funkload Documentation

Release 1.15.0

Benoit Delbosc

March 10, 2011

CONTENTS

This document describes the usage of the FunkLoad tool. This tool enables to do functional and load testing of web application.

CHAPTER

ONE

INTRODUCTION

FunkLoad is a functional and load web tester, written in Python, whose main use cases are:

- Functional testing of web projects, and thus regression testing as well.
- Performance testing: by loading the web application and monitoring your servers it helps you to pinpoint bottlenecks, giving a detailed report of performance measurement.
- Load testing tool to expose bugs that do not surface in cursory testing, like volume testing or longevity testing.
- Stress testing tool to overwhelm the web application resources and test the application recoverability.
- Writing web agents by scripting any web repetitive task, like checking if a site is alive.

1.1 Features

Main FunkLoad features are:

- Functional test are pure Python scripts using the pyUnit framework like normal unit test. Python enable complex scenarios to handle real world applications.
- Truly emulates a web browser (single-threaded) using an enhanced Richard Jones' webunit:
  - get/post/put/delete support
  - post any kind of content type like `application/xml`
  - basic authentication support
  - file upload and multipart/form-data submission
  - cookies support
  - referrer support
  - https support
  - https with ssl/tls by providing a private key and certificate (PEM formatted)
  - http_proxy support
  - fetching css, javascript and images
  - emulating a browser cache
- Advanced test runner with many command-line options:
  - set the target server url
  - display the fetched page in real time in your browser
  - debug mode to display http headers
  - check performance of a single page (or set of pages) inside a test

  - green/red color mode
  - select or exclude tests cases using a regex
  - support normal pyUnit test
  - support doctest from a plain text file or embedded in python docstring
- Turn a functional test into a load test: just by invoking the bench runner you can identify scalability and performance problems. If needed the bench can distributed over a group of worker machines.
- Detailed bench reports in ReST or HTML (and PDF via ps2pdf) containing:
  - the bench configuration
  - tests, pages, requests stats and charts.
  - the requets that took the most time.
  - monitoring one or many servers cpu usage, load average, memory/swap usage and network traffic charts.
  - an http error summary list
- Differential reports to compare 2 bench reports giving a quick overview of scalability and velocity changes.
- Easy test customization using a configuration file or command line options.
- Easy test creation using embeded TCPWatch as proxy recorder, so you can use your web browser and produce a FunkLoad test automatically, including file upload or any ajax call.
- Provides web assertion helpers to check expected results in responses.
- Provides helpers to retrieve contents in responses page using DOM.
- Easy to install (EasyInstall).
- Comes with examples look at the demo folder.
- Successfully tested with dozen of differents web servers: PHP, python, Java...

1.2 License

FunkLoad is free software distributed under the GNU GPL license.

CHAPTER

TWO

SCREENSHOTS

- HTML Benchmark report example. The default report.
- PDF Benchmark report. Generated from the emacs org-mode output.
- Differential report example. A differential report compares 2 benchmark reports giving a quick overview of scalability and velocity changes.
- Trend report example. A trend report analyzes many benchmark reports to display evolution of the page statistics over time.
- You can have browse some test case examples in the demo directory or check them out if you have already installed FunkLoad:

```
$ fl-install-demo
Extract FunkLoad examples into ./funkload-demo : ...  done.
$ ls funkload-demo
cmf  README.txt  seam-booking-1.1.5  simple  xmlrpc  zope
```

CHAPTER

THREE

INSTALLATION

This document describes how to install the FunkLoad tool.

3.1 Quick installation guide for Debian and Ubuntu

With Debian Lenny or Ubuntu 8.10, 9.04, 10.04 ..., you can install the latest snapshot this way

```
sudo aptitude install python-dev python-setuptools \
     python-webunit python-docutils gnuplot
sudo aptitude install tcpwatch-httpproxy --without-recommends
sudo easy_install -f http://funkload.nuxeo.org/snapshots/ -U funkload
```

To install the latest stable release replace the last line with

```
sudo easy_install -U funkload
```

If you want to test the new beta distributed mode (since 1.14.0) you need to install paramiko and virtualenv:

```
sudo aptitude install python-paramiko, python-virtualenv
```

That's all.

Unfortunatly the FunkLoad Debian package (old 1.6.2-3) does not work properly.

You can find an old 1.10.0 Debian package here.

3.2 Other OS

Some parts are OS specific:

- the monitor server works only on Linux
- the credential server is a unix daemon but can run on windows if launched in debug mode (-d)

Under windows there is a trick to install docutils (see below), you may also rename the scripts with a `.py` extension.

3.2.1 Required packages

- libexpat1
- funkload 1.14.0 requires python >= 2.5
- python 2.4 is required by the test runner only if you want to launch python doctest, other stuff work fine with a python 2.3.5. python >= 2.5 are supported with funkload > 1.6.0

- python distutils
- python xml
- EasyInstall, either use a package or download ez_setup.py, and run it:

  ```
  cd /tmp
  wget http://peak.telecommunity.com/dist/ez_setup.py
  sudo python ez_setup.py
  ```

  This will download and install the appropriate setuptools egg for your Python version.

3.2.2 Optional packages

- To produce charts with FunkLoad >= 1.9.0: gnuplot 4.2, either use a package or visit the http://www.gnuplot.info/ site.

  gnuplot (or wgnuplot for win) must be in the executable path.
- The recorder use TCPWatch which is not yet available using easy_install:

  ```
  cd /tmp
  wget http://hathawaymix.org/Software/TCPWatch/tcpwatch-1.3.tar.gz
  tar xzvf tcpwatch-1.3.tar.gz
  cd tcpwatch
  python setup.py build
  sudo python setup.py install
  ```

3.2.3 Installation

Easy installation

- Install the latest stable package:

  ```
  sudo easy_install -U funkload
  ```

  This will install FunkLoad, webunit and docutils eggs.
- Install the latest snapshot version:

  ```
  sudo easy_install -f http://funkload.nuxeo.org/snapshots/ -U funkload
  ```

- Install the development version

  If you want to try the latest unstable sources from git

  ```
  git clone git://github.com/nuxeo/FunkLoad.git
  ```

  then:

  ```
  cd FunkLoad/
  python setup.py build
  sudo python setup.py install
  ```

Installing without sudo power

Get the FunkLoad package from pypi: http://pypi.python.org/pypi/funkload/ or a snapshot from: http://funkload.nuxeo.org/snapshots/

Then use the buildout:

```
tar zxf funkload-1.14.0.tar.gz
cd funkload-1.14.0
python bootstrap.py
bin/buildout
```

Then your executable are ready on the bin directory.

