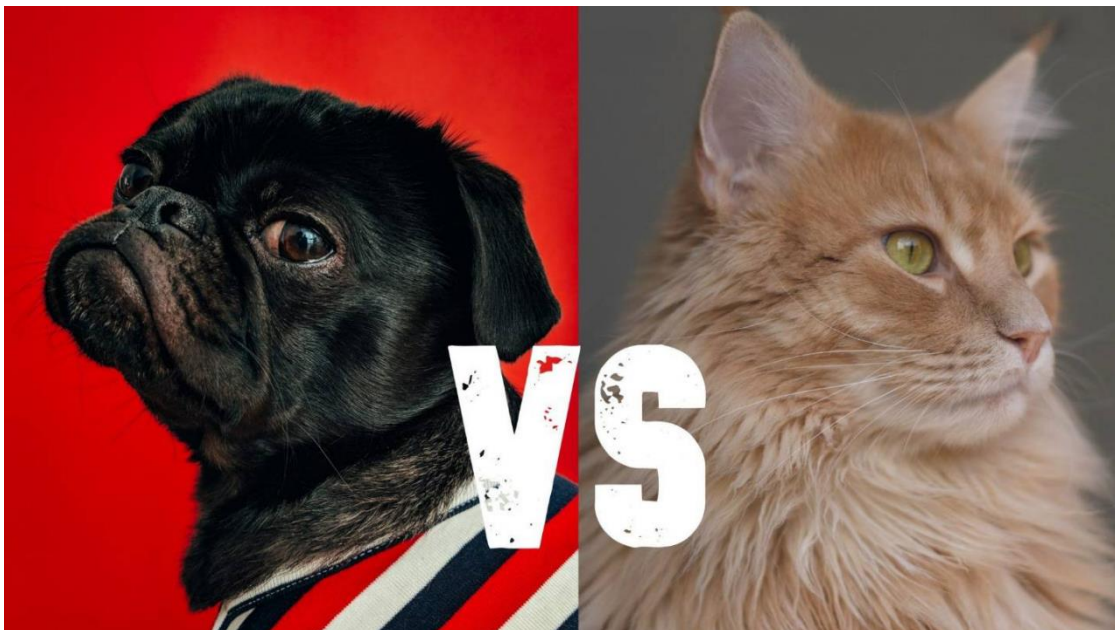


VIRTUALISATION/CONTENEURISATION

Rapport de projet

CATs or DOGs ?



Sommaire et Prérequis

Sommaire

Sommaire	2
Prérequis	2
I. Introduction	3
II. Mise en place de l'environnement	4
III. Mise en place de Docker	9
IV. Création des fichiers docker-compose.build.yml et docker-compose.yml	11
V. Lancer l'application avec Kubernetes	20
VI. Conclusion	23

Prérequis

Pour le bon déroulement de ce projet nous avons besoin de plusieurs logiciels et applications de supervisons afin de poursuivre ce projet dans de bonne conditions.

Voici une liste de quelque de ses applications :

- Une Licence VMware
- Une Machine Virtuelle ESXI
- Diverses Machines Virtuel
- Putty
- Un Compte Github

I. Introduction

Ce rapport présente la conception, la mise en œuvre et le déploiement d'une infrastructure de vote virtuelle basée sur des technologies de virtualisation et de conteneurisation. L'objectif principal de ce projet était de créer un environnement virtualisé à l'aide de VMware ESXi pour exécuter une machine virtuelle Ubuntu, et d'utiliser des conteneurs Docker pour mettre en place une application de vote en temps réel sur le thème des chats versus chiens.

Le projet se compose de plusieurs composants clés, chacun utilisant des technologies spécifiques pour permettre le fonctionnement complet de l'application de vote. Voici un aperçu des principales technologies et composants impliqués :

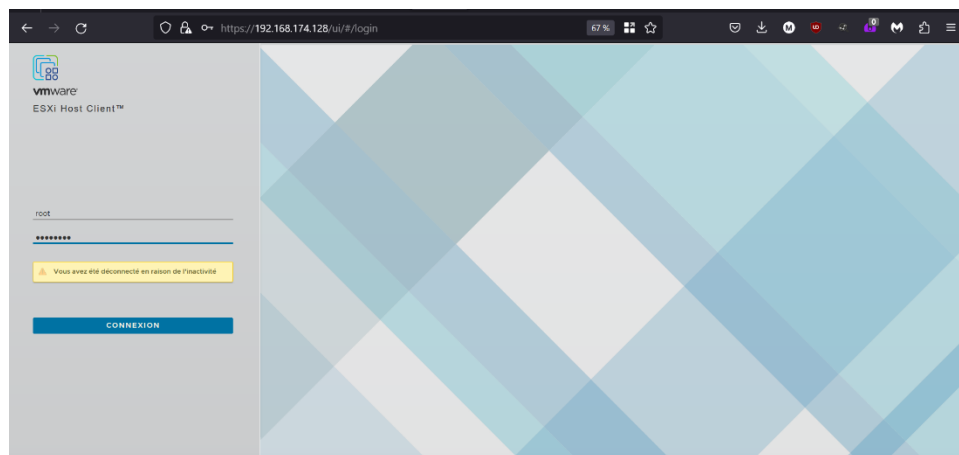
1. **Front-End Web App en Python pour Voter** : Cette application offre une interface utilisateur conviviale permettant aux utilisateurs de voter entre deux options, chats et chiens, dans un environnement web.
2. **Redis pour la Collecte des Votes** : Utilisé comme système de stockage clé-valeur, Redis joue un rôle essentiel dans la collecte instantanée et la gestion des nouveaux votes entrants.
3. **Travailleur .NET pour la Gestion des Votes** : Ce composant consomme les votes enregistrés et les stocke dans une base de données PostgreSQL, assurant ainsi la persistance des données de manière efficace et fiable.
4. **Base de Données PostgreSQL avec Volume Docker** : PostgreSQL est utilisé comme système de gestion de base de données relationnelle, intégré dans un conteneur Docker avec un volume dédié pour garantir la préservation des données.
5. **Application Web Node.js pour Afficher les Résultats en Temps Réel** : Cette application présente les résultats du vote de manière dynamique et en temps réel, offrant une expérience utilisateur enrichie.

II. Mise en place de l'environnement

Dans cette section, nous déployons notre premier outil sous VMware. L'ESXi agit comme un hyperviseur qui supervise et gère nos différentes machines. Il nous permet de créer un réseau virtuel entre ces éléments, offrant un contrôle total et simplifié de notre environnement, que ce soit pour nos machines virtuelles ou pour l'infrastructure réseau.

