TD - Conception d'une application conteneurisée générique (Docker)

Année 2025/2026

KABA Tasilimy

EPSI - Bachelor SYSOPS C2

Sommaire

1. Architecture technique

L'application est architecturée selon un modèle strict en 3-Tiers. Afin d'assurer l'isolation et la robustesse, tous les services sont encapsulés dans des conteneurs Docker et opèrent au sein d'un *bridge* réseau privé.

1.1 Schéma des flux de communication

Le protocole de communication est strictement unidirectionnel, garantissant que les requêtes suivent un chemin contrôlé depuis l'utilisateur jusqu'à la base de données (DB) isolée.

1. **Client :** La requête initiale est émise par l'utilisateur via le protocole HTTP standard (Port 80).
2. **Couche de présentation (Nginx Front-end) :** Ce service sert l'interface statique et fonctionne comme un proxy inverse essentiel, dirigeant les requêtes destinées à l'API vers le réseau interne.
3. **Couche applicative (API Flask) :** Ce service prend en charge la logique métier et est la seule entité autorisée à communiquer avec la couche de données.
4. **Couche de données (MySQL) :** SGBD, son accès est strictement cantonné au conteneur de la couche applicative pour une sécurité maximale.

1.2 Services et rôles au sein de l'orchestration docker

Service	Image Docker	Couche	Rôle Principal
Front-end	nginx:stable-alpine	Présentation (Tier 1)	Diffusion de l'interface utilisateur et fonction de Reverse Proxy.
API	flask-app:latest	Application (Tier 2)	Exécution de la logique métier et gestion de l'accès aux données.
Base de données	mysql:8.0	Données (Tier 3)	Stockage persistant et sécurisé des informations de l'application.

2. Guide de démarrage

Cette section présente la procédure complète pour initialiser, construire et déployer l'intégralité de la stack applicative.

2.1. Prérequis

L'intégralité des fichiers nécessaires à la construction de la solution est versionnée dans le dépôt Git (à l'exception des secrets).

Action	Commande
Clonage du dépôt	git clone https://github.com/Tasilimy/TD_Docker-SYSOPS-C2-KABA-Tasilimy.git
Configuration (secrets)	Le fichier .env doit être créé à la racine avec les identifiants de connexion.
Construction et Déploiement	docker compose up --build -d

2.2. Configuration des variables d'environnement

Pour assurer la portabilité et la gestion sécurisée des identifiants, les variables d'environnement sont externalisées via le fichier .env à la racine du projet.

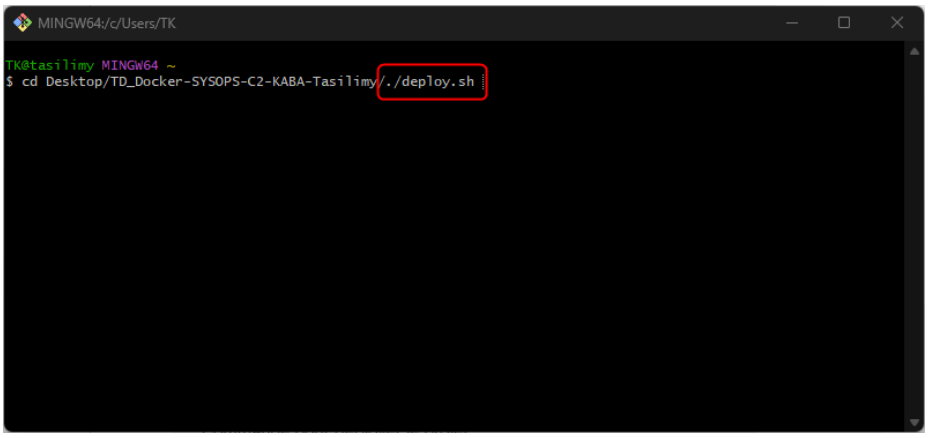
Variable	Rôle
API_PORT	Port d'écoute de l'API (ex: 8080).
MYSQL_ROOT_PASSWORD	Mot de passe administrateur de la DB.
MYSQL_DATABASE, MYSQL_USER, MYSQL_PASSWORD	Identifiants pour la connexion de la couche API à la DB.
DB_HOST	Nom du service MySQL pour la connexion inter-conteneurs.

