
TP Microservice - Sujet 2

Contents

Description de l'Application :	2
Étape 1 : Mise en Place du Projet	2
1.1. Structure du Répertoire	2
Étape 2 : Développement des Microservices	2
2.1. API des Utilisateurs (User Service)	2
Étape 3 : Configuration Multi-environnement	6
Étape 4 : Démarrage du Projet	6
Étape 5 : Tests et Communication entre Services	6
Ajout de la Base de Données PostgreSQL	9
Vérification du Fonctionnement	10

Conteneuriser et orchestrer une application composée de plusieurs microservices en utilisant Docker et Docker Compose. Vous allez développer une application web simple constituée de trois microservices interconnectés, apprendre à les faire communiquer, et déployer l'ensemble de l'application en local.

Description de l'Application :

Nous allons construire une application de gestion de tâches (To-Do List) composée de trois microservices : 1. **API des Utilisateurs** : Gère les utilisateurs de l'application. 2. **API des Tâches** : Gère les tâches à effectuer. 3. **Front-end Web** : Interface utilisateur qui permet d'interagir avec les deux API précédentes.

Chaque microservice sera développé dans un langage différent pour simuler une hétérogénéité technologique (par exemple, Node.js, Python, et PHP).

Étape 1 : Mise en Place du Projet

1.1. Structure du Répertoire

Créez un répertoire principal pour votre projet :

```
mkdir microservices-todo-app
cd microservices-todo-app
```

Créez des sous-répertoires pour chaque microservice :

```
mkdir user-service
mkdir task-service
mkdir web-frontend
touch .env .env.prod docker-compose.yml docker-compose.prod.yml
```

Étape 2 : Développement des Microservices

2.1. API des Utilisateurs (User Service)

Technologie : Node.js avec Express.js

```
cd user-service
npm init -y
npm install express
touch index.js
```

2.1.1. Initialisation du Projet

2.1.2. Code de l'API Éditez index.js :

```
const fs = require('fs');
const express = require('express');
const { Pool } = require('pg');
const app = express();
const port = process.env.PORT || 3000;

// Lire les secrets à partir des fichiers
const postgresUser = fs.readFileSync('/run/secrets/postgres_user',
  ↪ 'utf8').trim();
const postgresPassword = fs.readFileSync('/run/secrets/postgres_password',
  ↪ 'utf8').trim();

const pool = new Pool({
  user: postgresUser,
  host: process.env.POSTGRES_HOST || 'postgres',
  database: process.env.POSTGRES_DB || 'users_db',
  password: postgresPassword,
  port: 5432,
});

app.get('/users', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM users');
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Database query failed' });
  }
});

app.listen(port, () => {
  console.log(`User service listening at http://localhost:${port}`);
});
```

```
});
```

2.2. API des Tâches (Task Service) Technologie : Python avec Flask

```
cd ../task-service
mkdir app
cd app
touch app.py
```

2.2.1. Initialisation du Projet Créez un fichier requirements.txt :

```
Flask==3.0.0
```

2.2.2. Code de l'API Éditez app/app.py :

```
from flask import Flask, jsonify

app = Flask(__name__)

tasks = [
    {'id': 1, 'title': 'Faire les courses', 'completed': False},
    {'id': 2, 'title': 'Lire un livre', 'completed': False}
]

@app.route('/tasks', methods=['GET'])
def get_tasks():
    return jsonify(tasks)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=4000)
```

2.3. Front-end Web Technologie : PHP avec Slim Framework (ou un simple script PHP)

```
cd ../web-frontend
mkdir public
touch public/index.php
```

2.3.1. Initialisation du Projet

2.3.2. Code du Front-end Éditez public/index.php :

```
<?php
function fetch_data($url) {
    $response = file_get_contents($url);
    return json_decode($response, true);
}

$users = fetch_data('http://user-service:3000/users');
$tasks = fetch_data('http://task-service:4000/tasks');
?>

<!DOCTYPE html>
<html>
<head>
    <title>ToDo List</title>
</head>
<body>
    <h1>Utilisateurs</h1>
    <ul>
        <?php foreach ($users as $user): ?>
            <li><?php echo $user['name']; ?></li>
        <?php endforeach; ?>
    </ul>
    <h1>Tâches</h1>
    <ul>
        <?php foreach ($tasks as $task): ?>
            <li><?php echo $task['title']; ?> - <?php echo
                $task['completed'] ? 'Terminé' : 'En cours'; ?></li>
        <?php endforeach; ?>
    </ul>
</body>
</html>
```

Étape 3 : Configuration Multi-environnement

```
USER_SERVICE_PORT=3000  
TASK_SERVICE_PORT=4000  
WEB_PORT=8080
```

3.1. Fichier .env (Développement)

```
USER_SERVICE_PORT=80  
TASK_SERVICE_PORT=8080  
WEB_PORT=443
```

3.2. Fichier .env.prod (Production)

Étape 4 : Démarrage du Projet

4.1. Environnement de Développement Démarrer le projet avec les variables dev puis prod avec docker run.

4.2. Vérification du Fonctionnement

- **Accéder au Front-end :**

Ouvrez un navigateur et allez à `http://localhost:8080`. Vous devriez voir la liste des utilisateurs et des tâches.

