Continuous integration with Jenkins CI

Vojtěch Juránek

JBoss - a division by Red Hat

17. 2. 2012, Developer conference, Brno
What continuous integration (CI) is and why it’s useful.
Show you, that CI with Jenkins is easy (Python and Ruby examples).
Show you, that CI can be even more easy.
Continuous integration

When you are developing a piece of code, you probably
- compile the sources from time to time (if the code is compiled)
- check the functionality (run tests)

If something fails (compilation, tests etc.) you start to look for a wrong commit... Is it better and more easy to try to find a mistake in one commit (several/several dozen changed lines of code) or many commits (hundreds/thousands changed lines)?
Continuous integration

When you are developing a piece of code, you probably

- compile the sources from time to time (if the code is compiled)
- check the functionality (run tests)

If something fails (compilation, tests etc.) you start to look for a wrong commit. . .

Is it better and more easy to try to find a mistake in one commit (several/several dozen changed lines of code) or many commits (hundreds/thousands changed lines)?
When you are developing a piece of code, you probably

- compile the sources from time to time (if the code is compiled)
- check the functionality (run tests)

If something fails (compilation, tests etc.) you start to look for a wrong commit.

Is it better and more easy to try to find a mistake in one commit (several/several dozen changed lines of code) or many commits (hundreds/thousands changed lines)?
Continuous integration

Yup, as small change as possible is better!

Checking the status of the project very often (after each commit, if possible) is roughly what we call **continuous integration**. CI is necessary when you use agile development methodology, but very useful even if you use any other development methodology.
Advantages

- Immediate feedback to any change.
- Immediate overview of the status of your project - test results, test coverage, performance etc. (helps with decisions, planning).
- Complete history of the project (test results, build artifacts, etc.).
- Can improve your workflow (e.g. gerrit integration).
- Can be very easily extended to continuous deployment and eventually continuous delivery.
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests…

Later on:
I should also automate

- …checkout from SVN/git …
- …and setup some post commit hook to run it only after a commit …
- …analysis of test results …
- …some notification, to get alert only when something fails …

…and I can also …
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate
  …checkout from SVN/git . . .
  …and setup some post commit hook to run it only after a commit . . .
  …analysis of test results . . .
  …some notification, to get alert only when something fails . . .
  …and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate
- …checkout from SVN/git . . .
- …and setup some post commit hook to run it only after a commit . . .
- …analysis of test results . . .
- …some notification, to get alert only when something fails . . .
…and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate

- . . . checkout from SVN/git . . .
- . . . and setup some post commit hook to run it only after a commit . . .
- . . . analysis of test results . . .
- . . . some notification, to get alert only when something fails . . .

. . . and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate

- …checkout from SVN/git . . .
- …and setup some post commit hook to run it only after a commit . . .
- …analysis of test results . . .
- …some notification, to get alert only when something fails . . .

…and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate
  … checkout from SVN/git . . .
  … and setup some post commit hook to run it only after a commit . . .
  … analysis of test results . . .
  … some notification, to get alert only when something fails . . .
  … and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate

- . . . checkout from SVN/git . . .
- . . . and setup some post commit hook to run it only after a commit . . .
- . . . analysis of test results . . .
- . . . some notification, to get alert only when something fails . . .

. . . and I can also . . .
Don’t I spend too much time by running compilation and checking the results?

No, let the computer work instead of you. Hopefully, you time is more valuable than machine time:-)

Well, I can write some bash scripts which compile the code and run the tests. . .

Later on:
I should also automate
  . . . checkout from SVN/git . . .
  . . . and setup some post commit hook to run it only after a commit . . .
  . . . analysis of test results . . .
  . . . some notification, to get alert only when something fails . . .
  . . . and I can also . . .
... wait, don’t I re-invent the wheel??