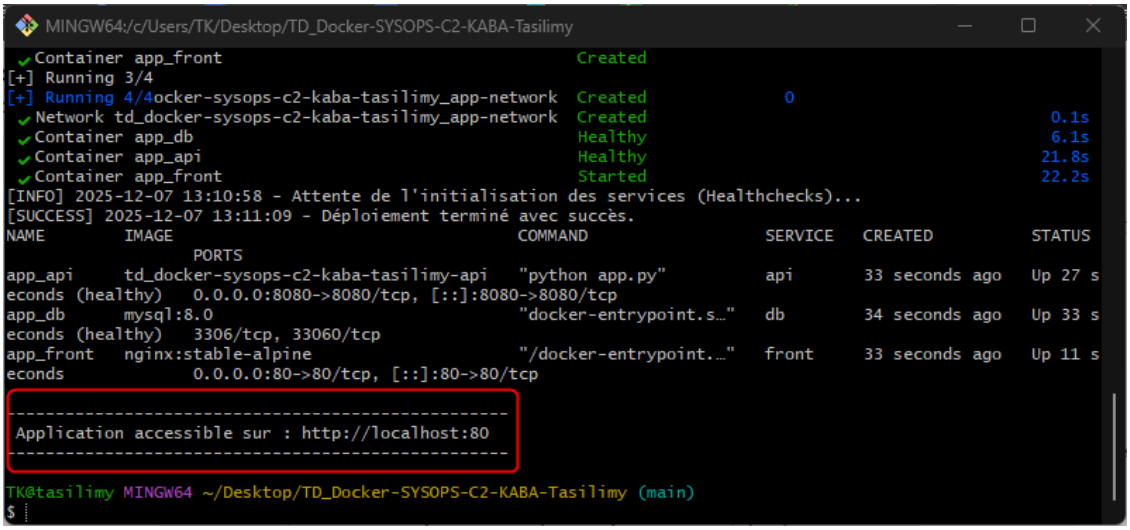
2.3. Script d'automatisation

Un script shell (deploy.sh) est mis en place pour automatiser la séquence de *build*, de *push* des images (vers un registre distant) et de déploiement final.

La commande pour lancer ce script est :

./deploy.sh





2.4. Construction des images et déploiement

Le déploiement est géré par Docker Compose, qui s'occupe de la construction des images (API et Front-end) en se basant sur les Dockerfile multi-étapes et de l'orchestration des trois services (db, api, front).

Les commandes suivantes permettent de vérifier la fonctionnalité de l'API et l'intégrité des

données après le déploiement.

Action	Commande
Vérifier la santé des services	docker compose ps
Tester le statut de l'API	curl http://localhost/status
Tester les données (schéma DB)	curl http://localhost/items
Arrêt total (suppression du volume de données)	docker compose down -v

Commande pour déployer le projet :

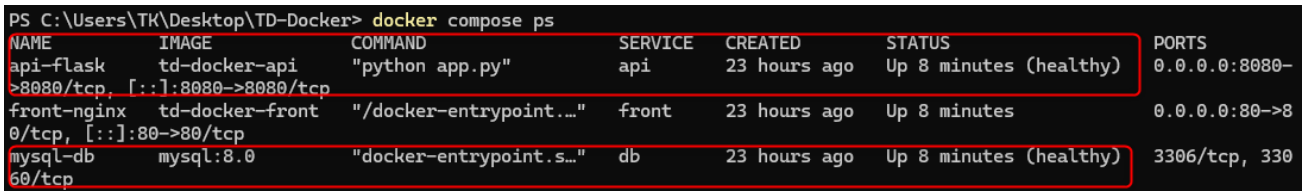
docker compose up --build -d

La commande permet la construction des images et lancement des conteneurs
L'option --build force la reconstruction des images locales
L'option -d lance le tout en mode détaché (en arrière-plan)

2.5. Vérification de l'état du déploiement

Les commandes suivantes permettent de vérifier que tous les services sont opérationnels, y compris les *healthchecks* :

1. Vérifier l'état des conteneurs (Doit afficher 'healthy' pour l'API et la DB)
- docker compose ps



2. Consulter les logs pour débugger
docker compose logs -f

```
PS C:\Users\TK\Desktop\TD-Docker> docker compose logs -f
front-nginx  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
mysql-db     | 2025-12-05 16:52:47+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.44-1.el9 started.
mysql-db     | 2025-12-05 16:52:48+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql-db     | 2025-12-05 16:52:48+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.44-1.el9 started.
mysql-db     | '/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'
mysql-db     | 2025-12-05T16:52:48.970442Z 0 [Warning] [MY-011068] [Server] The syntax '--skip-host-cache' is deprecated
 and will be removed in a future release. Please use SET GLOBAL host_cache_size=0 instead.
mysql-db     | 2025-12-05T16:52:48.973911Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.44) starting as
process 1
mysql-db     | 2025-12-05T16:52:48.985388Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
mysql-db     | 2025-12-05T16:52:49.463092Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
mysql-db     | 2025-12-05T16:52:49.933915Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
mysql-db     | 2025-12-05T16:52:49.934019Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS.
 Encrypted connections are now supported for this channel.
mysql-db     | 2025-12-05T16:52:49.942280Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Locat
ion '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
mysql-db     | 2025-12-05T16:52:49.994768Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address:
 '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
mysql-db     | 2025-12-05T16:52:49.995110Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Vers
ion: '8.0.44'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
```

