP 8, PyFlakes, ZPTLint), just extend your 'jenkins.cfg' with the 'jenkins-code-analysis.cfg' file and set the 'jenkins-test-directories' parameter to the package directories you want to analyze:

```
[buildout]
extends =
    buildout.cfg
    https://raw.github.com/plone/buildout.jenkins/master/jenkins.cfg
    https://raw.github.com/plone/buildout.jenkins/master/jenkins-code-analysis.cfg

jenkins-test-eggs = plone.app.discussion [test]
jenkins-test-directories = src/plone.app.collection/plone/app/collection
```

It is also possible to run jenkins-test and jenkins-code-coverage on multiple packages:

```
[buildout]
extends =
    buildout.cfg
    https://raw.github.com/plone/buildout.jenkins/master/jenkins.cfg
    https://raw.github.com/plone/buildout.jenkins/master/jenkins-code-analysis.cfg

jenkins-test-eggs =
    plone.app.collection [test]
    plone.app.contenttypes [test]
jenkins-test-directories =
```

```
    src/plone.app.discussion/plone/app/discussion
    src/plone.app.contenttypes/plone/app/contenttypes
```

CHAPTER 2

# Jenkins Job Shell Build Script

Configure a free-style Jenkins project and add a shell build script with the following lines:

```
python2.6 bootstrap.py
bin/buildout -c jenkins.cfg
```

If you want Jenkins to run the test only, append the following line:

```
bin/jenkins-test
```

If you want to run Robot Framework test, use:

```
bin/jenkins-test-robot
```

If you want Jenkins to run the test together with a test-coverage analysis (suitable for use with the Cobertura Plugin in Jenkins), append:

```
bin/jenkins-test-coverage
```

If you want Jenkins to run a code analysis, append:

```
bin/jenkins-code-analysis
```

It is also possible to run only certain code anlysis tasks.

CHAPTER 3

# Jenkins Configuration

For Jenkins to be able to understand the output of the tests and analyses, you will need to configure your instance accordingly.

- **Test results with `bin/jenkins-test` and `bin/jenkins-test-coverage`:** Configure your Jenkins project's options by enabling *Publish JUnit test result report* and setting *Test report XMLs* to be `parts/jenkins-test/testreports/*.xml`.
- **Test results with `bin/jenkins-test-robot`:** Configure your Jenkins project's options by enabling *Publish Robot Framework test results* setting in the *Post-build Actions* to be: - Directory of Robot output: parts/test - Log/Report link: robot_log.html - Output xml name: robot_output.xml - Report html name: robott_report.html - Log html name: robot_log.html
- Test coverage with `bin/jenkins-test-coverage`:

  Plugins recommended:

  - Cobertura Plugin read Cobertura installation instructions
  - Violations Plugin read Violations installation instructions

CHAPTER 4

# Post-build actions

All items are added through the *Add post-build action* button in your project.

**Publish JUnit test result report** Test report XMLs: `parts/jenkins-test/testreports/*.xml`

**Publish Coberture Coverage Report** Cobertura xml report pattern: `parts/jenkins-test/coverage.xml` Other options can stay on their default values.

**Report Violations** Following reports are available for Violations plugin:

- **csslint**

  XML filename pattern: `parts/jenkins-test/xml-csslint/**/*.xml`

- **jshint**

  XML filename pattern: `parts/jenkins-test/xml-jshint/**/*.xml`

- **pep8**

  XML filename pattern: `parts/jenkins-test/pep8.log`

- **cpd**

  XML filename pattern: `parts/jenkins-test/xml-clonedigger/**/clonedigger.xml`

**Clonedigger - Setup of violations plugin:** The clonedigger with `--cpd-output` will generate PMD's cpd similar output. (I hope there is no problem their XML schema in clonedigger).

http://clonedigger.sourceforge.net/hudson_integration.html

That should be all. Now run the build and watch for yourself! Enjoy.

