MEthodology in Data Science

Docker pour le Machine Learning et MLOps

Olivier Schwander
<olivier.schwander@sorbonne-universite.fr>

Master MIND
Sorbonne Université

2025-2026

Motivation

Développement

- Python 3.11 avec librairies spécifiques
- Conda/Poetry/uv pour les dépendances
- Une machine ubuntu/windows/mac
- Tout fonctionne en local

Production

- Serveur Debian avec Python 3.8
- Conflits de versions potentiels
- (Pensez à tous les soucis à PPTI)

Reproductiblité

Avantages

- Même environnement partout (dev, test, prod)
- Contrôler les versions des dépendances
- Déploiement fiable et rapide

Pour l'IA

- Reproductibilité des expériences
- Partage de modèles, des architectures, des poids
- Intérêt scientifique aussi: recherche reproductible

Dépendances et déploiements

Chaîne de dépendances

- CUDA, cuDNN pour le GPU
- pyTorch/TensorFlow et autres
- Autres bibliothèques (FFmpeg, OpenCV)

Scalabilité horizontale

- en train: même environnement pour tous les nœuds de train
- en inférence: besoin d'ajouter des nœuds facilement en cas de pic de demande

Devops et MLops

Devops (entre autre)

- Le déploiement, c'est du code comme tout le reste
- Images préconstruites avec tout déjà installé, juste à recopier
- Description de l'état attendu du système (ansible)

MLops

- Pareil pour les modèles et les poids
- Facilite la mise en production des modèles entraînés
- Numéros de version, tag, *hub* pour télécharger

Machines virtuelles

- OS complet invité: Linux, Windows (plus de possibilités)
- Hyperviseur (VirtualBox, VMware)
- Lourd, lent au démarrage
- Isolation forte (plus de sécurité) mais coût élevé
- Image: système complet (distribution installée de façon classique, paquets installés, etc)

Containers

- Partage du kernel hôte (linux-only)
- Léger, démarrage en quelques secondes
- Isolation des processus, des utilisateurs, des systèmes de fichier
- Image: distribution légère, + application + dépendances

En pratique

- **Docker**
- (mais il y a des alternatives)

Image docker

- Base immutable (système de base, outils linux de base)
- Ajouts de couches
- **Dockerfile**: règles pour décrire une nouvelle couche (installation de votre application)
- Distribué dans un registre

Un container qui tourne

- Instance d'une image (un même container peut tourner plusieurs fois, avec des données différentes)
- Processus au sens kernel, mais isolé du système hôtes (et entre eux)
- Éphémère par défaut: jetable (pas d'administration système classique, on fait évolue le Dockerfile)
- Données persistantes possibles (datasets, poids, etc)

Quelques commande: démo

```
docker pull python:3.11-slim

docker run -p 8080:80 -ti python:3.11-slim

docker ps -a
```

Architecture typique

Composants

- Votre application: API REST de prédiction
- Base de données (PostgreSQL, MongoDB)
- File d'attente (Redis, RabbitMQ)
- Serveur de modèles (Ollama, vLLM)

Votre application

- Juste l'API
- Le reste c'est des services autour
- Besoin d'orchestration pour gérer ça

Structure du projet

```
my-ml-app/
  - app/
    - __init__.py
    - main.py
    - model.py
  - pyproject.toml
  - uv.lock
  - Dockerfile
```

pyproject.toml

```toml
[project]
name = "my-ml-app"
version = "0.1.0"
requires-python = ">=3.11"
dependencies = [
    "fastapi>=0.109.0",
    "uvicorn>=0.27.0",
    "pydantic>=2.5.0",
    "numpy>=1.26.0",
]
```

Dockerfile

```
FROM ghcr.io/astral-sh/uv:python3.11-bookworm-slim

WORKDIR /app
COPY . . # Copie du code source

RUN uv sync --frozen

ENV PATH="/app/.venv/bin:$PATH"

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--por
```

Dockerfile multi-stage

```
FROM ghcr.io/astral-sh/uv:python3.11-bookworm-slim AS build

# [...] Instructions de build

# Stage 2: Runtime
FROM python:3.11-slim-bookworm

WORKDIR /app
COPY --from=builder /app/.venv /app/.venv
COPY --from=builder /app/app /app/app

# [...] Instructions de run
```

Pourquoi multi-stage ?

Taille

- Image finale légère
- Pas les outils de build

Sécurité

- Surface d'attaque plus faible
- Seulement le nécessaire pour l'exécution
- Meilleure sécurité

Composants

- **App API** (votre application)
- **Base de données** (PostgreSQL)
- **Serveur LLM** (Ollama)

docker-compose.yml

```yaml
services:
  api:
    # [...] Container poru votre application

  db:
    # [...] Base données SQL

  ollama:
    #[...] Serveur d'inférence

volumes:
  postgres_data:
  ollama_data:
```

docker-compose.yml

```yaml
api:
  build: . # Votre applicaiton si elle n'est pas sur le
  ports:
    - "8000:8000"
  depends_on:
    - db
    - ollama
  environment:
    - DATABASE_URL=postgresql://user:pass@db:5432/mydb
    - OLLAMA_URL=http://ollama:11434
```

docker-compose.yml

```yaml
db:
  image: postgres:15-alpine
  environment:
    - POSTGRES_USER=user
    - POSTGRES_PASSWORD=pass
    - POSTGRES_DB=mydb
  volumes:
    - postgres_data:/var/lib/postgresql/data
```

docker-compose.yml

```yaml
ollama:
  image: ollama/ollama:latest
  ports:
    - "11434:11434"
  volumes:
    - ollama_data:/root/.ollama
  command: ["ollama", "serve"]
```

