**Automatisation et Orchestration**

**Présenté par:**

**Mathis Douangpanya**

**Scénario choisi : Application Web avec Base de Données et Monitoring – Automatisée et Orchestrée**

Dans ce projet, je vais déployer une application web simple, reposant sur une architecture multi-conteneur. Cette application sera composée d’un serveur web, d’une base de données, ainsi que d’un système de surveillance (monitoring), le tout dans un environnement infonuagique simulé.  
L’objectif est de démontrer comment concevoir, automatiser et orchestrer une application distribuée en utilisant :

* Docker Compose pour le développement local,
* Docker Swarm pour la gestion multi-nœuds,
* Ansible pour l’automatisation des déploiements et des configurations.

Composants principaux

1. Conteneurs

* Serveur Web : Conteneur NGINX (ou équivalent) hébergeant une interface statique ou dynamique.
* Base de Données : Conteneur MySQL pour la gestion des données.
* Monitoring : Conteneur de surveillance (par exemple cAdvisor, Prometheus ou Grafana) permettant de suivre l’état des conteneurs et des ressources du système.

2. Automatisation

Création de playbooks Ansible pour :

* Installer Docker Engine sur toutes les machines.
* Configurer SSH, les utilisateurs et les permissions nécessaires.
* Initialiser et rejoindre un cluster Docker Swarm.
* Déployer automatiquement les services via docker stack deploy.

3. Orchestration

* Déploiement des services sur plusieurs machines virtuelles (exemple : 2 managers et 3 workers).
* Mise en place d’une haute disponibilité grâce à plusieurs réplicas pour les services critiques.
* Simulation d'une panne de conteneur pour vérifier la reprise automatique assurée par Swarm.

## Prérequis :

Docker engine :

A screenshot of a computer

AI-generated content may be incorrect.

Docker compose :

A screenshot of a computer

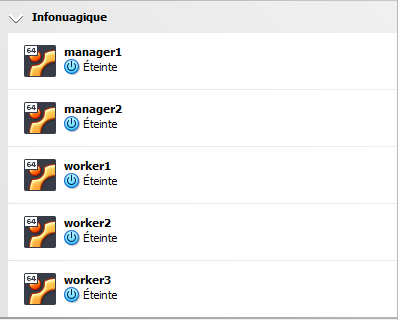
AI-generated content may be incorrect.

Ansible :

A computer screen shot of a code

AI-generated content may be incorrect.

Machines :



## Configurer le réseau des machines :

A computer screen shot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.Manager 1 : Manager 2 :

A screenshot of a computer

AI-generated content may be incorrect.Worker 1 : Worker 2 :

A screenshot of a computer

AI-generated content may be incorrect.

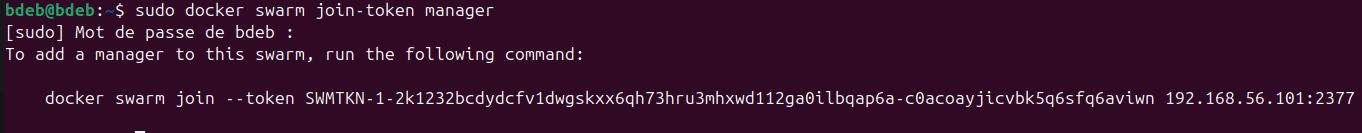
Worker 3 :

A screenshot of a computer program

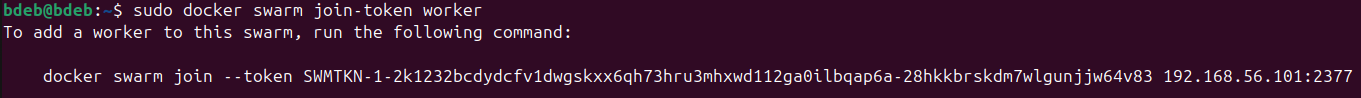
AI-generated content may be incorrect.

