

CI Jenkins

Daniel **Bracher** and Maximilian **Wech**

tgm Vienna
Wexstraße 19-23, 1200 Vienna

1 Tasks

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage." M.Fowler

Use as a team (2) a virtual server instance to get used with a Continuous Integration System.

1. Install CI "Jenkins"
2. Integrate your "Rock the net" project
3. Provide a stable build for public download
4. Show the whole documentation of your implementation (sourcecode / API, testdocumentation, coverage analysis, etc.)

Protocolize your work in a pdf-Document.

2.1 Work Distribution

	Bracher	Wech
Documentation	X	X
Installing Jenkins	X	X
Integrate "Rock the net" project	X	X
Providing a stable build	X	X

2.2 Time Notes (Hours)

	Estimated time [h]	Bracher	Estimated time [h]	Wech
Documentation	2	3	2	3
Installing Jenkins	1	2	1	2
Integrate "Rock the net" project	2	3	2	3
Providing a stable build	1	1	1	1

3 Execution Protocol

a)Installing Jenkins (in a virtual server instance such as Debian)

```
apt-get update
apt-get upgrade
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list'
apt-get install jenkins
```

b)Setting-up Jenkins

etc/default/jenkins -> change HTTP_HOST and AJP_HOST to your ip-address

```
# Listen address for HTTP connector
HTTP_HOST=192.168.0.13

# Listen address for AJP connector
AJP_HOST=192.168.0.13
```

etc/init.d/jenkins restart

c)Integrate the “Rock the net” project

First of all, it's necessary to create a new project in Jenkins:

- > Create “New Element”
- > Type in a “Project Name” without spaces
- > Choose “FreeStyle” project type

To get the source code from the “Rock the net” project, we integrated Git in Jenkins with a plugin:

- > Plugin-Manager
- > Git Plugin
- > Install without restart

After installing the plugin, a few configuration operations have to be done:

- > System Configuration
- > Navigate to the Git plugin tab
- > Do the following inputs:

Git plugin

Global Config user.name Value	<input type="text" value="jenkins"/>
Global Config user.email Value	<input type="text" value="jenkins@localhost"/>
Create new accounts base on author/committer's email <input type="checkbox"/>	

Some changes have to be done in the project configuration as well:

-> "Configure"

-> Navigate to the Source-Code-Management tab and type in the "Repository URL" and the necessary branch.

Source-Code-Management

<input checked="" type="radio"/> Git	
Repositories	<div>Repository URL <input type="text" value="git://github.com/austrargentina/Jenkins.git"/> Credentials <input type="text" value="- leer -"/> <input type="button" value="Add"/></div>
Branches to build	Branch Specifier (blank for 'any') <input type="text" value="*/Development"/>

If everything has worked, after a successful build you should have your "Rock the Net" project in jenkins:



d)Implement build

Several steps are mandatory to create a successful build:

- > Install ShiningPanda Plugin
- > Configure System
- > Navigate to Python and choose the python version as in the image below

Python Installationen

Name	Home or executable
System-CPython-3.3	/usr/bin/python3.4

☒ Automatisch installieren

[Installationsverfahren hinzufügen](#)

[Python entfernen](#)

[Python hinzufügen](#)

[Liste der Python Installationen auf diesem System](#)

In addition to the system settings, we have to configure some things in our project as well:

- > Go to your project
- > Configure
- > Navigate to "Buildverfahren"
- > Choose Execute Shell
- > To run a class just type in "python3 class.py"

Buildverfahren

Shell ausführen

Befehl `python3 test.py`

[Liste der verfügbaren Umgebungsvariablen](#)


[Löschen](#)

[Build-Schritt hinzufügen](#)

-> Save

-> Click on  [Jetzt bauen](#)

-> If it works, the output looks like that:

 **Build #60 (16.11.2014 22:35:08)**



Konsolenausgabe

```
Started by user anonymous
Building in workspace /var/lib/jenkins/jobs/RockTheNet/workspace
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git://github.com/austrargentina/Jenkins.git # timeout=10
Fetching upstream changes from git://github.com/austrargentina/Jenkins.git
> git --version # timeout=10
> git fetch --tags --progress git://github.com/austrargentina/Jenkins.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/Development^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/Development^{commit} # timeout=10
Checking out Revision b0df4914f286bd126f31elf40821979112e88df6 (refs/remotes/origin/Development)
> git config core.sparsecheckout # timeout=10
> git checkout -f b0df4914f286bd126f31elf40821979112e88df6
> git rev-list 854a1781d8ead4b0a4ee648f42786150e1993cbf # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson6711500927319191936.sh
+ python3 test.py
It works!
Finished: SUCCESS
```

e) Adding documentation of implementation to builds

For this part, following plugins are necessary:

-> Install those 4 plugins in the pluginManager

Javadoc Plugin	Erfolgreich
Maven Project Plugin	Erfolgreich
Cobertura Plugin	Erfolgreich
Violations	Erfolgreich

Running tests:

- > Project
- > Configure
- > Execute shell:

Shell ausführen

Befehl `pip install --quiet nosecover`
`nosetests --with-xunit UnitTests/TestConnector.py`

[Liste der verfügbaren Umgebungsvariablen](#)

Löschen

- > Post-Build-Actions
- > Publish JUnit - test results
- > Add nosetest.xml

Post-Build-Aktionen

Veröffentliche JUnit-Testergebnisse.

