Reproducible installation of applications using zc.buildout

Thomas Lotze
tl@gocept.com

Europython 2012
July 5, Florence, Italy

Outline

Outline

Outline

Outline

Outline

Outline

Contents

About zc.buildout

- Which problems shall the tool solve?
- Which problems shall we not concern ourselves with?
- history of zc.buildout
- terminology

About zc.buildout

- Which problems shall the tool solve?
- Which problems shall we not concern ourselves with?
- history of zc.buildout
- terminology

About zc.buildout

- Which problems shall the tool solve?
- Which problems shall we not concern ourselves with?
- history of zc.buildout
- terminology

About zc.buildout

- Which problems shall the tool solve?
- Which problems shall we not concern ourselves with?
- history of zc.buildout
- terminology

Problems to solve

“install and configure software in a reproducible way”

- both Python packages and any other software
- simple case: develop a Python package
- complex case: deploy a multi-part application
- simple description that is as complete as possible

Problems to solve

“install and configure software in a reproducible way”

- both Python packages and any other software
- simple case: develop a Python package
- complex case: deploy a multi-part application
- simple description that is as complete as possible

Problems to solve

“install and configure software in a reproducible way”

- both Python packages and any other software
- simple case: develop a Python package
- complex case: deploy a multi-part application
- simple description that is as complete as possible

Problems to solve

“install and configure software in a reproducible way”

- both Python packages and any other software
- simple case: develop a Python package
- complex case: deploy a multi-part application
- simple description that is as complete as possible

Problems to solve

“install and configure software in a reproducible way”

- both Python packages and any other software
- simple case: develop a Python package
- complex case: deploy a multi-part application
- simple description that is as complete as possible

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

Problems not to solve

- low-level: don't build software from source (such as C)
  - control existing specialised tools
  - configure/make/make install
  - distutils
- high-level: don't install into the host system
  - self-contained, isolated from other applications
  - provide pieces to be integrated with operating system
  - act as a building block for configuration management

What is zc.buildout?

- developed by Jim Fulton (Zope Corporation) in 2006
- builds on lessons learned from two earlier attempts
- used for much more than Zope projects today

What is zc.buildout?

- developed by Jim Fulton (Zope Corporation) in 2006
- builds on lessons learned from two earlier attempts
- used for much more than Zope projects today

What is zc.buildout?

- developed by Jim Fulton (Zope Corporation) in 2006
- builds on lessons learned from two earlier attempts
- used for much more than Zope projects today

Terminology: 3 meanings to “buildout”

- the software, zc.buildout
- the specification of an application’s build and configuration
- the build, i.e. a directory populated by running zc.buildout on a buildout configuration

Terminology: 3 meanings to “buildout”

- the software, zc.buildout
- the specification of an application’s build and configuration
- the build, i.e. a directory populated by running zc.buildout on a buildout configuration

Terminology: 3 meanings to “buildout”

- the software, zc.buildout
- the specification of an application’s build and configuration
- the build, i.e. a directory populated by running zc.buildout on a buildout configuration

Contents

A first simple buildout

- what's needed
- how to run zc.buildout
- what happens in a buildout run
- repeating buildout runs

A first simple buildout

- what's needed
- how to run zc.buildout
- what happens in a buildout run
- repeating buildout runs

A first simple buildout

- what's needed
- how to run zc.buildout
- what happens in a buildout run
- repeating buildout runs

A first simple buildout

- what's needed
- how to run zc.buildout
- what happens in a buildout run
- repeating buildout runs

What's needed?

- assume zc.buildout is not installed
- download bootstrap.py

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

- create a buildout configuration file

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

What's needed?

- assume zc.buildout is not installed
- download bootstrap.py

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

- create a buildout configuration file

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

What's needed?

- assume zc.buildout is not installed
- download bootstrap.py

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

- create a buildout configuration file

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

What's needed?