## Ajouter les nœuds au cluster Docker swarm :

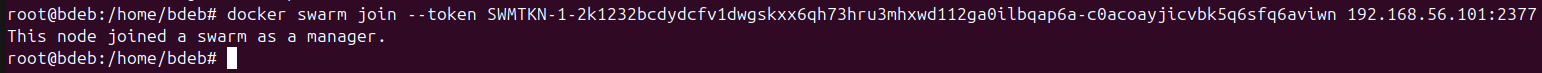
Token manager sur manager 1 :



Token worker sur manager 1:



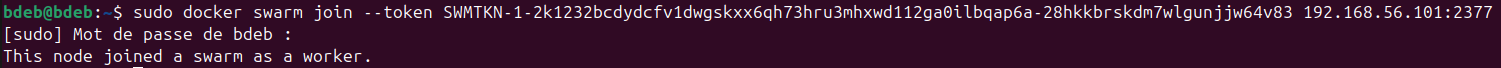
Manager 2:



Worker 1:



Worker 2 :



Worker 3 :



Vérif sur manager 1 :

A screenshot of a computer screen

AI-generated content may be incorrect.

## Partie 1 :

Les fichiers nécessaires

Docker-compose yml :

A screenshot of a computer screen

AI-generated content may be incorrect.Services : Définit tous les services (conteneurs) à lancer avec Docker Compose

Frontend : Nom du service représentant le serveur web (NGINX)

Image : Spécifie l'image Docker à utiliser pour le conteneur

Volumes te un dossier local ou un volume dans le à

Ports Fait le lien entre les ports de l'hôte et ceux du

Networks Connecte le service à un réseau Docker (app-net)

Backend  Nom du service pour l'application Python (Flask)

Build  Indique le dossier contenant le Dockerfile à construire une image personnalisée

Environnement  Définit des variables d’environnement accessibles dans le conteneur

Db  Nom du service pour la base de données MySQL

Restart Redémarre automatiquement le conteneur en cas d’arrêt ou crash

Volume Utilise un volume pour stocker les données MySQL de façon persistante

Monitoring  Nom du service qui représente le monitoring

A screenshot of a computer screen

AI-generated content may be incorrect.Port : Le service de monitoring roule sur le port 8081

Volumes Déclare les volumes persistants utilisés dans les services

Privileged Donne un accès complet au système hôte (utile pour cAdvisor)

Dbdate Nom du volume pour stocker les données de la base MySQL

A screenshot of a computer

AI-generated content may be incorrect.Dockerfile :

FROM Spécifie l’image de base à utiliser (ici Python 3.8 slim)

WORKDIR Définit le dossier de travail dans le conteneur (/app)

COPY Copie les fichiers du dossier backend dans l’image Docker

RUN Exécute une commande pendant la construction (ici pip install)

CMD Définit la commande à exécuter quand le conteneur démarre

Explication :  
Ce fichier permet à Docker de construire une image auto-exécutable de mon API Flask, que je pourrai lancer sur n’importe quelle machine, sans avoir besoin de réinstaller Python ou les autres dépendances.

Requirements.txt

A screen shot of a computer

AI-generated content may be incorrect.

Explication :

Ce fichier sert à déclarer les dépendances Python nécessaires pour faire fonctionner l’API Flask dans Docker. En général, indique que le backend a besoin du framework Flask pour fonctionner.

App.py

A computer screen shot of a program code

AI-generated content may be incorrect.

Explication :

Le fichier app.py lance un petit serveur web qui attend les connexions. Quand un client (comme un navigateur ou le frontend) envoie une requête à l’adresse du backend, ce fichier déclenche une réponse automatique : il renvoie simplement le message "Hello from the Backend!". C’est grâce à ce fichier que le backend répond aux utilisateurs.

Index.html

A computer screen shot of a computer code

AI-generated content may be incorrect.