La première chose à faire est l'installations de notre ESXi, pour cela nous pouvons aller sur le site de VMware et choisir la version que nous souhaitons. Une fois cela fait, nous lançons l'image que nous avons choisi sur VMware et configurons notre superviseur. Cette configuration est rapide et nous permettra de mieux exploiter nos outils.

Une fois la configuration finie, nous voici sur notre page de connexion comme sur l'image ci-dessous :

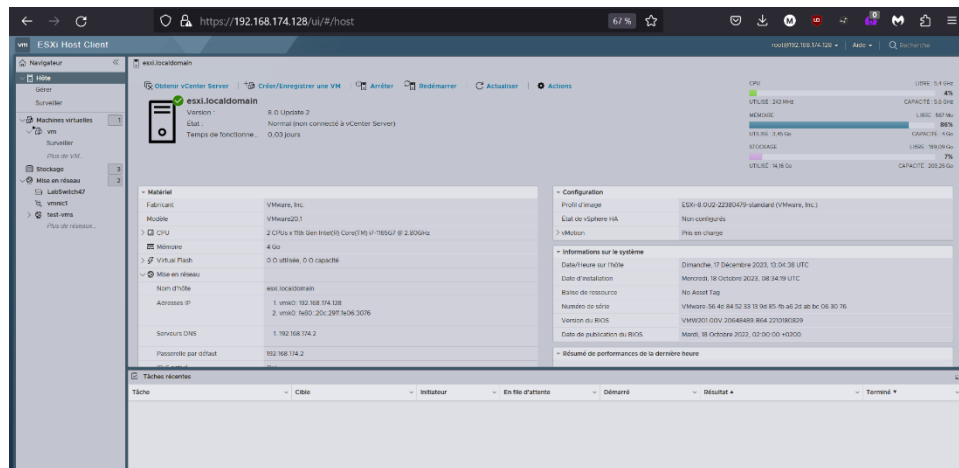


Nous saisissons nos identifiants précédemment enregistrés. Une fois dans notre interface ESXi, nous pouvons visualiser diverses informations comme l'utilisation de la CPU, de la mémoire et bien d'autres.

JANEZ Maxime
MAMAN Mayane
RODRIGUEZ Geoffroy

4AFISA ~ TC3 ~ TD47

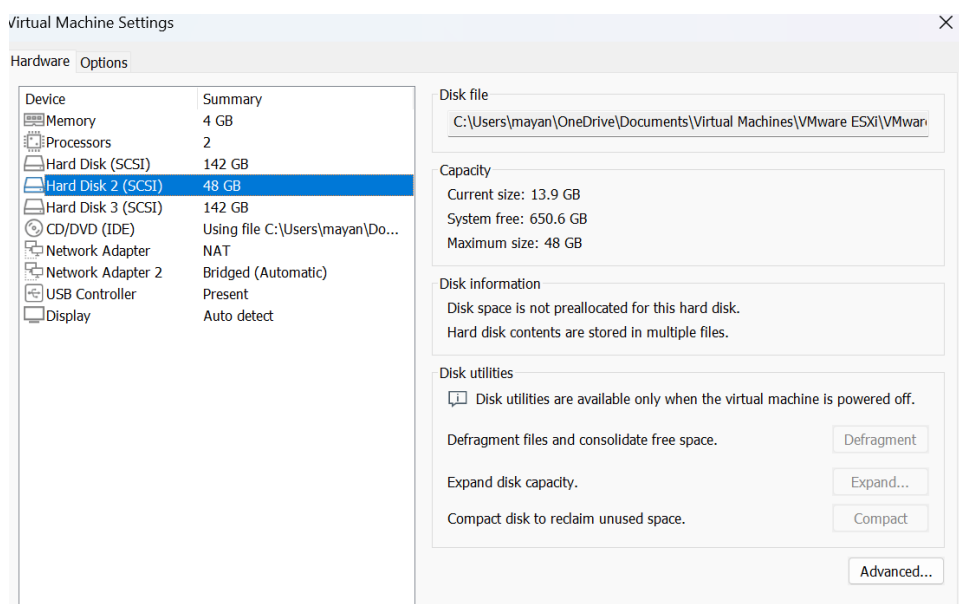
07/01/2024



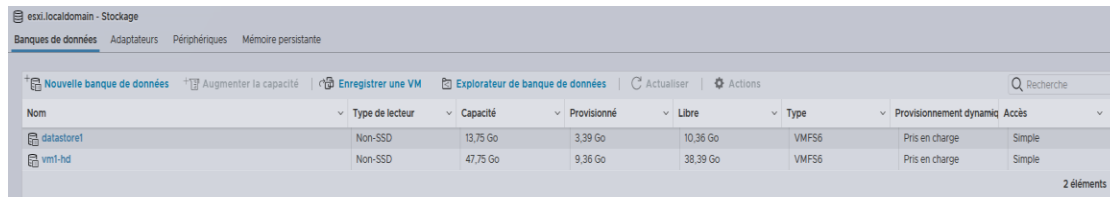
Afin de maintenir le bon fonctionnement de notre future machine virtuelle, nous allons rajouter un disque-dur virtuel sur notre ESXI. Celui-ci nous permettra de rajouter des emplacements de stockage libre pour nos machines Ubuntu.

Pour accomplir cela, nous arrêtons ESXI, accédons à ses paramètres de machine, ajoutons un disque dur, puis redémarrons ESXI.

Voici le menu pour ajouter un disque-dur à notre ESXI :



Nous intégrons une nouvelle banque de données qui intègre le disque dur nouvellement ajouté.



The screenshot shows the 'Storage' tab in the vSphere Client. It displays a table of datastores. The table has columns for Name, Type of reader, Capacity, Provisioned, Free, Type, Dynamic provisioning, and Access. Two datastores are listed: 'datastore1' and 'vm1-hd'.