CHAPTER 5

# Prerequisits

In order to be able to run some of the code analysis jobs you have to manually install some dependencies on the Jenkins machine:

## XMLLint

On Debian/Ubuntu just install the libxml2-utils:

```
$ sudo apt-get install libxml2-utils
```

## OHCount

On Debian/Ubuntu just install the ohcount package:

```
$ sudo apt-get install ohcount
```

## Sloccount

On Debian/Ubuntu just install the ohcount package:

```
$ sudo apt-get install sloccount
```

## Nodejs

Some code analysis steps require nodejs to be installed. On Debian/Ubuntu just install the nodejs and npm package:

```
$ sudo apt-get install nodejs npm
```

You can also install nodejs with a buildout recipe by adding this section to your buildout.cfg:

```
[jshint]
recipe = gp.recipe.node
npms = jshint
url = http://nodejs.org/dist/v0.8.9/node-v0.8.9.tar.gz
scripts = jshint
```

## JSLint

On Debian/Ubuntu you can run jslint on nodejs:

```
$ sudo apt-get install nodejs npm
$ sudo npm install -g jslint
```

## JSHint

On Debian/Ubuntu you can run jshint on nodejs:

```
$ sudo apt-get install nodejs npm
$ sudo npm install -g jshint
```

## CSSLint

On Debian/Ubuntu you can run csslint on nodejs:

```
$ sudo apt-get install nodejs npm
$ sudo npm install -g csslint
```

Contents:

### Jenkins Code Analysis

Expect jenkins code anlysis buildout.

#### PyFlakes

1. Install the warnings plugin:

   ```
   https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin
   ```

2. Set up a compiler warnings in jenkins configure.

Name:

```
pyflakes
```

Regular Expression:

^(.*):([0-9]*):(.*)$

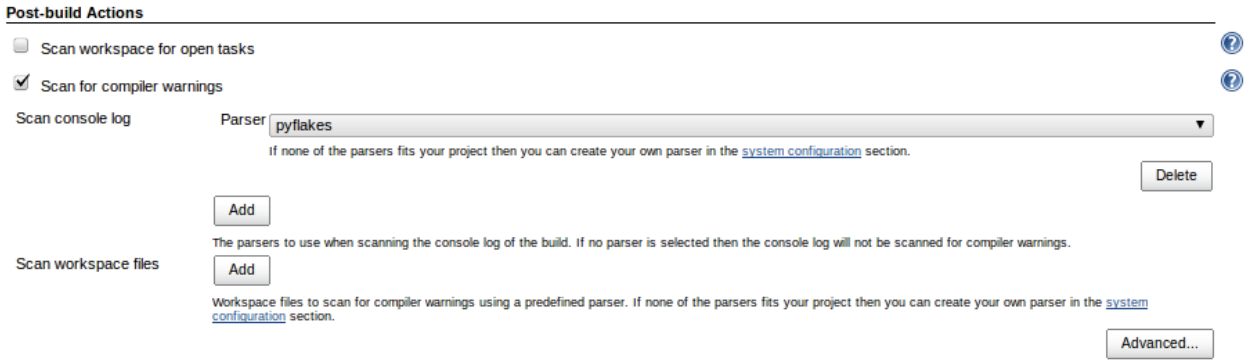
Mapping Script:

```
import hudson.plugins.warnings.parser.Warning
import hudson.plugins.analysis.util.model.Priority

String fileName = matcher.group(1)
String category = "PyFlakes Error"
String lineNumber = matcher.group(2)
String message = matcher.group(3)

return new Warning(fileName, Integer.parseInt(lineNumber), category, "PyFlakes Parser
↪", message, Priority.NORMAL);
```

3. Set up pyflakes warnings for job

Search for compiler warnings => search console =>

parser: pyflakes




## ZPTLint

1. Install the warnings plugin:

```
https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin
```

2. Set up a compiler warnings in jenkins configure.

Name:

```
zptlint
```

Regular Expression:

^.*in:s(S*)s*(?:<classs')?(S*)(?:'>)?:s(.*)s*,.*lines*([0-9]*).*$

Mapping Script:

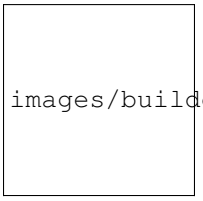
```
import hudson.plugins.warnings.parser.Warning
import hudson.plugins.analysis.util.model.Priority
```

```
String fileName = matcher.group(1)
String category = matcher.group(2)
String lineNumber = matcher.group(4)
String message = matcher.group(3)

return new Warning(fileName, Integer.parseInt(lineNumber), category, "ZPTLint Parser",
↪ message, Priority.HIGH);
```

