07/01/2024

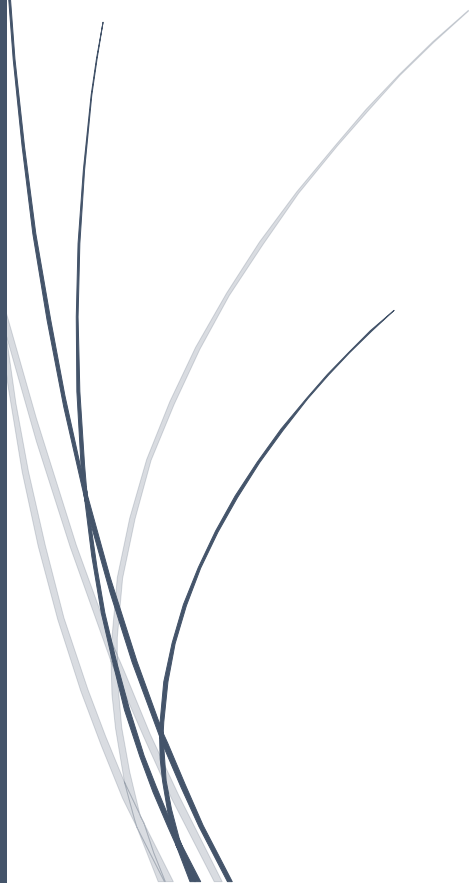
Virtualisation, conteneurisation

HumansBestFriend app?

CATs or DOGs ?

Les membres du groupe sont :

- Achref GAIEB
- Mehdi KRAMA
- Rayan BENLABIDI



esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

Table des matières

INTRODUCTION

Contexte

Le projet "Humans Best Friends" se présente comme une application exploitante l'architecture microservices à l'aide de conteneurs Docker. Python, Node.js et .NET, associés à Redis, sont les technologies centrales pour le déploiement de cette application.

Objectifs

Ce projet vise à illustrer de manière concrète les avantages de l'utilisation de Docker en tant que plateforme de conteneurisation pour la création d'une application distribuée basée sur une architecture microservices. L'objectif central est de mettre en évidence comment Docker simplifie les processus de développement, d'intégration et de déploiement, tout en garantissant la portabilité des applications. Cette étude détaillera les étapes mises en œuvre pour atteindre ces objectifs, mettant en lumière l'efficacité de Docker dans l'écosystème de développement logiciel moderne.

Nous aurons besoin d'utiliser les outils suivants pour ce projet:

- VMware
- ESXi
- VM Ubuntu
- Docker et docker-compose (installé sur le système d'exploitation Ubuntu)

Concepts Docker

1) Concept d'un DockerFile

Un DockerFile est un fichier texte décrivant les différentes étapes permettant de partir d'une base pour aller vers une application fonctionnelle. Docker lit les instructions que vous mettez dans le DockerFile pour créer automatiquement l'image requise.

2) Concept d'un Docker Compose

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services interconnectés.

3) Concept d'un Docker Registry

Une Docker Registry est une application qui permet de distribuer des images Docker au sein de votre organisation.

Configuration de l'Environnement

Après avoir installé la machine virtuelle sur ESXi, nous avons procédé à l'installation de l'environnement Docker en suivant les étapes suivantes : Tout d'abord, nous avons désinstallé les anciennes versions des packages Docker, même s'il s'agissait de la première installation de Docker sur la machine virtuelle, en utilisant la commande suivante :

for pkg in docker.io docker-doc docker-compose docker-compose-v2

podman-docker containerd runc; do sudo apt-get remove $pkg; done

Installation et mise a jour du repository :

#Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -0 /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

#Add the repository to Apt sources:

**echo **

"deb [arch=$(dpkg --print-architecture)

signed-by=/etc/apt/keyrings/docker.gpg]

**https://download.docker.com/linux/ubuntu **

**$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | **

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

Installation du Docker et du docker-compose :

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin docker-compose

Verification installation Docker:

esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

```
root@mehdi:~/esiea-ressources# docker -v
Docker version 24.0.7, build afdd53b
```

Vérification du démarrage correct des services Docker sur Linux Ubuntu.

Utiliser les commandes suivantes pour activer les services :

sudo systemctl enable docker.service

sudo systemctl enable containerd.service

Éviter la nécessité de préfixer chaque commande Docker avec "sudo" :

sudo groupadd docker

sudo usermod -aG docker <votre nom d'utilisateur>

Compose

Dans cette étape, nous souhaitons créer les images en utilisant Docker compose et Compose build. Les fichiers Docker nécessaires se trouvent déjà dans des dossiers distincts. Dans le fichier docker-compose.build.yml présenté ci-dessus, nous avons spécifié le contexte et le chemin d'accès aux fichiers Docker pour chaque service. Les sous-réseaux front-tier et back-tier ont également été inclus, avec certains services exposant des ports vers l'extérieur. Certains services sont lancés en fonction d'autres services (par exemple, le service worker dépend du lancement préalable de redis et postgres).