3. Tester la route de statut de l'API (accessible via le service 'front' qui est le proxy)

- curl http://localhost/status

Réponse attendue : {"status": "OK"}

```
PS C:\Users\TK\Desktop\TD-Docker> curl http://localhost/status


StatusCode        : 200
StatusDescription : OK
Content           : {"database_connection":"OK","message":"<<< OK >>","status":"OK"}

RawContent        : HTTP/1.1 200 OK
                    Connection: keep-alive
                    Content-Length: 65
                    Content-Type: application/json
                    Date: Sat, 06 Dec 2025 16:11:44 GMT
                    Server: nginx/1.28.0

                    {"database_connection":"OK","message":"<<< OK ...
Forms             : {}
Headers           : {[Connection, keep-alive], [Content-Length, 65], [Content-Type, application/json], [Date, Sat, 06
                    Dec 2025 16:11:44 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 65
```

3. Choix techniques et sécurité

Cette partie explique les choix techniques et les méthodes utilisées pour sécuriser et optimiser le projet.

3.1. Bonnes pratiques d'optimisation et de légèreté docker

L'utilisation des builds multi-étapes a permis d'alléger considérablement les images Docker :

Métrique	Valeur Avant (Builder)	Valeur Après (Production)	Gain / Optimisation
Taille Image API	~1000 MB (python:3.11)	242 MB	-76% de réduction
Contenu	Compilateurs, SDK, Caches	Runtime strict uniquement	Surface d'attaque réduite

3.2. Mesures de sécurité du conteneur et de l'orchestration

La sécurité est mise en œuvre à la fois au niveau du build et du runtime.

- **Principe du moindre privilège (Utilisateur non-root) :** L'exécution des processus est systématiquement réalisée sous un utilisateur dédié et non-root pour limiter les dommages potentiels en cas de compromission d'un conteneur.
- **Scan des images et content trust :** Pour valider la robustesse, un scan de vulnérabilités est recommandé avant le déploiement. Si la commande native `docker scan` n'est pas disponible (dépendance Snyk), l'utilisation de l'outil open-source **Trivy** est préconisée via Docker : `docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy image td_docker-sysops-c2-kaba-tasilimy-api`. De plus, l'intégrité de l'image est garantie par l'activation du Docker Content Trust via la variable d'environnement : `export DOCKER_CONTENT_TRUST=1` avant le *push*.
- **Healthchecks et Dépendances :** La définition de healthchecks couplée à `depends_on: service_healthy` (dans le `docker-compose.yml`) assure que les dépendances critiques sont saines avant l'exécution du service, résolvant le problème de race condition.
- **Isolation Réseau :** L'utilisation d'un **Réseau Privé (Bridge Network) explicite** garantit que seule l'API est capable d'accéder à la DB, assurant une isolation complète.

Corrections de sécurité appliquées :

- **Conteneurs API et Front-end :** La directive cap_drop: ALL a été appliquée pour retirer la majorité des capacités kernel inutiles (moindre privilège au runtime).
- **Conteneur Nginx (Front) :** La capacité cap_add: CHOWN a été réintégrée spécifiquement et uniquement pour Nginx afin de résoudre l'erreur de permission lors de l'initialisation du cache (l'erreur chown).
- **Conteneur DB (MySQL) :** Les restrictions cap_drop ont été levées pour le service DB afin de résoudre les conflits de permission critiques au démarrage (setgid), un compromis nécessaire pour le déploiement de l'image standard.

4. Analyse des difficultés et solutions implémentées

Cette section présente les principaux défis rencontrés et les solutions efficaces adoptées.

4.1. Difficultés rencontrées

- **Problèmes de séquence de démarrage :** L'API tentait régulièrement de se connecter à la DB avant que le schéma ne soit prêt, conduisant à des échecs d'initialisation.
- **Conflit de permissions sécuritaires :** L'application rigoureuse des directives de sécurité (cap_drop: ALL) a provoqué des échecs critiques au démarrage de MySQL (erreur setgid) et de Nginx (erreur chown).
- **Erreurs de contexte de build et routage :** Des erreurs initiales ont été rencontrées (chemins relatifs dans le Dockerfile incorrects, puis défaillance du Reverse Proxy Nginx).
- **Défaillance du reverse proxy :** Problème de routage entre le Front-end Nginx et l'API qui a causé une latence excessive ou un échec de connexion.

