Architecture micro-service

Point tech - Pipenv

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Compétence outil : gérer ses dépendances avec `pipenv`

Dans le TP, chaque micro-service Python a ses propres dépendances (Flask, requests, PyJWT, etc). Pour les gérer proprement, **on utilise `pipenv`** : un outil moderne qui combine un environnement virtuel + un fichier de dépendances.

1.1 Pourquoi utiliser `pipenv` ?

- Crée automatiquement un **environnement virtuel** isolé
- Gère deux fichiers : `Pipfile` (dépendances) et `Pipfile.lock` (versions gelées)
- Plus propre que `pip install` global

1.2 Commandes utiles

1.2.1 Créer un projet et installer une lib (ex : Flask)

```
pipenv install flask
```

Cela crée :

- `Pipfile` → liste des paquets
- `Pipfile.lock` → versions exactes

1.2.2 Lancer un shell dans l'environnement

```
pipenv shell
```

1.2.3 Installer un autre paquet (ex : PyJWT)

```
pipenv install pyjwt
```

1.2.4 Lister les paquets

```
pipenv graph
```

1.3 Et pour Docker ?

Les conteneurs Python ont besoin d'un fichier `requirements.txt`. Voici comment le générer depuis `pipenv` :

```
pipenv lock --requirements > requirements.txt
```

Cela crée un fichier standard `requirements.txt` compatible avec tous les `Dockerfile` Python.

1.4 Exemple de `Dockerfile` avec pipenv converti

```
FROM python:3.10-slim

WORKDIR /app
COPY . .

# Installer les dépendances via requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

CMD ["python", "app.py"]
```

2 Bonnes pratiques

— Ne versionnez pas `.venv/` → ajoutez-le à `.gitignore`
— Pensez à régénérer `requirements.txt` après chaque changement
— Vous pouvez aussi ajouter `pipenv install --dev pytest` pour les tests