Yes, you do! There are several good CI servers. And probably the most popular is Jenkins.
Kohsuke Kawaguchi started Hudson project in 2006 while working in Sun.
Trademark and other issues after acquisition of Sun by Oracle.
Community decided to rename Hudson to Jenkins in January 2011 (Hudson is still developed by Oracle and Sonatype, now moving under Eclipse foundation).
Jenkins now affiliated with Software in the Public Interest (SPI) NPO.
All important information can be found at
https://wiki.jenkins-ci.org/display/JENKINS/Governance+Document

Jenkins usage grows steeply - more then 30k instances in anonymous usage statistics (i.e. actual # of instances is probably higher as not all instances send the stats.)
670+ repositories and 350 developer on GitHub!
Kohsuke Kawaguchi started Hudson project in 2006 while working in Sun. Trademark and other issues after acquisition of Sun by Oracle. Community decided to rename Hudson to Jenkins in January 2011 (Hudson is still developed by Oracle and Sonatype, now moving under Eclipse foundation).

Jenkins now affiliated with Software in the Public Interest (SPI) NPO

All important information can be found at https://wiki.jenkins-ci.org/display/JENKINS/Governance+Document

Jenkins usage grows steeply - more then 30k instances in anonymous usage statistics (i.e. actual # of instances is probably higher as not all instances send the stats.)

670+ repositories and 350 developer on GitHub!

![Graph showing the growth of Jenkins instances over time.](image)
Releases, packages

http://jenkins-ci.org

Release cycle:
- Released usually weekly - release early, release often.
- Long term support (LTS) release - every 3 months, every month minor release with backports of major bug fixes.

Distribution (Jenkins is a Java servlet):
- Web archive (WAR).
- Native package.
Running Jenkins

- Download war file,
  - deploy it on your favorite servlet container like JBoss AS 7 or Tomcat,
  - or just simply run `java -jar jenkins.war`.
- Install native package e.g. via `yum` and start it as a service

```
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
yum -y install jenkins
service jenkins start
```

![Jenkins UI](image)
Installing plug-ins

Plug-ins are one of the strong point of Jenkins, around 400 plug-ins, plug-ins for almost everything.

Visit Manage Jenkins -> Manage Plugins -> Available, see also https://wiki.jenkins-ci.org/display/JENKINS/Plugins

Few tips for language agnostic plugins:

- **SCM:**
  - CVS and Subversion are already pre-installed.
  - Git plugin - integrates Jenkins with git.
  - GitHub plugin - better integration with GitHub, e.g. browse changelog on GitHub etc.
  - Plugins for most of the SCMs are available.

- **Issue tracker integration:**
  - Jira plugin.
  - Bugzilla plugin.

- **Ingratiation with other tools / services:**
  - EC2 / Delta cloud plugins - using machines from cloud for builds.
  - Gerrit plugin.

- **Notifications / post-build actions - almost whatever you like :-)**
Few tips for language specific plug-ins

- Java is very well supported, plugins for all commonly used tools.
- C++:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin
- Python:
  - Python plugin
  - Shining Panda plugin
- Ruby:
  - Ruby plugin
  - Rake plugin
  - Ruby metrics plugin
- PHP:

Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Few tips for language specific plug-ins

- **Java** is very well supported, plugins for all commonly used tools.
- **C++**:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin
- **Python**:
  - Python plugin
  - Shining Panda plugin
- **Ruby**:
  - Ruby plugin
  - Rake plugin
  - Ruby metrics plugin
- **PHP**:

Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Few tips for language specific plug-ins

- **Java** is very well supported, plugins for all commonly used tools.
- **C++**:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin
- **Python**:
  - Python plugin
  - Shining Panda plugin
- **Ruby**:
  - Ruby plugin
  - Rake plugin
  - Ruby metrics plugin
- **PHP**:
- Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Java is very well supported, plugins for all commonly used tools.

