Corso web developer junior con Python e Django

Alessandro Dentella

THUX

Alessandro Dentella
adentella@thundersystems.it

Ambienti virtuali

come creare/riprodurre un ambiente

THUX

Alessandro Dentella
adentella@thundersystems.it

Setup.py

- L'installazione di un pacchetto singolo, che sia nel nostro filesystem avviene tramite un comando fra i seguenti:
  python setup.py install
  pip install .
- Il file `setup.py` ha all'interno la configurazione di
  - Nome pacchetto
  - URL sorgente
  - Autore
  - Catalogazione
  - Dipendenze (solo se usa setuptools, caso più comune)
  - Cartelle/file da installare
  - Test
  - ...
- Negli ultimi anni setup.py è stato sostituito da pyproject.toml, che invece che essere un moduo python è una descrizione del progetto, delle dipendenze, ...

Alessandro Dentella
adentella@thundersystems.it

```
from setuptools import setup, find_packages
setup(
    name='jmb.webposte',
    namespace_packages = ['jmb'],
    version = "0.1",
    description='django interface to send italian “Raccomandate”',
    url='http://hg/thundersystems.it/jmb/jmb.webposte',
    author_email='dev@thundersystems.it',
    packages=find_packages(exclude=['tests', 'tests.*']),
    platforms='any',
    classifiers=[
        'Programming Language :: Python',
        'Topic :: Internet :: WWW/HTTP :: Dynamic Content',
    ],
    install_requires = [
        'jmb.core>=0.3.3',
        'jmb.i18n',
    ],
    package_data=get_data_files('jmb.webposte')
)
```

Alessandro Dentella
adentella@thundersystems.it

Limiti di `setup.py`

- Non sempre le dipendenze scritte da terze parti sono corrette (es: non vengono scritte limitazioni non ancora esistenti `django>=1.4`)
- `setup.py` non permette di decidere se un pacchetto deve essere usato come sorgente (ad esempio perché in sviluppo)
- Alcuni pacchetti non sono vere dipendenze, ma sono necessarie solo in sviluppo o nel deploy:
  - django_debug_toolbar, nose, ipdb, Sphinx
  - Psycopg2
  - Gunicorn, Werkzeug
  - ...

Alessandro Dentella
adentella@thundersystems.it

Necessità

- In Python quando viene importato un pacchetto viene scelta l'ultima versione (esistono possibili configurazioni alternative in verità, ma non particolarmente comode...)
- Se in una macchina vogliamo fare coesistere release differenti dobbiamo isolare i 2 ambienti, ovvero creare dei virtualenv differenti. In questi virtualenv, la scelta particolare del `sys.path` regola la visibilità dei pacchetti presenti nel sistema.

Alessandro Dentella
adentella@thundersystems.it

Virtualenv

- Il pacchetto base per fare un ambiente virtuale è il comando `virtualenv` di Ian Biking
- Crea una sottocartella con quel nome e all'interno un pacchetto python ed una cartella bin
- `source bin/activate` attiva quell'ambiente virtuale, ovvero modifica il `PATH` in modo che l'interprete Python lì contenuto sia visto per primo
- Per reimpostare il `PATH`, useremo il comando `deactivate`

Alessandro Dentella
adentella@thundersystems.it

```
sandro@bluff:/tmp$ python3 -c "import ipdb"
sandro@bluff:/tmp$ python3 -m venv test
New python executable in test/bin/python
Installing setuptools, pip...done.
sandro@bluff:/tmp$ cd test
sandro@bluff:/tmp/test$ source bin/activate
(test)sandro@bluff:/tmp/test$ python -c "import ipdb"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named ipdb
(test)sandro@bluff:/tmp/test$ pip install ipdb
Downloading/unpacking ipdb
  Using download cache from /home/sandro/.buildout/download-cache/pip/https%3A%2F
%2Fpypi.python.org%2Fpackages%2Fsource%2Fi%2Fipdb%2Fipdb-0.8.zip
  Running setup.py (path:/tmp/test/build/ipdb/setup.py) egg_info for package ipdb

Downloading/unpacking ipython>=0.10 (from ipdb)
  Using download cache from /home/sandro/.buildout/download-cache/pip/https%3A%2F
%2Fpypi.python.org%2Fpackages%2F2.7%2Fi%2Fipython%2Fipython-2.4.0-py2-none-any.whl
Installing collected packages: ipdb, ipython
  Running setup.py install for ipdb

    Installing ipdb script to /tmp/test/bin
Successfully installed ipdb ipython
Cleaning up...
(test)sandro@bluff:/tmp/test$ python -c "import ipdb"
(test)sandro@bluff:/tmp/test$ pip freeze
argparse==1.2.1
ipdb==0.8
ipython==2.4.0
wsgiref==0.1.2
(test)sandro@bluff:/tmp/test$ deactivate
```