Installing without network access

Transfer all the archives from http://funkload.nuxeo.org/3dparty/ in a directory:

```
mkdir -p /tmp/fl
cd /tmp/fl
wget -r -l1 -nd http://funkload.nuxeo.org/3dparty/
```

Transfer the latest FunkLoad package

```
# get setuptools package and untar
python setup.py install

easy_install docutils*
easy_install tcpwatch*
easy_install webunit*
easy_install funkload*
```

3.3 Test it

Install the FunkLoad examples:

```
fl-install-demo
```

Go to the demo/xmlrpc folder then:

```
cd funkload-demo/xmlrpc/
make test
```

To test benching and report building just:

```
make bench
```

See `funkload-demo/README` for others examples.

3.4 Problems ?

- got a

  ```
  error: invalid Python installation: unable to open /usr/lib/python2.4/config/Makefile (No suc
  ```

  Install python2.4-dev package.

- easy_install complains about conflict:

  ```
  ...
  /usr/lib/site-python/docutils

  Note: you can attempt this installation again with EasyInstall, and use
  either the --delete-conflicting (-D) option or the
  --ignore-conflicts-at-my-risk option, to either delete the above files
  and directories, or to ignore the conflicts, respectively.  Note that if
  you ignore the conflicts, the installed package(s) may not work.
  -------------------------------------------------------------------------
  error: Installation aborted due to conflicts
  ```

  If FunkLoad, webunit or docutils were previously installed without using EasyInstall. You need to reinstall the package that raises the conflict with the `--delete-conflicting` option, see easy_install documentation.

- If you still have conflict try to remove FunkLoad from your system:

  ```
  easy_install -m funkload
  rm -rf /usr/lib/python2.3/site-packages/funkload*
  rm -rf /usr/local/funkload/
  rm /usr/local/bin/fl-*
  rm /usr/bin/fl-*
  ```

  then reinstall

- easy_install ends with:

  ```
  error: Unexpected HTML page found at
  http://prdownloads.sourceforge.net...
  ```

  Source Forge has changed their donwload page you need to update your setuptools:

  ```
  sudo easy_install -U setuptools
  ```

- Failed to install docutils 0.4 with easy_install 0.6a9 getting a:

  ```
  ...
  Best match: docutils 0.4
  Downloading http://prdownloads.sourceforge.net/docutils/docutils-0.4.tar.gz?download
  Requesting redirect to (randomly selected) 'mesh' mirror
  error: No META HTTP-EQUIV="refresh" found in Sourceforge page at http://prdownlcads.sourcefor
  ```

  It looks like sourceforge change their download page again :(

  - download manually the docutils tar gz from http://prdownloads.sourceforge.net/docutils/docutils-0.4.tar.gz?download
  - then `sudo easy_install /path/to/docutils-0.4.tar.gz`

- When testing `make test` return

  ```
  ### credentialctl: Stopping credential server.
  python: can't open file '/usr/lib/python2.4/site-packages/funkload-1.2.0-py2.4.egg/funkload/c
  ```

  Starting with FunkLoad 1.2.0 scripts are installed in /usr/bin, previously they were in /usr/local/bin, you need to remove them:

  ```
  sudo rm /usr/local/bin/fl-*
  ```

- While runing a test you got

```
Traceback (most recent call last):
File "/usr/local/bin/fl-run-test", line 8, in <module>
  load_entry_point('funkload==1.11.0', 'console_scripts', 'fl-run-test')()
File "build/bdist.linux-i686/egg/funkload/TestRunner.py", line 478, in main
File "build/bdist.linux-i686/egg/funkload/TestRunner.py", line 337, in __init__
File "build/bdist.linux-i686/egg/funkload/TestRunner.py", line 347, in loadTests
File "/usr/lib/python2.6/doctest.py", line 2412, in DocFileSuite
  suite.addTest(DocFileTest(path, **kw))
File "/usr/lib/python2.6/doctest.py", line 2331, in DocFileTest
  doc, path = _load_testfile(path, package, module_relative)
File "/usr/lib/python2.6/doctest.py", line 219, in _load_testfile
  return open(filename).read(), filename
IOError: [Errno 2] No such file or directory: 'path/to/your/testcase'
```

This means that your test file can not be loaded as a python module, (may be due to import error) FunkLoad then badly report it as an invalid doc test file.

To view the original error just run the testcase using python instead of fl-run-test (`python your_test_case.py`).

This is fixed since 1.14.

CHAPTER

FOUR

FIRST STEPS WITH FUNKLOAD

A FunkLoad test is made of a typical unittest and a configuration file. Let's look at a simple test script that is comming with the FunkLoad examples.

To get the demo examples you just need to run:

```
fl-install-demo
# Extract FunkLoad examples into ./funkload-demo : ...  done.
cd funkload-demo/simple
```

4.1 The test case

Here is an extract of the simple demo test case `test_Simple.py`:

```
import unittest
from random import random
from funkload.FunkLoadTestCase import FunkLoadTestCase

class Simple(FunkLoadTestCase):
    """This test use a configuration file Simple.conf."""
    def setUp(self):
        """Setting up test."""
        self.server_url = self.conf_get('main', 'url')

    def test_simple(self):
        # The description should be set in the configuration file
        server_url = self.server_url
        # begin of test ---------------------------------------------
        nb_time = self.conf_getInt('test_simple', 'nb_time')
        for i in range(nb_time):
            self.get(server_url, description='Get url')
        # end of test -----------------------------------------------

if __name__ in ('main', '__main__'):
    unittest.main()
```

The Simple test case extend `FunkLoadTestCase` and implement a test case named test_simple. this test case loop on a get request.

The `FunkLoadTestCase` extends the `unittest.TestCase` to add methods:

- to send HTTP request (get, post, put, delete or xmlrpc)
- to help building assertion with the response (getBody, getLastUrl, ...)
- to customize the test by accessing a configuration file (conf_getInt)
- ...

The target url, the number of requests are defined in the configuration files.

By convention the name of the configuration file is the name of the test case class with “.conf” extension in our case: `Simple.conf`.

