Packaging Python Projects in 2023

2023-01-26 @ Pykonik

Hi, I’m Piotr

or GwynBleidD @ discord

https://discord.pykonik.org

Let’s go back in time

Aka (not so) quick history of packaging

2000 - setup.py and distutils

- In standard library of Python 1.6
- Using setup.py executable file
- Offers a way to build and install a package in a “standardized” way, via simple commands
- Just works?

```
from distutils.core import setup
setup (name = "foo",
       version = "1.0",
       py_modules = ["foo"])
```

```
> python setup.py sdist
> python setup.py install
> python setup.py bdist_exe
> python setup.py bdist_rpm
> python setup.py bdist_deb
```

Note: it’s a “modern” example, I’m not going into details of distutils evolution

2001 - Packages Metadata (PEP 241)

- Gives ability to describe a package in a standardized way, providing it’s name, authors, short and long description etc...
- later updated by PEP 314 in 2003
- Support added to distutils

2003 - PyPI - Python Package Index (Cheeseshop)

- Documented since PEP 301 in 2002
- Online since 2003
- Gives the ability to upload Python Packages into one, central location.
- And download them too!
- easy_install ftw!

Problem solved!
We have everything, right?

Right?

Not really…

Problems?

- Packages have to be built on the target PC (or packaged in rpm/deb/exe)
- New features to distutils are hard to release (it’s a part of stdlib!)
- No way to upgrade distutils without Python (again, it’s a part of stdlib!)
- Targeting backwards compatibility for older Python versions makes it even harder
- Anything not supported by distutils needs manual workarounds

2004 - setuptools

- Set of extensions to the distutils

  namespace packages, optional dependencies, automatic manifest building by inspecting version control systems, web scraping to find packages in unusual places, recognition of complex version numbering schemes, and so on, and so on
- Package installation tool
- Pulls in dependencies as well
- Introduces binary pre-built packages! (eggs)
- Allowed the creation of easy_install

Advantages of setuptools

- It’s NOT in a standard lib
- Fast development
- Just a thin wrapper around the distutils!

```
10043 find . -type f -print0 | xargs -0 cat | wc -l
25189
```

All you need use it:

- Replace `from distutils.core import setup` with `from setuptools import setup`

Problem (again) solved!
We have everything, right?

(again) not really…

Just one problem…

It’s NOT in a standard lib

That means you need to install it first…

And ask users of your package to do the same…

2005 - Wacky? solution: easy_install

- First “package manager” for Python
- Built on top of setuptools (that means setuptools included)
- Easy to use

2008 - pip

- Second(?) package manager for Python
- “Real” package manager
- Built on top of setuptools, again (that means setuptools included)
- Even easier to use
- Much more features compared to easy_install

Worth mentioning

- 2006 - buildout - also packages things outside of python
- 2007 - virtualenv
- 2008 - distribute - fork of setuptools (merged back to setuptools in 2013)
- 2013 - wheels
- 2014 - virtualenv now installs setuptools and pip in new envs

Still some problems...

- setuptools backwards compatibility
- What if I need a specific version of setuptools?
- What if I need a specific plugin?
- What if I need anything else?
- What if I don’t want to install anything in my system/environment?

And we need to get rid of setup.py

2016 - pyproject.toml (PEP 518)

- Lists required packages for build process
- Separate ad-hoc virtualenv to execute the build
- Extended (?) by PEP 517 to specify build tool
- Extended by PEP 621 to add metadata
- Can contain any tool configuration
- And probably many more in the future!

So how it looks like?

```
[build-system]
requires = ["setuptools>=40.8.0", "wheel"]
build-backend = "setuptools.build_meta:__legacy__"
```

`requires` - list of packages required for the project build (PEP 518)
`build-backend` - what to execute to build the project (PEP 517)

The `setuptools.build_meta:__legacy__` will just execute the setup.py file

That's it!

`backend-path` - extend the python path with your project

How it works with pip?

- If no pyproject.toml, execute the setup.py, limited build isolation
- If no `[build-system]` in it, assume defaults, full build isolation
- If no `build-backend` in it, error
- If `build-system.build-backend` exists, use this backend to build package

You can force build isolation on or off when installing the package by cmd params

As end-user, you can also

- `--no-build-isolation` - Force build isolation off
- `--no-use-pep517` - force direct setup.py execution
- `--use-pep517` - force `build-backend` (default one if none set)

But if your package has `pyproject.toml` file, it's pep517 build or nothing. There is no setting allowing you to have `pyproject.toml` but not use pep517 builds.

Compliant packaging tools:

- Setuptools
- Poetry
- Flit

Demo time!

It’s a trap!

- Poetry doesn’t follow PEP 508 dependency specification
- In poetry `^` allows to bump minor and patch segments (2nd and 3rd)
- In poetry `~` allows to bump only patch segment (3rd)
  NO MATTER HOW IT’S SPECIFIED!

- In PEP 508 there is no `^`
- In PEP 508 `~` allows to bump last specified segment
  So if you specify `~=2.1.0`, `2.1.1` is allowed, `2.2.0` or `2.2.1` are not
  But if you specify `~=2.1`, both `2.1.1` and `2.2.0` are allowed
  In both cases, `3.0.0` and `2.0.x` are not allowed

It’s not that easy…

- Not every tool supports PEP 621 (standardized project metadata)
- Even if it does, there are some tool-specific options

But it’s still working fairy well :)

Some other tools

Other tools can store config in pyproject.toml too

- Black
- Pytest
- Mypy
- …
- Setuptools-scm
- Poetry-dynamic versioning

Let’s see last 2 in action!

Demo time!

Some additional notes

- In python 3.12 distutils will be REMOVED from stdib
  It will still reside in setuptools for backwards compatibility
- Poetry will also join standardized project metadata at some point
- At some point, pip will build source packages with FULL isolation (without access to global packages)
- You can still use setup.py and pyproject.toml together (just use `setuptools.build_meta:__legacy__` and define everything needed in `build_system.requires`)

Summary

- Pyproject.toml is awesome
- Setup.py is not going anywhere
- Distutils is no longer needed, but that’s not a problem
- It’s handy to keep your version in git tags (and only there)

Thank you!

Questions?