Packaging Python Applications

A short tour. At PyCon Balkan 2018-11-17, 16:00

Martin Czygan

About me

- Software developer at Leipzig University Library (475)
- Co-author “Getting Started with Python Data Analysis” (2015)
- Pythonista (and Gopher)

Packaging is not just boring

- unrelated to features, hard to sell
- hundreds of options
- less formalized, project-dependent

So why care?

- Aug 9, 2000: “The Joel Test: 12 Steps to Better Code”

There are two items related to packaging:

- #2 Can you make a build in one step?
- #3 Do you make daily builds?

On item #2

*By this I mean: how many steps does it take to make a **shipping build** from the latest source snapshot? On good teams, there's a single script you can run that does a full checkout from scratch, rebuilds every line of code, **makes the EXEs**, in all their various versions, languages, and #ifdef combinations, creates the **installation package**, and creates the final media — CDROM layout, download website, whatever.*

On item #2

If the process takes any more than one step, it is prone to errors. And when you get closer to shipping, you want to have a very fast cycle of fixing the "last" bug, making the final EXEs, etc. If it takes 20 steps to compile the code, run the installation builder, etc., you're going to go crazy and you're going to make silly mistakes.

Personal experience

- used to write long ant scripts
- most of my projects still have some kind of makefile (code, writing, data)
- running builds on ephemeral VM mirroring production environment
- tried to ease adoption of Python by using various tools (e.g. Linux distribution packages, to have upgrade and downgrade paths)

Probably signs of a slight OCD. But there is some satisfaction in being able to just run *one command*. It's also hard, and might not be worth your time.

Packaging is only a part of a larger story

- the decade of devops (2009-)
- code, build, test, **package**, **release**, configure, monitor

There slides reflect only a few small pieces of the puzzle.

The Packaging Gradient

There is an interesting talk called *The Packaging Gradient* by Mahmoud Hashemi at PyBay 2017 (601), BayPiggies 2017 (82) - blob.

> *One lesson threaded throughout Enterprise Software with Python is that deployment is* ***not the last step of development****.*

What will we look at?

- just do nothing
- from module to package to pypi
- single file deployments (PEP 441)
- reusing linux distribution infrastructure
- images and containers
- frozen software

Is there a best things. No. In the best case some inspiration or awareness of tools.

A single Python file (module)

- with a large standard library, it is possible to write useful things in Python and stdlib only

Deployment cannot get simpler:

```
# curl https://secure.site/script.py > /usr/local/bin && ch
```

Requirements: Python (matching version). Script should have no dependencies. Also, a way to get a script onto the target machine (ssh).

Beautiful, if possible (*simple is better than complex*).

From module to package

- module is a file, package is a directory (containing an `__init__.py`)
- the source can be grouped into a *source distribution* called *sdist*

```
$ python setup.py sdist --formats=gztar,zip
...
creating dist
Creating tar archive
...
```

This includes module, packages, extensions, scripts, readmes, assets and other files that you control through an `MANIFEST.in` file.

From module to package

The sdist file will contain some metadata such as PKG-INFO, a setup.cfg, some .egg-info files.

You can install the `sdist` with pip:

```
$ pip install myproj-0.1.2.tar.gz
```

Or uninstall:

```
$ pip uninstall myproj
```

From module to package

Writing a setup.py file can be simple.

> *The smallest python project is two files. A setup.py file which describes the metadata about your project, and a file containing Python code to implement the functionality of your project.*
>
> *However, there are only three required fields: name, version, and packages.*
>
> *The name field must be unique if you wish to publish your package on the Python Package Index (PyPI). The version field keeps track of different releases of the project.*
>
> *The packages field describes where you've put the Python source code within your project.*

Sources, binaries and uploads

Now that you are happy that you can create a valid source distribution, it's time to upload the finished product to PyPI. We'll also create a bdist_wininst distribution file of our project, which will create a Windows installable file. There are a few different file formats that Python distributions can be created for. Each format must be specified when you run the upload command, or they won't be uploaded (even if you've previously built them previously). You should always upload a source distribution file. The other formats are optional, and will depend upon the needs of your anticipated user base:

```
$ python setup.py sdist bdist_wininst upload
```