Nachvollziehbare Anwendungsinstallation mit zc.buildout

Thomas Lotze

gocept gmbh & co. kg
Halle (Saale)

6.10.2011, PyCon DE, Leipzig

Gliederung

Gliederung

Gliederung

Gliederung

Gliederung

Gliederung

Gliederung

Welches Problem wollen wir lösen?

- Software installieren und konfigurieren
- Python-Pakete und beliebige andere Software
- einfacher Fall: Python-Paket entwickeln
- komplexer Fall: vielteilige Anwendung in Produktion
- einfache, aber möglichst vollständige Beschreibung

Welches Problem wollen wir lösen?

- Software installieren und konfigurieren
- Python-Pakete und beliebige andere Software
- einfacher Fall: Python-Paket entwickeln
- komplexer Fall: vielteilige Anwendung in Produktion
- einfache, aber möglichst vollständige Beschreibung

Welches Problem wollen wir lösen?

- Software installieren und konfigurieren
- Python-Pakete und beliebige andere Software
- einfacher Fall: Python-Paket entwickeln
- komplexer Fall: vielteilige Anwendung in Produktion
- einfache, aber möglichst vollständige Beschreibung

Welches Problem wollen wir lösen?

- Software installieren und konfigurieren
- Python-Pakete und beliebige andere Software
- einfacher Fall: Python-Paket entwickeln
- komplexer Fall: vielteilige Anwendung in Produktion
- einfache, aber möglichst vollständige Beschreibung

Welches Problem wollen wir lösen?

- Software installieren und konfigurieren
- Python-Pakete und beliebige andere Software
- einfacher Fall: Python-Paket entwickeln
- komplexer Fall: vielteilige Anwendung in Produktion
- einfache, aber möglichst vollständige Beschreibung

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Welche Probleme wollen wir nicht lösen?

- Software aus den Quellen (z.B. C) bauen
  - configure/make/make install
  - distutils
- systemweite Installation
  - Zusammenspiel mit Betriebssystem-Distribution
  - Konflikte mit anderen Anwendungen im gleichen System

Was ist Buildout?

- 2006 von Jim Fulton (Zope Corporation) entwickelt
- baut auf Erfahrungen mit zwei Vorgängersystemen auf
- inzwischen weit über Zope-Projekte hinaus verwendet
- Begriff: zc.buildout, Konfiguration und konkrete Installation

Was ist Buildout?

- 2006 von Jim Fulton (Zope Corporation) entwickelt
- baut auf Erfahrungen mit zwei Vorgängersystemen auf
- inzwischen weit über Zope-Projekte hinaus verwendet
- Begriff: zc.buildout, Konfiguration und konkrete Installation

Was ist Buildout?

- 2006 von Jim Fulton (Zope Corporation) entwickelt
- baut auf Erfahrungen mit zwei Vorgängersystemen auf
- inzwischen weit über Zope-Projekte hinaus verwendet
- Begriff: zc.buildout, Konfiguration und konkrete Installation

Was ist Buildout?

- 2006 von Jim Fulton (Zope Corporation) entwickelt
- baut auf Erfahrungen mit zwei Vorgängersystemen auf
- inzwischen weit über Zope-Projekte hinaus verwendet
- Begriff: zc.buildout, Konfiguration und konkrete Installation

Gliederung

Was braucht man?

- entweder zc.buildout bereits installiert, oder bootstrap.py
- zwei Dateien:

```
$ ls

bootstrap.py buildout.cfg
```

- einfache Konfiguration:

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

Was braucht man?

- entweder zc.buildout bereits installiert, oder bootstrap.py
- zwei Dateien:

```
$ ls

bootstrap.py buildout.cfg
```

- einfache Konfiguration:

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

Was braucht man?