Explication :

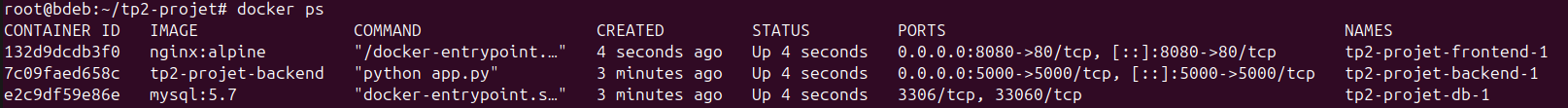
Le fichier index.html est la page d’accueil de l’application. Lorsqu’un utilisateur ouvre l’adresse du site dans son navigateur, ce fichier s’affiche automatiquement. Il montre un message de bienvenue et permet de confirmer visuellement que le frontend est bien en ligne et connecté au reste de l’application.

Test docker-compose.yml

A screenshot of a computer program

AI-generated content may be incorrect.

Ici, j’ai lancer la commande docker compose up --build, qui va donc exécuter fichier docker-compose.yml pour lancer les conteneurs et le reste des commandes.



Test backend

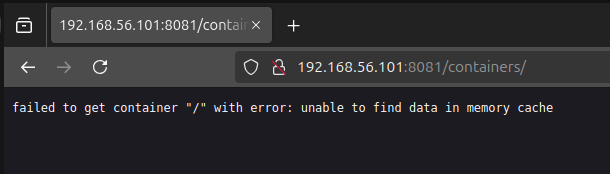


Elle envoie une requête HTTP GET à l’adresse localhost sur le port 5000 (celui du backend Flask). Elle permet de tester que le backend fonctionne et répond correctement avec le message qui retourne « Hello from the backend ».

Test frontend, backend et base de données



Test monitoring cAdvisor :



En accédant à 192.168.56.101:8081, je teste si cAdvisor fonctionne correctement pour surveiller l’activité des conteneurs sur ta VM manager1. Il permet d’afficher en temps réel l’utilisation des ressources de mon application Docker.

Erreur : Cette erreur signifie que cAdvisor n’a pas pu récupérer les informations du conteneur racine ("/"), car les données ne sont pas encore présentes dans le cache mémoire de l’outil.

## Partie 2 :

Docker login pour link à mon compte Dockerhub :

A screenshot of a computer error

AI-generated content may be incorrect.

Push backend sur docker hubA computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Explication: Lors du déploiement de services dans un cluster, les nœuds du cluster vont chercher l’image directement sur Docker Hub. Cela garantit que tous les nœuds ont la même version de ton application.

Déployer stack Docker sur le Docker swarm multi nœuds:

A screen shot of a computer screen

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

Après le déploiement de la stack Docker avec Docker Swarm, on observe que les services sont bien en cours d'exécution avec 1 réplique active pour chaque service. Le service tp2\_backend fonctionne correctement, et on voit que l'application Flask est en mode développement. L'application écoute sur toutes les adresses disponibles, avec un accès sur le port 5000 à l'adresse 0.0.0.0 et 172.18.0.6:5000, ce qui signifie que le service est accessible depuis plusieurs adresses réseau du cluster.

Test docker swarm sur manager 2:

A screenshot of a computer screen

AI-generated content may be incorrect.

## Partie 3:

Fichier hosts.ini :

A computer screen with white text

AI-generated content may be incorrect.

Tests de ping :

A screenshot of a computer program

AI-generated content may be incorrect.

A purple and black screen

AI-generated content may be incorrect.

Test ssh de la machine manager1 :









A screenshot of a computer program

AI-generated content may be incorrect.Fichier deploy\_swarm.yml :

A screenshot of a computer program

AI-generated content may be incorrect.

Le fichier deploy\_swarm.yml est un playbook Ansible qui va automatiser la configuration d’un cluster Docker Swarm. Il commence par initialiser le cluster sur le manager1 en exécutant la commande docker swarm init, puis récupère les tokens d'adhésion (un pour les managers, un pour les workers) et l'adresse IP de ce nœud. Ensuite, il connecte manager2 au cluster en utilisant le token manager, et fait de même pour les nœuds workers (worker1, worker2, worker3) en utilisant le token worker. Chaque étape utilise des conditions intelligentes (when) pour éviter les erreurs si les nœuds font déjà partie d’un cluster. Grâce à l’utilisation des modules shell, register, et des faits dynamiques (set\_fact), le playbook adapte automatiquement les commandes selon le contexte de chaque hôte, rendant le déploiement totalement automatisé.