Testberichte in XML-Format `nosetests.xml`

Es sind reguläre Ausdrücke wie z.B. 'myproject/target/test-reports/*.xml' erlaubt. Das genaue Format können Sie [der Spezifikation für @includes eines Ant-Filesets](#) entnehmen. Das Ausgangsverzeichnis ist der [Arbeitsbereich](#).

☐ Lange Standard-Out/-Error Ausgaben aufbewahren

Löschen

Post-Build-Schritt hinzufügen ▼

-> Run a build

The result will be shown in the test interface:

As you can see, all 7 passed.

Testergebnisse

Fehlschläge

Tests

Dauer: 6 Sekunden

[Beschreibung hinzufügen](#)

Alle Tests

Package	Dauer	Fehlgeschlagen (Diff.)	Übersprungen (Diff.)	Pass (Diff.)	Summe (Diff.)
UnitTests.TestConnector	6 Sekunden	0	0	7	+7

Coverage

-> Go to the project

-> Type in the shell code as in following image:

Buildverfahren

Shell ausführen

Befehl `nosetests --with-xcoverage --with-xunit UnitTests/TestConnector.py`

[Liste der verfügbaren Umgebungsvariablen](#)

-> At Post-Build-Action type in `**/coverage.xml`

Veröffentliche die Cobertura Testabdeckung

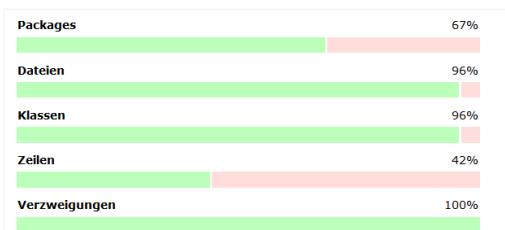
Dateien Cobertura XML Ergebnisse `**/coverage.xml`

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use `**/target/site/cobertura` multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the Cobertura must be configured to generate XML reports for this plugin to function.

Testabdeckung

Cobertura Coverage Report

Trend



Zusammenfassung der Testabdeckung nach Project

Name	Packages	Dateien	Klassen	Zeilen	Verzweigungen
Cobertura Coverage Report	67% 2/3	96% 95/99	96% 95/99	42% 3407/8039	100% 0/0

Aufschlüsselung der Testabdeckung nach Package

Name	Dateien	Klassen	Zeilen	Verzweigungen
.usr.lib.python3	0% 0/4	0% 0/4	0% 0/683	-
.usr.local.lib.python3	100% 73/73	100% 73/73	44% 2277/5118	-
<default>	100% 22/22	100% 22/22	50% 1130/2238	-


Violations


- >Go to your project's configuration
- >Add a 'Post-Build-Schritt'
- >Choose 'Report Violations'
- >In the line 'pylint' add '**/pylint.out' in the column "XML filename pattern "

pylint	10	999	999	**/pylint.out
--------	----	-----	-----	---------------




Show the sourcecode / API





- [.git](#)
- [.pyenv](#)
- [Doc](#)
- [GUI](#)
- [GUI_Testing](#)
- [MIB_Files](#)
- [Objekte](#)
- [RockTheNet](#)
- [UnitTests](#)
- [Util](#)
- [.coverage](#) 21.12 KB [anzeigen](#)
- [coverage.xml](#) 310.59 KB [anzeigen](#)
- [nosetests.xml](#) 1019 B [anzeigen](#)
- [README.md](#) 16 B [anzeigen](#)
- [setup.py](#) 293 B [anzeigen](#)
- [test.py](#) 19 B [anzeigen](#)


[\(Alle Dateien als ZIP-Archiv herunterladen\)](#)

List of references:

[1]: <http://www.alexconrad.org/2011/10/jenkins-and-python.html>, zuletzt besucht: 17.11.2014

[2]: <http://www.vogella.com/tutorials/Jenkins/article.html>, zuletzt besucht: 17.11.2014

[3]: <http://www.uvd.co.uk/blog/configuring-jenkins-continuous-integration-server-to-work-with-git/>, zuletzt besucht: 17.11.2014