Il python di sistema trova ipdb

Creo un virtualenv che NON ha ipdb

Windows: .\Scripts\activate

Chiedo al virtualenv quali pacchetti ha

virtualenv

- Il virtualenv non deve necessariamente stare nella cartella di lavoro
- Possiamo indicare a pip una cartella dove mettere la cache dei pacchetti scaricati

```
sandro@bluff:~$ cat .pip/pip.conf
[global]
download-cache=~/.buildout/download-cache/pip
```

- È possibile preparare un file di configurazione (es: `requirements.txt`) dove sono contenuti tutti i pacchetti che verranno installati con un solo comando:

```
pip install -r requirements.txt
```

- Questo file di configurazione accetta anche URL di repository hg/git/...

Alessandro Dentella
adentella@thundersystems.it

autoenv

Ogni volta che cambiamo progetto dobbiamo ricordarci di entrare nell'environment. C'è una script di shell che può leggere un file `.env` ed eseguirlo. (leggere le istruzioni sul sito di autoenv)

```
pip install autoenv
```

Ci basta configurare l'env:

```
echo workon my_env > .env
```

per

Alessandro Dentella
adentella@thundersystems.it

Limiti

- Ogni ambiente virtuale copia ogni pacchetto presente
- Ogni volta che cambiamo ambiente virtuale dobbiamo disattivare il vecchio ed attivare il nuovo
- Non più vero da quando è invalso l'uso delle wheels.
  Per ogni pacchetto che abbia una estensione in c (database, lxml, Pillow) la compilazione avviene per ogni ambiente virtuale.

Alessandro Dentella
adentella@thundersystems.it

virtualwrapper

- Fornisce alcuni script per rendere più semplice l'uso degli ambienti virtuali
  - `mkvirtualenv` (crea l'ambiente virtuale ma mette gli ambienti tutti in una cartella unica `$WORKON_HOME`
  - `workon label`: attiva l'ambiente virtuale contraddistinto da quella etichetta
- Con linux
  `apt-get install virtualenvwrapper`

Alessandro Dentella
adentella@thundersystems.it

pyenv

una valida alternativa a virtualenvwrapper è pyenv che lavora in modo differente:

viene installata **una script unica** in tutto il sistema che viene eseguita *al posto* di Python (o pip) e che invoca il python dentro al virtualenv scelto secondo una logica precisa

Alessandro Dentella
adentella@thundersystems.it

pyenv: logica operativa

Possiamo configurare:

- il default per il sistema (pyenv global)
- il python per la shell in corso (pyenv shell)
- il python per la cartella e le sottocartelle (pyenv local)

Alessandro Dentella
adentella@thundersystems.it

Pyenv: interpreti

- è possibile installare velocemente altre versioni di Python
- è possibile creare virtualenv

Alessandro Dentella
adentella@thundersystems.it

Altri ambienti: anaconda

In ambito scientifico aumentano molto le librerie compilate che si usano è nata quindi l'esigenza di un package manager capace di gestire librerie compilate: `conda` è questo package manager, mentre l'ecosistema di chiama `anaconda`.

L'installazione è piuttosto semplice e ben spiegata sul sito ufficiale, una volta installato creare un ambiente virtuale è semplice:

```
conda create -n mio_virtenv anaconda
```

Dove l'ultima parola indica che nel virtualenv devono finire tutti i comandi normalmente presenti nell'ecosistema.

Alessandro Dentella
adentella@thundersystems.it

Installazione sorgenti nell'env

Una volta creato l'environment diventa vitale potere raggiungere il codice che si scrive nell'environment stesso. A questo scopo si può

- Personalizzare l'env usando `PYTHONPATH` per arrivare alla cartella dei sorgenti
- Usare `pip install -e <src-folder>`. L'opzione `-e` crea un file .pth che estende il path del virtualenv alla cartella `<src-folder>`

Alessandro Dentella
adentella@thundersystems.it

conda install

L’installazione di un pacchetto aggiuntivo viene fatta sempre con questo `conda`

```
conda install pystan
```


Alessandro Dentella
adentella@thundersystems.it