- **Tester les API directement :**

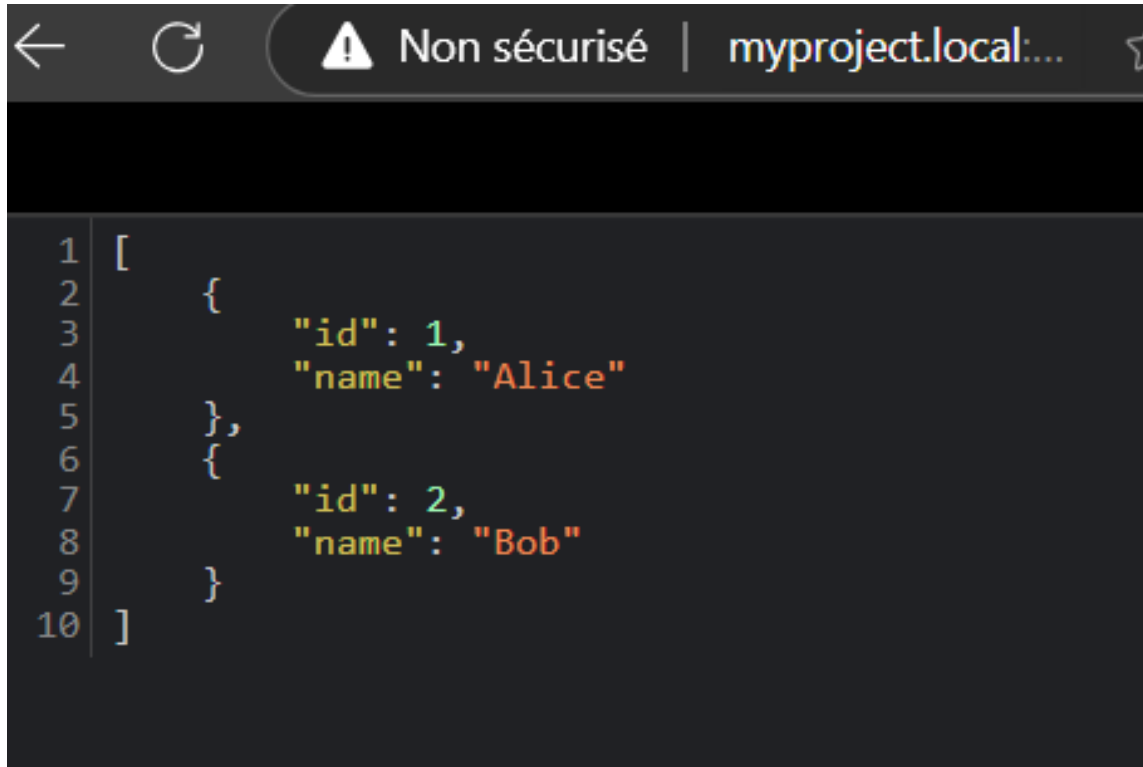
- **Utilisateurs** : `http://localhost:3000/users`
- **Tâches** : `http://localhost:4000/tasks`

Étape 5 : Tests et Communication entre Services

5.1. Communication Inter-Services

- Les services utilisent les noms de service définis dans Docker Compose (`user-service` et `task-service`) pour communiquer.

- Dans index.php, les URLs `http://user-service:3000/users` et `http://task-service:4000/tasks` font référence aux conteneurs respectifs via le DNS interne de Docker.



