Background / Deployment: PyPI - the package index

Wilber Hernandez
wilberhdez@gmail.com
https://www.linkedin.com/in/wilberhdez26/

Content

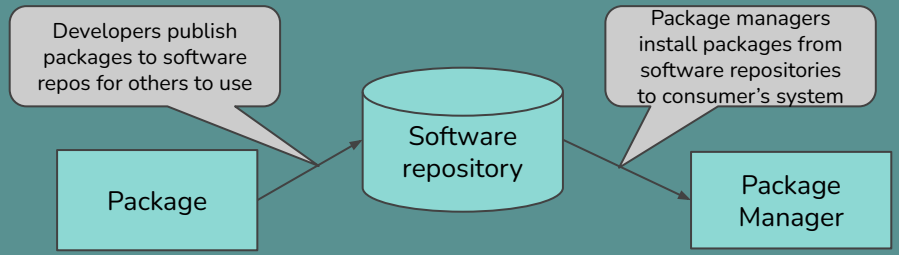
- Intro
  - Python Package
  - Package Management Systems -vs- Package Managers
  - Facts About PyPI
- Contents of a Distribution Package
- Benefits of Packaging
- Python Build Workflow
- The Python Build System
- Distribution Package Artifacts Created
- Metadata between project and distribution package
- A Continuous Integration Pipeline
- Deployment / Publishing Your Package
- PIP Package Manager
- Resources

Python Package

- What is a package?
  - Software and metadata rolled together
- Two concepts defined by the Python Packaging Authority
  - import packages
  - **distribution packages**
    - archiving Python projects for publishing for others to use

Package Management Systems -vs- Package Managers

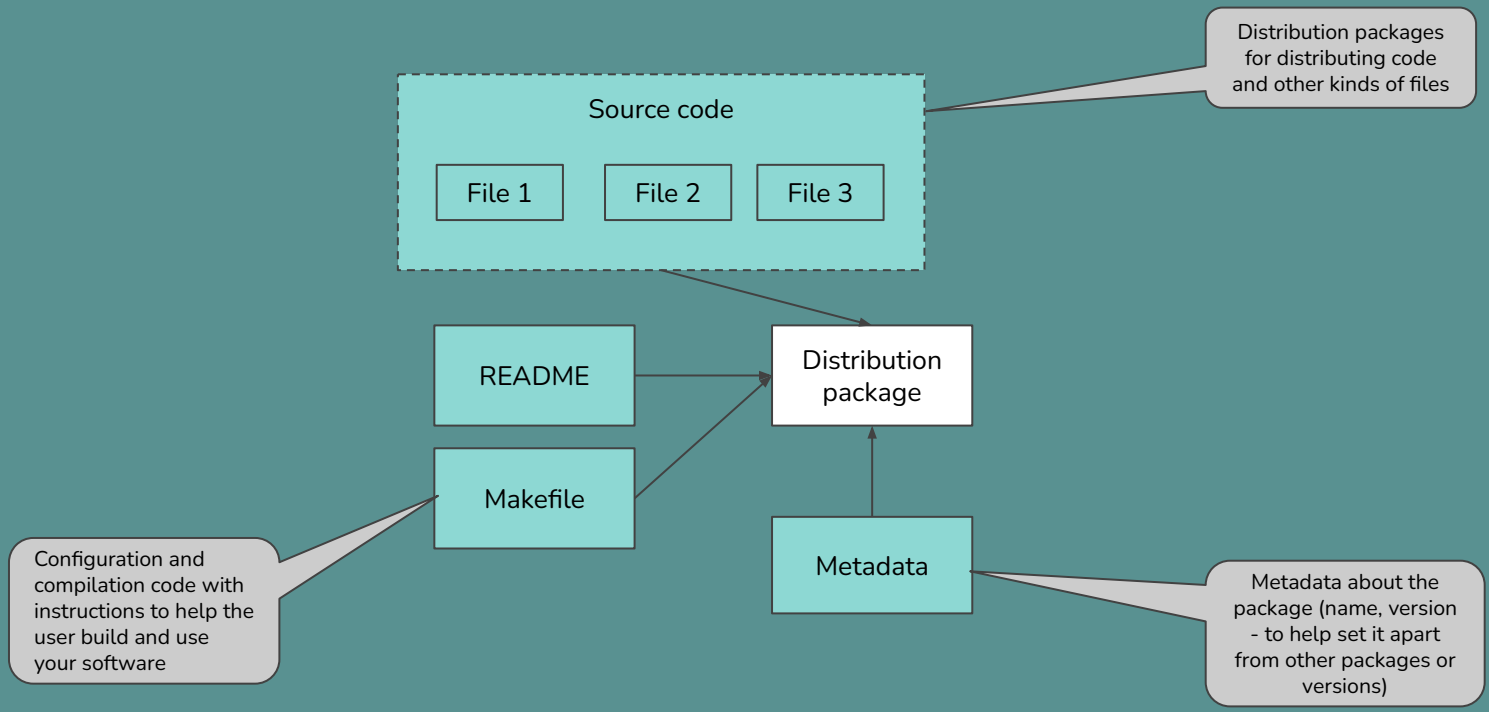
- Package Management System
  - **PyPI (Python Package Index)**
    - Python’s community official repository for installing packages
    - to manage (standardize) the experience of installing and using others’ code
  - Conda-forge
- Package Managers
  - pip
  - conda