4.2 The configuration file

It is a plain text file with sections:

```
# main section for the test case
[main]
title=Simple FunkLoad tests
description=Simply testing a default static page
url=http://localhost/index.html

# a section for each test
[test_simple]
description=Access %(nb_time)s times the main url
nb_time=20

<<snip>>
# a section to configure the test mode
[ftest]
log_to = console file
log_path = simple-test.log
result_path = simple-test.xml
sleep_time_min = 0
sleep_time_max = 0

# a section to configure the bench mode
[bench]
cycles = 50:75:100:125
duration = 10
startup_delay = 0.01
sleep_time = 0.01
cycle_time = 1
log_to =
log_path = simple-bench.log
result_path = simple-bench.xml
sleep_time_min = 0
sleep_time_max = 0.5
```

4.3 Runing the test

Check that the url present in the `main` section is reachable, then invoking `fl-run-test` will run all the tests present in the test_Simple module:

```
$ fl-run-test -dv test_Simple.py
test_simple (test_Simple.Simple) ... test_simple: Starting -----------------------------------
        Access 20 times the main url
test_simple: GET: http://localhost/index.html
        Page 1: Get url ...
test_simple:  Done in 0.006s
test_simple:  Load css and images...
test_simple:   Done in 0.002s
test_simple: GET: http://localhost/index.html
        Page 2: Get url ...
<<snip>>
```

```
       Page 20: Get url ...
test_simple:  Done in 0.000s
test_simple:  Load css and images...
test_simple:   Done in 0.000s
Ok
----------------------------------------------------------------------
Ran 1 test in 0.051s

OK
```

4.4 Runing a benchmark

To run a benchmark you invoke `fl-run-bench` instead of the test runner, you also need to select which test case to run.

The result of the bench will be saved in a single xml file `simple-bench.xml`, the name of this result file is set in the configuration file in the `bench` section.

You can override the configuration file using command line option, here we ask for 3 cycles with 1, 10 and 20 concurrents users (CUs).

```
$ fl-run-bench -c 1:10:20 test_Simple.py Simple.test_simple
========================================================================
Benching Simple.test_simple
========================================================================
Access 20 times the main url
------------------------------------------------------------------------

Configuration
=============

* Current time: 2011-01-26T23:22:51.267757
* Configuration file: /tmp/funkload-demo/simple/Simple.conf
* Log xml: /tmp/funkload-demo/simple/simple-bench.xml
* Server: http://localhost/index.html
* Cycles: [1, 10, 20]
* Cycle duration: 10s
* Sleeptime between request: from 0.0s to 0.5s
* Sleeptime between test case: 0.01s
* Startup delay between thread: 0.01s

Benching
========

* setUpBench hook: ... done.

Cycle #0 with 1 virtual users
-----------------------------

* setUpCycle hook: ... done.
* Start monitoring localhost: ... failed, server is down.
* Current time: 2011-01-26T23:22:51.279718
* Starting threads: . done.
* Logging for 10s (until 2011-01-26T23:23:01.301664): .. done.
* Waiting end of threads: . done.
* Waiting cycle sleeptime 1s: ... done.
* tearDownCycle hook: ... done.
* End of cycle, 14.96s elapsed.
* Cycle result: **SUCCESSFUL**, 2 success, 0 failure, 0 errors.
```

```
Cycle #1 with 10 virtual users
------------------------------

* setUpCycle hook: ... done.
* Current time: 2011-01-26T23:23:06.234422
* Starting threads: .......... done.
* Logging for 10s (until 2011-01-26T23:23:16.360602): .............. done.
* Waiting end of threads: .......... done.
* Waiting cycle sleeptime 1s: ... done.
* tearDownCycle hook: ... done.
* End of cycle, 16.67s elapsed.
* Cycle result: **SUCCESSFUL**, 14 success, 0 failure, 0 errors.

Cycle #2 with 20 virtual users
------------------------------

* setUpCycle hook: ... done.
* Current time: 2011-01-26T23:23:06.234422
* Starting threads: .......... done.
* Logging for 10s (until 2011-01-26T23:23:16.360602): .............. done.
* Waiting end of threads: .......... done.
* Waiting cycle sleeptime 1s: ... done.
* tearDownCycle hook: ... done.
* End of cycle, 16.67s elapsed.
* Cycle result: **SUCCESSFUL**, 14 success, 0 failure, 0 errors.

* tearDownBench hook: ... done.

Result
======

* Success: 40
* Failures: 0
* Errors: 0

Bench status: **SUCCESSFUL**
```

4.5 Generating a report

The xml result file can be turn into an html report this way:

```
$ fl-build-report --html simple-bench.xml
Creating html report: ...done:
/tmp/funkload-demo/simple/test_simple-20110126T232251/index.html
```

It should generate something like this: http://funkload.nuxeo.org/report-example/test_simple-20110126T232251/

Note that there were no monitoring in our simple benchmark.

4.6 Write your own test

The process to write a new test is the following:

- Use the recorder to initialize the test case and the configuration files and to grab requests.
- Play the test and display each response in firefox, this will help you to add assertion and check the response:

```
fl-run-test -dV test_BasicNavigation.py
```

- Implement the dynamic part:
  - For each request add an assertion to make sure the page is the one you expect. this can be done by checking if a term is present in a response:

    ```
    self.assert_('logout' in self.getBody(), "Login failure")
    ```

  - Generates random input, you can use the FunkLoad.Lipsum module:

    ```
    from FunkLoad import Lipsum
    ...
    lipsum = Lipsum()
    # Get a random title
    title = lipsum.getSubject()
    ```

  - Extracts a token from a previous response:

    ```
    from FunkLoad.utils import extract_token
    ...
    jsf_state = extract_token(self.getBody(), ' id="javax.faces.ViewState" value="', '"')
    ```

  - Uses a credential server if you want to make a bench with different users or simply don't want to hard code your login/password:

    ```
    from funkload.utils import xmlrpc_get_credential
    ...
    # get an admin user
    login, pwd = xmlrpc_get_credential(host, port, "admin")
    ```

- Configure the monitoring and automate your benchmark using a Makefile.

CHAPTER

FIVE

WRITING TEST SCRIPT

5.1 Submiting requests

- HTTP GET

  Here are some example on how to perform a get request:

  ```
  self.get(self.server_url + "/logout", description="Logout ")
  self.get(self.server_url + "/search?query=foo",
           description="Search with params in the URL")
  self.get(self.server_url + "/search", params=[['query', 'foo']],
           description="Search using params")
  ```

- HTTP POST

  Here are some example on how to perform a post request:

  ```
  from webunit.utility import Upload
  from funkload.utils import Data
  ...
  # simple post
  self.post(self.server_url + "/login",
            params=[['user_name', 'scott'],
                    ['user_password', 'tiger']],
            description="Login as scott")

  # upload a file
  self.post(self.server_url + "/uploadFile",
            params=[['file', Upload('/tmp/foo.pdf'),
                    ['title', 'foo file']],
            description="Upload a file")

  # post with text/xml content type
  self.post(self.server_url + "/xmlAPI",
            params=Data('text/xml', '<foo>bar</foo>'),
            description="Call xml API")
  ```

- HTTP PUT/DELETE:

  ```
  from funkload.utils import Data
  ...
  self.put(self.server_url + '/xmlAPI",
           Data('text/xml', '<foo>bar</foo>',
           description="Put query")
  self.delete(self.server_url + '/some/rest/path/object',
              description="Delete object')
  ```

- xmlrc helper:

```
    ret = self.xmlrpc(server_url, 'getStatus',
                      description="Check getStatus")
```

You should set a description when submiting a request, this improves the readability of the report.

