Youssef GRINE
AbdelJabbar HAKEM
Souhail EL MAKTAFI

Groupe 48

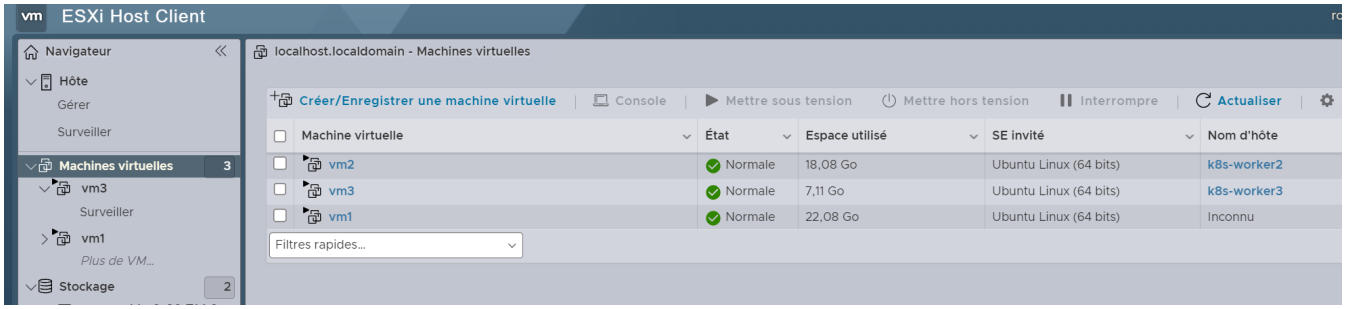

# HumansBestFriend app? CATs or DOGs ?

# Introduction

Ce rapport fait partie de notre projet de virtualisation. Il consiste à déployer une application en utilisant docker et kubernetes, le tout en utilisant 3 VM avec EsXi.
Une Vm est le master(“k8s-master”), et les deux autres sont des workers(“k8s-worker2 et k8s-worker3”).

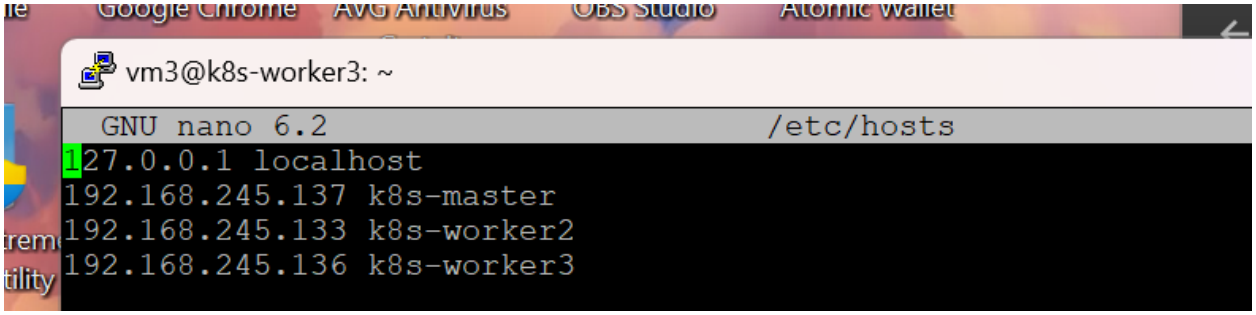
# Configuration de EsXI et des VM

Voici une capture d'écran de mon interface Esxi et on peut voir qu’on a configuré trois VM, qui sont reliés à un switch pour pouvoir communiquer.




Ces trois VM tournent sous ubuntu server 20.04. On a eu des problèmes avec la VM master car elle n’avait pas assez de RAM ce qui a provoqué pas mal de problèmes lors de l’utilisation de kubernetes.

Ensuite, pour chaque VM, on a modifié le fichier /etc/hosts pour attribuer des noms facilement utilisables pour chaque VM.




# Premier demarrage

Après avoir cloné le dépôt Github dans la VM master, on a dû créer un fichier docker build et un docker build compose pour pouvoir utiliser l’application.