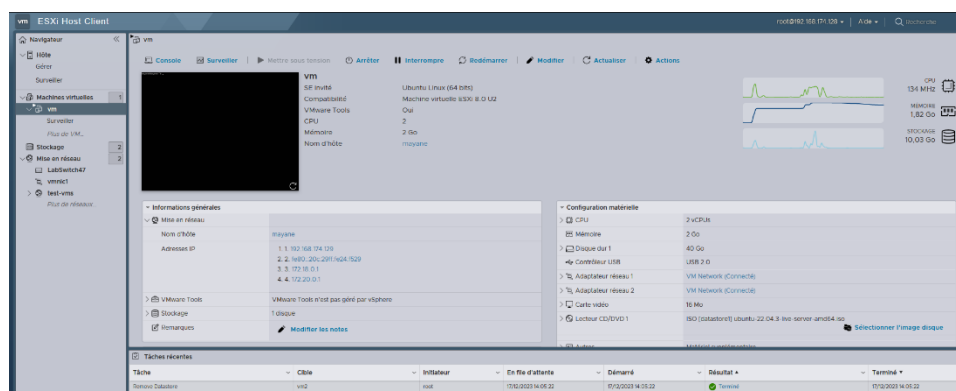
Nom	Type de lecteur	Capacité	Provisionné	Libre	Type	Provisionnement dynamique	Accès
datastore1	Non-SSD	13,75 Go	3,39 Go	10,36 Go	VMFS6	Pris en charge	Simple
vm1-hd	Non-SSD	47,75 Go	9,36 Go	38,39 Go	VMFS6	Pris en charge	Simple

2 éléments

Nous allons maintenant ajouter une nouvelle machine virtuelle directement depuis notre ESXI, pour cela nous allons dans l'onglet " Machines virtuelles " et nous cliquons sur " Ajouter une nouvelle machine virtuelle".

Maintenant que nous avons installé notre première machine virtuelle sur notre ESXI, nous allons accéder à son interface afin de visualiser certaines informations et vérifier si elle a été correctement mise en œuvre. Nous pouvons depuis cette interface avoir un système de log complet ainsi que les caractéristiques de notre nouvelle machine.

L'interface de supervision de la nouvelle machine virtuelle :

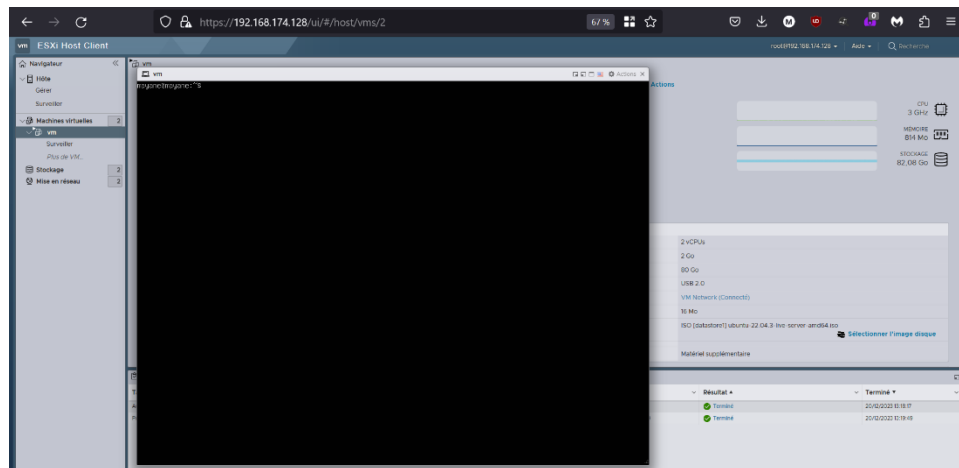


The screenshot shows the 'ESXi Host Client' interface. The left sidebar shows the navigation tree with 'Machines virtuelles' selected. The main area displays the configuration of a new virtual machine. The 'Informations générales' section shows the name 'vmt1', the OS 'Ubuntu Linux (64 bits)', and the hardware version 'VMware Tools'. The 'Configuration matérielle' section shows the hardware configuration: 2 vCPUs, 2 GB of memory, 60 GB of disk, USB 2.0 controller, VM Network connection, 16 Mo of cache, and 1 SCSI controller. The 'Tâches récentes' section shows a list of tasks.

Tâche	Cible	Initiateur	En file d'attente	Démarré	Résultat	Terminé
Remise à jour	vmt1	root	17/12/2023 14:05:22	17/12/2023 14:05:22	Terminé	17/12/2023 14:05:22

Nous n'avons plus qu'à démarrer notre machine virtuelle depuis notre interface. Une fois démarré nous entrons dans l'invite de commande pour installer les packages qui nous seront utiles lors de notre utilisation.

Voici l'invite de commande une fois la machine mise en route :



Pour commencer, nous procédons à l'installation du logiciel qui nous autorise à accéder à distance à notre machine, à savoir le package de connexion à distance tel que le protocole SSH. Voici les lignes de commandes à taper pour son installation.

Installation de ssh sur Ubuntu :

```
$ sudo apt-get install ssh
```

Modification des données pour autoriser la connexion avec mot de passe :

```
$ sudo vim /etc/ssh/sshd_config
```

Décommenter la ligne :

```
PasswordAuthentication yes
```

Grâce à l'infrastructure réseau configurée sur notre ESXi, notre machine virtuelle peut obtenir une adresse IP. Cette adresse IP nous autorise à prendre le contrôle à distance de cette machine virtuelle depuis notre ordinateur physique, en utilisant l'outil Putty, par exemple.

Voici les commandes et les retours de notre invite de commande où nous visualisons l'IP de la machine :

JANEZ Maxime
MAMAN Mayane
RODRIGUEZ Geoffroy

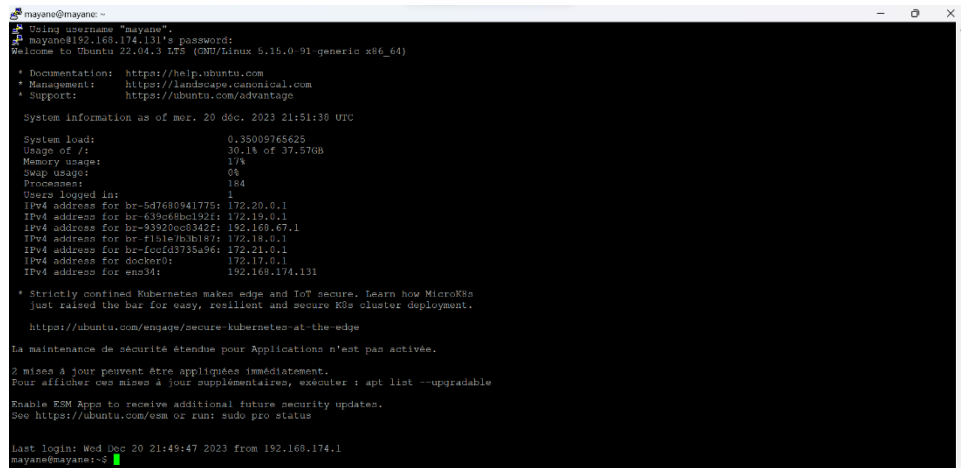
4AFISA ~ TC3 ~ TD47

07/01/2024



Une fois la prise de contrôle à distance effectuée en saisissant correctement les paramètres sur Putty, nous pouvons passer aux étapes suivantes de notre projet.