If you run your test in debug mode you can see what is being send, the debug mode is activated with the `--debug --debug-leve=3` options.

By running your test with the `-V` option you will see each response in your running instance of firefox.

5.2 Adding assertion

After each request you should add an assertion to make sure you are on the expected page.

You can check the response content using `self.getBody()`

```
self.get(server_url, description="home page")
self.assert_('Welcome' in self.getBody(), "Wrong home page")
```

You can check an expected HTTP return code:

```
ret = self.get(...)
self.assert_(ret.code in [200, 301], "expecting a 200 or 301")
```

Note that FunkLoad is testing the HTTP return code by default, assuming that a different code thant 200:301:302 is an error, this can be changed using the ok_codes parameters or config file option:

```
ret = self.get(self.server_url + '/404.hmtl', ok_codes=[200, 404],
           description="Accept 404")
self.assert_(ret.code == 404)
```

You can check the returned URL wich may be different if you have been redirected:

```
self.post(self.server_url + "/login",
          params=[['user_name', 'scott'],
                  ['user_password', 'tiger']],
          description="Login as scott")
self.assert_('dashboard' in self.getLastURL(), "Login failure")
```

5.3 Basic Authentication

```
self.setBasicAuth('scott', 'tiger')
self.get(self.server_url, description="Get using basic auth")
# remove basic auth
self.clearBasicAuth()
```

5.4 Extra headers

```
self.setHeader('Accept-Language', 'de')
# this header is set for all the next requests
...
# Remove all additional headers
self.clearHeaders()
```

5.5 Extracting information

At some point you will need to extract information from the response. When possible the best way to do it is using string find or regex. Parsing XML or HTML has such an extra cost that it will prevent you to submit hight load.

FunkLoad comes with a simple extract_token working with string finds:

```
from FunkLoad.utils import extract_token
...
token = extract_token(self.getBody(), 'id="mytoken" value="', '"')
```

Of course for pure functional testing you can use FunkLoad helpers:

```
ret = self.get(self.server_url, description="Get some page")
urls = self.listHref(url_pattern="view_document",
                     content_pattern="View")
base_url = self.getLastBaseUrl()
```

Or the WebUnit minidom:

```
title = self.getDom().getByName('title')[0].getContents()
```

Or any python XML/HTML processing library including beautiful soup.

5.6 Using the configuration file

You can get information from the configuration file, using the approriate `self.conf_get*(section, key)` methods:

```
# Getting value from the main section
value = self.get_conf('main', 'key', 'default')
count = self.get_confInt('main', 'nb_docs', 10)
percent = self.get_confFloat('main', 'percent', 5.5)
items = self.get_confList('main', 'names')
# The names in the conf file are separated with a semi column
# names=name1:name2:name3
```

5.7 Sharing credentials

If you need to share credentials among your tests you can use the FunkLoad credential server. Here is an example to request credentials:

```
from funkload.utils import xmlrpc_get_credential
...
# get the credential host and port from the config file
credential_host = self.conf_get('credential', 'host')
credential_port = self.conf_getInt('credential', 'port')
# get a login/pwd from the members group
login, password = xmlrpc_get_credential(credential_host,
                                        credential_port,
                                        'members')
```

Since FunkLoad 1.15 the credential server can return a sequence:

```
from funkload.utils import xmlrpc_get_seq
...
seq = xmlrpc_get_seq()
```

The sequence starts with 0 but can be initialized in the credential server configuration file.

5.8 Generating data

FunkLoad comes with a simple text random generator a Lipsum like:

```
>>> from funkload.Lipsum import Lipsum
>>> print 'Word: %s\n' % (Lipsum().getWord())
Word: albus

>>> print 'UniqWord: %s\n' % (Lipsum().getUniqWord())
UniqWord: fs3ywpxg

>>> print 'Subject: %s\n' % (Lipsum().getSubject())
Subject: Fulvus orientalis albus hortensis dorsum

>>> print 'Subject uniq: %s\n' % (Lipsum().getSubject(uniq=True))
Subject uniq: F26v3y fuscus variegatus dolicho caulos cephalus

>>> print 'Sentence: %s\n' % (Lipsum().getSentence())
Sentence: Argentatus arvensis diplo familiaris tetra trich ; vulgaris montanus folius tetra so ecl

>>> print 'Paragraph: %s\n' % (Lipsum().getParagraph())
Paragraph: Sit pteron, tetra dermis viridis cyanos. Tetra novaehollandiae cyanos indicus major ort

>>> print 'Message: %s\n' % (Lipsum().getMessage())
Message: Familiaris fulvus flora xanthos tomentosus lutea lineatus ?, dolicho campus maculatus ad

Dermis zygos, ventrus oeos glycis dulcis chloreus verrucosus lineatus, pteron sinensis officinali

Variegatus deca fuscus petra rubra biscortborealis familiaris sativus leucus xanthos phyton argent

>>> print 'Phone number: %s\n' % Lipsum().getPhoneNumber()
Phone number: 07 20 25 56 06

>>> print 'Phone number fr short: %s\n' % Lipsum().getPhoneNumber(
...     lang="fr", format="short")
Phone number fr short: 0787117995

>>> print 'Phone number fr medium: %s\n' % Lipsum().getPhoneNumber(
...     lang="fr", format="medium")
Phone number fr medium: 07 88 31 30 06

>>> print 'Phone number fr long: %s\n' % Lipsum().getPhoneNumber(
...     lang="fr", format="long")
Phone number fr long: +33 (0)7 41 08 36 56

>>> print 'Phone number en_US short: %s\n' % Lipsum().getPhoneNumber(
...     lang="en_US", format="short")
Phone number en_US short: 863-3655

>>> print 'Phone number en_US medium: %s\n' % Lipsum().getPhoneNumber(
...     lang="en_US", format="medium")
Phone number en_US medium: (327) 129-2863

>>> print 'Phone number en_US long: %s\n' % Lipsum().getPhoneNumber(
...     lang="en_US", format="long")
Phone number en_US long: +00 1 (283) 158-7134

>>> print 'Address default: %s' % Lipsum().getAddress()
Address default: 85 place Brevis
99612 Trich
```

5.9 Adding information to the report

- At runtime a bench can add metadata to the report using the setUpBench hook and the addMetadata method:

  ```
  def setUpBench(self):
     ret = self.get(self.server_url + "/getVersion",
                    description="Get the server version")
     self.addMetadata('Application version', ret.getBody())
  ```

- At runtime from the command line using the `--label` option of the bench runner.
- After the bench using a file named `funkload.metadata` with a list of `key:value`. At the moment this file is only used by the trend reports to add charts label and bench description. This file must be put on the report directory:

  ```
  label: label used by trend report
  build: 666
  builtOn: hostname
  Text taken as description `using ReST power <http://url/>`__
  Can be multine text.
  ```

5.10 API

More info on the API doc: FunkLoadTestCase.