Automatisation ansible docker swarm:

Commande : ansible-playbook -i hosts.ini deploy\_swarm.yml

A computer screen shot of a computer code

AI-generated content may be incorrect.

Résultat automatisation:

A screenshot of a computer screen

AI-generated content may be incorrect.

## Partie 4 modification des fichiers:

Fichier index.html :

A screenshot of a computer code

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

Résultat :

A screenshot of a computer

AI-generated content may be incorrect.

J’ai ici, juste modifié fichier index.html pour rendre ma page web plus intéressante.

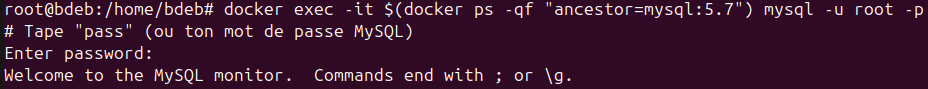
Ajout d’un script JS dans index.html :

A computer screen shot of a program code

AI-generated content may be incorrect.

Ce script JavaScript permet de récupérer les données des visiteurs depuis mon backend Flask, disponible à l’adresse http://192.168.56.101:5000/visiteurs, en utilisant fetch pour faire une requête HTTP GET. Une fois les données reçues et converties en JSON, il parcourt chaque objet (chaque visiteur) et crée dynamiquement une "carte" HTML pour l'afficher. Chaque carte contient le nom, le courriel, la profession, le pays et la date d’inscription du visiteur, puis elle est ajoutée dans le conteneur HTML avec l’ID visiteurs. Cela permet donc d’avoir une interface frontend qui se met à jour automatiquement avec les données présentes en base sans avoir besoin de modifier le HTML à la main.

Insertion de données :



A computer screen shot of a program

AI-generated content may be incorrect.

J’ai inséré des données à travers conteneur MySQL pour les afficher.

Fichier app.py :

A screenshot of a computer program

AI-generated content may be incorrect.

Ce fichier est un script Python qui utilise le micro-framework Flask pour créer une API web simple. Il définit deux routes : la première ('/') retourne simplement un message de bienvenue "Hello from the Backend!" pour tester si le backend fonctionne. La seconde route ('/visiteurs') se connecte à ma base de données MySQL nommée tp2\_db, puis exécute une requête SQL pour récupérer toutes les entrées de la table visiteurs. Les résultats sont ensuite retournés au format JSON grâce à jsonify. La ligne CORS(app) active le support CORS (Cross-Origin Resource Sharing), ce qui permet à des pages web hébergées sur un autre port ou domaine (comme le frontend sur le port 8080) d'accéder à cette API. Enfin, l’application Flask démarre en écoutant sur toutes les interfaces (0.0.0.0) au port 5000.

Fichier DockerFile :

A computer screen shot of white text

AI-generated content may be incorrect.

- FROM python:3.12-slim  
→ Cette ligne indique que l’image de base est une version allégée de Python 3.12, idéale pour créer des conteneurs légers.

- WORKDIR /app  
→ Définit le répertoire de travail à l’intérieur du conteneur comme /app. Toutes les commandes suivantes s’exécuteront à partir de ce dossier.

- COPY requirements.txt .  
→ Copie le fichier requirements.txt de mon projet local dans le répertoire /app du conteneur.

- RUN pip install --no-cache-dir -r requirements.txt  
→ Installe les dépendances Python listées dans requirements.txt, sans garder les fichiers en cache (ce qui allège l’image).

- COPY app.py .  
→ Copie le fichier app.py dans le conteneur, dans le dossier /app.

- CMD ["python", "app.py"]  
→ Spécifie la commande qui sera exécutée quand le conteneur démarre : ici, le script app.py est lancé avec Python.