Facts about PyPI

- Pronounced "pie pea eye" (aka, the “Cheese Shop”)
- PyPI came about around a decade after Python was released in Feb 1991
- Contribute to PyPI
  - Lots of work at the Warehouse project
  - Donations are appreciated
- Changes announced on
  - pypi-announce mailing list
  - PSF blog (under label “pypi”)

Contents of a Distribution Package



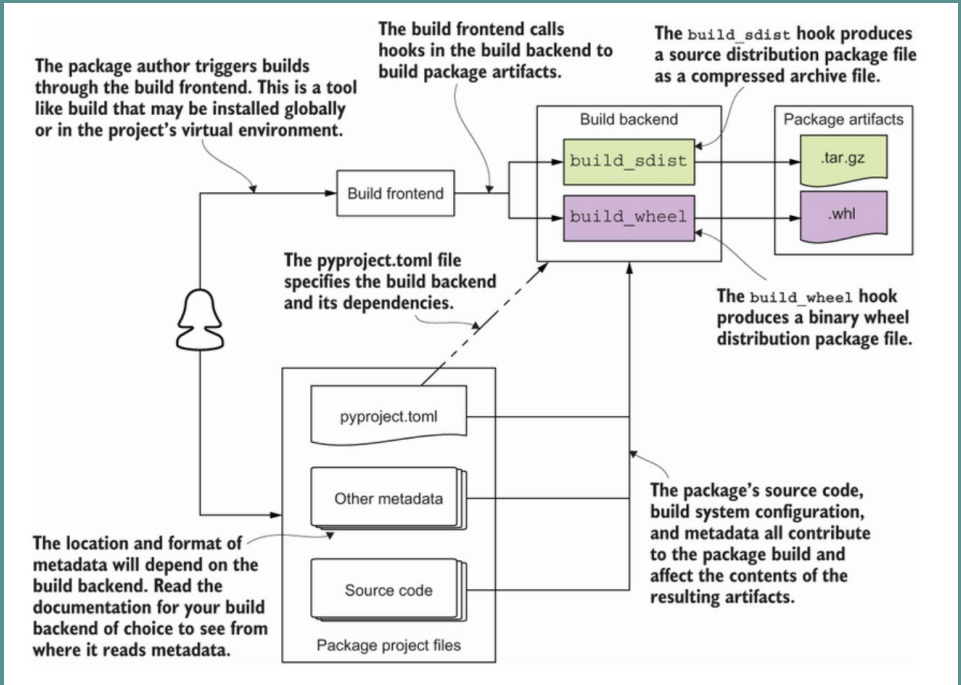
Benefits of Packaging

- Sharing software with people
- Stronger cohesion and encapsulation
- Clearer definition of ownership
- Looser coupling between areas of the code
- More opportunity for composition