Docker.build.yml

```yaml
version: '3.8'

networks:
  front-tier:
  back-tier:

services:
  worker:
    image: esiea-ressources-worker
    networks:
      - back-tier
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy

  vote:
    build:
      context: ./vote
    ports:
      - "5002:80"
    networks:
      - front-tier
      - back-tier
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5002"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
    volumes:
      - ./vote:/usr/local/app

  redis:
    image: redis
    networks:
```

```
    - back-tier
  volumes:
    - "./healthchecks:/healthchecks"
  healthcheck:
    test: /healthchecks/redis.sh
    interval: "5s"

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

seed-data:
  image: esiea-ressources-seed-data
  networks:
    - front-tier
  profiles:
    - seed
  depends_on:
    vote:
      condition: service_healthy
  restart: "no"

result:
  image: esiea-ressources-result
  ports:
    - "5001:80"
    - "127.0.0.1:9229:9229"
  networks:
    - back-tier
  depends_on:
```

```
      db:
        condition: service_healthy

volumes:
  db-data:
```

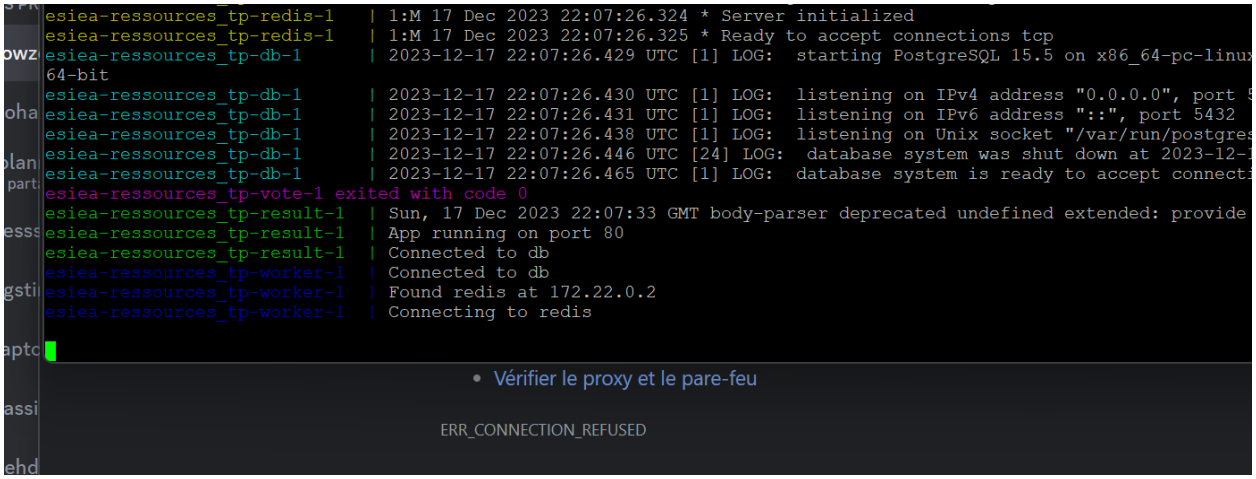
Docker.compose.build.yml

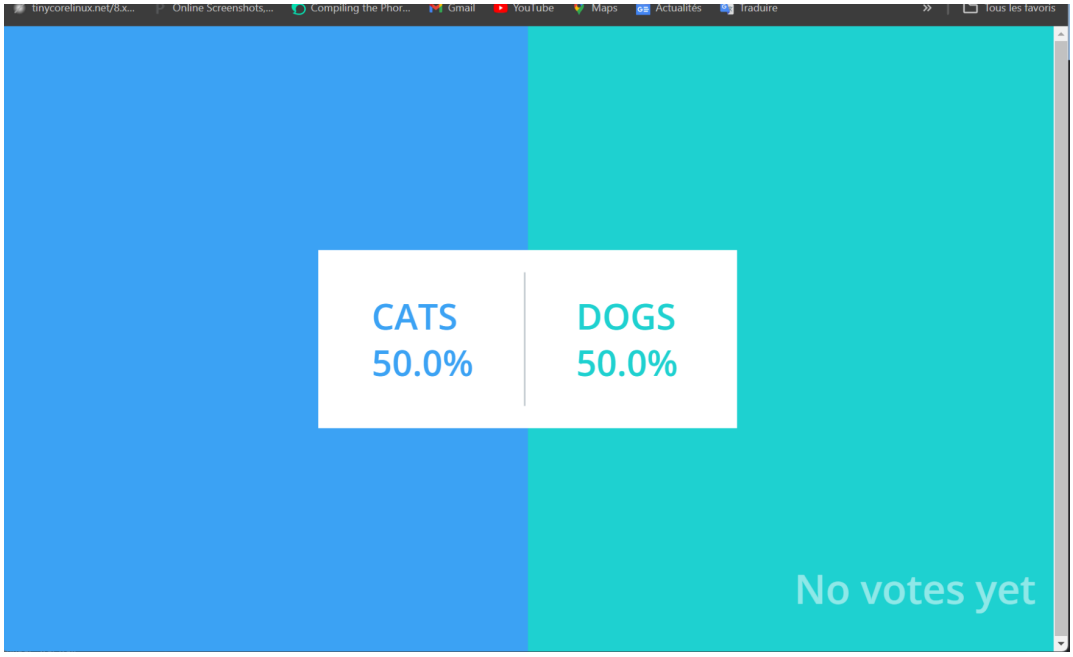
```
version: '3.8'

networks:
  front-tier:
  back-tier:

services:
  worker:
    build:
      context: ./worker
    networks:
      - back-tier
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy

  vote:
    build:
      context: ./vote
    ports:
      - "5002:80"
    networks:
      - front-tier
      - back-tier
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5002"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 10s
```