Fichier requirements.txt :

A black and white text

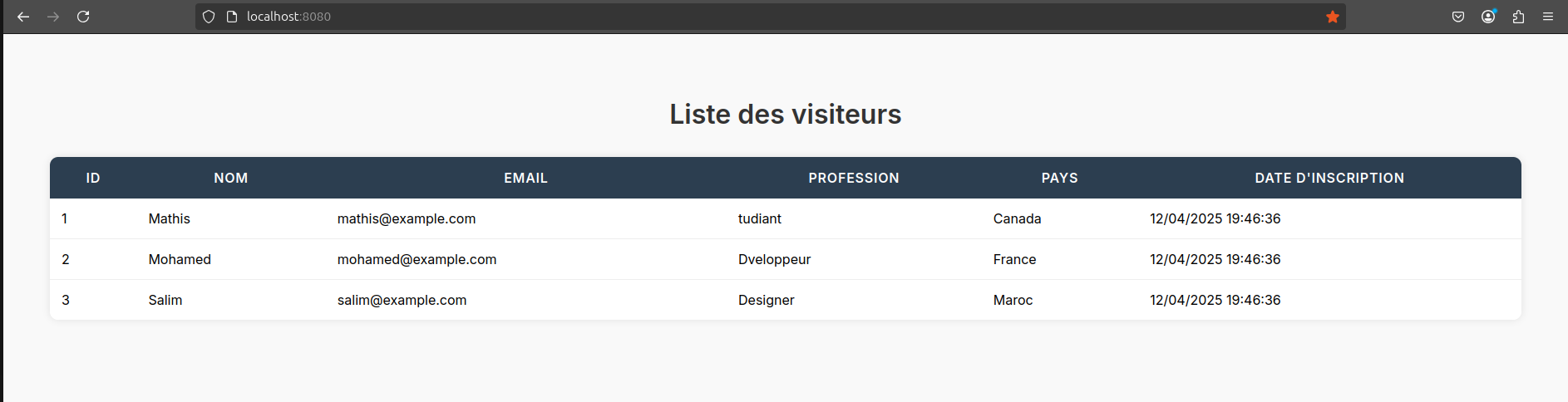
AI-generated content may be incorrect.

- flask  
→ Installe Flask, le micro-framework web Python utilisé pour créer mon API.

- flask-cors  
→ Installe Flask-CORS, une extension qui permet de gérer les CORS (Cross-Origin Resource Sharing), c’est-à-dire les permissions d’accès depuis des domaines différents (comme frontend HTML accédant à backend Flask).

- mysql-connector-python  
→ Installe le connecteur MySQL pour Python, utilisé dans code pour te connecter à la base de données MySQL (mysql.connector.connect(...)).s

Résultat final :

Frontend:

Backend (API GET):

A screenshot of a computer

AI-generated content may be incorrect.

- Le navigateur (via code JavaScript) envoie une requête HTTP GET à l'API Flask (/visiteurs) pour récupérer la liste des visiteurs.

- Mon backend Flask se connecte à la base de données MySQL, exécute une requête SELECT \* FROM visiteurs, et récupère les données.

- Flask convertit les résultats de la base en JSON (un format texte structuré, facile à lire en JS).

- Le navigateur reçoit cette réponse JSON, et l'affiche (comme dans la capture).

cAdvisor :

A screenshot of a computer

AI-generated content may be incorrect.

A blue and white striped background with white gauges

AI-generated content may be incorrect.

A graph with blue lines

AI-generated content may be incorrect.

A graph with red and blue lines

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

A graph with red line

AI-generated content may be incorrect.

cAdvisor (Container Advisor) est un outil open-source développé par Google qui permet de surveiller en temps réel l’utilisation des ressources système des conteneurs Docker. Il fournit une interface graphique intuitive avec des graphiques et jauges pour visualiser l’utilisation du CPU, de la mémoire, du réseau et des processus actifs. Grâce à cette interface, je peux rapidement détecter une surcharge système, surveiller la performance des conteneurs, et assurer une meilleure stabilité et efficacité de application.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.Test sur la machine manager 2 :

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.Test sur la machine worker 1:

A screenshot of a computer

AI-generated content may be incorrect.

Fichier docker-compose.yml final :

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

## Auto-évaluation :