CHAPTER

SIX

BENCHMARKS CONCEPTS

The same FunkLaod test can be turned into a load test, just by invoking the bench runner `fl-run-bench`.

6.1 Page

A page is an http get/post request with associated sub requests like redirects, images or links (css, js files). This is what users see as a single page.

Note that an xmlrpc call or a put/delete is also taken in account as a page.

6.2 Test

A test is made with 3 methods: setUp/test_name/tearDown. During the test_name method each get/post request is called a page.

```
[setUp][page 1]    [page 2] ... [page n]   [tearDown]
======================================================> time
       <----------------------------------> test method
               <--> sleeptime_min to sleeptime_max
       <-----> page 1 connection time
```

6.3 Cycle

A cycle is a load of n concurrents test during a 'duration' period. Threads are launched every 'startupdelay' seconds, each thread executes test in a loop.

Once all threads have been started we start to record stats.

Only tests that end during the 'duration' period are taken into account for the test stats (in the representation below test like [—X are not take into account).

Only pages and requests that finish during the 'duration' are taken into account for the request and pages statistic

Before a cycle a setUpCycle method is called, after a cycle a tearDownCycle method is called, you can use these methods to test differents server configuration for each cycle.

```
Threads
^
|
|
|n                     [---test--]   [--------]   [--|---X
|...
|                      |                              |
|2            [------|--]   [--------]   [-------] |
|                    |                             |
|1          [------X | [--------]   [-------]   [--|--X
|                    |                             |
|[setUpCycle]        |                             |    [tearDownCycle]
===========================================================> time
                     <------ cycle duration ------->
 <----- staging ----->                             <---- staging ----->
            <-> startupdelay    <---> sleeptime
```

6.4 Cycles

FunkLoad can execute many cycles with different number of CUs (Concurrent Users), this way you can find easily the maximum number of users that your application can handle.

Running n cycles with the same CUs is a good way to see how the application handles a writing test over time.

Running n cycles with the same CUs with a reading test and a setUpCycle that change the application configuration will help you to find the right tuning.

```
cvus = [n1, n2, ...]

Threads
^
|
|
|n2                            __________
|                             /          \
|                            /            \
|n1   _________             /              \
|    /         \           /                \
|   /           \         /                  \
|  /             \       /                    \
 ==================================================> time
      <------->   duration     <-------->
                  <-----> cycle sleep time
```

6.5 Tips

Here are few remarks/advices to obtain workable metrics.

- Since it may use significant CPU resources, make sure that performance limits are not hit by FunkLoad before your server's limit is reached. Check this by launching a bench from another host.
- Having a cycle with one user gives a usefull reference.
- Run multiple cycles for a bench.
- A bench is composed of a benching test (or scenario) run many times. A good benching test should not be too long so you have a higher testing rate (that is, more benching tests can come to their end).
- The cycle duration for the benching test should be long enough. Around 5 times the duration of a single benching test is a value that is usually a safe bet. You can obtain this duration of a single benching test by running `fl-run-test myfile.py MyTestCase.testSomething`.

  Rationale : Normally a cycle duration of a single benching test should be enough. But from the testing platform side if there are more than one concurrent user, there are many threads to start and it takes some time. And on from the tested platform side it is common that a benching test will last longer and longer as the server is used by more and more users.

- You should use many cycles with the same step interval to produce readable charts (1:10:20:30:40:50:60 vs 1:10:100)
- A benching test must have the same number of page and in the same order.
- Use a Makefile to make reproductible bench.
- There is no debug option while doing a bench (since this would be illegible with all the threads). So, if a bench fails (that is using *fl-run-bench*), use `fl-run-test -d` to debug.

CHAPTER

SEVEN

RECORDING A TEST

You can use `fl-record` to record your navigator activity, this requires the TCPWatch python proxy see installation for information on how to install TCPWatch.

1. Start the recorder:

   ```
   fl-record basic_navigation
   ```

   This will output something like this:

   ```
   Hit Ctrl-C to stop recording.
   HTTP proxy listening on :8090
   Recording to directory /tmp/tmpaYDky9_funkload.
   ```

1. Setup your browser proxy and play your scenario

- in Firefox: Edit > Preferencies > General; Connection Settings set *localhost:8090* as your HTTP proxy
- Play your scenario using your navigator
- Hit Ctrl-C to stop recording:

  ```
  ^C
  # Saving uploaded file: foo.png
  # Saving uploaded file: bar.pdf
  Creating script: ./test_BasicNavigation.py.
  Creating configuration file: ./BasicNavigation.conf.
  ```

  You now have a new module test_BaiscNavigation file with its configuration file. Ready to be tested, refer to the tutorial to learn how to turn it into a workable test.

To add more requests to your test, just use `fl-record` without parameters, performs your requests on the browser then hit Ctrl-C. `fl-record` will output the code ready to be paste in your test case.

```
$ fl-record
HTTP proxy listening on :8090
Recording to directory /tmp/tmpt0l7jh_funkload.
^C
TCPWatch finished.
      self.post(server_url + "/booking/register.seam", params=[
        ['registration', 'registration'],
        ['registration:usernameDecorate:username', 'scott'],
        ['registration:nameDecorate:name', 'scott'],
        ['registration:passwordDecorate:password', 'tiger'],
        ['registration:verifyDecorate:verify', 'tiger'],
        ['registration:register', 'Register'],
        ['javax.faces.ViewState', '_id6407']],
        description="Post /booking/register.seam")
$
```

Note that `fl-record` :

- works fine with multi-part encoded form and file upload.
- handles automaticly JSF Myfaces token, which enable to easily record and play any JBoss Seam application.
- doesn't support HTTPS, the work around is to first record a scenario on HTTP, and then change the *url* back to *https* in the configuration file.

CHAPTER

EIGHT

THE CREDENTIAL SERVER

If you are writing a bench that requires to be logged with different users FunkLoad provides an xmlrpc credential server to serve login/pwd between the different threads.

It requires 2 files with the same pattern as unix /etc/passwd and /etc/group. The password file have the following format:

```
login1:pwd1
login2:pwd2
...
```

The group file format is:

```
group1:user1, user2
group2:user2
# you can split group declaration
group1:user3
...
```

The credential server use a configuration file to point to the user and group file and define the listening port:

```
[server]
host=localhost
port=22207
credentials_path=passwords.txt
groups_path=groups.txt
# to loop over the entire file set the value to 0
loop_on_first_credentials=0
```

To start the credential server:

```
fl-credential-ctl credential.conf start
```

You can find more option in the usage page.