- entweder zc.buildout bereits installiert, oder bootstrap.py
- zwei Dateien:

```
$ ls

bootstrap.py buildout.cfg
```

- einfache Konfiguration:

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

Wie geht es los?

```
$ python bootstrap.py -d

Downloading http://pypi..../distribute-0.6.21.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'.
```

Nach dem Bootstrap-Lauf

```
$ ls *

bootstrap.py  buildout.cfg

bin:
buildout

develop-eggs:

eggs:
distribute-0.6.21-py2.7.egg
zc.buildout-1.5.2-py2.7.egg

parts:
buildout
```

Wie geht es weiter?

```
$ bin/buildout

Getting distribution for 'zc.recipe.egg'.
Got zc.recipe.egg 1.3.2.
Installing sphinx.
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
warning: ...
Got docutils 0.8.1.
Getting distribution for 'Jinja2>=2.2'.
warning: ...
Got Jinja2 2.6.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
```

Nach dem Buildout-Lauf

```
$ ls *

bootstrap.py  buildout.cfg

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

develop-eggs:

eggs:
distribute-0.6.21-py2.7.egg
docutils-0.8.1-py2.7.egg
Jinja2-2.6-py2.7.egg
Sphinx-1.0.8-py2.7.egg
zc.buildout-1.5.2-py2.7.egg
zc.recipe.egg-1.3.2-py2.7.egg

parts:
buildout
```

Was ist passiert?

- Konfiguration:

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

- Buildout-Part „sphinx“ wird installiert
- Arbeit wird von einem Rezept erledigt
- Rezept kommt aus einem Egg
- Buildout-Lauf:

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

Was ist passiert?

- Konfiguration:

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

- Buildout-Part „sphinx“ wird installiert
- Arbeit wird von einem Rezept erledigt
- Rezept kommt aus einem Egg
- Buildout-Lauf:

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

Was ist passiert?

- Konfiguration:

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

- Buildout-Part „sphinx“ wird installiert
- Arbeit wird von einem Rezept erledigt
- Rezept kommt aus einem Egg
- Buildout-Lauf:

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

Was ist passiert?

- Konfiguration:

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

- Buildout-Part „sphinx“ wird installiert
- Arbeit wird von einem Rezept erledigt
- Rezept kommt aus einem Egg
- Buildout-Lauf:

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

Was ist passiert?

- Konfiguration:

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

- Buildout-Part „sphinx“ wird installiert
- Arbeit wird von einem Rezept erledigt
- Rezept kommt aus einem Egg
- Buildout-Lauf:

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

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Was ist passiert?

- Aufgabe des Rezepts:

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

- Quellen von sphinx herunterladen, Egg bauen
- deklarierte Abhängigkeiten verfolgen
- Buildout-Lauf:

```
Getting distribution for 'sphinx'.
Got Sphinx 1.0.8.
Getting distribution for 'docutils>=0.5'.
Got docutils 0.8.1.
...
```

- explizit genanntes Egg auf Skripte untersuchen
- Buildout-Lauf:

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

- keine Skripte aus den anderen Eggs

Gliederung

Wie funktioniert die Egg-Installation?

- Inhalt eines Skripts:

```
1  #!/usr/bin/python
2
3  import sys
4  sys.path[0:0] = [
5      '/home/thomas/py/eggs/Sphinx-1.0.8-py2.7.egg',
6      '/home/thomas/py/eggs/docutils-0.8.1-py2.7.egg',
7      '/home/thomas/py/eggs/Jinja2-2.6-py2.7.egg',
8      '/usr/lib/python2.7/dist-packages',
9      ]
10
11 import sphinx.quickstart
12
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- jedes Skript setzt seinen eigenen Python-Pfad
- jedes Skript ruft einen Entry-Point des Eggs auf

Wie funktioniert die Egg-Installation?

- Inhalt eines Skripts:

```
1  #!/usr/bin/python
2
3  import sys
4  sys.path[0:0] = [
5      '/home/thomas/py/eggs/Sphinx-1.0.8-py2.7.egg',
6      '/home/thomas/py/eggs/docutils-0.8.1-py2.7.egg',
7      '/home/thomas/py/eggs/Jinja2-2.6-py2.7.egg',
8      '/usr/lib/python2.7/dist-packages',
9      ]
10
11 import sphinx.quickstart
12
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- jedes Skript setzt seinen eigenen Python-Pfad
- jedes Skript ruft einen Entry-Point des Eggs auf

Wie funktioniert die Egg-Installation?