- C++:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin

Python:
- Python plugin
- Shining Panda plugin

Ruby:
- Ruby plugin
- Rake plugin
- Ruby metrics plugin

PHP:

Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Few tips for language specific plug-ins

- Java is very well supported, plugins for all commonly used tools.
- C++:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin
- Python:
  - Python plugin
  - Shining Panda plugin
- Ruby:
  - Ruby plugin
  - Rake plugin
  - Ruby metrics plugin
- PHP:

Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Few tips for language specific plug-ins

- **Java** is very well supported, plugins for all commonly used tools.
- **C++**:
  - xUnit plugins - supports (besides other) UnitTest++, CppUnit, Boost Test Library
  - qmakebuilder plugin
- **Python**:
  - Python plugin
  - Shining Panda plugin
- **Ruby**:
  - Ruby plugin
  - Rake plugin
  - Ruby metrics plugin
- **PHP**:
- Far not a complete list, list above is just my personal selection of some interesting plugins! Also plugins for other languages exists.
Plug-ins roadmap

- Java plugins more mature, e.g. most Java tools plugins has auto-installer - if the tool is not installed, plugin is able to install and set it up (very useful when running in the cloud), such features in plug-ins for other languages (Python, Ruby) will hopefully follow.

- Moving Java specific features into the plug-ins.

- Some originally Java features can be used also for other languages, e.g. JUnit test results can be used for any JUnit compatible format (see examples), JavaDoc can publish arbitrary documentation etc.

- If you are missing something, you can always use shell task to execute command you want ...

... or implement missing plugin/functionality.
Support for other languages - not only plug-ins for different languages but even possibility to develop plugin in different languages! Already done for Ruby, support for other languages should follow.
Fresh Fedora 16 installation

# Install Java
yum -y install java-1.6.0-openjdk-devel

# Python setup
yum -y install python-pip python-nose python-zmq python-virtualenv
dependencies = os.path.expanduser('~')
pip-pip install nose-cov

# Install Jenkins
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
yum -y install jenkins
service jenkins start

Via Jenkins UI install Git, Cobertura and ShiningPanda plug-ins.
Setup git repository:

Source Code Management

- CVS
- Git

Repositories

Repository URL: git://github.com/ipython/ipython.git

Advanced...

Delete Repository

Add

Branches to build

Branch Specifier (blank for default): master

Delete Branch

Add

Repository browser

(Auto)

Advanced...

None

Subversion
Optionally set up repository browser if you want to browse commits directly on GitHub:
Add a build step: execute shell command (install tools, ipython and run tests) in virtualenv:

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
</table>
| pip install nose coverage readline  
| python setup.py install  
| ipptest --with-xml-coverage --with-xunit |
Setup paths to unit and coverage reports:

- Publish Cobertura Coverage Report
  - Cobertura xml report pattern: `*.coverage.xml`
    - This is a file name pattern that can be used to locate the cobertura.xml report files (for example with Maven use `**/target/site/cobertura/coverage.xml`). The path is relative to the module root unless you have configured your SCM with 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 workspace root.
    - Cobertura must be configured to generate XML reports for this plugin to function.

- Consider only stable builds
  - Include only stable builds, i.e. exclude unstable and failed ones.

- Source Encoding
  - Encoding when showing files: ASCII

- Coverage Metric Targets
  - Conditionals: 70%
  - Lines: 80%
  - Methods: 80%

- Publish JUnit test result report
  - Test report XMLs: `*.xunit.xml`
    - Fileset 'includes' setting that specifies the generated raw XML report files, such as '*myproject/target/test-reports/*.xml'. Base dir of the fileset is the workspace root.