- assume zc.buildout is not installed
- download bootstrap.py

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

- create a buildout configuration file

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

Getting started

```
$ python bootstrap.py -d

Downloading http://pypi..../distribute-0.6.27.tar.gz
...
Creating directory '/home/thomas/py/bin'.
Creating directory '/home/thomas/py/parts'.
Creating directory '/home/thomas/py/eggs'.
Creating directory '/home/thomas/py/develop-eggs'.
Generated script '/home/thomas/py/bin/buildout'.
```

After bootstrapping

```
$ ls *

bootstrap.py  buildout.cfg

bin:
buildout

develop-eggs:

eggs:
distribute-0.6.27-py2.7.egg
zc.buildout-1.5.2-py2.7.egg

parts:
buildout
```

What's next?

```
$ bin/buildout

Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
Getting distribution for 'sphinx'.
Got Sphinx 1.1.3.
Getting distribution for 'docutils>=0.7'.
warning: ...
Got docutils 0.9.1.
Getting distribution for 'Jinja2>=2.3'.
warning: ...
Got Jinja2 2.6.
Getting distribution for 'Pygments>=1.2'.
Got Pygments 1.5.
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
```

After the buildout run

```
$ ls *

bootstrap.py  buildout.cfg

bin:
buildout  sphinx-apidoc  sphinx-autogen
sphinx-build  sphinx-quickstart

develop-eggs:

eggs:
distribute-0.6.27-py2.7.egg    docutils-0.9.1-py2.7.egg
Jinja2-2.6-py2.7.egg           Pygments-1.5-py2.7.egg
Sphinx-1.1.3-py2.7.egg         zc.buildout-1.5.2-py2.7.egg
zc.recipe.egg-1.3.2-py2.7.egg

parts:
buildout
```

What happened?

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

- buildout part “sphinx” is installed
- work is done by a recipe: plug-in point
- recipe comes as an egg

```
Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
```

What happened?

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

- buildout part “sphinx” is installed
- work is done by a recipe: plug-in point
- recipe comes as an egg

```
Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
```

What happened?

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

- buildout part “sphinx” is installed
- work is done by a recipe: plug-in point
- recipe comes as an egg

```
Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
```

What happened?

```
1 [buildout]
2 parts = sphinx
3
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
```

- buildout part “sphinx” is installed
- work is done by a recipe: plug-in point
- recipe comes as an egg

```
Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
```

What happened? The sphinx part

zc.recipe.egg invokes zc.buildout's easy_install API

- download sphinx sources (as configured)

```
Getting distribution for 'sphinx'.
```

- build the egg

```
Got Sphinx 1.1.3.
```

- follow declared dependencies

```
Getting distribution for 'docutils>=0.5'.
Got docutils 0.9.1.
...
```

- detect and install scripts provided by explicitly listed eggs

```
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
...
```

What happened? The sphinx part

zc.recipe.egg invokes zc.buildout's easy_install API

- download sphinx sources (as configured)

```
Getting distribution for 'sphinx'.
```

- build the egg

```
Got Sphinx 1.1.3.
```

- follow declared dependencies

```
Getting distribution for 'docutils>=0.5'.
Got docutils 0.9.1.
...
```

- detect and install scripts provided by explicitly listed eggs

```
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
...
```

What happened? The sphinx part

zc.recipe.egg invokes zc.buildout's easy_install API

- download sphinx sources (as configured)

```
Getting distribution for 'sphinx'.
```

- build the egg

```
Got Sphinx 1.1.3.
```

- follow declared dependencies

```
Getting distribution for 'docutils>=0.5'.
Got docutils 0.9.1.
...
```

- detect and install scripts provided by explicitly listed eggs

```
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
...
```

What happened? The sphinx part

zc.recipe.egg invokes zc.buildout's easy_install API

- download sphinx sources (as configured)

```
Getting distribution for 'sphinx'.
```

- build the egg

```
Got Sphinx 1.1.3.
```

- follow declared dependencies

```
Getting distribution for 'docutils>=0.5'.
Got docutils 0.9.1.
...
```

- detect and install scripts provided by explicitly listed eggs

```
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
...
```

What happened? The sphinx part

zc.recipe.egg invokes zc.buildout's easy_install API

- download sphinx sources (as configured)

```
Getting distribution for 'sphinx'.
```

- build the egg

```
Got Sphinx 1.1.3.
```

- follow declared dependencies

```
Getting distribution for 'docutils>=0.5'.
Got docutils 0.9.1.
...
```

- detect and install scripts provided by explicitly listed eggs

```
Generated script '/home/thomas/py/bin/sphinx-apidoc'.
...
```

Repeating the buildout run

- with configuration unchanged:

```
$ bin/buildout
Updating sphinx.
```

- already installed, not installed again
- unconditional update phase
  - looks for new releases by default

