Explication ligne par ligne de mon Dockerfile

```
FROM python:3.9-alpine3.13
```

- Base de l'image Docker : Python 3.9 sur Alpine Linux 3.13 (image très légère).

```
LABEL maintainer="Axel.com"
```

- Ajoute une métadonnée « maintainer » pour indiquer qui maintient cette image.

```
ENV PYTHONUNBUFFERED 1
```

- Configure Python pour qu'il n'utilise pas de buffering sur stdout/stderr, ce qui facilite le debug/logs en temps réel dans Docker.

```
COPY ./requirements.txt /tmp/requirements.txt
COPY ./requirements.dev.txt /tmp/requirements.dev.txt
COPY ./app /app
```

- Copie les fichiers requirements.txt et requirements.dev.txt dans le dossier temporaire `/tmp` de l'image.
- Copie le dossier `app` dans `/app` dans l'image.

```
WORKDIR /app
```

- Définit le répertoire de travail pour toutes les commandes suivantes (CMD, RUN, etc.) à `/app`.

```
EXPOSE 8000
```

- Indique que le conteneur écoute sur le port 8000 (celui utilisé souvent par Django).

```
ARG DEV=false
```

- Définit une variable d'argument (qui peut être passée au build) pour indiquer si on est en mode dev ou pas.

```
RUN python -m venv /py && \
    /py/bin/pip install --upgrade pip && \
    apk add --update --no-cache postgresql-client jpeg-dev && \
    apk add --update --no-cache --virtual .tmp-build-deps \
        build-base postgresql-dev musl-dev zlib zlib-dev && \
    /py/bin/pip install -r /tmp/requirements.txt && \
    if [ "$DEV" = "true" ]; then \
        /py/bin/pip install -r /tmp/requirements.dev.txt ; \
    fi && \
    rm -rf /tmp && \
    apk del .tmp-build-deps && \
    adduser --disabled-password --no-create-home \
    django-user && \
    mkdir -p /vol/web/media && \
    mkdir -p /vol/web/static && \
    chown -R django-user:django-user /vol && \
    chmod -R 755  /vol
```

- Crée un environnement virtuel Python dans `/py`.
- Met à jour pip dans cet environnement.
- Installe `postgresql-client` et `jpeg-dev` (packages nécessaires à ta stack).
- Installe un groupe temporaire de paquets de compilation nécessaires à la compilation de dépendances Python (build-base, postgresql-dev, etc.).
- Installe les dépendances Python listées dans requirements.txt.
- Si `DEV=true`, installe aussi les dépendances de dev.
- Supprime le dossier temporaire `/tmp` pour alléger l'image.
- Supprime les paquets de compilation (gain de poids et sécurité).
- Crée un utilisateur `django-user` sans mot de passe ni dossier home.
- Crée les dossiers `/vol/web/media` et `/vol/web/static` (pour les fichiers médias et statiques).
- Change la propriété de `/vol` et ses sous-dossiers à `django-user`.
- Définit les permissions à 755 sur `/vol`.

```
ENV PATH=/py/bin:$PATH
```

- Ajoute le dossier des binaires de l'environnement virtuel `/py/bin` dans la variable PATH pour que les commandes Python et pip pointent vers cet environnement.

```
USER django-user
```

- Exécute les prochaines commandes et le conteneur avec l'utilisateur non root `django-user` (meilleure sécurité).

Profil minimal d'un Dockerfile pour une app Django

```
FROM python:3.9-slim

ENV PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirements.txt .

RUN pip install --upgrade pip && pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Commandes essentielles Dockerfile (à retenir)

- `FROM` : base de l'image
- `LABEL` : métadonnées
- `ENV` : variables d'environnement
- `WORKDIR` : dossier de travail
- `COPY` / `ADD` : copier des fichiers dans l'image
- `RUN` : exécuter une commande lors du build (installer des dépendances, créer des dossiers, etc.)
- `EXPOSE` : indiquer un port exposé
- `CMD` : commande par défaut à exécuter au démarrage du conteneur
- `ENTRYPOINT` : pour configurer un exécutable principal (optionnel)
- `USER` : définir l'utilisateur qui lance le conteneur (important pour la sécurité)