- Retain long standard output/error
Run a build and check the results:

**Project iptyhon**

- **Coverage Report**
- **Workspace**
- **Recent Changes**
- **Latest Test Result** (no failures)

### Permalinks
- Last build (#33): 1 day 2 hr ago
- Last stable build (#35): 1 day 2 hr ago
- Last successful build (#35): 1 day 2 hr ago
- Last unstable build (#35): 3 days 22 hr ago
- Last unsuccessful build (#35): 3 days 22 hr ago

**Code Coverage**

- Classes: 84%
- Conditionals: 100%
- Files: 84%
- Lines: 52%
- Packages: 78%

**Test Result Trend**
# Unit test results

## Test Result

1 failures (±0), 14 skipped (-5)

---

## All Failed Tests

```
>>> IPython.core.tests.test_run.TestMagicRunSimple.test_aggressive_namespace_cleanup 1 ms 1
```

## All Tests

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Fail</th>
<th>(diff)</th>
<th>Skip</th>
<th>(diff)</th>
<th>Total</th>
<th>(diff)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPython</td>
<td>12 ms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>IPython.config.configurable</td>
<td>4 ms</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IPython.config.loader</td>
<td>5 ms</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IPython.config.tests.test_application</td>
<td>0.11 sec</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>IPython.config.tests.test_configurable</td>
<td>24 ms</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>IPython.config.tests.test_loader</td>
<td>74 ms</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>IPython.core</td>
<td>12 ms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>IPython.core.interactiveshell</td>
<td>18 ms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>IPython.core.magic</td>
<td>0.31 sec</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>IPython.core.inspect</td>
<td>9 ms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>IPython.core.tests</td>
<td>11 sec</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>118</td>
<td></td>
</tr>
</tbody>
</table>
...and eventually easily check failed tests:

Failed

IPython.core.tests.test_run.TestMagicRunSimple.test_aggressive_namespace_cleanup (from nose tests)

Failing for the past 1 build (Since 📄 #35 )
 Took 1 ms.

Error Message

This test is known to fail

Stacktrace

Traceback (most recent call last):
  File "/usr/lib/python2.7/unittest/case.py", line 327, in run
testMethod()
  File "/var/lib/jenkins/shiningpanda/jobs/c6b345e9/virtualenvs/d41d8cd9/lib/python2.7/site-packages/nose/case.py", line 197, in runTest
    self.test(+self.arg)
  File "/var/lib/jenkins/shiningpanda/jobs/c6b345e9/virtualenvs/d41d8cd9/lib/python2.7/site-packages/IPython/external/decorators/_decorators.py", line 220, in knownfailer
    raise KnownFailureTest, msg
KnownFailureTest: This test is known to fail
Test coverage report:

## Code Coverage

### Cobertura Coverage Report

#### Trend

![Coverage trend graph]

#### Project Coverage summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Classes</th>
<th>Conditionals</th>
<th>Files</th>
<th>Lines</th>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cobertura Coverage Report</td>
<td>89%</td>
<td>100%</td>
<td>84%</td>
<td>83%</td>
<td>78%</td>
</tr>
</tbody>
</table>

#### Coverage Breakdown by Package

<table>
<thead>
<tr>
<th>Name</th>
<th>Classes</th>
<th>Conditionals</th>
<th>Files</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python_config</td>
<td>4/4</td>
<td>N/A</td>
<td>4/4</td>
<td>57%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python config profile</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python_config tests</td>
<td>3/3</td>
<td>N/A</td>
<td>3/3</td>
<td>99%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python core</td>
<td>42/42</td>
<td>N/A</td>
<td>42/42</td>
<td>59%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python core tests</td>
<td>28/28</td>
<td>N/A</td>
<td>28/28</td>
<td>94%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python extensions</td>
<td>5/5</td>
<td>N/A</td>
<td>5/5</td>
<td>49%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python extensions tests</td>
<td>1/1</td>
<td>N/A</td>
<td>1/1</td>
<td>99%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python frontend</td>
<td>0/1</td>
<td>N/A</td>
<td>0/1</td>
<td>0%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python frontend.html</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python frontend.html notebook</td>
<td>0/6</td>
<td>N/A</td>
<td>0/6</td>
<td>0%</td>
</tr>
<tr>
<td>var/lib/jenkins/shrungpanda.jobs.os455-virtualenv.d41d8c89.lib python2.7 site-packages /Python frontend.html notebook tests</td>
<td>0/1</td>
<td>N/A</td>
<td>0/1</td>
<td>0%</td>
</tr>
</tbody>
</table>
Ruby example: Rails tutorial blog project