Python Build Workflow

- Setup a virtual environment, and install dependencies using pip
- Types of build systems
  - Frontend tool
    - Python package build system ⇒ $ pyproject-build
    - pip
  - Backend tools
    - Traditional:
      - Setuptools
    - Modern:
      - Hatch, Poetry, Flit, Maturin (Rust), Sphinx doc themes

The Python Build System



Distribution Package Artifacts Created

```
$ ls -a1 $HOME/code/first-python-package/
.
..
.venv/
UNKNOWN.egg-info/
build/
dist/
pyproject.toml
```

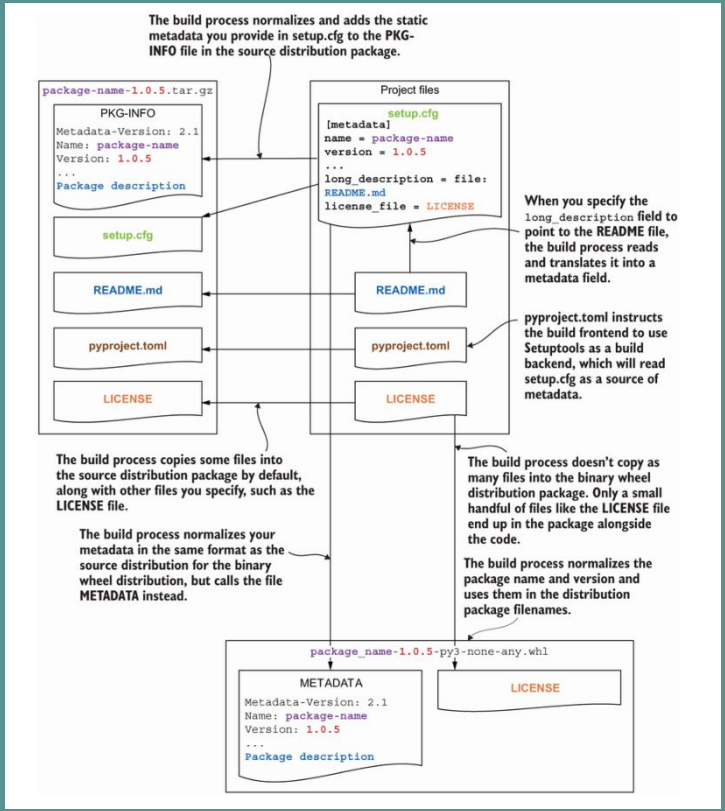
```
$ ls -a1 dist/
.
..
first-python-package-0.0.1.tar.gz
first_python_package-0.0.1-py3-none-any.whl
```

```
$ cd $HOME/code/first-python-package/dist/
$ tar -xzf first-python-package-0.0.1.tar.gz
```