Commandes Docker Compose: démo

```
docker compose up -d

docker compose logs -f

docker compose down

docker compose up -d --force-recreate
```

Différents outils

Docker tout seul

- Application simple: un seul processus

Docker Compose

- Déploiement complee: application qui utilise plusieurs services
- Un seul hôte

Docker Swarm

- Multi-hôtes: plusieurs nœuds pour mieux passer à l'échelle
- Load balancing intégré
- Haute disponiblité

Kubernetes

- Orchestration avancée
- Auto-scaling

MLOps methodoloy

Engineering techniques

- SysOps: infrastructre as code
- DevOps: software development as code
- MLOps: ML as code

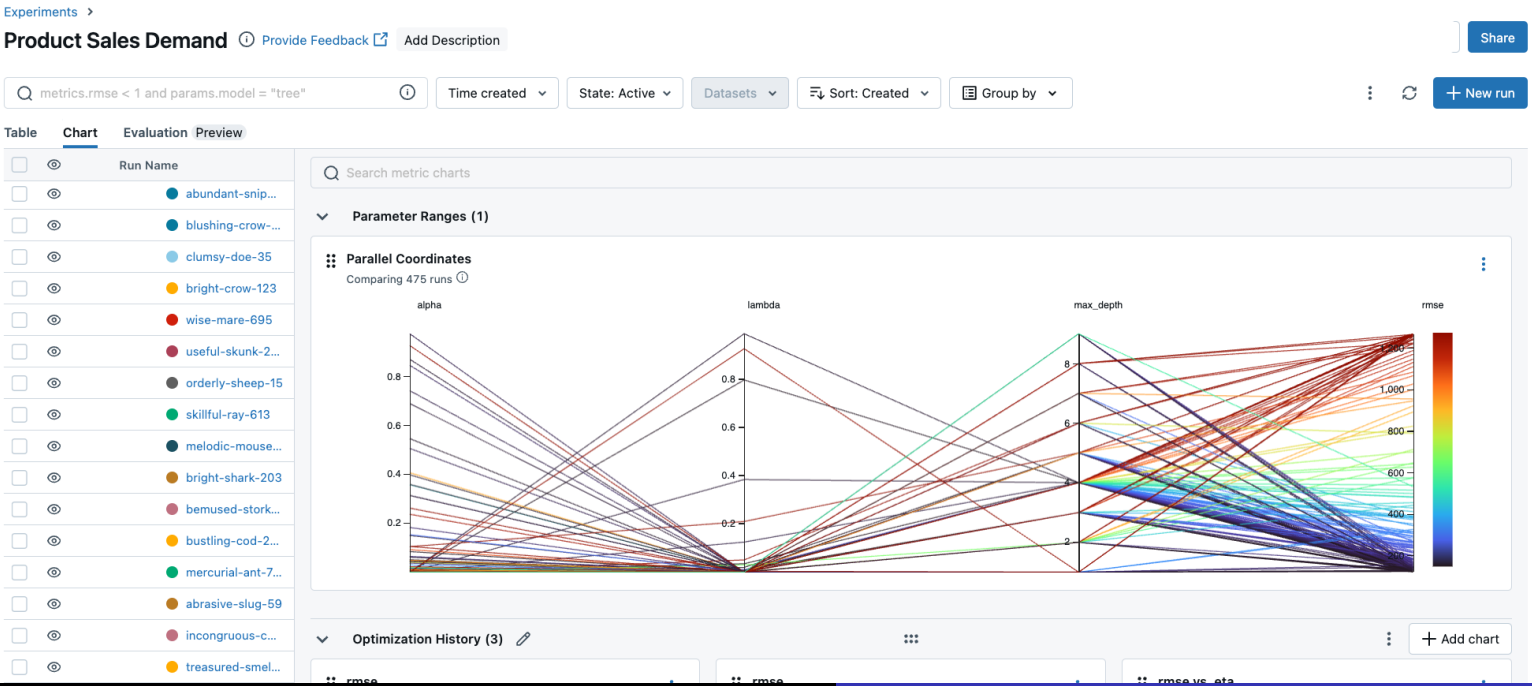
Keypoints

- Data pipelines: building, validation, deployment
- Model training pipelines: **reproducible** experiments, hyperparameter tuning
- Model **versioning**: tracking models, datasets, and code together
- Model **deployment**: putting models in production
- Model monitoring: performance drift, data drift, bias detection, odd cases
- Automated retraining: scheduled or drift-triggered

MLflow

Features

- Record hyperparameters and model
- Plot learning curves and other figures
- Model register to distribute models
- Version number, tags
- Dashboard to monitor training and usage of models



MLflow integration

```
with mlflow.start_run():
    params = {
        "epochs": epochs,
        "learning_rate": 1e-3,
        "batch_size": 64,
        "optimizer": "SGD",
    }
    mlflow.log_params(params)
    model = ... # Training here
    mlflow.pytorch.log_model(model, "model")
```

Conclusion

Besoin de reproductibilté

- Science: validtié des publications
- Développement/prototypage/train: relancer facilement des tâches
- Production/inférence: déploiement facile des modèles

Docker

- Image qui contient tout le nécessaire
- Référence pour le déploimeent de toutes les applications web de nos jours
- Utile aussi en ML

MLops

- Méthologie pour gérer les modèles entraînés
- Tout simplement: savoir ce qu'on utilise en production