In your test case to get a credential:

```
from funkload.utils import xmlrpc_get_credential
...
login, pwd = xmlrpc_get_credential(host, port, "group2")
```

CHAPTER

NINE

THE MONITOR SERVER

If you want to monitor a linux server health during the bench, you have to run a monitor xmlrpc server on the target server, this requires to install a minimal FunkLoad package, on Debian/Ubunutu

```
sudo aptitude install python-dev python-setuptools \
   python-webunit python-docutils
sudo easy_install -f http://funkload.nuxeo.org/snapshots/ -U funkload
```

Then create a configuration file `monitor.conf`:

```
[server]
host = <IP or FQDN>
port = 8008
interval = .5
interface = eth0
[client]
host = <IP or FQDN>
port = 8008
```

And start the monitor server:

```
fl-monitor-ctl monitor.conf start
```

On the bench server add to your test configuration file the following section:

```
[monitor]
hosts = <IP or FQDN>

[<IP or FQDN>]
description = The application server
port = 8008
```

Then run the bench, the report will include server stats.

Note that you can monitor multiple hosts and that the monitor is linux specific.

A new contribution will be added in 1.16 to extend the monitoring, it will enable to use Nagios or Munin plugins.

CHAPTER

TEN

FUNKLOAD COMMAND LINE USAGE

All the FunkLoad command starts with `fl-` and have a `--help` options.

10.1 Recorder `fl-record` usage

fl-record [options] [test_name]

fl-record launch a TCPWatch proxy and record activities, then output a FunkLoad script or generates a FunkLoad unit test if test_name is specified.

The default proxy port is 8090.

Note that tcpwatch.py executable must be accessible from your env.

See http://funkload.nuxeo.org/ for more information.

10.1.1 Examples

fl-record foo_bar Run a proxy and create a FunkLoad test case, generates test_FooBar.py and FooBar.conf file. To test it: fl-run-test -dV test_FooBar.py

fl-record -p 9090 Run a proxy on port 9090, output script to stdout.

fl-record -i /tmp/tcpwatch Convert a tcpwatch capture into a script.

10.1.2 Options

--version	show program's version number and exit
--help, -h	show this help message and exit
--verbose, -v	Verbose output
--port=PORT, -p PORT	The proxy port.
--tcp-watch-input=TCPWATCH_PATH, -i TCPWATCH_PATH	Path to an existing tcpwatch capture.
--loop=LOOP, -l LOOP	Loop mode.

10.2 Test runner `fl-run-test` usage

fl-run-test [options] file [class.method|class|suite] [...]

fl-run-test launch a FunkLoad unit test.

A FunkLoad unittest use a configuration file named [class].conf, this configuration is overriden by the command line options.

See http://funkload.nuxeo.org/ for more information.

10.2.1 Examples

fl-run-test myFile.py Run all tests.

fl-run-test myFile.py test_suite Run suite named test_suite.

fl-run-test myFile.py MyTestCase.testSomething Run a single test MyTestCase.testSomething.

fl-run-test myFile.py MyTestCase Run all 'test*' test methods in MyTestCase.

fl-run-test myFile.py MyTestCase -u http://localhost Same against localhost.

fl-run-test –doctest myDocTest.txt Run doctest from plain text file (requires python2.4).

fl-run-test –doctest -d myDocTest.txt Run doctest with debug output (requires python2.4).

fl-run-test myfile.py -V Run default set of tests and view in real time each page fetch with firefox.

fl-run-test myfile.py MyTestCase.testSomething -l 3 -n 100 Run MyTestCase.testSomething, reload one hundred time the page 3 without concurrency and as fast as possible. Output response time stats. You can loop on many pages using slice -l 2:4.

fl-run-test myFile.py -e [Ss]ome Run all tests that match the regex [Ss]ome.

fl-run-test myFile.py -e '!xmlrpc$' Run all tests that does not ends with xmlrpc.

fl-run-test myFile.py –list List all the test names.

fl-run-test -h More options.

10.2.2 Options

--version show program's version number and exit

--help, -h show this help message and exit

--quiet, -q Minimal output.

--verbose, -v Verbose output.

--debug, -d FunkLoad and doctest debug output.

--debug-level=DEBUG_LEVEL Debug level 3 is more verbose.

--url=MAIN_URL, -u MAIN_URL Base URL to bench without ending '/'.

--sleep-time-min=FTEST_SLEEP_TIME_MIN, -m FTEST_SLEEP_TIME_MIN Minumum sleep time between request.

--sleep-time-max=FTEST_SLEEP_TIME_MAX, -M FTEST_SLEEP_TIME_MAX Maximum sleep time between request.

--dump-directory=DUMP_DIR Directory to dump html pages.

--firefox-view, -V Real time view using firefox, you must have a running instance of firefox in the same host.

--no-color Monochrome output.

--loop-on-pages=LOOP_STEPS, -l LOOP_STEPS Loop as fast as possible without concurrency on pages, expect a page number or a slice like 3:5. Output some statistics.

--loop-number=LOOP_NUMBER, -n LOOP_NUMBER Number of loop.

--accept-invalid-links Do not fail if css/image links are not reachable.

--simple-fetch Don't load additional links like css or images when fetching an html page.

--stop-on-fail Stop tests on first failure or error.

--regex=REGEX, -e REGEX The test names must match the regex.

--list Just list the test names.

--doctest Check for a doc test.

--pause Pause between request, press ENTER to continue.

10.3 Bench runner `fl-run-bench` usage

fl-run-bench [options] file class.method

fl-run-bench launch a FunkLoad unit test as load test.

A FunkLoad unittest use a configuration file named [class].conf, this configuration is overriden by the command line options.

See http://funkload.nuxeo.org/ for more information.

10.3.1 Examples

fl-run-bench myFile.py MyTestCase.testSomething Bench MyTestCase.testSomething using MyTestCase.conf.

fl-run-bench -u http://localhost:8080 -c 10:20 -D 30 myFile.py

MyTestCase.testSomething Bench MyTestCase.testSomething on localhost:8080 with 2 cycles of 10 and 20 users during 30s.

fl-run-bench -h More options.

10.3.2 Options

--version show program's version number and exit

--help, -h show this help message and exit

--url=MAIN_URL, -u MAIN_URL Base URL to bench.

--cycles=BENCH_CYCLES, -c BENCH_CYCLES Cycles to bench, this is a list of number of virtual concurrent users, to run a bench with 3 cycles with 5, 10 and 20 users use: -c 2:10:20

--duration=BENCH_DURATION, -D BENCH_DURATION Duration of a cycle in seconds.

--sleep-time-min=BENCH_SLEEP_TIME_MIN, -m BENCH_SLEEP_TIME_MIN Minimum sleep time between requests.