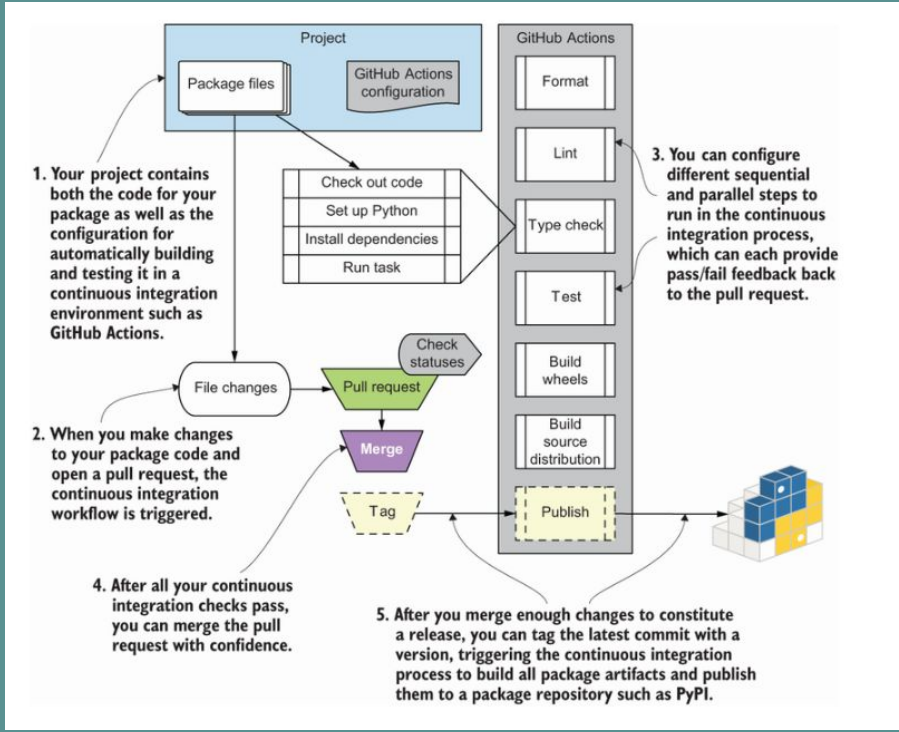
```
$ ls -1R first-python-package-0.0.1/
PKG-INFO
first_python_package.egg-info
pyproject.toml
setup.cfg
```

```
first-python-package-0.0.1/first_python_package.egg-info:
PKG-INFO
SOURCES.txt
dependency_links.txt
top_level.txt
```

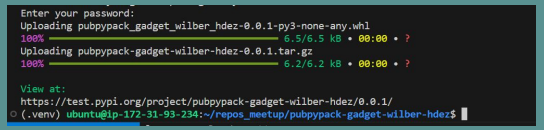
Metadata bet. project and distribution package