- Inhalt eines Skripts:

```
1  #!/usr/bin/python
2
3  import sys
4  sys.path[0:0] = [
5      '/home/thomas/py/eggs/Sphinx-1.0.8-py2.7.egg',
6      '/home/thomas/py/eggs/docutils-0.8.1-py2.7.egg',
7      '/home/thomas/py/eggs/Jinja2-2.6-py2.7.egg',
8      '/usr/lib/python2.7/dist-packages',
9      ]
10
11 import sphinx.quickstart
12
13 if __name__ == '__main__':
14     sphinx.quickstart.main()
```

- jedes Skript setzt seinen eigenen Python-Pfad
- jedes Skript ruft einen Entry-Point des Eggs auf

Wie nutzt man die Eggs im Interpreter?

- Interpreter mit jeweiligem Pfad installieren (Skript)
- Konfiguration:

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

- Buildout-Lauf:

```
$ bin/buildout

Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
Generated interpreter '/home/thomas/py/bin/py'.
```

- Parts bei Änderung der Konfiguration neu erzeugt

Wie nutzt man die Eggs im Interpreter?

- Interpreter mit jeweiligem Pfad installieren (Skript)
- Konfiguration:

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

- Buildout-Lauf:

```
$ bin/buildout

Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
Generated interpreter '/home/thomas/py/bin/py'.
```

- Parts bei Änderung der Konfiguration neu erzeugt

Wie nutzt man die Eggs im Interpreter?

- Interpreter mit jeweiligem Pfad installieren (Skript)
- Konfiguration:

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

- Buildout-Lauf:

```
$ bin/buildout

Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
Generated interpreter '/home/thomas/py/bin/py'.
```

- Parts bei Änderung der Konfiguration neu erzeugt

Wie nutzt man die Eggs im Interpreter?

- Interpreter mit jeweiligem Pfad installieren (Skript)
- Konfiguration:

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

- Buildout-Lauf:

```
$ bin/buildout

Uninstalling sphinx.
Installing sphinx.
Generated script '/home/thomas/py/bin/sphinx-build'.
Generated script '/home/thomas/py/bin/sphinx-quickstart'.
Generated script '/home/thomas/py/bin/sphinx-autogen'.
Generated interpreter '/home/thomas/py/bin/py'.
```

- Parts bei Änderung der Konfiguration neu erzeugt

Gliederung

Welche Egg-Versionen werden installiert?

- deklarierte Versionsabhängigkeiten werden immer erfüllt
- Eggs sollten keine exakten Versionsabhängigkeiten haben
- Eggs im Paketindex und bei weiteren Quellen gesucht
- neueste passende Versionen werden benutzt
- Suche nach neuen Versionen kann unterdrückt werden, das garantiert aber immer noch keine Stabilität

Welche Egg-Versionen werden installiert?

- deklarierte Versionsabhängigkeiten werden immer erfüllt
- Eggs sollten keine exakten Versionsabhängigkeiten haben
- Eggs im Paketindex und bei weiteren Quellen gesucht
- neueste passende Versionen werden benutzt
- Suche nach neuen Versionen kann unterdrückt werden, das garantiert aber immer noch keine Stabilität

Welche Egg-Versionen werden installiert?

- deklarierte Versionsabhängigkeiten werden immer erfüllt
- Eggs sollten keine exakten Versionsabhängigkeiten haben
- Eggs im Paketindex und bei weiteren Quellen gesucht
- neueste passende Versionen werden benutzt
- Suche nach neuen Versionen kann unterdrückt werden, das garantiert aber immer noch keine Stabilität

Welche Egg-Versionen werden installiert?

- deklarierte Versionsabhängigkeiten werden immer erfüllt
- Eggs sollten keine exakten Versionsabhängigkeiten haben
- Eggs im Paketindex und bei weiteren Quellen gesucht
- neueste passende Versionen werden benutzt
- Suche nach neuen Versionen kann unterdrückt werden, das garantiert aber immer noch keine Stabilität

Welche Egg-Versionen werden installiert?

