RAPPORT DE SYNTHÈSE – PRÊT À RENDRE

📘 Rapport de synthèse – Projet Docker & Virtualisation

Projet : DailyHabits
Étudiante : Jeannette Ngele Bissengue
Formation : EPSI – Docker & Virtualisation

DailyHabits est une application web qui permet de gérer et suivre des habitudes quotidiennes. Elle expose une API REST pour récupérer et créer des habitudes, et un front-end léger pour visualiser les données. L'application utilise **Flask**, **SQLite** et est entièrement containerisée avec **Docker** pour faciliter le déploiement et l'automatisation.

1️⃣ Architecture de la solution

La solution repose sur une architecture **multi-services conteneurisée** à l'aide de Docker et Docker Compose.

🔹 Service API (Backend)

- Technologie : **Python / Flask**
- Fournit des endpoints REST :
  - `/status` : vérification de l'état de l'API
  - `/items` : récupération des habitudes stockées
- Base de données : **SQLite**, stockée dans un volume Docker
- Exécution avec un utilisateur **non-root**
- Healthcheck intégré pour vérifier la disponibilité du service

🔹 Service Frontend

- Application web statique
- Dépend du service API
- Démarré uniquement lorsque l'API est saine (`depends_on`)

🔹 Réseau & Volumes

- Réseau Docker bridge dédié : `dailyhabits-net`
- Volume persistant : `db_data` pour la base SQLite

2️⃣ Commandes clés utilisées

Construction des images

`docker build -t dailyhabits-api ./api docker build -t dailyhabits-frontend ./frontend`

Tests unitaires

`docker run --rm dailyhabits-api pytest`

Vérification de la configuration Compose

`docker compose config`

Push vers Docker Hub

`docker push jeannette329/dailyhabits-api:1.1 docker push jeannette329/dailyhabits-frontend:1.1`

Déploiement

`docker compose up -d`

3️⃣ Bonnes pratiques mises en œuvre

- ✔ Images légères (`python:3.11-slim`)
- ✔ Utilisateur non-root dans les conteneurs
- ✔ Séparation claire API / Frontend
- ✔ Variables d'environnement (`.env`)
- ✔ Volumes pour la persistance des données
- ✔ Healthchecks Docker
- ✔ Tests unitaires exécutés dans l'image
- ✔ Script d'automatisation (`deploy.ps1`)
- ✔ Docker Content Trust testé pour la signature des images

4️⃣ Difficultés rencontrées

🔸 Docker Content Trust

- Gestion complexe des clés et passphrases
- Erreurs de signature (`no hashes specified`)
- Solution : désactivation contrôlée pour le déploiement local

🔸 Tests unitaires

- Problème d'import Flask lié à `app.run`
- Correction via un `if __name__ == "__main__"`

🔸 Endpoint `/items`

- Erreur 500 lorsque la base n'est pas initialisée
- Tests adaptés pour accepter ce comportement

🔸 service API était marqué comme `/unhealthy`

Une difficulté a été rencontrée lors du déploiement : le service API était marqué comme unhealthy.
L'origine du problème provenait du healthcheck Docker utilisant curl, absent de l'image python:3.11-slim.
La solution a consisté à installer explicitement curl dans l'image API, permettant au healthcheck de fonctionner correctement.

le service API était fonctionnel mais incorrectement déclaré unhealthy en raison d'un healthcheck exécuté avant la fin de l'initialisation de la base de données. L'ajout d'un délai de démarrage (`start_period`) a permis de stabiliser le déploiement et de garantir un lancement fiable de l'ensemble de la stack.

5️⃣ Améliorations possibles

- 🔧 Intégration CI/CD (GitHub Actions / GitLab CI)
- 🔐 Sécurisation avancée (Secrets Docker, Vault)
- 📈 Scaling horizontal avec Docker Swarm ou Kubernetes
- 🧪 Tests d'intégration avec base de données mockée
- 📝 Monitoring (Prometheus / Grafana)

✅ Conclusion

Ce projet met en œuvre une stack Docker complète, automatisée et opérationnelle, respectant les bonnes pratiques de conteneurisation moderne.
L’ensemble est reproductible, sécurisé et prêt pour une industrialisation future.