Repeating the buildout run

- with configuration unchanged:

```
$ bin/buildout
Updating sphinx.
```

- already installed, not installed again
- unconditional update phase
  - looks for new releases by default

Repeating the buildout run

- with configuration unchanged:

```
$ bin/buildout
Updating sphinx.
```

- already installed, not installed again
- unconditional update phase
  - looks for new releases by default

Repeating the buildout run

- modify configuration:

```
4  [sphinx]
5  recipe = zc.recipe.egg
6  eggs = sphinx
7  scripts = sphinx-build sphinx-apidoc
```

```
$ bin/buildout
Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/sphinx-apidoc'.
Generated script '/home/thomas/py/sphinx-build'.
```

- part with modified configuration is re-installed from scratch
- previously created files (e.g. scripts) are removed
- parts with unchanged configuration are updated

Repeating the buildout run

- modify configuration:

```
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
7 scripts = sphinx-build sphinx-apidoc
```

```
$ bin/buildout
Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/sphinx-apidoc'.
Generated script '/home/thomas/py/sphinx-build'.
```

- part with modified configuration is re-installed from scratch
- previously created files (e.g. scripts) are removed
- parts with unchanged configuration are updated

Repeating the buildout run

- modify configuration:

```
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
7 scripts = sphinx-build sphinx-apidoc
```

```
$ bin/buildout
Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/sphinx-apidoc'.
Generated script '/home/thomas/py/sphinx-build'.
```

- part with modified configuration is re-installed from scratch
- previously created files (e.g. scripts) are removed
- parts with unchanged configuration are updated

Repeating the buildout run

- modify configuration:

```
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
7 scripts = sphinx-build sphinx-apidoc
```

```
$ bin/buildout
Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/sphinx-apidoc'.
Generated script '/home/thomas/py/sphinx-build'.
```

- part with modified configuration is re-installed from scratch
- previously created files (e.g. scripts) are removed
- parts with unchanged configuration are updated

Repeating the buildout run

- modify configuration:

```
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
7 scripts = sphinx-build sphinx-apidoc
```

```
$ bin/buildout
Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/sphinx-apidoc'.
Generated script '/home/thomas/py/sphinx-build'.
```

- part with modified configuration is re-installed from scratch
- previously created files (e.g. scripts) are removed
- parts with unchanged configuration are updated

Contents

How Python code is installed

- scripts and their environment
- a Python interpreter
- compare to virtualenv + pip

How Python code is installed

- scripts and their environment
- a Python interpreter
- compare to virtualenv + pip

How Python code is installed

- scripts and their environment
- a Python interpreter
- compare to virtualenv + pip

How egg installation works

```
$ cat bin/sphinx-quickstart
#!/usr/bin/python

import sys
sys.path[0:0] = [
    '/home/thomas/py/eggs/Sphinx-1.1.3-py2.7.egg',
    '/home/thomas/py/eggs/docutils-0.9.1-py2.7.egg',
    '/home/thomas/py/eggs/Jinja2-2.6-py2.7.egg',
    '/home/thomas/py/eggs/Pygments-1.5-py2.7.egg',
    ]

import sphinx.quickstart

if __name__ == '__main__':
    sphinx.quickstart.main()
```

How egg installation works

- each script calls one of the egg's entry points

```
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- each script sets up its own Python path

```
4 sys.path[0:0] = [
5     '/home/thomas/py/eggs/Sphinx-1.1.3-py2.7.egg',
6     ...
```

- use a Python installation without modifying it

How egg installation works

- each script calls one of the egg's entry points

```
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- each script sets up its own Python path

```
4 sys.path[0:0] = [
5     '/home/thomas/py/eggs/Sphinx-1.1.3-py2.7.egg',
6     ...
```

- use a Python installation without modifying it

How egg installation works

- each script calls one of the egg's entry points

```
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- each script sets up its own Python path

```
4 sys.path[0:0] = [
5     '/home/thomas/py/eggs/Sphinx-1.1.3-py2.7.egg',
6     ...
```

- use a Python installation without modifying it

Using eggs with an interpreter

- configure the eggs' part to create an interpreter:

```
4  [sphinx]
5  recipe = zc.recipe.egg
6  eggs = sphinx
7  interpreter = py
```

- the egg recipe creates an executable script:

```
$ bin/buildout
...
Generated interpreter '/home/thomas/py/bin/py'.
```

Using eggs with an interpreter

- configure the eggs' part to create an interpreter:

```
4 [sphinx]
5 recipe = zc.recipe.egg
6 eggs = sphinx
7 interpreter = py
```

- the egg recipe creates an executable script:

```
$ bin/buildout
...
Generated interpreter '/home/thomas/py/bin/py'.
```

Using eggs with an interpreter

```
#!/usr/bin/python

import sys
sys.path[0:0] = [
    '/var/lib/python-eggs/Sphinx-1.1.3-py2.7.egg',
    '/var/lib/python-eggs/docutils-0.9.1-py2.7.egg',
    '/var/lib/python-eggs/Jinja2-2.6-py2.7.egg',
    '/var/lib/python-eggs/Pygments-1.5-py2.7.egg',
    ]

...
    exec _val
    __import__("runpy").run_module(...)
    execfile(...)
    __import__("code").interact(...)
```

- just another script that sets up its path
- invokes Python interpreter according to options

Using eggs with an interpreter

```
1  #!/usr/bin/python
2
3  import sys
4  sys.path[0:0] = [
5      '/var/lib/python-eggs/Sphinx-1.1.3-py2.7.egg',
6      '/var/lib/python-eggs/docutils-0.9.1-py2.7.egg',
7      '/var/lib/python-eggs/Jinja2-2.6-py2.7.egg',
8      '/var/lib/python-eggs/Pygments-1.5-py2.7.egg',
9      ]
10
11 ...
12     exec _val
13     __import__("runpy").run_module(...)
14     execfile(...)
15     __import__("code").interact(...)
```

- just another script that sets up its path
- invokes Python interpreter according to options

Using eggs with an interpreter

```
1  #!/usr/bin/python
2
3  import sys
4  sys.path[0:0] = [
5      '/var/lib/python-eggs/Sphinx-1.1.3-py2.7.egg',
6      '/var/lib/python-eggs/docutils-0.9.1-py2.7.egg',
7      '/var/lib/python-eggs/Jinja2-2.6-py2.7.egg',
8      '/var/lib/python-eggs/Pygments-1.5-py2.7.egg',
9      ]
10
11 ...
12     exec _val
13     __import__("runpy").run_module(...)
14     execfile(...)
15     __import__("code").interact(...)
```

- just another script that sets up its path
- invokes Python interpreter according to options

Contrast: virtualenv + pip

- creates a Python installation meant to be modified
- pip requirements file: minimal set of packages
- defines the Python path as a well-known directory
- Python path implicitly set up by using the local interpreter
- Python path may be exported: “activate” the environment

Contrast: virtualenv + pip

- creates a Python installation meant to be modified
- pip requirements file: minimal set of packages
- defines the Python path as a well-known directory
- Python path implicitly set up by using the local interpreter
- Python path may be exported: “activate” the environment

Contrast: virtualenv + pip

- creates a Python installation meant to be modified
- pip requirements file: minimal set of packages
- defines the Python path as a well-known directory
- Python path implicitly set up by using the local interpreter
- Python path may be exported: “activate” the environment

Contrast: virtualenv + pip

- creates a Python installation meant to be modified
- pip requirements file: minimal set of packages
- defines the Python path as a well-known directory
- Python path implicitly set up by using the local interpreter
- Python path may be exported: “activate” the environment

Contrast: virtualenv + pip

- creates a Python installation meant to be modified
- pip requirements file: minimal set of packages
- defines the Python path as a well-known directory
- Python path implicitly set up by using the local interpreter
- Python path may be exported: “activate” the environment

Contents

Reproducibility

- specifying what to install: pinning versions
- enforcing a complete specification
- known-good sets of software packages

Reproducibility

- specifying what to install: pinning versions
- enforcing a complete specification
- known-good sets of software packages

Reproducibility

- specifying what to install: pinning versions
- enforcing a complete specification
- known-good sets of software packages

Which eggs are installed?

- determined by buildout configuration and dependencies
- full paths baked into scripts: no random additions
- eggs are looked up at the package index
- eggs may also come from files or an on-line list of links

Which eggs are installed?

- determined by buildout configuration and dependencies
- full paths baked into scripts: no random additions
- eggs are looked up at the package index
- eggs may also come from files or an on-line list of links

Which eggs are installed?

- determined by buildout configuration and dependencies
- full paths baked into scripts: no random additions
- eggs are looked up at the package index
- eggs may also come from files or an on-line list of links

Which eggs are installed?

- determined by buildout configuration and dependencies
- full paths baked into scripts: no random additions
- eggs are looked up at the package index
- eggs may also come from files or an on-line list of links

Which egg versions are installed?

- declared dependencies on versions are always fulfilled
- newest matching versions are used
- search for newer versions may be suppressed
- versions still depend on first installation

Which egg versions are installed?

- declared dependencies on versions are always fulfilled
- newest matching versions are used
- search for newer versions may be suppressed
- versions still depend on first installation

Which egg versions are installed?

- declared dependencies on versions are always fulfilled
- newest matching versions are used
- search for newer versions may be suppressed
- versions still depend on first installation

Which egg versions are installed?

- declared dependencies on versions are always fulfilled
- newest matching versions are used
- search for newer versions may be suppressed
- versions still depend on first installation

How to pin versions with buildout

- global option:

```
1 [buildout]
2 versions = versions
3
4 [versions]
5 sphinx = 1.1.2
```

- version pinnings are always honoured
- versions of other packages are still unpredictable

How to pin versions with buildout

- global option:

```
1 [buildout]
2 versions = versions
3
4 [versions]
5 sphinx = 1.1.2
```

- version pinnings are always honoured
- versions of other packages are still unpredictable

How to pin versions with buildout

- global option:

```
1 [buildout]
2 versions = versions
3
4 [versions]
5 sphinx = 1.1.2
```

- version pinnings are always honoured
- versions of other packages are still unpredictable

Forcing all versions to be pinned

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4  allow-picked-versions = false
5
6  [versions]
7  Jinja2 = 2.6
8  Pygments = 1.5
9  distribute = 0.6.27
10 docutils = 0.9.1
11 sphinx = 1.1.2
12 zc.buildout = 1.5.2
13 zc.recipe.egg = 1.3.2
14
15 [sphinx]
16 recipe = zc.recipe.egg
17 eggs = sphinx
```

Forcing all versions to be pinned

- recipes and even zc.buildout itself are pinned
- known-good build in addition to known-good code:
  - be sure that pieces of the build system match
  - predictable configuration (e.g. paths, generated scripts)
- still not pinned: Python itself

Forcing all versions to be pinned

- recipes and even zc.buildout itself are pinned
- known-good build in addition to known-good code:
  - be sure that pieces of the build system match
  - predictable configuration (e.g. paths, generated scripts)
- still not pinned: Python itself

Forcing all versions to be pinned

- recipes and even zc.buildout itself are pinned
- known-good build in addition to known-good code:
  - be sure that pieces of the build system match
  - predictable configuration (e.g. paths, generated scripts)
- still not pinned: Python itself

Forcing all versions to be pinned

- recipes and even zc.buildout itself are pinned
- known-good build in addition to known-good code:
  - be sure that pieces of the build system match
  - predictable configuration (e.g. paths, generated scripts)
- still not pinned: Python itself

Forcing all versions to be pinned

- recipes and even zc.buildout itself are pinned
- known-good build in addition to known-good code:
  - be sure that pieces of the build system match
  - predictable configuration (e.g. paths, generated scripts)
- still not pinned: Python itself

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Isolation and sharing

- one version of each egg per buildout (one versions section)
- (possible future feature: egg versions per part)
- doesn't modify the Python installation or the OS
- still, be careful about site-packages (e.g., OS packages)
- any number of buildouts may coexist
- egg files on disk may be shared among buildouts:

```
1 [buildout]
2 eggs-directory = /var/lib/python-eggs
```

Maintaining version pinnings

- add a version pinning for each new package
- update versions consciously at a convenient time
- pinnings describe known good sets (KGS) of eggs
- maintain KGS of related packages in a central place

Maintaining version pinnings

- add a version pinning for each new package
- update versions consciously at a convenient time
- pinnings describe known good sets (KGS) of eggs
- maintain KGS of related packages in a central place

Maintaining version pinnings

- add a version pinning for each new package
- update versions consciously at a convenient time
- pinnings describe known good sets (KGS) of eggs
- maintain KGS of related packages in a central place