- deklarierte Versionsabhängigkeiten werden immer erfüllt
- Eggs sollten keine exakten Versionsabhängigkeiten haben
- Eggs im Paketindex und bei weiteren Quellen gesucht
- neueste passende Versionen werden benutzt
- Suche nach neuen Versionen kann unterdrückt werden, das garantiert aber immer noch keine Stabilität

Wie legt Buildout Versionen fest?

- globale Option:

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4
5  [versions]
6  sphinx = 1.0.7
7
8  [sphinx]
9  recipe = zc.recipe.egg
10 eggs = sphinx
```

- festgelegte Versionen immer beachtet
- Versionen anderer Pakete immer noch unvorhersagbar

Wie legt Buildout Versionen fest?

- globale Option:

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4
5  [versions]
6  sphinx = 1.0.7
7
8  [sphinx]
9  recipe = zc.recipe.egg
10 eggs = sphinx
```

- festgelegte Versionen immer beachtet
- Versionen anderer Pakete immer noch unvorhersagbar

Wie legt Buildout Versionen fest?

- globale Option:

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4
5  [versions]
6  sphinx = 1.0.7
7
8  [sphinx]
9  recipe = zc.recipe.egg
10 eggs = sphinx
```

- festgelegte Versionen immer beachtet
- Versionen anderer Pakete immer noch unvorhersagbar

Wie legt Buildout Versionen fest?

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4  allow-picked-versions = false
5
6  [versions]
7  Jinja2 = 2.6
8  distribute = 0.6.21
9  docutils = 0.8.1
10 sphinx = 1.0.7
11 zc.buildout = 1.5.2
12 zc.recipe.egg = 1.3.2
13
14 [sphinx]
15 recipe = zc.recipe.egg
16 eggs = sphinx
```

- selbst die Art und Weise der Installation festgeschrieben

Wie legt Buildout Versionen fest?

```
1  [buildout]
2  parts = sphinx
3  versions = versions
4  allow-picked-versions = false
5
6  [versions]
7  Jinja2 = 2.6
8  distribute = 0.6.21
9  docutils = 0.8.1
10 sphinx = 1.0.7
11 zc.buildout = 1.5.2
12 zc.recipe.egg = 1.3.2
13
14 [sphinx]
15 recipe = zc.recipe.egg
16 eggs = sphinx
```

- selbst die Art und Weise der Installation festgeschrieben

Wie vertragen sich unterschiedliche Versionen?

- nur eine Version eines Eggs pro Buildout
- installierte Eggs isoliert von der Außenwelt
- beliebig viele Buildouts können nebeneinander existieren

Wie vertragen sich unterschiedliche Versionen?

- nur eine Version eines Eggs pro Buildout
- installierte Eggs isoliert von der Außenwelt
- beliebig viele Buildouts können nebeneinander existieren

Wie vertragen sich unterschiedliche Versionen?

- nur eine Version eines Eggs pro Buildout
- installierte Eggs isoliert von der Außenwelt
- beliebig viele Buildouts können nebeneinander existieren

Wie pflegt man Versionslisten?

- Versionsliste mit jedem neuen Paket erweitern
- Versionen zu passender Zeit kontrolliert aktualisieren
- Listen beschreiben als gut anerkannte Sätze von Eggs (KGS)
- KGS zusammengehöriger Pakete zentral pflegen
- einzelne Versionen trotzdem im Projekt überschreiben

Wie pflegt man Versionslisten?

- Versionsliste mit jedem neuen Paket erweitern
- Versionen zu passender Zeit kontrolliert aktualisieren
- Listen beschreiben als gut anerkannte Sätze von Eggs (KGS)
- KGS zusammengehöriger Pakete zentral pflegen
- einzelne Versionen trotzdem im Projekt überschreiben

Wie pflegt man Versionslisten?

- Versionsliste mit jedem neuen Paket erweitern
- Versionen zu passender Zeit kontrolliert aktualisieren
- Listen beschreiben als gut anerkannte Sätze von Eggs (KGS)
- KGS zusammengehöriger Pakete zentral pflegen
- einzelne Versionen trotzdem im Projekt überschreiben

Wie pflegt man Versionslisten?