--sleep-time-max=BENCH_SLEEP_TIME_MAX, -M BENCH_SLEEP_TIME_MAX Maximum sleep time between requests.

--test-sleep-time=BENCH_SLEEP_TIME, -t BENCH_SLEEP_TIME Sleep time between tests.

--startup-delay=BENCH_STARTUP_DELAY, -s BENCH_STARTUP_DELAY Startup delay between thread.

--as-fast-as-possible, -f Remove sleep times between requests and between tests, shortcut for -m0 -M0 -t0

--no-color Monochrome output.

--accept-invalid-links Do not fail if css/image links are not reachable.

--simple-fetch Don't load additional links like css or images when fetching an html page.

--label=LABEL, -l LABEL Add a label to this bench run for easier identification (it will be appended to the directory name for reports generated from it).

--enable-debug-server Instantiates a debug HTTP server which exposes an interface using which parameters can be modified at run-time. Currently supported parameters: /cvu?inc=<integer> to increase the number of CVUs, /cvu?dec=<integer> to decrease the number of CVUs, /getcvu returns number of CVUs

--debug-server-port=DEBUGPORT Port at which debug server should run during the test

--distribute distributes the CVUs over a group of worker machines that are defined in the section [workers]

--distribute-workers=WORKERLIST this parameter will over-ride the list of workers defined in the config file. expected notation is uname@host,uname:pwd@host or just host...

--is-distributed this parameter is for internal use only. it signals to a worker node that it is in distributed mode and shouldn't perform certain actions.

10.4 Report builder `fl-build-report` usage

fl-build-report [options] xmlfile [xmlfile...]

or

fl-build-report –diff REPORT_PATH1 REPORT_PATH2

fl-build-report analyze a FunkLoad bench xml result file and output a report. If there are more than one file the xml results are merged.

See http://funkload.nuxeo.org/ for more information.

10.4.1 Examples

fl-build-report funkload.xml ReST rendering into stdout.

fl-build-report –html -o /tmp funkload.xml Build an HTML report in /tmp

fl-build-report –html node1.xml node2.xml node3.xml Build an HTML report merging test result from 3 nodes.

fl-build-report –diff /tmp/test_reader-20080101 /tmp/test_reader-20080102 Build a differential report to compare 2 bench reports, requires gnuplot.

fl-build-report -h More options.

10.4.2 Options

--version show program's version number and exit

--help, -h show this help message and exit

--html, -H Produce an html report.

--with-percentiles, -P Include percentiles in tables, use 10%, 50% and 90% for charts, default option.

--no-percentiles No percentiles in tables display min, avg and max in charts (gdchart only).

--diff, -d	Create differential report.
--output-directory=OUTPUT_DIR, -o OUTPUT_DIR	Parent directory to store reports, the directoryname of the report will be generated automatically.
--report-directory=REPORT_DIR, -r REPORT_DIR	Directory name to store the report.
--apdex-T=APDEX_T, -T APDEX_T	Apdex T constant in second, default is set to 1.5s. Visit http://www.apdex.org/ for more information.

10.5 Credential server `fl-credential-ctl` Usage

fl-credential-ctl config_file action

action can be: start|startd|stop|restart|status|test

Execute action on the XML/RPC server.

10.5.1 Options

--version	show program's version number and exit
--help, -h	show this help message and exit
--quiet, -q	Verbose output

10.6 Monitor server `fl-monitor-ctl` usage

fl-monitor-ctl config_file action

action can be: start|startd|stop|restart|status|test

Execute action on the XML/RPC server.

10.6.1 Options

--version	show program's version number and exit
--help, -h	show this help message and exit
--quiet, -q	Verbose output

CHAPTER

ELEVEN

FAQ

11.1 What does all these dots mean ?

During a bench cycle the “Starting threads” dots means the number running threads:

```
Cycle #1 with 10 virtual users
------------------------------

* setUpCycle hook: ... done.
* Current time: 2011-01-26T23:23:06.234422
* Starting threads: ........
```

During the cycle logging the green dots means a successful test while the red ‘F’ are for test failure:

```
* Logging for 10s (until 2011-01-26T23:23:16.360602): ......F......
```

During the stagging down the dots are the number of stopped threads:

```
* Waiting end of threads: .........
```

11.2 How to accept invalid Cookies ?

- `Error :  COOKIE ERROR: Cookie domain "<DOMAINE>" doesn't start with "."`

  Comment the lines in file /usr/lib/python2.6/site-packages/webunit-1.3.8-py2.6.egg/webunit/cookie.py:

  ```
  #if domain[0] != '.':
  #   raise Error, 'Cookie domain "%s" doesn\'t start with "."' % domain
  ```

- `Error :  COOKIE ERROR: Cookie domain "."<DOMAINE>" doesn't match request host "<DOMAINE>"`

  Comment the lines in the file /usr/lib/python2.6/site-packages/webunit-1.3.8-py2.6.egg/webunit/cookie.py:

  ```
  #if not server.endswith(domain):
  #   raise Error, 'Cookie domain "%s" doesn\'t match '
  #   'request host "%s"'%(domain, server)
  ```

11.3 How to submit high load ?

High load works fine for IO Bound test, not on CPU bound test. The test script must be light:

- When possible don't parse html/xml page, using simple find or regexp are much much faster than any html parsing including getDOM, html parser or beautifulsoup. If you start emulating a browser then you will be as slow as a browser.
- Always use `--simple-fetch` option to prevent parsing html page to retrieve resources use explicit GET in your code.
- Try to generate or prepare the data before the test to minimize the processing during the test.

On 32b OS install psyco, it gives a 50% boost (`aptitude install python-psyco` on Debia/Ubuntu OS).

On multi CPU server, GIL is getting infamous, to get all the power you need to use CPU affinity `taskset -c 0 fl-run-bench` is always faster than `fl-run-bench`. Using one bench runner process per CPU is a work around to use the full server power.

Use multiple machine to perform the load, see the next section.

11.4 How to run multiple bencher ?

Bench result file can be merged by the `fl-build-report` command, but how to run multiple bencher ?

There are many ways:

- Use the new distribute mode (still in beta), it requires paramiko and virtualenv:

  ```
  sudo aptitude install python-paramiko, python-virtualenv
  ```

  It adds 2 new command line options:

  - `--distribute`: to enable distributed mode
  - `--distribute-workers=uname@host,uname:pwd@host...`: user:password can be skipped if using pub-key.

