TP Noté : Orchestration, Résilience et Industrialisation Docker

Niveau : Master 2
Durée estimée : 3h - 4h
Contexte : Architecture Microservices, Docker Compose, Python FastAPI, Nginx, Redis, Postgres.

1. Contexte et Objectifs

Vous récupérez le "Proof of Concept" (POC) d'une application existante. Cette stack a été développée rapidement : elle est instable, tourne en root et ne gère pas la résilience.

L'équipe précédente a déjà mis en place un début de persistance avec un script migration-v001.sql qui initialise une table métier spécifique.

Votre mission : Transformer ce POC en une stack de production **résiliente**, **sécurisée** et capable de supporter la nouvelle fonctionnalité "Dashboard Étudiant".

L'Architecture Cible

L'application finale devra respecter l'architecture 3-tiers suivante :

1. **Frontend** : Un serveur Nginx servant une SPA statique (fichier fourni).
2. **Backend** : Une API Python FastAPI existante (à adapter).
3. **Data Layer** :
    - **Postgres** : Doit contenir les données existantes + la nouvelle table students.
    - **Redis** : Pour le comptage de vues en temps réel.

2. Ressources Fournies

Arborescence attendue :

```
/project-root
├── docker-compose.yml
├── python.Dockerfile
├── requirements.txt
├── sqlfiles/
│   └── migration-v001.sql
├── app/
│   └── main.py          (Anciennement postgres-test.py)
└── frontend/
    └── index.html       (Contrat d'interface fourni ci-dessous)
```

Le Contrat d'Interface (Frontend)

Le fichier frontend/index.html est immuable. **Interdiction de modifier le JS.** Votre infrastructure backend doit s'adapter pour répondre aux appels de ce client.

(Récupérez le code HTML en annexe).

3. Travail à réaliser

Étape 1 : Exposition du Frontend

L'application ne doit pas être ouverte en local (problèmes CORS).

- Créez un service de reverse proxy pour servir l'application sur le port 8080 qui va servir le fichier html annexe.
- Suivez les bonnes pratiques Docker.

Validation : http://localhost:8080 affiche le dashboard.

Étape 2 : Stratégie d'initialisation de données (Postgres)

Sur la base de votre script de migration existant vous allez :

1. Ajoutez un nouveau script d'initialisation pour créer la table students et y insérer des données de test.
2. Configurez le service Database pour que **tous** les scripts .sql (l'existant et le vôtre) soient joués au premier démarrage.
3. **Contrainte forte :** La persistance des données doit être garantie.

Étape 3 : Logique Métier et Cache (Redis)

Actuellement, l'API renvoie uniquement la version de la base de données sur la route /. Vous devez implémenter la logique métier attendue par le frontend.

1. **Intégration Redis :** Ajoutez le service Redis à la stack (avec persistance).
2. **Modification de l'API (main.py) :**
   - Modifiez la route racine / .
   - Elle doit désormais faire une requête SQL (SELECT) pour récupérer la liste des étudiants.
   - Pour chaque requête, incrémentez un compteur atomique dans Redis.
3. **Format de réponse :** L'API doit renvoyer une liste d'objets JSON incluant les données Postgres ET le compteur Redis.

   Indice : Référez-vous à la documentation de la librairie redis-py.

Étape 4 : Orchestration et Résilience (Critique)

L'API plante actuellement si la DB n'est pas prête. Ce comportement est inacceptable en prod. Mettez en place des mécanismes de sécurité entre les différents services.

Graceful Degradation (Code) : Modifiez le code Python. Si le service de Cache (Redis) est indisponible (crash, surcharge), l'API **ne doit pas** renvoyer d'erreur 500. Elle doit renvoyer la liste des étudiants avec une valeur par défaut pour les vues (ex: 0 ou null).

- *Test : Coupez Redis (docker stop) et vérifiez que le dashboard s'affiche toujours.*

Étape 5 : Durcissement et Sécurité

1. **Moindre Privilège :** Les conteneurs ne doivent pas tourner en root. Modifiez le Dockerfile de l'API pour utiliser un utilisateur système dédié.
2. **Isolation Réseau :**
   - Seuls les points d'entrée nécessaires (Frontend:8080, API:8000) doivent être exposés sur l'hôte.
   - La base de données et le cache ne doivent être accessibles que via le réseau interne Docker.

4. Livrables

À déposer sur Moodle :

- Un fichier texte contenant le lien vers votre **dépôt Git** (GitHub, GitLab, etc.).
- Assurez-vous que le dépôt est **public** ou que l'enseignant a les droits d'accès.
- Votre dépôt doit contenir un README.md expliquant comment lancer le projet (commandes, pré-requis).

Critères d'évaluation (Expertise M2) :

- **Fonctionnel :** La stack démarre en une commande up et le dashboard est opérationnel après un git clone.
- **Résilience :** Gestion des dépendances de démarrage et tolérance aux pannes (Redis).
- **Utilisation de la documentation :** Utilisation correcte des variables d'environnement et des mécanismes natifs des images officielles.
- **Qualité :** Propreté du Dockerfile, .gitignore pertinent (pas de fichiers inutiles/env), gestion des secrets.

Annexe : Fichier Frontend Immuable

Sauvegardez ce contenu dans frontend/index.html.

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Student Dashboard - M2 Info</title>
    <!-- Styles CSS simplifiés pour la lisibilité -->
```

```
    <style>
        body { font-family: sans-serif; padding: 20px; background: #f4f4f4; }
        .card { background: white; padding: 15px; margin-bottom: 10px; border-radius: 5px;
box-shadow: 0 2px 5px rgba(0,0,0,0.1); }
        .status { padding: 10px; margin-bottom: 20px; border-radius: 4px; font-weight: bold;
text-align: center;}
        .ok { background-color: #d4edda; color: #155724; }
        .error { background-color: #f8d7da; color: #721c24; }
    </style>
</head>
<body>
    <h1>Dashboard Promo M2</h1>
    <div id="status" class="status">Chargement...</div>
    <div id="list"></div>

    <script>
        const API_URL = "http://localhost:8000";

        async function load() {
            const statusDiv = document.getElementById('status');
            try {
                const res = await fetch(`${API_URL}/`);
                if (!res.ok) throw new Error("API Error");
                const data = await res.json();

                // Vérification de version (Si l'étudiant n'a pas fait l'étape 2)
                if(data.version) {
                    statusDiv.innerText = "⚠️ API connectée : Mode Version (Table Students non
trouvée)";
                    statusDiv.className = "status error";
                    return;
                }

                statusDiv.innerText = "✅ Système Opérationnel";
                statusDiv.className = "status ok";

                document.getElementById('list').innerHTML = data.map(s => `
                    <div class="card">
                        <h3>${s.nom || 'Anonyme'}</h3>
                        <p>Promo: ${s.promo || 'N/A'}</p>
                        <p><small>Vues: ${s.views !== undefined ? s.views : 'N/A (Cache
HS)'}</small></p>
                    </div>
```

```
            `).join('');
        } catch (e) {
            statusDiv.innerText = "❌ Erreur connexion API";
            statusDiv.className = "status error";
            console.error(e);
        }
    }
    setInterval(load, 2000);
    load();
  </script>
</body>
</html>
```