

Introduction to buildout

A system for deploying python applications



Distributed under Creative Commons BY 3.0

June 29, 2011

Fabien ANDRÉ <fandre@bearstech.com>

Outline

- 1 Goals of this presentation
- 2 Issues addressed by buildout
- 3 Basic principles of buildout
- 4 Conclusion

Buildout : most common questions

For people unfamiliar with deployment of python applications, it is difficult to understand :

- *What is buildout for ? Do I need buildout ?*
- *How does it work ? What are its basic principles ?*

Outline

- 1 Goals of this presentation
- 2 Issues addressed by buildout
 - Common application deployment issues
 - Issue #1 : Dependency management
 - Issue #2 : Isolation of environments
 - Issue #3 : Other sysadmin operations
 - Buildout : one command to rule them all
- 3 Basic principles of buildout
- 4 Conclusion

Common application deployment issues

Three common issues while deploying applications :

Dependency management How to automatically all your application dependencies ? Your project may require the pyramid web framework which itself depends on paste, zope.event, zope.configuration, chameleon...

Environment isolation What to do if one of your project requires sqlalchemy 0.6.x and another sqlalchemy ≥ 0.7 ?

Other sysadmin operations You may need to generate configuration files, copy files, run commands etc. to make your application ready to use.

Issue description

- Without dependency management tool it is very tedious to install all the modules required by your project.
- You download one dependency of your project and then discover this package itself requires another package. . .

Solution

Python eggs

- .egg files, python packages including metadata which specifies which packages they depend on
- Located in a central repository (PyPI, the python package index)
- easy_install tool able to install packages directly from PyPI and all of their dependencies
- Python eggs can be built easily using Distribute setuptools

Python eggs are similar to .deb/.rpm packages, easy_install to apt-get or yum and PyPI to distribution repositories

Issue description

Issue not solved by python eggs.

- 1 You install project1 which requires sqlalchemy 0.6.x.
easy_install installs this version
- 2 You install project2 which requires sqlalchemy ≥ 0.7 .
easy_install replaces sqlalchemy 0.6.4 by sqlalchemy 0.7.1
- 3 easy_install keeps the version required by the latest installed module
- 4 If you start project1, it crashes because it requires sqlalchemy 0.6.x

Need for **isolated python environments** (different configuration, different site-packages directories).

Solution

virtualenv

- Enables you to set up a separate environment for each project
- Installs dependencies of each project inside its specific environment
- Therefore two projects can use two different versions of the same library

virtualenvwrapper : lightweight wrapper around virtualenv to allow creation/deletion of python environments with a single command

Issue description

Additional tasks may need to be performed to install your application :

- Generation of configuration files
- Installation of init scripts
- Copy files or folders

For example, if your project is a web application, you may need to configure gunicorn and supervisord.

Solution

Common solutions to solve this problem :

- Use of shell scripts
- Better solution : Fabric + ConfigParser

Fabric

- Python package
- Makes it possible to run locally or remotely (through ssh) shell commands from python
- A fabric task is a python function (in which you can use classical python statements and shell commands)
- Using the fab tool, you can run fabric tasks from the command line

Overview of the previous solutions

Each solution focuses on **one** specific issue.

They don't address the whole problem.

Therefore, to install your application, the user needs to :

- Setup a virtual environment
- Activate it
- Install your package
- Read and understand your README file
- Run a bunch of scripts to finish the installation

Review of the previous solutions

Many drawbacks :

- Need for virtualenv to be installed
- Complex install procedure
- Above all, install procedure not standardized. Each application has its own and specific install procedure
- Need to document extensively your install procedure
- Difficult to write reusable shell/fabric scripts

Introducing buildout

Standardized, simple (2 commands !) way to deploy python applications :

- 1 `python bootstrap.py`
 - To be run inside the directory of a *buildout enabled project*
 - Downloads and installs buildout scripts
- 2 `bin/buildout`
 - Installs dependencies in an isolated python environment.
 - Runs recipes (sort of configurable, reusable administration scripts)

You can think of `bin/buildout` as the `./configure && make && make install` of python.