  For instance to use 2 workers you can do something like this:

  ```
  $ fl-run-bench -c 1:2:3 -D 5 -f --simple-fetch  test_Simple.py Simple.test_simple --distribut
  ========================================================================
  Benching Simple.test_simple
  ========================================================================
  Access 20 times the main url
  ------------------------------------------------------------------------

  Configuration
  =============

  * Current time: 2011-02-13T23:15:15.174148
  * Configuration file: /tmp/funkload-demo/simple/Simple.conf
  * Distributed output: log-distributed
  * Server: http://node0/
  * Cycles: [1, 2, 3]
  * Cycle duration: 5s
  * Sleeptime between request: from 0.0s to 0.0s
  * Sleeptime between test case: 0.0s
  * Startup delay between thread: 0.01s
  * Workers :octopussy,simplet

  * Preparing sandboxes for 2 workers.....
  * Starting 2 workers..
  ```

```
* [node1] returned
* [node2] returned
* Received bench log from [node1] into log-distributed/node1-simple-bench.xml
* Received bench log from [node2] into log-distributed/node2-simple-bench.xml

# Now building the report
$ fl-build-report --html log-distributed/node1-simple-bench.xml  log-distributed/node2-simple
Merging results files: ..
nodes: node1, node2
cycles for a node:    [1, 2, 3]
cycles for all nodes: [2, 4, 6]
Results merged in tmp file: /tmp/fl-mrg-o0MI8L.xml
Creating html report: ...done:
/tmp/funkload-demo/simple/test_simple-20110213T231543/index.html
```

Note that the version of FunkLoad installed on nodes is defined in the configuration file:

```
[distribute]
log_path = log-distributed
funkload_location=http://pypi.python.org/packages/source/f/funkload/funkload-1.14.0.tar.gz
```

- Using BenchMaster http://pypi.python.org/pypi/benchmaster
- Using Fabric http://tarekziade.wordpress.com/2010/12/09/funkload-fabric-quick-and-dirty-distributed-load-system/
- Old school pssh/Makefile:

```
# clean all node workspaces
parallel-ssh -h hosts.txt rm -rf /tmp/ftests/
# distribute tests
parallel-scp -h hosts.txt -r ftests /tmp/ftests
# launch a bench
parallel-ssh -h hosts.txt -t -1 -o bench "(cd /tmp/ftests&& make bench URL=http://target/)"
# get the results
parallel-slurp -h hosts.txt -o out -L results-date -u '+%Y%m%d-%H%M%S' -r /tmp/ftests/report
# build the report with fl-build-report, it supports the results merging
```

11.5 How to mix different scenarii in a bench ?

Simple example with percent of users:

```python
import random
...
def testMixin(self):
    if random.randint(1, 100) < 30:
        # 30% writer
        return self.testWriter()
    else:
        # 70% reader
        return self.testReader()
```

Example with fixed number of users:

```python
def testMixin(self):
    if self.thread_id < 2:
        # 2 importer threads
        return self.testImporter()
    elif self.thread_id < 16:
        # 15 back office with sleep time
        return self.testBackOffice()
    else:
        # front office users
        return self.testFrontOffice()
```

Note that when mixing tests the detail report for each page is meaningless because you are mixing pages from multiple tests.

11.6 How to modify a report ?

The report is in reStructuredText, the `index.rst` can be edited in text mode, to rebuild the html version:

```
rst2html --stylesheet=funkload.css   index.rst --traceback > index.html
```

Charts are build with gnuplot the gplot script file are present in the report directory to rebuild the pages charts for instance:

```
gnuplot pages.gplot
```

Since FunkLoad 1.15 you can also use an org-mode output to edit or extend the report before exporting it as a PDF.

11.7 How to automate stuff ?

Here is a sample Makefile

```
CREDCTL := fl-credential-ctl credential.conf
MONCTL := fl-monitor-ctl monitor.conf
LOG_HOME := ./log

ifdef URL
    FLOPS = -u $(URL) $(EXT)
else
    FLOPS = $(EXT)
endif

ifdef REPORT_HOME
    REPORT = $(REPORT_HOME)
else
    REPORT = report
endif

all: test

test: start test-app stop

bench: start bench-app stop

start:
    -mkdir -p $(REPORT) $(LOG_HOME)
    -$(MONCTL) restart
    -$(CREDCTL) restart

stop:
    -$(MONCTL) stop
    -$(CREDCTL) stop
```

```
test-app:
    fl-run-test -d --debug-level=3 --simple-fetch test_app.py App.test_app $(FLOPS)

bench-app:
    -fl-run-bench --simple-fetch test_app.py App.test_app -c 1:5:10:15:20:30:40:50 -D 45 -m 0.1 -
    -fl-build-report $(LOG_HOME)/app-bench.xml --html -o $(REPORT)

clean:
    -find . "(" -name "*~" -or  -name ".#*" -or  -name "*.pyc" ")" -print0 | xargs -0 rm -f
```

It can be used like this:

```
make test
make test URL=http://override-url/
# add extra parameters to the FunkLoad command
make test EXT="-V"
make bench
```

11.8 How to write fluent tests ?

You can use the PageObject and fluent interface patterns as in the Nuxeo DM tests to write test like this:

```
class MySuite(NuxeoTestCase):
     def testMyScenario(self):
         (LoginPage(self)
          .login('Administrator', 'Administrator')
          .getRootWorkspaces()
          .createWorkspace('My workspace', 'Test ws')
          .rights().grant('ReadWrite', 'members')
          .view()
          .createFolder('My folder', 'Test folder')
          .createFile('My file', 'Test file', 'foo.pdf')
          .getRootWorkspaces().deleteItem("My workspace")
          .logout())
```

CHAPTER

TWELVE

DEVELOPMENT

The API can be browse here

To check out the source code:

```
https://github.com/nuxeo/FunkLoad
```

You can contribute using the github fork process or by sending patch or feedback to me: bdelbosc _at_ nuxeo.com.

Thanks to all the contributors !

CHAPTER

THIRTEEN

REPORTING BUGS

If you want to report a bug or if you think that something is missing, either send me an email bdelbosc _at_ nuxeo.com or use the github issue tracker.

The list of open tasks and bugs are sceduled in the TODO file.

CHAPTER

FOURTEEN

LINKS

- How to write funkload test in few minutes http://blog.redturtle.it/redturtle-blog/how-to-write-funkload-test-in-few-minutes
- Fabic + Funkload distributed load http://tarekziade.wordpress.com/2010/12/09/funkload-fabric-quick-and-dirty-distributed-load-system/
- Testing plone site with FunkLoad http://plonexp.leocorn.com/xp/plonedemo/xps9
- BenchMaster http://pypi.python.org/pypi/benchmaster
- Tools for a successful Plone project http://www.martinaspeli.net/articles/tools-for-a-successful-plone-project
- Using Apdex http://www.apdex.org/using.html
- Apdex with FunkLoad (in french) http://www.wikigento.com/tag/funkload/

CHAPTER

FIFTEEN

INDICES AND TABLES

- PDF version of the documentation
- Changes
- Glossary
- Links
- *Index*
- *Module Index*
- *Search Page*