3. Set up zptlint warnings for job

Search for compiler warnings => search console =>

parser: zptlint




### Flake 8

1. Install the warnings plugin:

```
https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin
```

2. Set up a compiler warnings in jenkins configure.

Name:

```
Flake8
```

Regular Expression:

^(.*):([0-9]*):([0-9]*):(.E[0-9]*)(.*)$

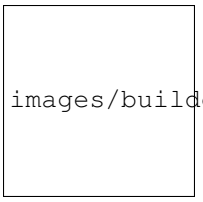
Mapping Script:

```
import hudson.plugins.warnings.parser.Warning
import hudson.plugins.analysis.util.model.Priority

String fileName = matcher.group(1)
String lineNumber = matcher.group(2)
String message = matcher.group(5)
String category = matcher.group(4)

return new Warning(fileName, Integer.parseInt(lineNumber), category, "PyFlakes Parser
↪", message, Priority.NORMAL);;
```

3. Set up flakes8 warnings for job

Search for compiler warnings => search console =>

parser: flakes8




### UTF-8 headers

1. Install the warnings plugin:

```
https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin
```

2. Set up a compiler warnings in jenkins configure.

Name:

```
utf-8 headers
```

Regular Expression:

(.*)

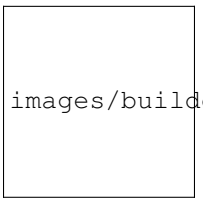
Mapping Script:

```
import hudson.plugins.warnings.parser.Warning
import hudson.plugins.analysis.util.model.Priority

String fileName = matcher.group(1)

return new Warning(fileName, 1, "utf-8", "encoding", "Missing # -*- coding: utf-8 -*-␣
↪in python file.", Priority.LOW);
```

3. Set up utf-8 warnings for job

Search for compiler warnings => search console =>

parser: utf-8 headers




### XMLLint

1. Install xmllint on the Jenkins server:

```
$ sudo apt-get install libxml2-utils
```

2. ...

### Open Tasks

1. Install the task scanner plugin:

```
https://wiki.jenkins-ci.org/display/JENKINS/Task+Scanner+Plugin
```

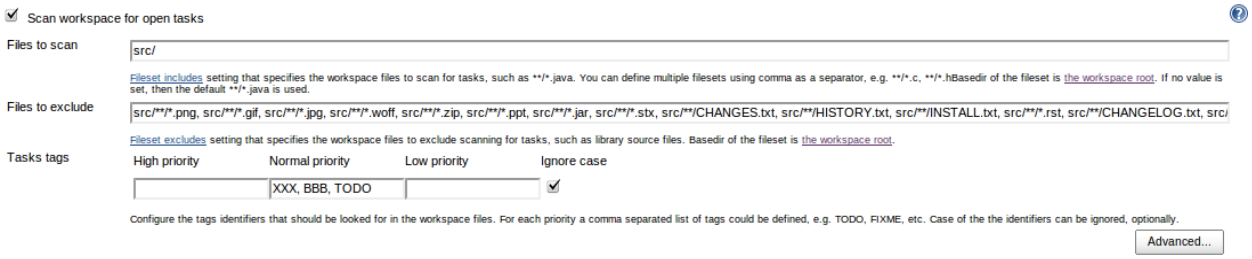
2. Configure your Jenkins job:

Go to section “Post-build Actions”. Check the “Scan workspace for open tasks” checkbox and fill out the settings:

Files to scan: packages/**/*

Files to exclude: packages/**/*.png, packages/**/*.gif, packages/*/*.jpg, packages/*/*.zip, packages/*/*.ppt, packages/*/*.jar, packages/*/*.stx, packages/*/CHANGES.txt, packages/**/HISTORY.txt, packages/**/INSTALL.txt, packages/**/*.rst, packages/**/CHANGELOG.txt, packages/**/ChangeLog

Tasks tags: Normal priority: XXX, BBB, TODO



CHAPTER 6

# Indices and tables

- genindex
- modindex
- search