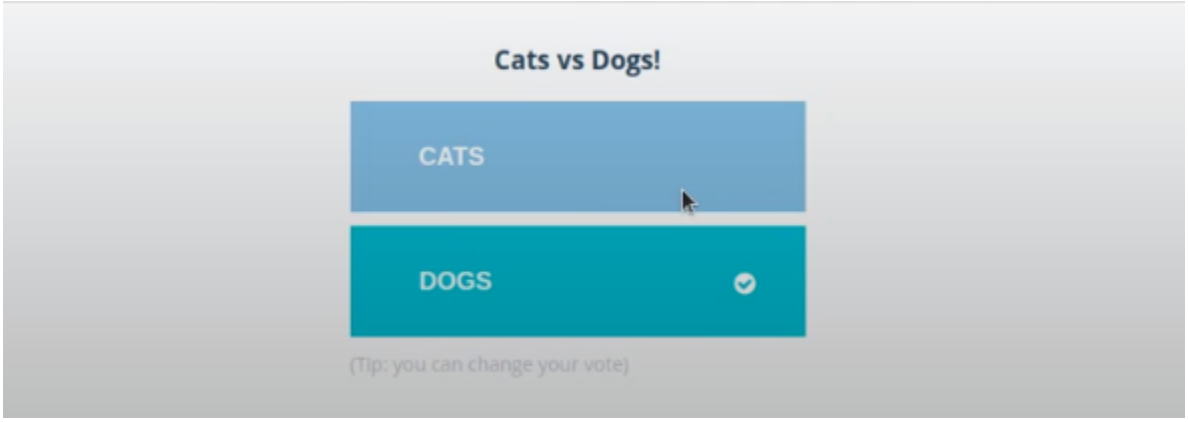
```
    volumes:
      - ./vote:/usr/local/app

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

  db:
    image: postgres:15-alpine
    networks:
      - back-tier
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"

  seed-data:
    build:
      context: ./seed-data
    networks:
      - front-tier
    profiles:
      - seed
    depends_on:
      vote:
        condition: service_healthy
    restart: "no"

  result:
    build:
      context: ./result
    ports:
```

```
      - "5001:80"
    networks:
      - back-tier
    depends_on:
      db:
        condition: service_healthy
    entrypoint: nodemon --inspect=0.0.0.0 server.js
    volumes:
      - ./result:/usr/local/app

volumes:
  db-data:
```

Après avoir placé ces fichiers dans le dossier de l'application, on a utilisé la commande docker compose up et on a obtenu la page de vote et la page de résultat dans le port 5001 et 5002.