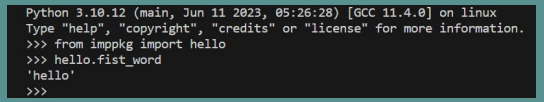


A Continuous Integration Pipeline



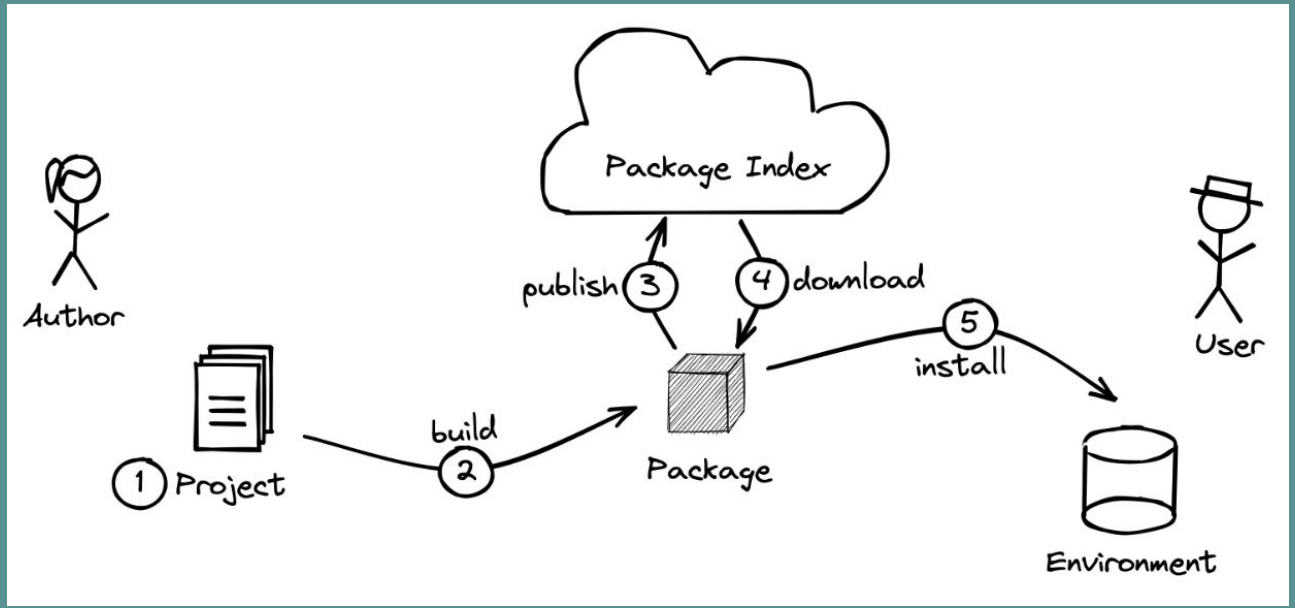
Deployment / Publishing Your Package





- Create an PyPI user account
  - https://pypi.org/account/register/
- “Claim” the package name you want to use on PyPI
  - Check if your package name already exists on PyPI, https://pypi.org
  - Manually upload your package
    - You can test on https://test.pypi.org before trying on live home page
- Use Twine-tool to publish your package, https://twine.readthedocs.io/en/stable/

Package Lifecycle



PIP Package Manager (Configuring PIP)

- Reasons for adding a configuration file:
  - Company hosting a private package index
  - Issues with cache so force PIP to redownload
  - Control where PIP puts downloaded and extracts files

PIP Package Manager (Configuring PIP continues)

- Here's an example of the configuration file (also available in CLI)

```
[global]
# Specify the default index URL. This is where pip looks for packages.
index-url = https://pypi.org/simple/

# Specify additional index URLs. This can be useful for fallback or custom repositor
extra-index-url = https://example.com/simple/
extra-index-url = https://another-mirror.com/simple/

# Ignore the local cache. This means that pip will not use the cache for package ret
no-cache-dir = true

# Set the location where wheels are cached.
wheel-dir = /path/to/custom/wheel/cache
```

This could be your company's internal package index

This ignores the local cache

This puts wheels in a custom location

PIP Package Manager (Configuring PIP continues)

- Why install packages from a local directory?
  - You're developing a package/library and want to make sure it works

```
python -m pip install path/to/SomeProject
```

Keep in mind that you can also install an *editable* version of your package. This lets you develop and test your library without reimporting and reinstalling!

```
python -m pip install -e path/to/SomeProject
```

PIP Package Manager
(When PIP isn't enough)

- The pipenv package is meant to streamline your workflow

Instead of:

python -m venv venv
source venv/bin/activate
pip install requests

+ Manually managing dependencies

You could do:

pipenv install requests

and pipenv will take care of the rest

Q&A

- What is the difference between a module and a package?
- What are absolute and relative imports in Python?
- What is PyPA?
- What is Test PyPI and why do we need it?
- Is an init file a requirement to build a package?

Q&A

- What is the difference between a module and a package?
  - A module is meant to organize functions, variables, and classes into separate Python code files. A Python package is like a folder to organize multiple modules or sub-packages.
- What are absolute and relative imports in Python?
  - Absolute import requires the use of the absolute path of a package starting from the top level, whereas relative import is based on the relative path of the package as per the current location of the program in which the import statement is to be used.
- What is PyPA?
  - The Python Packaging Authority (PyPA) is a working group that maintains a core set of software projects used in Python packaging.
- What is Test PyPI and why do we need it?
  - Test PyPI is a repository of software for the Python programming language for testing purposes.
- Is an init file a requirement to build a package?
  - The init file is optional since Python version 3.3.

Resources

- Books
  - Publishing Python Packages, by Dane Hillard (February 2023)
  - Hypermodern Python Tooling, by Claudio Jolowicz (April 2024)
- Emily Charles (Boston Meetup group)
- Webpage - “Package management: a brief history”, https://blog.tidelift.com/a-brief-history-of-package-management, by by Jeremy Katz (December 19, 2017)
- Sample pyproject.toml metadata file, https://packaging.python.org/en/latest/tutorials/packaging-projects/#configuring-metadata
- Writing Your pyproject.toml file, https://packaging.python.org/en/latest/guides/writing-pyproject-toml/#writing-your-pyproject-toml
- Adding a trusted publisher to an existing PyPI project, https://docs.pypi.org/trusted-publishers/adding-a-publisher/