Interface de commande via Putty :



III. Mise en place de Docker

Nous allons maintenant installer Docker sur notre machine Ubuntu avec la prise de contrôle à distance de Putty. La mise en place de Docker est simple et rapide mais est essentielle dans notre projet.

Docker est largement adopté comme plateforme de conteneurisation en raison de sa grande flexibilité, et c'est avec cette flexibilité que nous comptons l'intégrer à notre projet VotingApp.

Voici les diverses commandes utilisées pour notre installation et configuration de Docker :

```
# 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 -o /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
```

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

Pour simplifier notre configuration et gagner du temps, nous avons recours à des commandes permettant de contourner l'utilisation systématique du mot-clé "sudo" devant nos diverses instructions.

Voici les commandes pour ne plus utiliser "sudo" :

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ sudo systemctl enable docker.service
```

JANEZ Maxime
MAMAN Mayane
RODRIGUEZ Geoffroy

4AFISA ~ TC3 ~ TD47

07/01/2024

Nous avons donc vu grâce à cette partie comment installer et configurer l'outil Docker sur notre machine virtuel, passons maintenant à la partie suivante.

IV. Création des fichiers docker-compose.build.yml et docker-compose.yml

Pour mener à bien cette étape du projet, nous débutons par revisiter le [lien](#) du dépôt GitHub attribué à ce projet. Ensuite, nous procédons au clonage du projet existant sur notre machine Ubuntu, nous permettant ainsi d'accéder aux ressources déjà présentes pour leur utilisation. Voici la commande pour cloner le projet :

```
$ git clone https://github.com/pascalito007/esiea-ressources.git
```

Grace aux ressources qui nous a été donné sur le GitHub du projet, il ne nous reste plus qu'à écrire un fichier "docker-compose.build.yml" et un fichier " docker-compose.yaml " pour construire le projet et exécuter notre sondage sur le web. Toutefois, nous devons tout de même faire attention aux ports que nous choisissons lors de la configuration de notre fichier .yaml. Ces ports seront les portes d'entrée pour arriver sur nos pages web.

Avant de passer à l'automatisation des fichiers avec la commande `docker-compose`, nous allons d'abord nous concentrer sur la partie des commandes `docker` afin de construire les images et de les lancer.

Avant de lancer chaque commande, il faut se positionner sur les répertoires respectifs des images que l'on veut créer.

Exemple : `$ cd vote`

```
$ docker build -t vote-app:latest . #Le point fait référence au
répertoire où se trouve le fichier Docker
$ docker build -t result-app:latest .
$ docker build -t worker-app:latest .
$ docker build -t seed-app:latest .
```

Exemple de rendu :

[illegible]

07/01/2024

[illegible]

```
$ docker run -d -p 5001:80 --name vote-container vote-app:latest
$ docker run -d -p 5002:80 --name result-container result-app:latest
$ docker run -d --name worker-container worker-app:latest
$ docker run -d --name seed-container seed-app:latest
```

```
mayane@mayane:~/votingApp/vote$ docker run -d -p 5001:80 --name name vote-container vote-app:latest
b0ec18b73d8877b1d991d8d0a5de2414e7b9dfe4d531d9254cf2e7056119c0ea

mayane@mayane:~/votingApp/result$ docker run -d -p 5002:80 --name result-container result-app:latest
f46853d778309ecbe48610b5f7b643189461316bde8ab8edb7524171f2dcfe5a

mayane@mayane:~/votingApp/worker$ docker run -d --name worker-container worker-app:latest
1002d2e3dbfee2a30213b72b2abe7bd58a1b78843e16846a8c8b94873737ebc0

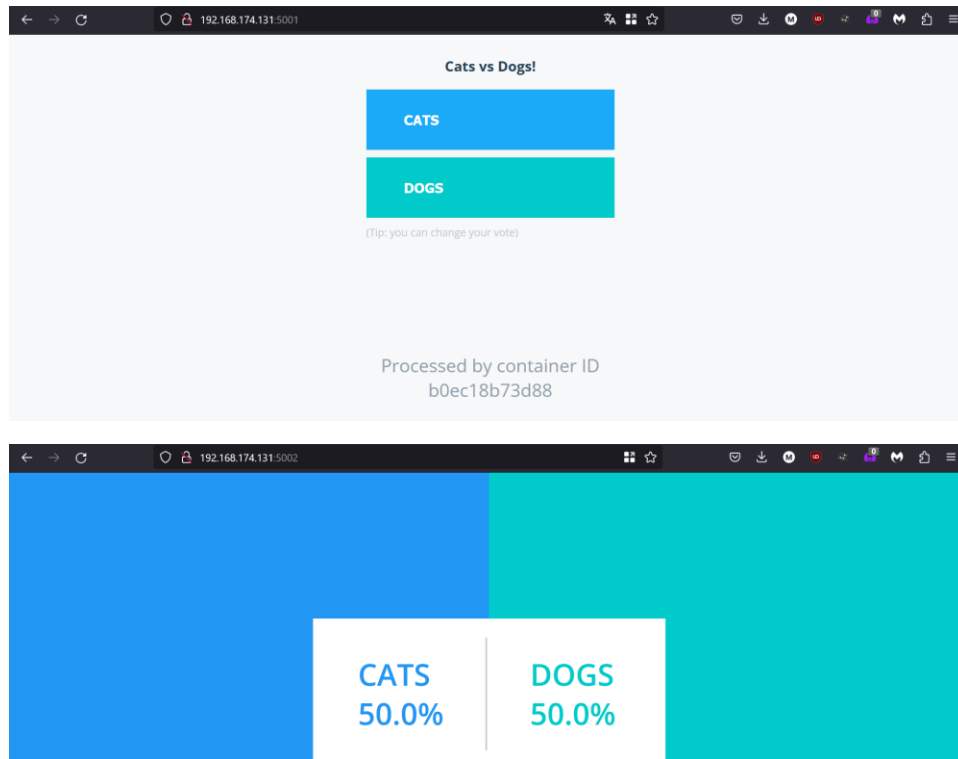
mayane@mayane:~/votingApp/seed-data$ docker run -d --name seed-container seed-app:latest
6b96b551569e3e89d3238f31491f687d9736891412397699013059fba664d233
```

Vérification si les conteneurs se sont bien lancés :

Lançons la commande : `$ docker ps`