Maintaining version pinnings

- add a version pinning for each new package
- update versions consciously at a convenient time
- pinnings describe known good sets (KGS) of eggs
- maintain KGS of related packages in a central place

Overriding version pinnings

- buildout configurations may extend each other
- use externally maintained KGS:

```
[buildout]
extends = http://example.com/versions.cfg
parts = sphinx
allow-picked-versions = false

[versions]
sphinx = 1.1.2
```

- contents of versions.cfg:

```
[buildout]
versions = versions

[versions]
Jinja2 = ...
```

Overriding version pinnings

- buildout configurations may extend each other
- use externally maintained KGS:

```
[buildout]
extends = http://example.com/versions.cfg
parts = sphinx
allow-picked-versions = false

[versions]
sphinx = 1.1.2
```

- contents of versions.cfg:

```
[buildout]
versions = versions

[versions]
Jinja2 = ...
```

Overriding version pinnings

- buildout configurations may extend each other
- use externally maintained KGS:

```
[buildout]
extends = http://example.com/versions.cfg
parts = sphinx
allow-picked-versions = false

[versions]
sphinx = 1.1.2
```

- contents of versions.cfg:

```
[buildout]
versions = versions

[versions]
Jinja2 = ...
```

Contents

How to install non-Python software

```
1  [buildout]
2  parts = frontend
3
4  [nginx]
5  recipe = zc.recipe.cmmi
6  url = http://nginx.org/download/nginx-1.2.2.tar.gz
7
8  [frontend]
9  recipe = gocept.nginx
10 nginx = nginx
11 configuration =
12     worker_processes 1;
13     events {
14         worker_connections 1024;
15     ...
```

How to install non-Python software

- recipe for doing configure/make/make install

```
$ ls parts/nginx
conf  html  logs  sbin
```

- custom recipes for specialised tasks

```
$ cat parts/frontend/frontend.conf
pid /home/thomas/py/parts/frontend/frontend.pid;
...
worker_processes 1;
events {
worker_connections 1024;
...

$ cat bin/frontend
#!/bin/sh
ARGV="$@"
NGINX='/home/thomas/py/parts/nginx/sbin/nginx'
PIDFILE='/home/thomas/py/parts/frontend/frontend.pid'
...
```

How to install non-Python software

- recipe for doing configure/make/make install

```
$ ls parts/nginx
conf  html  logs  sbin
```

- custom recipes for specialised tasks

```
$ cat parts/frontend/frontend.conf
pid /home/thomas/py/parts/frontend/frontend.pid;
...
worker_processes 1;
events {
worker_connections 1024;
...

$ cat bin/frontend
#!/bin/sh
ARGV="$@"
NGINX='/home/thomas/py/parts/nginx/sbin/nginx'
PIDFILE='/home/thomas/py/parts/frontend/frontend.pid'
...
```

How to install non-Python software

- recipe for doing configure/make/make install

```
$ ls parts/nginx
conf  html  logs  sbin
```

- custom recipes for specialised tasks

```
$ cat parts/frontend/frontend.conf
pid /home/thomas/py/parts/frontend/frontend.pid;
...
worker_processes 1;
events {
worker_connections 1024;
...
```

```
$ cat bin/frontend
#!/bin/sh
ARGV="$@"
NGINX='/home/thomas/py/parts/nginx/sbin/nginx'
PIDFILE='/home/thomas/py/parts/frontend/frontend.pid'
...
```

Interaction between configuration sections

- dependencies between configuration sections
- dependent part can reuse information from others
- depended-upon parts will be handled first in each phase
- also possible: referring to part options in config file

Interaction between configuration sections

- dependencies between configuration sections
- dependent part can reuse information from others
- depended-upon parts will be handled first in each phase
- also possible: referring to part options in config file

Interaction between configuration sections

- dependencies between configuration sections
- dependent part can reuse information from others
- depended-upon parts will be handled first in each phase
- also possible: referring to part options in config file

Interaction between configuration sections

- dependencies between configuration sections
- dependent part can reuse information from others
- depended-upon parts will be handled first in each phase
- also possible: referring to part options in config file

Using buildout for large systems

```
[database]
recipe = zc.recipe.filestorage
blob-dir = ${buildout:directory}/parts/database/blobs

[zeo]
recipe = zc.zodbrecipes:server
address = 8100
pack-keep-old = true
zeo.conf =
   <zeo>
    address ${zeo:address}
   </zeo>
   <filestorage 1>
     blob-dir ${database:blob-dir}
   ...

[app-server]
recipe = zc.zope3recipes:instance
zodb-client-cache-size = 200MB
zodb-object-cache-size = 20MB
blob-dir = ${database:blob-dir}
```

Other commonly used recipes

- file templates, directories
- deployment
- development tools: test runner, omelette
- more specific software: django, sphinx, supervisor, ...