Overview of buildout

- Created by Jim Fulton of Zope Corporation
- Distributed under Zope Public Licence (ZPL) 2.1 (BSD-like)
- First release in 2006, latest release in the 1.5 branch in November 2010
- 2.0 branch under development, compatible with Python 3 (latest release in April 2011)
- More than 300 recipes available on PyPI ranging from recipes to install apache to recipes to build pyGtk and pyCairo
- Used by Zope, Grok and Plone as their main tool to install and manage software

« *Buildout is an exceedingly civilized way to develop an app.* »

–*Jacob Kaplan-Moss, creator of Django*

Outline

- 1 Goals of this presentation
- 2 Issues addressed by buildout
- 3 Basic principles of buildout**
 - Adding buildout to a new project
 - The buildout.cfg file
- 4 Conclusion

```
$ cd mywebsite
$ ls
  build  buildout.cfg  dist  mywebsite  MyWebSite.egg-info  setup.py
$ cat setup.py
setup(
    name = "MyWebSite", version = "0.1",
    packages = find_packages(),
    install_requires = ["pyramid", "sqlalchemy<=0.6.4"],
    entry_points = {'console_scripts' :
        ["sqlver = mywebsite:sqlversion"]} )
$ cat mywebsite/__init__.py
import pyramid
import sqlalchemy

if sqlalchemy.__version__.split('.')[1] != '6':
    raise Exception("This project requires sqlalchemy 0.6.x")

def sqlversion():
    print "This project uses SQLAlchemy", sqlalchemy.__version__
```

Overview of a simple project

- Two dependencies pyramid and sqlalchemy
- One entry point sqlversion

Adding buildout :

- 1 Fetch the bootstrap.py file

```
$ cd mywebsite  
$ wget http://svn.zope.org/*checkout*/\  
    zc.buildout/trunk/bootstrap/bootstrap.py
```

- 2 Create a basic buildout.cfg file

```
$ cat buildout.cfg  
[buildout]  
parts =
```

Directory Structure of a Buildout

You can now run buildout :

```
$ bin/buildout
```

Buildout did nothing (because our buildout.cfg file is empty) except creating a directory structure :

- `bin/` : Receives executable scripts created by buildout from entrypoints defined in `setup.py` files
- `eggs/` : Receives python packages downloaded by buildout
- `part/` : Each part may create a directory beneath `part/` for its specific use

Example of a buildout.cfg file

```
$ cat buildout.cfg
[buildout]
parts = python console
develop = .

[python]
recipe = zc.recipe.egg
interpreter = python
eggs = mywebsite

[console]
recipe = zc.recipe.egg:scripts
eggs = mywebsite
```

Structure of a buildout.cfg file

- When it is invoked buildout runs all the *parts* listed in the `parts` option of the `[buildout]` section.
- To each *part* listed in the `parts` option corresponds a section in the `buildout.cfg` file

Each part consists of :

- A recipe to execute (`zc.recipe.egg`,
`zc.recipe.egg:scripts`)
- Arguments for this recipe (`interpreter`, `eggs ...`)

Buildout recipes

Recipes : Python scripts which carry out a certain task according to the arguments they are given

Example of recipes :

- `zc.recipe.egg` Installs specified eggs (eggs argument) into the buildout eggs directory. Also able to create a python interpreter with these eggs baked in (interpreter argument).
- `zc.recipe.egg:scripts` Creates scripts in `bin/` based on `setup.py` entry_points.
- `zc.recipe.testrunner` Creates a test script in `bin/` which runs tests for the project.
- Many recipes available on PyPI : apache installation etc...

Outline

- 1 Goals of this presentation
- 2 Issues addressed by buildout
- 3 Basic principles of buildout
- 4 Conclusion**

Conclusion

Common use case :

- As soon as you need a complete, extensible build system to deploy python applications (install project in isolated environment and carry out any installation task)

Cases where buildout is not suitable :

- If you don't need to carry out many installation tasks and just need to package your project, bare setuptools may be enough.
- If you only need to carry many specific systems administration tasks (especially if you need to connect to remote hosts), fabric may be a better choice as fabric tasks are faster to write than buildout recipes.