Pour initier la construction des images, nous utilisons la commande suivante :

docker-compose -f docker-compose.build.yml build

Code du docker-compose.yml :

```
version: '3.8'

services:
  vote:
    image: esiea-ressources/vote
    ports:

"5002:80"
  networks:
front-tier
back-tier

  result:
    image: esiea-ressources/result
    ports:

"5001:80"
  networks:
front-tier
back-tier

  worker:
    image: esiea-ressources/worker
    networks:

back-tier

  seed-data:
    image: esiea-ressources/seed-data
    networks:

front-tier
  depends_on:
vote

  db:
    image: postgres:16.1
    volumes:

db-data:/var/lib/postgresql/data
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: root
    POSTGRES_DB: postgres
  networks:
back-tier

  redis:
    image: redis:latest
    networks:

back-tier

networks:
  back-tier:
  front-tier:

volumes:
  db-data:
```

Création des images

docker-compose -f docker-compose.build.yml build

```
 => => exporting layers                                                    0.0s
 => => writing image sha256:12adf0f4cbb5d732fdfa9f97d6e3d01ee7a4f7c15cc8b  0.0s
 => => naming to docker.io/esiea-ressources/seed-data                      0.0s
Building result
[+] Building 1.3s (12/12) FINISHED                          docker:default
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 528B                                       0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 54B                                           0.0s
 => [internal] load metadata for docker.io/library/node:18-slim            1.1s
 => [1/7] FROM docker.io/library/node:18-slim@sha256:fe687021c06383a2bc5e  0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 593B                                          0.0s
 => CACHED [2/7] RUN apt-get update &&     apt-get install -y --no-instal  0.0s
 => CACHED [3/7] WORKDIR /usr/local/app                                    0.0s
 => CACHED [4/7] RUN npm install -g nodemon                                0.0s
 => CACHED [5/7] COPY package*.json ./                                     0.0s
 => CACHED [6/7] RUN  npm ci &&  npm cache clean --force &&  mv /usr/loca  0.0s
 => CACHED [7/7] COPY . .                                                  0.0s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:c073e516a5561332b8324f27e4e7615710e0cd9ccad96  0.0s
 => => naming to docker.io/esiea-ressources/result                         0.0s
root@mehdi:~/esiea-ressources#
```

Compose build

docker-compose.build.yml

```yaml
version: '3.8'
services:
  worker:
    build:
      context: ./worker
    image: esiea-ressources/worker

  vote:
    build:
      context: ./vote
    image: esiea-ressources/vote

  seed-data:
    build:
      context: ./seed-data
    image: esiea-ressources/seed-data

  result:
    build:
      context: ./result
    image: esiea-ressources/result
```

Lacement de l’application avec :

docker-compose up -d

```
root@mehdi:~/esiea-ressources# docker-compose up -d
Starting esiea-ressources_vote_1    ... done
Starting esiea-ressources_result_1  ... done
Starting esiea-ressources_redis_1   ... done
Starting esiea-ressources_db_1      ... done
Starting esiea-ressources_worker_1  ... done
Starting esiea-ressources_seed-data_1 ... done
```

