TP2 – Déploiement d’une stack multi-services avec Docker Compose

Contexte

Dans ce TP, vous allez déployer une application **multi-services** à l’aide de **Docker Compose**.
L’objectif est de comprendre comment orchestrer plusieurs conteneurs (application, base de données, services annexes) au sein d’une même stack, en respectant les bonnes pratiques de configuration, de persistance et de réseau.

Ce TP s’inscrit dans une logique **Data / Backend / Infrastructure**, proche des environnements professionnels.

Objectifs pédagogiques

À l’issue de ce TP, l’étudiant sera capable de :

- Déployer plusieurs services avec Docker Compose
- Comprendre la communication inter-conteneurs
- Externaliser la configuration via des variables d’environnement
- Mettre en place la persistance des données avec des volumes
- Diagnostiquer le fonctionnement d’une stack Docker (logs, exec, healthchecks)

Stack technique imposée

La stack devra contenir **au minimum** les services suivants :

- **API Backend** : FastAPI
- **Base de données** : PostgreSQL
- **Outil d’administration de la base** : Adminer ou pgAdmin

- **Service complémentaire** : Redis

Architecture cible (logique)

- L'API communique avec PostgreSQL via le réseau Docker
- PostgreSQL stocke ses données dans un volume persistant
- Redis est utilisé comme cache ou service annexe
- L'outil d'administration permet de visualiser la base de données
- Tous les services sont orchestrés via un unique fichier `docker-compose.yml`

Arborescence attendue

```
tp2-docker-compose/
│
├── app/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── main.py
│
├── docker-compose.yml
├── .env
└── README.md
```

Contraintes techniques obligatoires

Votre stack doit respecter les points suivants :

1. Le fichier `docker-compose.yml` doit contenir **au moins 3 services**
2. Les paramètres sensibles doivent être stockés dans un fichier `.env`
3. La base de données doit utiliser un **volume Docker** pour la persistance

4. Un **réseau Docker** dédié doit être défini
5. Au moins un service doit avoir un **healthcheck**
6. L'API doit exposer un endpoint de test (`/health` par exemple)

Fonctionnalités minimales attendues

API FastAPI

- Endpoint `/health` retournant un statut `OK`
- Endpoint permettant de vérifier la connexion à la base de données

Base de données

- PostgreSQL configurée via variables d'environnement
- Données persistantes après redémarrage des conteneurs

Administration

- Accès à Adminer ou pgAdmin via un navigateur

Commandes à maîtriser

Les commandes suivantes doivent être utilisées et comprises :

```
docker compose up -d --build
docker compose ps
docker compose logs -f
docker compose exec <service> bash
docker compose down
docker compose down -v
```

Tests à réaliser

- Accéder à l'API via le navigateur ou `curl`
- Vérifier le statut `/health`
- Vérifier l'accès à la base depuis l'outil d'administration
- Redémarrer la stack et vérifier que les données sont conservées

---

Livrables attendus

À rendre dans un dossier compressé ou dépôt Git :

- `docker-compose.yml` commenté
- Code de l'API (FastAPI)
- Fichier `.env` (sans secrets réels)
- `README.md` contenant :
    - Instructions de lancement
    - Ports exposés
    - Méthode de test
    - Problèmes rencontrés / solutions