4.2. Solutions et améliorations implémentées

- **Résolution de la dépendance au démarrage :** Le problème de *Race Condition* a été corrigé par la mise en œuvre des Healthchecks et de depends_on: service_healthy dans le docker-compose.yml.
- **Correction des capacités (permissions) :** L'erreur setgid a été résolue en retirant les restrictions du conteneur DB. L'erreur Nginx a été corrigée via l'ajout ciblé de cap_add: CHOWN.
- **Implémentation du multi-stage build et initialisation DB :** L'adoption du build multi-étapes a garanti la légèreté. Le script db/init.sql est monté pour l'initialisation automatique et robuste du schéma.
- **Correction du reverse proxy :** Le fichier front/nginx.conf a été ajusté pour garantir le routage correct des chemins /status et /items vers l'API, validant ainsi le point d'entrée unique.

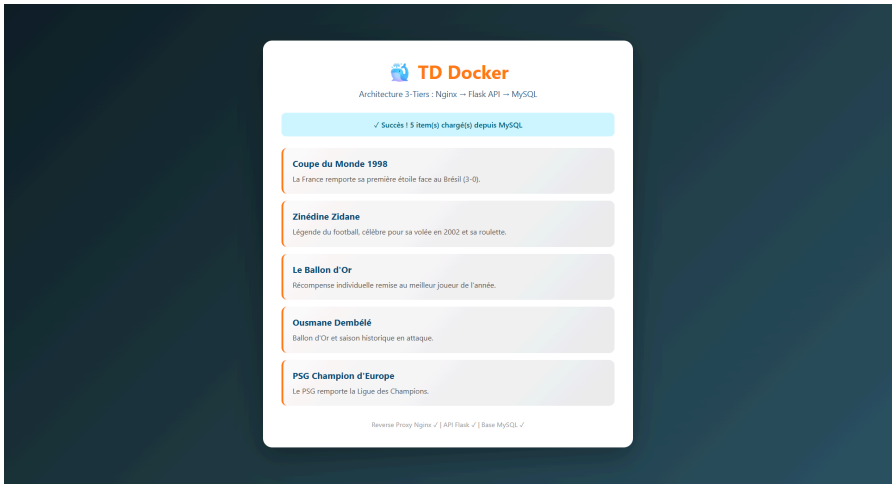
4.3. Perspectives d'amélioration

- **Optimisation du déploiement via registre distant (Docker Hub) :** Pour la production, l'utilisation d'images pré-construites sur un registre distant (plutôt que le *build* local) réduirait le temps de mise en service et garantit l'invariabilité de l'environnement.
- **Sécurisation avancée (Docker Secrets) :** Remplacement des variables .env par Docker Secrets (ou HashiCorp Vault) pour une gestion professionnelle des identifiants sensibles.
- **Intégration et déploiement continus (CI/CD) :** L'intégration à un pipeline CI/CD pour automatiser les tests, le scan de sécurité et le déploiement est l'étape suivante logique.
- **Supervision et métriques (Prometheus) :** L'ajout d'un outil de supervision pour le suivi des performances de l'API est essentiel pour le maintien en condition opérationnelle.

Résultats :

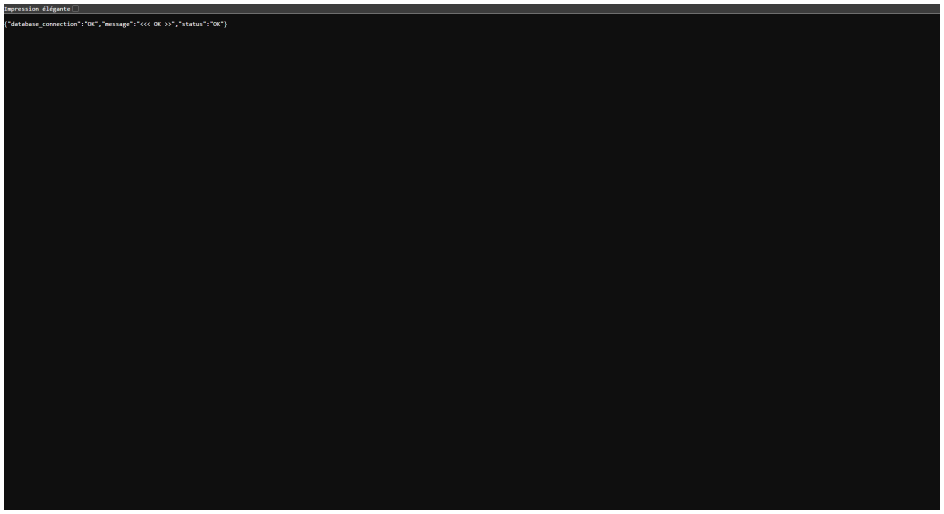
Frontend (Nginx) - Port 80

http://localhost/ = Page d'accueil principale - Interface web affichant la liste des items



API Flask - Port 8080

http://localhost:8080/status = Healthcheck API/DB - Vérifie l'état de l'API et de la base de données



http://localhost:8080/items = Liste des items - Récupère tous les items de la base de données