- Versionsliste mit jedem neuen Paket erweitern
- Versionen zu passender Zeit kontrolliert aktualisieren
- Listen beschreiben als gut anerkannte Sätze von Eggs (KGS)
- KGS zusammengehöriger Pakete zentral pflegen
- einzelne Versionen trotzdem im Projekt überschreiben

Wie pflegt man Versionslisten?

- Versionsliste mit jedem neuen Paket erweitern
- Versionen zu passender Zeit kontrolliert aktualisieren
- Listen beschreiben als gut anerkannte Sätze von Eggs (KGS)
- KGS zusammengehöriger Pakete zentral pflegen
- einzelne Versionen trotzdem im Projekt überschreiben

Wie benutzt man Versionslisten?

- Buildout-Konfigurationen können aufeinander aufbauen
- extern gepflegtes KGS nutzen:

```
1 [buildout]
2 extends = http://example.com/versions.cfg
3 parts = sphinx
4 allow-picked-versions = false
5
6 [versions]
7 sphinx = 1.0.7
```

- Inhalt von versions.cfg:

```
1 [buildout]
2 versions = versions
3
4 [versions]
5 Jinja2 = ...
```

Wie benutzt man Versionslisten?

- Buildout-Konfigurationen können aufeinander aufbauen
- extern gepflegtes KGS nutzen:

```
1 [buildout]
2 extends = http://example.com/versions.cfg
3 parts = sphinx
4 allow-picked-versions = false
5
6 [versions]
7 sphinx = 1.0.7
```

- Inhalt von versions.cfg:

```
1 [buildout]
2 versions = versions
3
4 [versions]
5 Jinja2 = ...
```

Wie benutzt man Versionslisten?

- Buildout-Konfigurationen können aufeinander aufbauen
- extern gepflegtes KGS nutzen:

```
1  [buildout]
2  extends = http://example.com/versions.cfg
3  parts = sphinx
4  allow-picked-versions = false
5
6  [versions]
7  sphinx = 1.0.7
```

- Inhalt von versions.cfg:

```
1  [buildout]
2  versions = versions
3
4  [versions]
5  Jinja2 = ...
```

Gliederung

Wie installiert man Nicht-Python-Software?

- Beispiel: LDAP

```
1  [buildout]
2  parts = python-ldap
3
4  [openldap]
5  recipe = zc.recipe.cmmi
6  url = http://.../openldap-2.4.23.tgz
7  extra_options= --disable-slapd --disable-backends
8
9  [python-ldap]
10 recipe = zc.recipe.egg:custom
11 egg = python-ldap
12 include-dirs = ${openldap:location}/include
13 library-dirs = ${openldap:location}/lib
14 rpath = ${openldap:location}/lib
```

Wie installiert man Nicht-Python-Software?

- Rezept für configure/make/make install

```
$ ls parts/openldap/
bin  etc  include  lib  share
```

- Rezept für Eggs mit besonderen Anforderungen beim Bauen
- Abhängigkeiten zwischen Konfigurationsabschnitten

Wie installiert man Nicht-Python-Software?

- Rezept für configure/make/make install

```
$ ls parts/openldap/
bin  etc  include  lib  share
```

- Rezept für Eggs mit besonderen Anforderungen beim Bauen
- Abhängigkeiten zwischen Konfigurationsabschnitten

Wie installiert man Nicht-Python-Software?

- Rezept für configure/make/make install

```
$ ls parts/openldap/
bin  etc  include  lib  share
```

- Rezept für Eggs mit besonderen Anforderungen beim Bauen
- Abhängigkeiten zwischen Konfigurationsabschnitten

Wie hilft Buildout bei großen Systemen?

```
[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}
```

Gliederung

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert

Wie geht es weiter?

- Egg-Unterstützung
  - distribute neben setuptools als Anfang
  - Ziel: wieder nur eine Implementierung
  - distutils2/distribute/setuptools/???
- Python-3-Unterstützung
  - zc.buildout 2.0.0 alpha1
  - größtenteils portiert