http://192.168.86.134:5001/

http://192.168.86.134:5002/

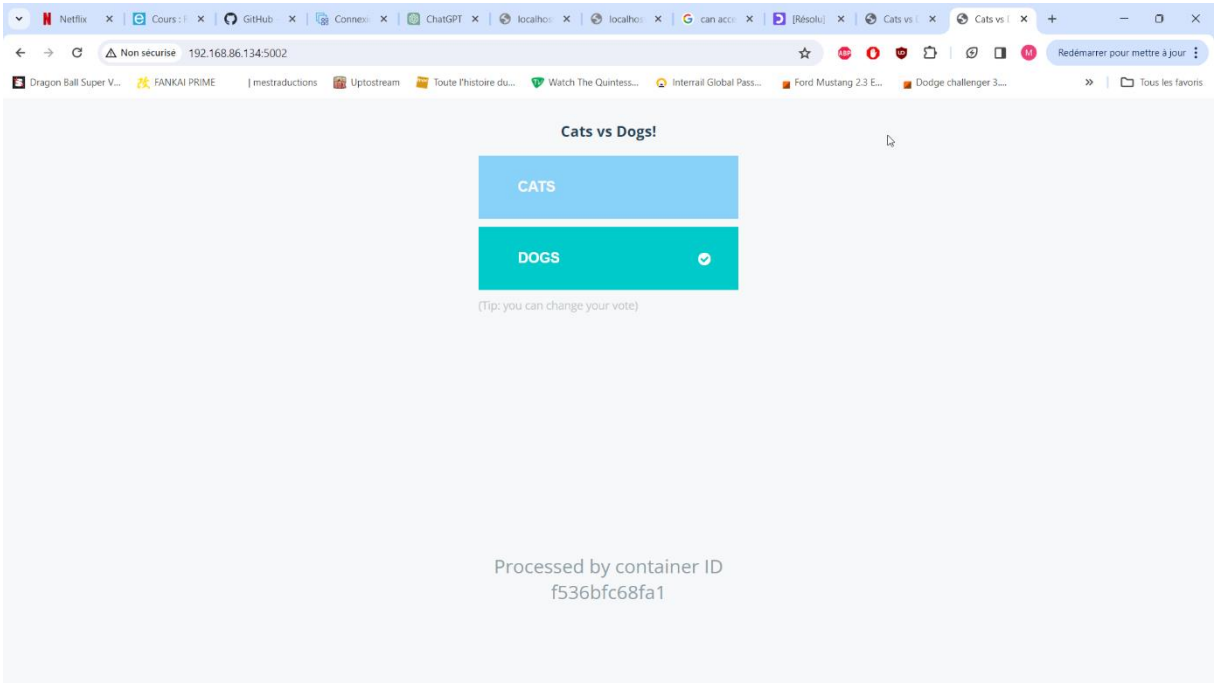
pour vérifié faire **ip -a**

prendre l'adresse **ip:port** et mettre dans le moteur de recherche :

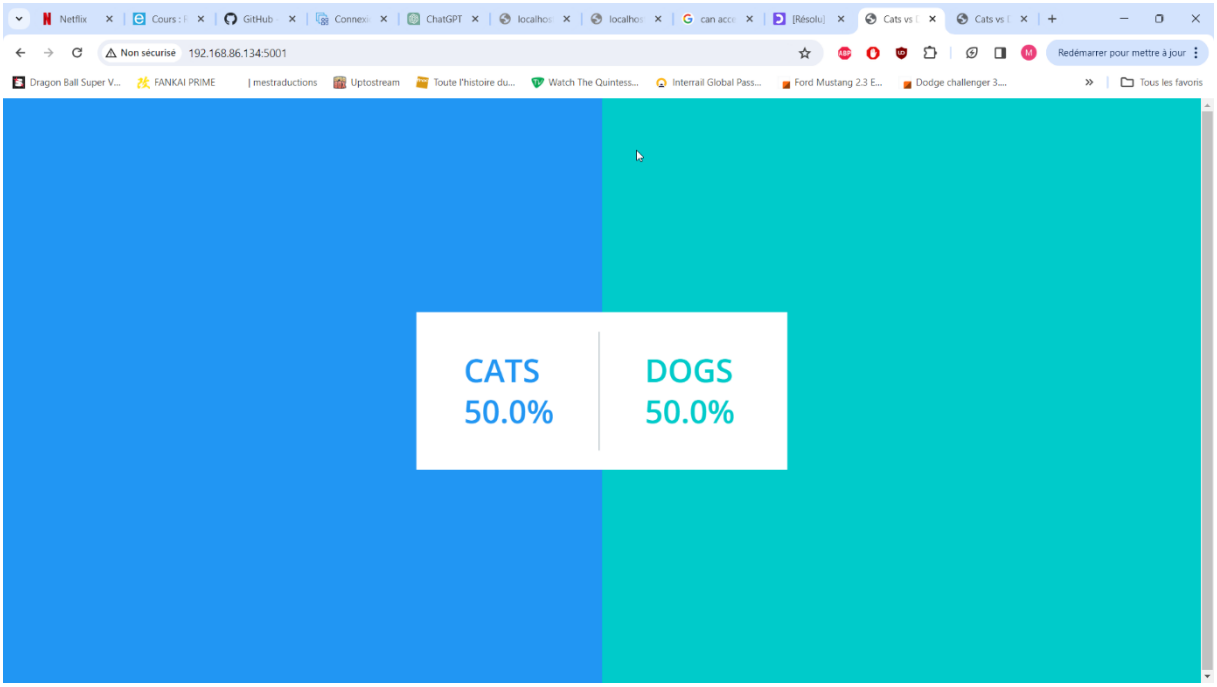
ip:5002 et ip:5001

Développement des Services Docker

Illustration du sites web capturé après la mise en œuvre réussie des conteneurs, mettant en avant le processus de vote destiné aux chiens.


Cats vs Dogs!
CATS
DOGS
(Tip: you can change your vote)
Processed by container ID
f536bfc68fa1

Resultats :


CATS
50.0%
DOGS
50.0%

Lien github : https://github.com/Mehdiiiii94/virtu-esiea

Conclusion

En résumé, le déploiement de l'application HumansBestFriend avec Docker et Docker Compose a été une expérience instructive. Nous avons réussi à mettre en place une architecture distribuée avec des services interagissant de manière fluide. Cependant, nous avons rencontré un défi fréquent lié à la vérification de l'état de santé des conteneurs Redis et Postgre. Les vérifications de santé fournies ont souvent conduit à déclarer l'un des deux conteneurs comme "non fonctionnel", ce qui bloquait le déploiement de l'application. Cela nécessitait un redémarrage du conteneur concerné à chaque fois. Cette situation souligne l'importance cruciale des mécanismes de surveillance et de gestion de la santé des conteneurs dans des déploiements complexes.