```
mayane@mayane:~/votingApp/seed-data$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
5fce74650f8f   seed-app:latest "/bin/sh -c /seed/ge..." 4 seconds ago  Up 3 seconds  :::5002->80/tcp                    seed-container
1002d2e3dbfe   worker-app:latest "dotnet Worker.dll"        20 minutes ago Up 20 minutes  :::5002->80/tcp                    worker-container
f46853d77830   result-app:latest "/usr/bin/tini -- no..." 26 minutes ago Up 26 minutes  0.0.0.0:5002->80/tcp, :::5002->80/tcp result-container
b0ec18b73d88   vote-app:latest  "gunicorn app:app -b..." 28 minutes ago Up 28 minutes  0.0.0.0:5001->80/tcp, :::5001->80/tcp vote-container
```

Voyons si nous arrivons à accéder à notre application via le navigateur. Prenons le port que nous lui avons attribué ainsi que l'adresse IP de notre machine :



Maintenant passons à la partie automatisé avec les fichiers que nous allons créer.

- Commençons par la rédaction du fichier `docker-compose.build.yml` :

Ce fichier permet de définir les différents services composant une application, spécifiant comment ils doivent être construits et configurés. Chaque service correspond à une partie de l'application. En résumé, ce script permet de décrire et d'orchestrer les différentes parties de l'application, simplifiant ainsi le

processus de déploiement et de gestion des conteneurs. Cependant il ne lance pas directement l'application, nous lancerons l'application avec le second fichier docker-compose.yml.

```
version: '3.8'
services:
  # Worker service
  worker:
    build:
      context: ./worker
      dockerfile: Dockerfile

  # Vote service
  vote:
    build:
      context: ./vote
      dockerfile: Dockerfile

  # Seed data service
  seed-data:
    build:
      context: ./seed-data
      dockerfile: Dockerfile

  # Result service
  result:
    build:
      context: ./result
      dockerfile: Dockerfile

  # Database service
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - db-data:/var/lib/postgresql/data
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - back-tier
      - cats-or-dogs-network

  # Redis service
  redis:
    image: redis
```

```
networks:
  back-tier:
  cats-or-dogs-network:

volumes:
  db-data:
```

Pour lancer ce fichier nous allons utiliser la commande :

```
$ docker-compose -f docker-compose.build.yml build
```

Voici le résultat :

```
mayane@mayane:~/votingApp$ docker-compose -f docker-compose.build.yml build
db uses an image, skipping
redis uses an image, skipping
Building worker
[*] Building 0.4s (16/16) FINISHED
-> [internal] load build definition from Dockerfile
-> [internal] load build definition from Dockerfile
-> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0
-> [internal] load metadata for mcr.microsoft.com/dotnet/runtime:7.0
[*] Build 1/7 FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:dc5f17c847f9c7a1a0194920b0e6d01956af95f534b61a2ce6db8563471c
-> [internal] load build context
-> transferring context: 85k
-> [stage 1/7] FROM mcr.microsoft.com/dotnet/runtime:7.0@sha256:b41a241da8624e65448d3b0ec42152f10a7b10b2d1aa1a912e230b285631
-> CACHED [stage 1/7] WORKDIR /app
-> CACHED [build 2/7] RUN echo "I am running on linux/amd64, building for linux/amd64"
-> CACHED [build 3/7] WORKDIR /source
-> CACHED [build 4/7] COPY *.csproj .
-> CACHED [build 5/7] RUN dotnet restore -a amd64
-> CACHED [build 6/7] COPY . .
-> CACHED [build 7/7] RUN dotnet publish -o release -s /app -s amd64 --self-contained false --no-restore
-> CACHED [stage 1/3] COPY --from=build /app .
-> exporting to image
-> exporting layers
-> writing image sha256:b0c6f0e1ae89e6c5bed850c45dae52b25e76decaf4db1d1e63623319443cf95e
-> naming to docker.io/library/votingapp_worker
Building vote
[*] Building 1.3s (11/11) FINISHED
-> [internal] load build definition from Dockerfile
-> [internal] load build definition from Dockerfile
-> [internal] load metadata for docker.io/library/python:3.11-slim
-> [internal] load build context
-> transferring context: 274k
-> [stage 1/5] FROM docker.io/library/python:3.11-slim@sha256:f64ae771f30991cf30064d9f9fd6e1acc07927f165f4e37ea3f0b893e0c3f
-> CACHED [stage 2/5] RUN apt-get update && apt-get install -y --no-install-recommends curl && rm -rf /var/lib/apt/lists/*
-> CACHED [stage 3/5] WORKDIR /usr/local/app
-> CACHED [stage 4/5] COPY requirements.txt ./requirements.txt
-> CACHED [stage 5/5] RUN pip install --no-cache-dir -r requirements.txt
Building seed-data
[*] Building 1.0s (10/10) FINISHED
-> [internal] load build definition from Dockerfile
-> [internal] load build definition from Dockerfile
-> [internal] load metadata for docker.io/library/python:3.9-slim
-> [1/5] FROM docker.io/library/python:3.9-slim@sha256:90ea53c4437a761f30e0c05b43c9f0b4d902ec23594f804739c50da3a0bed
-> [internal] load build context
-> transferring context: 101k
-> CACHED [2/5] RUN apt-get update && apt-get install -y --no-install-recommends apache2-utils && rm -rf /var/lib/apt/lists/*
-> CACHED [3/5] WORKDIR /seed
-> CACHED [4/5] COPY .
-> CACHED [5/5] RUN python make_data.py
-> exporting to image
-> exporting layers
-> writing image sha256:8557d913ecf0baf674fccc59774c238e60349b46015917eb646975e8345e2
-> naming to docker.io/library/votingapp_seed-data
Building result
[*] Building 1.3s (12/12) FINISHED
-> [internal] load build definition from Dockerfile
-> [internal] load build definition from Dockerfile
-> [internal] load metadata for docker.io/library/node:18-slim
-> [internal] load build context
-> transferring context: 54k
-> [internal] load metadata for docker.io/library/node:18-slim
-> [1/7] FROM docker.io/library/node:18-slim@sha256:f6e97021c86353a2bc5eaf6dc29b627ed28a55f6bdfb0ca193f0c24b763c37
-> CACHED [2/7] RUN apt-get update && apt-get install -y --no-install-recommends curl tini && rm -rf /var/lib/apt/lists/*
-> CACHED [3/7] WORKDIR /usr/local/app
-> CACHED [4/7] RUN npm install --global node_modules
-> CACHED [5/7] COPY package.json .
-> CACHED [6/7] RUN npm ci && npm cache clean --force && mv /usr/local/app/node_modules /node_modules
-> CACHED [7/7] COPY . .
-> exporting to image
-> exporting layers
-> writing image sha256:127fa0d93208a171bdad40a8bba1a288471b035a7b72a80bd4f071bf98c27
-> naming to docker.io/library/votingapp_result
```

Nous allons utiliser cette commande pour voir si les images sont bien lancées afin de vérifier qu'il fonctionne bien :

