Python Packaging

2023-01-31

Pipenv

Pipenv is for ... applications

Pipenv might be a great example of porcelain software. It gathers together existing tooling with a sprinkle of policy and philosophy to enable you to work smarter and safer.

The problems that Pipenv seeks to solve are multi-faceted:

- You no longer need to use pip and virtualenv separately. They work together.
- Managing a requirements.txt file can be problematic, so Pipenv uses Pipfile and Pipfile.lock to separate abstract dependency declarations from the last tested combination.
- Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.
- Strongly encourage the use of the latest versions of dependencies to minimize security risks arising from outdated components.
- Give you insight into your dependency graph (e.g. $ pipenv graph).
- Streamline development workflow by loading .env files.

---

https://pipenv.pypa.io/en/latest/index.html

And what about my libraries?

When you're focusing on determinism and supply-chain security for your application deployments, Pipenv can be an indispensable tool. With the proper tooling, it's quite straightforward to capture and encode the characteristics of your requirements, ensure that they're stored in such a way that automated tooling can help alert you to vulnerabilities and provide a simple and repeatable mechanism for ensuring that all deployments use the exact set of requirements considered compatible and secure.

For libraries, define abstract dependencies via install_requires in setup.py. The decision of which version exactly to be installed and where to obtain that dependency is not yours to make!

---

https://pipenv.pypa.io/en/latest/advanced/#pipfile-vs-setup-py

Enhance!

PEP 517, 518 … 621, Oh, my!

TLDR; In the beginning, Python packaging was a source of sanity. Eventually the design, made for a different, simpler time, began to show its limits.

The following PEP represent an (incomplete) summary of community discussions on how packaging in Python can be guided into the future and define several new interfaces:

- A build-system independent interface (*i.e.*, package your application without setuptools) https://peps.python.org/pep-0517/
- A new interface for defining dependencies (*i.e.*, define your requirements without setuptools) https://peps.python.org/pep-0518/
- A specification for writing project metadata https://peps.python.org/pep-0621/ https://packaging.python.org/en/latest/specifications/declaring-project-metadata/#declaring-project-metadata
- Using editable requirements in a PEP 517 world https://peps.python.org/pep-0660/

Why all the fuss?

1. Software supply-chain security!
2. Deterministic deployments!
3. Portability!
4. Reduce false positives and noise from automated systems!
5. Fun!
6. Profit!

Further reading

https://setuptools.pypa.io/en/latest/userguide/quickstart.html

https://python-poetry.org

https://www.pantsbuild.org/

https://flit.pypa.io/en/stable/

https://pypi.org/project/enscons/