A screenshot of a web browser window. The address bar shows a warning icon, the text "Non sécurisé", and the URL "myproject.local:...". The main content area displays a JSON array of two user objects. The first object has "id": 1 and "name": "Alice". The second object has "id": 2 and "name": "Bob". The JSON is formatted with syntax highlighting and line numbers 1 through 10 on the left.

```
1  [  
2    {  
3      "id": 1,  
4      "name": "Alice"  
5    },  
6    {  
7      "id": 2,  
8      "name": "Bob"  
9    }  
10 ]
```



A screenshot of a web browser window. The address bar shows a warning icon, the text "Non sécurisé", and the URL "myproject.local:...". The main content area displays a JSON array of two task objects. The first object has "completed": false, "id": 1, and "title": "Faire les courses". The second object has "completed": false, "id": 2, and "title": "Lire un livre". The JSON is formatted with syntax highlighting and line numbers 1 through 12 on the left.

```
1  [  
2    {  
3      "completed": false,  
4      "id": 1,  
5      "title": "Faire les courses"  
6    },  
7    {  
8      "completed": false,  
9      "id": 2,  
10     "title": "Lire un livre"  
11   }  
12 ]
```


5.2. Tester la Résilience

- **Arrêter un Service :**

Dans un autre terminal, exécutez un stop sur le nom du container de votre service :

```
docker stop user-service
```

- **Actualiser le Front-end :**

Vous devriez constater que la section des utilisateurs ne s'affiche plus, tandis que les tâches sont toujours visibles.

```
15:32:31 root@multi-webapp ~ # docker ps
CONTAINER ID   IMAGE                                     COMMAND                                     CREATI
a2d769f6baf8   microservices-todo-app-web-frontend    "docker-php-entrypoi..."   51 mir
5bab93cef998   microservices-todo-app-user-service     "docker-entrypoint.s..."   51 mir
70a6b60e61c1   microservices-todo-app-task-service     "python ./app/app.py"       51 mir
15:32:38 root@multi-webapp ~ # docker stop microservices-todo-app-user-service-1
```

Warning: file_get_contents(): php_network_getaddresses: getaddrinfo for user-service failed: Name or service not known in /var/www/html/index.php on line 3

Warning: file_get_contents(http://user-service:3000/users): Failed to open stream: php_network_getaddresses: getaddrinfo for user-service failed: Name or service not known in /var/www/html/index.php on line 3

Utilisateurs

Warning: foreach() argument must be of type array/object, null given in /var/www/html/index.php on line 19

Tâches

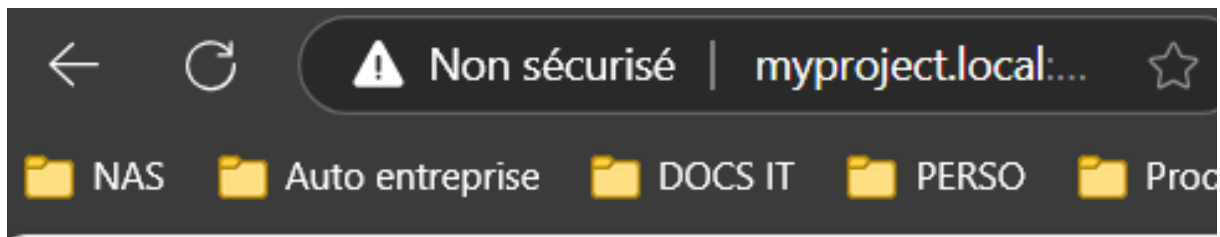
- Faire les courses - En cours
- Lire un livre - En cours

- **Redémarrer le Service :**

•

```
docker start user-service
```

```
15:32:52 root@multi-webapp ~ # docker start microservices-todo-app-user-service-1
microservices-todo-app-user-service-1
```



Utilisateurs

- Alice
- Bob

Tâches

- Faire les courses - En cours
- Lire un livre - En cours

- Actualiser à Nouveau :

La section des utilisateurs devrait réapparaître.

Ajout de la Base de Données PostgreSQL

Création du Script d'Initialisation de la Base de Données Créez le fichier `db/init.sql` :

db/init.sql :

```
CREATE TABLE IF NOT EXISTS users (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(100)  
);
```

```
INSERT INTO users (name) VALUES ('Alice'), ('Bob');
```

Créer un conteneur base de donnée postgres avec les éléments suivants :

Environnement variables : - POSTGRES_USER - POSTGRES_PASSWORD - POSTGRES_DB

Volumes : - postgres-data:/var/lib/postgresql/data - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
Sur le même réseau que les autres.

Vérification du Fonctionnement

Accéder à l'Interface Web Ouvrez un navigateur et accédez à :

`http://localhost:8080`

Vous devriez voir la liste des utilisateurs (Alice et Bob) et des tâches.

Tester les API Directement

- **API des Utilisateurs :**

```
curl http://[IP DU CONTAINER]:3000/users
```

- **API des Tâches :**

```
curl http://[IP DU CONTAINER]:4000/tasks
```

Vérifier les healthchecks Pour vérifier l'état de santé des services :

```
docker ps
docker inspect --format='{{json .State.Health}}' user-service
```