Pourquoi utiliser Poetry

Pourquoi utiliser Poetry pour gérer un projet Python ?

Poetry est un outil de gestion de projet Python qui centralise la gestion des dépendances, l'environnement virtuel, et le packaging dans une seule interface cohérente. Contrairement aux outils classiques comme pip + virtualenv ou pipenv, Poetry offre une approche plus rigoureuse et professionnelle pour développer, partager ou déployer des projets Python.

Gestion des dépendances précise et fiable

- Poetry gère automatiquement un fichier `poetry.lock` qui **fige toutes les versions des dépendances**, garantissant que l'environnement soit strictement le même pour chaque utilisateur ou environnement (CI, prod, etc.).
- Il résout les conflits de dépendances de manière claire et fiable.

Isolation automatique de l'environnement

- À chaque projet, Poetry crée un **environnement virtuel dédié**, isolé du système. Pas besoin de `virtualenv` ou d'activer quoi que ce soit manuellement.
- Cela garantit qu'un projet A ne casse pas un projet B à cause d'un conflit de versions.

Syntaxe simple et explicite avec `pyproject.toml`

- Toutes les métadonnées du projet (nom, version, auteurs, dépendances, scripts, etc.) sont centralisées dans un seul fichier : `pyproject.toml`.
- Ce format est **standardisé** (PEP 518) et utilisé par de nombreux outils modernes.

Packaging et publication facilités

- Poetry permet de **packager et publier facilement** un module ou une application sur un registre (comme PyPI).
- Il gère tout le cycle de vie du projet, de l'installation à la publication.

Utilisation fluide en équipe et en CI/CD

- Grâce à `poetry.lock`, tous les développeurs d'une équipe ou une machine CI utilisent exactement le même environnement.
- Aucune surprise entre développement local et tests automatisés.

Exemple de projet Python simple avec Poetry

Prenons un exemple concret : un petit script Python qui se connecte à une base PostgreSQL pour extraire et afficher des données. Voici comment l'organiser avec Poetry.

Structure du projet

```
mon-projet/
│
├── app/
│   └── main.py
│
├── pyproject.toml
├── poetry.lock
└── docker-compose.yml
```

Initialisation du projet avec Poetry

```
git:(master) poetry init
```

Poetry vous pose quelques questions (nom du projet, description, etc.), puis génère un fichier pyproject.toml.

Ajout de dépendances

```
git:(master) poetry add pandas sqlalchemy psycopg2-binary
```

Exemple de pyproject.toml

```toml
[tool.poetry]
name = "mon-projet"
version = "0.1.0"
description = "Exemple de projet Poetry"
authors = ["Ton Nom <tonmail@example.com>"]

[tool.poetry.dependencies]
python = "^3.11"
pandas = "^2.2"
sqlalchemy = "^2.0"
psycopg2-binary = "^2.9"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Exemple de `main.py`

```python
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("postgresql://myuser:mypassword@db:5432/mydb")
df = pd.read_sql("SELECT * FROM ventes", engine)
print(df.head())
```

Exemple de docker-compose.yml

```yaml
version: '3.9'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - db
    volumes:
      - .:/app
    working_dir: /app
    command: poetry run python app/main.py
    networks:
      - backend

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - backend

volumes:
  pgdata:

networks:
  backend:
```

Dockerfile minimal pour Poetry

```
FROM python:3.11-slim

RUN pip install --no-cache-dir poetry

WORKDIR /app

COPY pyproject.toml poetry.lock* ./
RUN poetry install --no-root

COPY . .

CMD ["poetry", "run", "python", "app/main.py"]
```

Lancement

```
git:(master) docker-compose up --build
```

Conclusion

Poetry simplifie et professionnalise la gestion d’un projet Python. Il apporte une rigueur comparable à ce qu’offrent des langages comme Rust ou Node.js avec leur gestionnaire de paquets. Moins d’erreurs, une installation reproductible, un packaging propre : Poetry s’impose comme l’outil idéal pour tout projet Python sérieux.