```
$ docker-compose -f docker-compose.build.yml up
```

```
mayane@mayane:~/VotingApp$ docker-compose -f docker-compose.build.yml up
Starting votingapp_redis_1 ... done
Starting votingapp_vote_1 ... done
Starting votingapp_db_1 ... done
Starting votingapp_seed-data_1 ... done
Starting votingapp_result_1 ... done
Starting votingapp_worker_1 ... done
Attaching to votingapp_vote_1, votingapp_db_1, votingapp_seed-data_1, votingapp_result_1, votingapp_redis_1, votingapp_worker_1
db_1 | PostgreSQL Database directory appears to contain a database. Skipping initialization
db_1 |
db_1 | 2024-01-02 21:09:13.136 UTC [1] LOG: starting PostgreSQL 15.5 on x86_64-pc-linux-musl, compiled by gcc (Alpine 13.2.1_git20231014) 13.2.1 20
31014, 64-bit
db_1 |
db_1 | 2024-01-02 21:09:13.137 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db_1 | 2024-01-02 21:09:13.137 UTC [1] LOG: listening on IPv6 address "::", port 5432
db_1 | 2024-01-02 21:09:13.149 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1 | 2024-01-02 21:09:13.160 UTC [23] LOG: database system was shut down at 2024-01-02 19:45:19 UTC
db_1 | 2024-01-02 21:09:13.223 UTC [1] LOG: database system is ready to accept connections
seed-data_1 | This is ApacheBench, Version 2.3 <Revision: 1903618 >
seed-data_1 | Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
seed-data_1 | Licensed to The Apache Software Foundation, http://www.apache.org/
seed-data_1 | Benchmarking vote (be patient)
seed-data_1 | apr_socket_recv: Connection refused (111)
seed-data_1 | This is ApacheBench, Version 2.3 <Revision: 1903618 >
seed-data_1 | Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
seed-data_1 | Licensed to The Apache Software Foundation, http://www.apache.org/
seed-data_1 | Benchmarking vote (be patient)
seed-data_1 | apr_socket_recv: Connection refused (111)
seed-data_1 | This is ApacheBench, Version 2.3 <Revision: 1903618 >
seed-data_1 | Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
seed-data_1 | Licensed to The Apache Software Foundation, http://www.apache.org/
seed-data_1 | Benchmarking vote (be patient)
seed-data_1 | apr_socket_recv: Connection refused (111)
seed-data_1 | This is ApacheBench, Version 2.3 <Revision: 1903618 >
seed-data_1 | Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
seed-data_1 | Licensed to The Apache Software Foundation, http://www.apache.org/
seed-data_1 | Benchmarking vote (be patient)
seed-data_1 | apr_socket_recv: Connection refused (111)
redis_1 | 11c 02 Jan 2024 21:09:14.045 # WARNING Memory overcommit must be enabled! Without it, a background save or replication may fail under low memem
ry condition. Being disabled, it can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. To fix this issue
add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
redis_1 | 11c 02 Jan 2024 21:09:14.046 * c0000000c000 Redis is starting c0000000c000
redis_1 | 11c 02 Jan 2024 21:09:14.046 * Redis version=7.2.3, bits=64, commit=000000000, modified=0, pid=1, just started
redis_1 | 11c 02 Jan 2024 21:09:14.046 # Warning: no config file specified, using the default config. In order to avoid a warning, you should specify a config file in /etc/postgrescon
```

- Passons au second fichier maintenant qui permettrait le lancement de l'application.

Ce fichier permet de définir et gérer un ensemble de services Docker. Ce fichier définit les paramètres pour démarrer les conteneurs de manière cohérente et pour interconnecter les différents services d'une application, ce qui facilite le déploiement et la gestion de l'application dans différents environnements, comme le développement, le test et la production.

```
services:
  # Service for voting
  vote:
    build:
      context: ../vote
      target: dev
    depends_on:
      redis:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
    volumes:
      - ../vote:/usr/local/app
    ports:
      - "5001:80"
    networks:
      - front-tier
      - back-tier

# Service for result handling
```



```
result:
  build: ./result
  # use nodemon rather than node for local dev
  entrypoint: nodemon --inspect=0.0.0.0 server.js
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - ./result:/usr/local/app
  ports:
    - "5002:80"
    - "127.0.0.1:9229:9229"
  networks:
    - front-tier
    - back-tier

# Worker service
worker:
  build:
    context: ./worker
  depends_on:
    redis:
      condition: service_healthy
    db:
      condition: service_healthy
  networks:
    - back-tier

# Redis service
redis:
  image: redis:alpine
  volumes:
    - ./healthchecks:/healthchecks
  healthcheck:
    test: /healthchecks/redis.sh
    interval: "5s"
  networks:
    - back-tier

# Postgres database service
db:
  image: postgres:15-alpine
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
  volumes:
    - db-data:/var/lib/postgresql/data
    - ./healthchecks:/healthchecks
```

```
healthcheck:
  test: /healthchecks/postgres.sh
  interval: "5s"
networks:
  - back-tier

# Service to seed the database with votes
# Will run only with the "seed" profile
# docker compose --profile seed up -d
seed:
  build: ./seed-data
  profiles: ["seed"]
  depends_on:
    vote:
      condition: service_healthy
  networks:
    - front-tier
  restart: "no"

volumes:
  db-data:

networks:
  front-tier:
  back-tier:
  cats-or-dogs-network:
    driver: bridge
```

Nous allons ensuite écrire une commande qui va nous permettre de démarrer, d'exécuter et de connecter entre eux tous les services du fichier " docker-compose.yml ", simplifiant ainsi le déploiement et la gestion d'une application complexe. :