# Install Ruby and Rails
yum -y install ruby ruby-devel rubygems sqlite sqlite-devel js
gem install rails
gem install ci_reporter

# Create tutorial app
rails new blog
rails generate scaffold Post name:string title:string content:text

Add into Rakefile

require 'rubygems'
require 'ci/reporter/rake/test_unit'

Push your new app into git repo.
Via Jenkins UI install Ruby metrics plugin (Rake plugin will be installed as dependency).
Add build step: execute Rake tasks (db migration, test setup and test execution):

```
Invoke Rake
Rake Version (Default)
Tasks
  db:migrate
ci:setup:testunit
test
```

Specify Rake task(s) to run.
Setup path to unit test results and also you can turn on generation of Rails reports (annotations and statistics):

- Publish JUnit test result report
  - Test report XMLs: `test/reports/*.xml`
  - Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.
  - Retain long standard output/error

- Publish Rails Notes report
  - Rake Version: (Default)

- Publish Rails stats report
  - Rake Version: (Default)
Run a build and check the results:

**Project rails-blog**

- Annotations report
- Rails stats report
- Recent Changes

**Permalinks**
- Last build (#5, 1 hr 46 min ago)
- Last stable build (#5, 1 hr 46 min ago)
- Last successful build (#5, 1 hr 46 min ago)
Unit tests results and Rails annotations report

Test Result

0 failures (±0)

All Tests

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Fail (diff)</th>
<th>Skip (diff)</th>
<th>Total</th>
<th>(diff)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>0.17 sec</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Annotations (Rails notes) report

Output

```bash
workspace -s rails --silent_notes
app/controllers/posts_controller.rb:
  * [ 2] [TODO] improve this class
```
Rails statistics report

Rails stats report

<table>
<thead>
<tr>
<th>Ratios</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>LOC/M</td>
</tr>
<tr>
<td>9</td>
<td>Lines/LOC</td>
</tr>
<tr>
<td>8</td>
<td>M/C</td>
</tr>
<tr>
<td>7</td>
<td>Test/Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllers</td>
<td>88</td>
</tr>
<tr>
<td>Helpers</td>
<td>4</td>
</tr>
<tr>
<td>Models</td>
<td>2</td>
</tr>
<tr>
<td>Libraries</td>
<td>0</td>
</tr>
<tr>
<td>Integration tests</td>
<td>0</td>
</tr>
<tr>
<td>Functional tests</td>
<td>49</td>
</tr>
<tr>
<td>Unit tests</td>
<td>11</td>
</tr>
<tr>
<td>Total</td>
<td>154</td>
</tr>
</tbody>
</table>

Code LOC: 67  Test LOC: 45  Code to Test Ratio: 10.7
● Only basic examples.

● Real use cases can be much more sophisticated. Jenkins has many advanced interesting features which allows you to create very robust builds.

● Check Jenkins: The Definitive Guide
  http://www.wakaleo.com/books/jenkins-the-definitive-guide
Still not persuaded?

- Don’t want to setup a build machine for Jenkins server/builds (or slow down your dev machine)?
- Still think it’s too difficult to setup CI?
- Just lazy to do the setup?
- Java hater (*Java never ever on my server*)?
- *Any other reason*?
… then consider some cloud PaaS (platform as a service) offering, e.g. OpenShift by Red Hat!
https://openshift.redhat.com

Still some initial set up (like registration) needs to be done, but you needn’t
- find suitable machine for Jenkins server
- set up Jenkins server and build environment
- maintain CI environment

Support for
- Java
- PHP
- Python
- Ruby
- Perl

... and all this is **for free!** (in case of OpenShift Express)
Setup OpenShift app

Install OpenShift tools:

```
wget https://openshift.redhat.com/app/repo/openshift.repo
    -O /etc/yum.repos.d/openshift.repo
yum -y install rhc
```

If you don’t use `yum`, check OpenShift tutorial how to install it using `gem`.