Other commonly used recipes

- file templates, directories
- deployment
- development tools: test runner, omelette
- more specific software: django, sphinx, supervisor, ...

Other commonly used recipes

- file templates, directories
- deployment
- development tools: test runner, omelette
- more specific software: django, sphinx, supervisor, ...

Other commonly used recipes

- file templates, directories
- deployment
- development tools: test runner, omelette
- more specific software: django, sphinx, supervisor, ...

Contents

Closing remarks

- summary, zc.buildout's strengths
- issues with zc.buildout
- roadmap for further development

Closing remarks

- summary, zc.buildout's strengths
- issues with zc.buildout
- roadmap for further development

Closing remarks

- summary, zc.buildout's strengths
- issues with zc.buildout
- roadmap for further development

Summary: zc.buildout's strengths

- very well suited for installing Python code (dependencies, scripts)
- extensible by recipes to cover complex applications
- works from a complete plain-text specification
- able to pin versions including those of the build system
- isolation of the build from uncontrolled environment

Summary: zc.buildout's strengths

- very well suited for installing Python code (dependencies, scripts)
- extensible by recipes to cover complex applications
- works from a complete plain-text specification
- able to pin versions including those of the build system
- isolation of the build from uncontrolled environment

Summary: zc.buildout's strengths

- very well suited for installing Python code (dependencies, scripts)
- extensible by recipes to cover complex applications
- works from a complete plain-text specification
- able to pin versions including those of the build system
- isolation of the build from uncontrolled environment

Summary: zc.buildout's strengths

- very well suited for installing Python code (dependencies, scripts)
- extensible by recipes to cover complex applications
- works from a complete plain-text specification
- able to pin versions including those of the build system
- isolation of the build from uncontrolled environment

Summary: zc.buildout's strengths

- very well suited for installing Python code (dependencies, scripts)
- extensible by recipes to cover complex applications
- works from a complete plain-text specification
- able to pin versions including those of the build system
- isolation of the build from uncontrolled environment

Issues

- Python path is not readily inspectable
- no way to recognise a failed buildout by its state
- simple domain model that leaves all details up to recipes
- combination of configuration by non-programming language plus recipes feels unwieldy
- no concept of convergence: parts installed depending on configuration, not state

Issues

- Python path is not readily inspectable
- no way to recognise a failed buildout by its state
- simple domain model that leaves all details up to recipes
- combination of configuration by non-programming language plus recipes feels unwieldy
- no concept of convergence: parts installed depending on configuration, not state

Issues

- Python path is not readily inspectable
- no way to recognise a failed buildout by its state
- simple domain model that leaves all details up to recipes
- combination of configuration by non-programming language plus recipes feels unwieldy
- no concept of convergence: parts installed depending on configuration, not state

Issues

- Python path is not readily inspectable
- no way to recognise a failed buildout by its state
- simple domain model that leaves all details up to recipes
- combination of configuration by non-programming language plus recipes feels unwieldy
- no concept of convergence: parts installed depending on configuration, not state

Issues

- Python path is not readily inspectable
- no way to recognise a failed buildout by its state
- simple domain model that leaves all details up to recipes
- combination of configuration by non-programming language plus recipes feels unwieldy
- no concept of convergence: parts installed depending on configuration, not state

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Roadmap

- egg support
  - currently by reusing easy_install from distribute
  - switch to distutils2/packaging when it's "stable enough"
- Python 3 support
  - attempt at porting with 2to3: zc.buildout 2.0.0 alpha2
  - current plans: start over using a single code base
- features, efficiency, refactoring

Where to work with zc.buildout

Of course, with us!

gocept is looking for developers.

http://gocept.com

Thank you.