```
$ docker-compose up
```

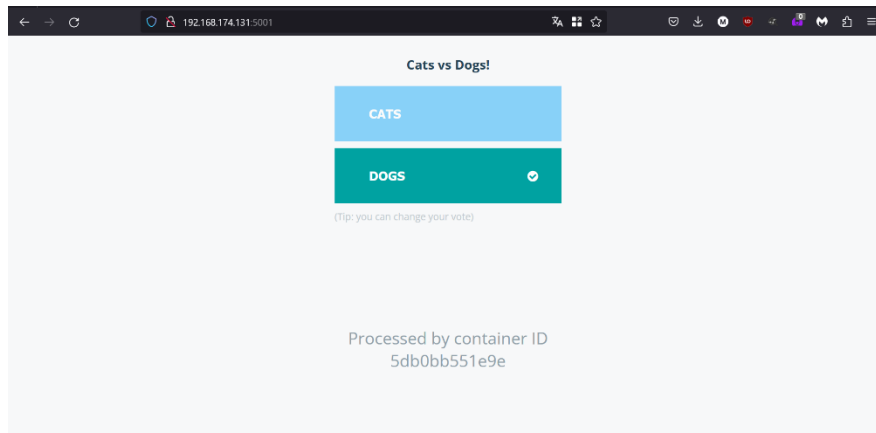
Cette commande aura d'abord pour effet l'extraction des données, elle peut durer quelques minutes. Puis la création de nos divers conteneurs comme visible sur l'image ci-dessous :



```
mayane@mayane:~/votingApp$ docker compose up
Creating network for votingApp: votingApp_default
Creating votingapp-redis-1
Creating votingapp-db-1
Creating votingapp-vote-1
Creating votingapp-worker-1
Creating votingapp-result-1
Attaching to votingapp-redis-1, votingapp-db-1, votingapp-result-1, votingapp-vote-1, votingapp-worker-1
votingapp-redis-1 |
votingapp-db-1 | PostgreSQL database directory appears to contain a database. Skipping initialization
votingapp-db-1 |
votingapp-redis-1 |
```

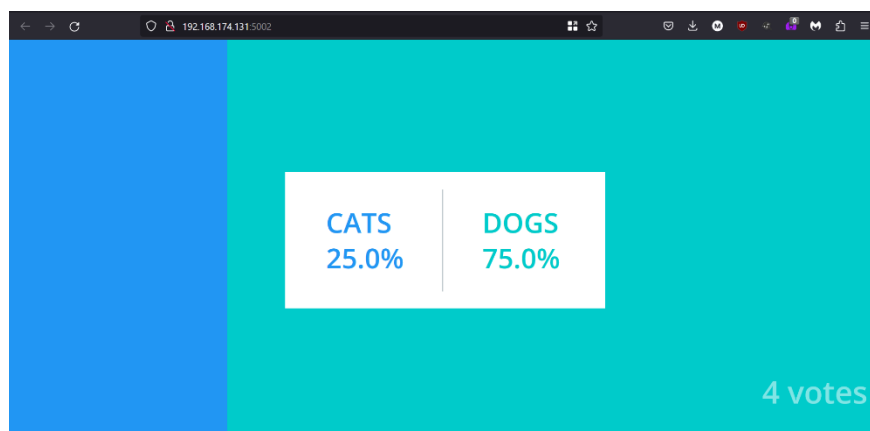
À présent, ouvrons notre navigateur et saisissons l'URL de recherche que nous avons définie antérieurement dans le fichier "docker-compose.yml" que nous avons rédigé : adresseIP:port

Nous arrivons alors sur l'interface où nous pouvons choisir notre animal de compagnie favori :



Pour visualiser le résultat de notre sondage, il nous suffit de modifier le port de notre url et nous sommes redirigés sur l'autre interface contenant le résultat.

Pour effectuer des tests fonctionnels de notre projet, nous avons également installé l'extension “ [Firefox Multi-Account Containers](#) “. Cet outil nous autorise l'ouverture simultanée de plusieurs onglets avec différents comptes, facilitant ainsi la possibilité de voter plusieurs fois sur un seul navigateur. Cette approche nous permet de vérifier le bon fonctionnement de notre page web et la réactivité de ses fonctionnalités pour afficher les résultats dynamiques.



V. Lancer l'application avec Kubernetes

Initialement, nous avons compris que le projet incluait la mise en place d'un cluster Kubernetes, même si cela ne faisait finalement pas partie des spécifications initiales. Cependant, nous avons décidé de quand même ajouter cette composante pour deux raisons principales : tout d'abord, notre compréhension initiale nous laissait penser que la configuration d'un cluster Kubernetes était nécessaire pour une gestion avancée des conteneurs. De plus, nous avons reconnu que l'intégration de Kubernetes offrirait une expérience d'apprentissage supplémentaire et pourrait potentiellement enrichir la robustesse et la flexibilité de notre infrastructure. Par conséquent, bien que ce ne fût pas initialement spécifié, nous avons volontairement ajouté cette partie à notre projet.

Installons d'abord kubernetes :

```
$ sudo apt-get install kubernetes  
  
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubec  
tl"  
  
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubect1.sha  
256"
```

Vérifier l'output s'il est ok :

```
$ echo "$(cat kubect1.sha256) kubect1" | sha256sum --check  
  
mayane@mayane:~/votingApp$ echo "$(cat kubect1.sha256) kubect1" | sha256sum --check  
kubect1: OK
```

Installer kubect1 :

```
$ sudo install -o root -g root -m 0755 kubect1  
/usr/local/bin/kubect1
```

Nous avons ensuite utilisé Minikube qui permettrait de mettre en place un cluster Kubernetes local sur une machine individuelle, nous n'aurons pas besoin d'environnement complexe grâce à cela.

Installation et lancement de Minikube :

```
$ curl -LO  
https://storage.googleapis.com/minikube/releases/latest/minikube-  
linux-amd64
```

```
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube  
$ minikube start
```

Maintenant écrivons les fichiers yaml pour lancer l'application dans un cluster Kubernetes.

Dans le contexte de déploiement d'applications, l'utilisation de fichiers YAML pour Kubernetes offre une approche modulaire et efficace. Chaque fichier spécifie la configuration nécessaire pour déployer des éléments distincts d'une application, allant de la base de données aux services de vote et de file d'attente.

L'ensemble de ces fichiers permet une gestion simplifiée et évolutive de l'application, en utilisant les fonctionnalités de conteneurisation de Docker et les capacités d'orchestration de Kubernetes. Cette approche offre une manière pratique et structurée de déployer et de gérer des applications complexes, en les séparant en blocs configurables et interconnectés, simplifiant ainsi leur maintenance et leur évolution.

Les fichiers se trouvent dans un dossier spécifique sur notre [repository GitHub](#).

À présent, il est temps d'instancier les déploiements à partir des fichiers YAML :