Register your domain:

```
rhc-create-domain -n mydomain -l rhlogin
```

Create your app with Jenkins CI enabled:

- `rhc-create-app -a jbosstest -t jbossas-7.0 --enable-jenkins ci`
- `rhc-create-app -a rubytest -t ruby-1.8 --enable-jenkins ci`
- `rhc-create-app -a pythontest -t python-2.6 --enable-jenkins ci`

Add existing app under Jenkins CI by

```
rhc-ctl-app -a myapp -e add-jenkins-client-1.4
```
What OpenShift does

OpenShift will
- create git repo for your application,
- create Jenkins instance it already doesn’t exists,
- create CI job for your app and do some basic setup,
- after each commit run CI build,
- deploy your application.

You have admin login for your Jenkins instance so you can install arbitrary plugin and modify jobs as in examples above.

See also video tutorial on https://openshift.redhat.com/app/express.
Some OpenShift example apps: https://github.com/openshift.
After each commit OpenShift starts CI builds

vjuranek@localhost rubytest\$ git push
Enter passphrase for key ‘/home/vjuranek/.ssh/libra_id_rsa’:
Counting objects: 133, done.
Delta compression using up to 2 threads.
Compressing objects: 100\% (89/89), done.
Writing objects: 100\% (124/124), 90.96 KiB, done.
Total 124 (delta 20), reused 120 (delta 18)
remote: Executing Jenkins build.
remote:
remote: You can track your build at http://ci-vjuranek.rhcloud.com/job/rubytest-build
remote:
remote: Waiting for build to schedule....Done
remote: Waiting for job to complete.......Done
remote: SUCCESS
remote: New build has been deployed.
To ssh://fef2974bd2e1403f92f873358cf360c5@rubytest2-vjuranek.rhcloud.com/~git/rubytest2.git/
  8e4a040..3fb2bd3  master -> master
Other PaaS offerings providing Jenkins

- **CloudBees**
  - For FOSS projects free 2,000 minutes/month of m1.small and 500/month minutes of m1.large build/test capacity
  - Examples (FOSS projects):

- **Shining Panda**
  - [https://www.shiningpanda.com](https://www.shiningpanda.com)
  - For FOSS projects free 1 hour/day
  - For more details check [https://www.shiningpanda.com/pricing](https://www.shiningpanda.com/pricing)
  - Examples (FOSS projects):
    - [https://www.shiningpanda.com/public](https://www.shiningpanda.com/public)
Other (possibly) interesting stuff

- Mobile Jenkins
  http://www.jenkins-ci.mobi/

- Integration with Eclipse
  http://www.cloudbees.com/eclipse-plugin.cb
  http://tasktop.com/connectors/hudson-jenkins

- KDE tray app
  https://gitorious.org/fargies-misc-tools/jenkins-tray

- Python API
  http://packages.python.org/python-jenkins

- Ruby API
  https://github.com/cowboyd/jenkins.rb
Want to know more about Jenkins?

Something unclear? Doesn’t work? Need some help with setup CI for your project?
Room **B411, tomorrow 12:45-13:15**, I’ll be there for tomorrow and we can discuss your issues.

Jenkins channels:

- **Mailing users lists:**
  http://groups.google.com/group/jenkinsci-users
- **Mailing dev list:**
  http://groups.google.com/group/jenkinsci-dev
- **IRC channel:** #jenkins on http://www.freenode.net/
- **Twitter:** http://twitter.com/jenkinsci

Interested in Jenkins and also virtualization? RHEV team is searching Jenkins specials!