```
$ kubectl create -f k8s-specifications/
```

Vérifier que les configurations ont bien été déployés :

```
$ kubectl get deployments
```

```
mayane@mayane:~/example-voting-app$ kubectl get deployments  
NAME      READY   UP-TO-DATE   AVAILABLE   AGE  
db        1/1     1             1           110s  
redis     1/1     1             1           110s  
result    1/1     1             1           110s  
vote      1/1     1             1           110s  
worker    1/1     1             1           110s  
mayane@mayane:~/example-voting-app$
```

```
$ kubectl get pods
```

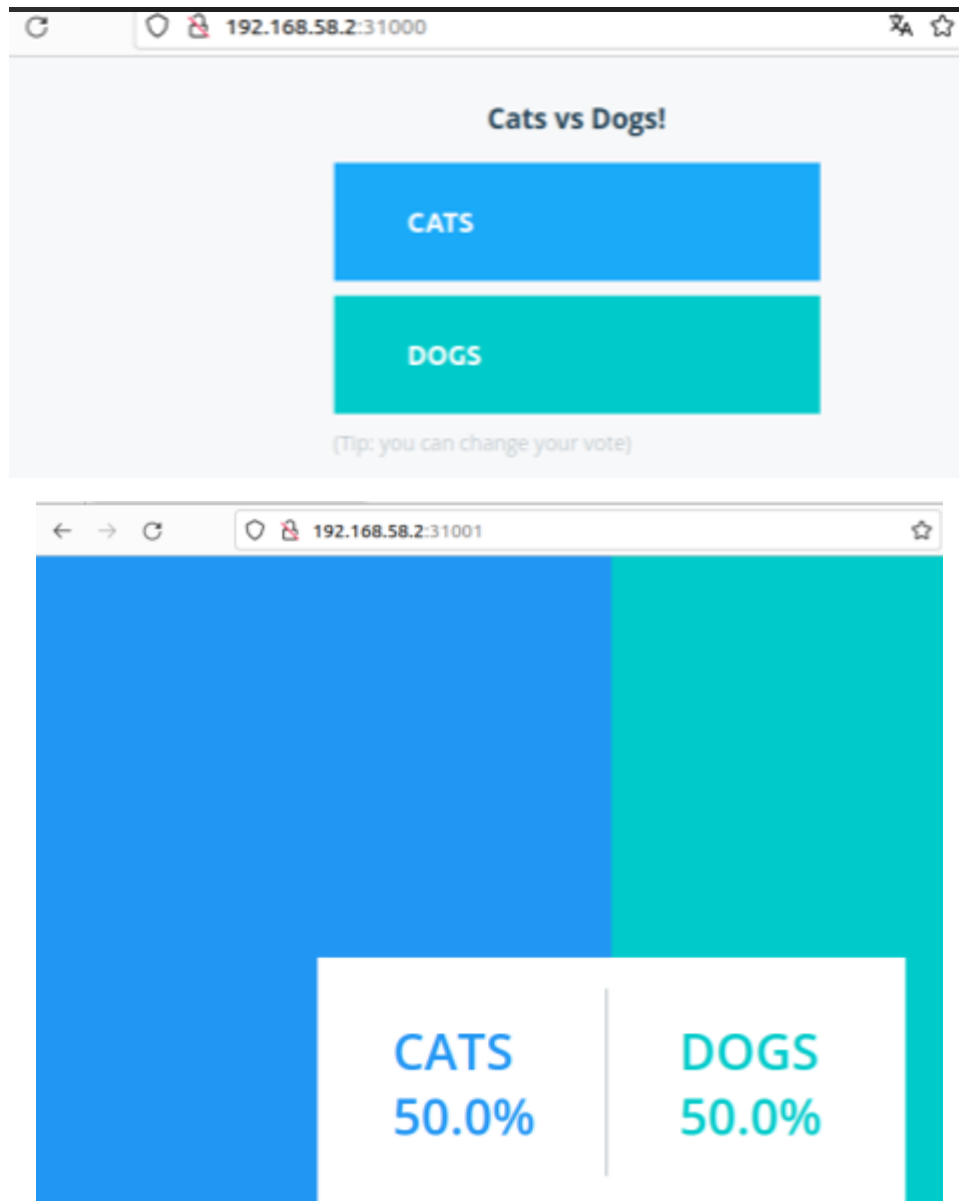
```
mayane@mayane:~/example-voting-app$ kubectl get pods  
NAME                                READY   STATUS    RESTARTS   AGE  
db-6d9f87bb9b-rfrc                 1/1     Running   0           75s  
redis-77fccb7f9-6jh0s              1/1     Running   0           75s  
result-4b4b0cfe0-ab7bb             1/1     Running   0           75s  
vote-565bd759-9jkq                 1/1     Running   0           75s  
worker-7dd74ebbb-7nq3              1/1     Running   0           75s  
mayane@mayane:~/example-voting-app$
```

Maintenant allons sur le navigateur et vérifions si tout s'est bien configuré.

Tout d'abord il faut passer par l'IP du cluster de Minikube :

```
$ minikube ip  
192.168.58.20
```

Ouvrons une fenêtre sur le navigateur :



Nous avons enfin un cluster Kubernetes qui fonctionne.

VI. Conclusion

Ce projet de déploiement d'une infrastructure de vote virtuelle basée sur des technologies de virtualisation et de conteneurisation a démontré avec succès la puissance et la flexibilité de ces approches pour la création d'applications modernes et évolutives.

L'intégration d'VMware ESXi pour l'hébergement de la machine virtuelle Ubuntu, associée à l'utilisation de Docker pour la conteneurisation des différents composants de l'application de vote, a permis de créer un environnement agile et hautement configurable. Cette approche a facilité le déploiement, la gestion et la mise à l'échelle des différents services sans impacter la stabilité globale du système.

L'ajout ultérieur d'un cluster Kubernetes a apporté une dimension nouvelle à l'infrastructure, offrant une orchestration et une gestion plus avancées des conteneurs. Cela a permis une optimisation supplémentaire en termes de scalabilité, de gestion des ressources et de déploiement automatique des services, même si cette partie n'était pas initialement demandée.

De plus, notre [compte GitHub](#) a été publié avec l'ensemble des fichiers et configurations utilisés au cours de ce projet, offrant ainsi une opportunité pour d'autres passionnés ou professionnels de s'inspirer de notre démarche et de nos réalisations.

En conclusion, ce projet a mis en évidence l'impact essentiel de la virtualisation, de la conteneurisation et de l'orchestration dans la construction d'infrastructures modernes.