# Utilisation de kubernetes

On a installé sur chaque vm kubernetes avec kubelet, kubeadm et kubectl.
Ensuite, on a choisi une vm comme master et les deux autres comme workers.
Dans la VM master, on a creer le cluster avec la commande:

*sudo kubeadm init --pod-network-cidr=192.168.245.0/24*

Avec cette commande, nous avons créer le cluster mais aussi on a obtenu des instruction par la suite, comme le token ou la creation de fichiers de configuration:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Ceci est la commande pour joindre les workers au cluster :
kubeadm join 192.168.245.132:6443 --token 4t9pfh.l4fmyou3r5wod2bt --discovery-token-cacert-hash sha256:84d55e1d6d9e34f7c2e35446f9ff87777362abf389aab3bee90cf020f90300da

Et maintenant on peut vérifier la mise en place du cluster:

```
worker@master-virtual-machine:~/Desktop/New Folder$ kubectl get nodes
NAME                     STATUS   ROLES           AGE   VERSION
master-virtual-machine   Ready    control-plane   79s   v1.28.2
worker-virtual-machine   Ready    <none>          7s    v1.28.2
```

Voici un exemple d'un cluster avec un master et un worker.

Une fois le cluster mis en place, il nous a fallu créer ce qu'on appelle des "manifests" pour kubernetes. Voici un exemple de manifeste:

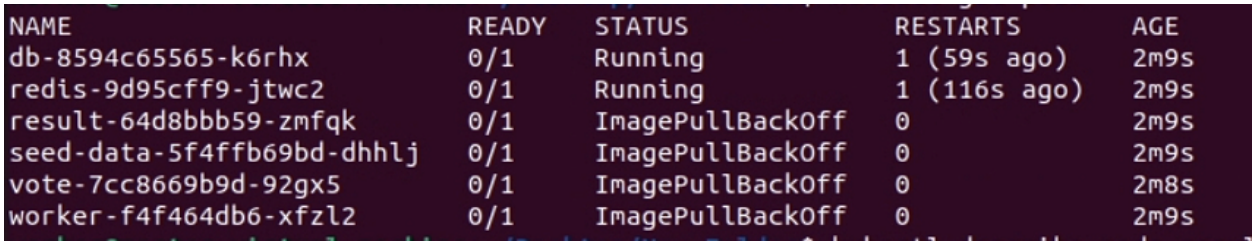
```
apiVersion: v1
kind: Service
metadata:
  name: result
spec:
  selector:
        app: result
  ports:
        - name: http
        protocol: TCP
        port: 80
        targetPort: 5001
        - name: debugger
        protocol: TCP
        port: 9229
        targetPort: 9229

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result
spec:
  replicas: 1
  selector:
        matchLabels:
        app: result
  template:
        metadata:
        labels:
        app: result
        spec:
        containers:
        - name: result
        image: esiea-ressources-result
```

Il comporte une partie Service, celle-ci permet de déclarer et de définir les services réseau qui exposent des applications déployées dans le cluster.
Ainsi qu'une partie Deployment, qui quant à elle permet de déclarer et de définir comment déployer et gérer des applications dans le cluster, en spécifiant des détails tels que le nombre de réplicas, l'image Docker à utiliser, et les stratégies de mise à jour.

Il nous faut appliquer les manifestes avec cette commande :

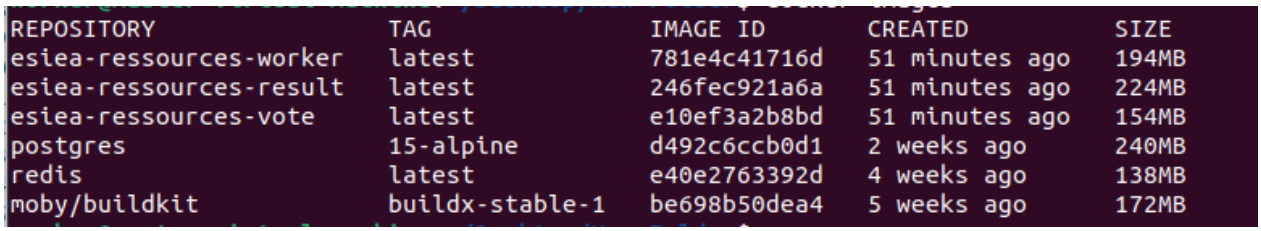
kubectl apply -f ./

Malheureusement, suite à un problème dont nous n'avons pas réussi à comprendre la source, voici les déploiements :
Avec la commande kubectl get pods



| NAME | READY | STATUS | RESTARTS | AGE |
|---|---|---|---|---|
| db-8594c65565-k6rhx | 0/1 | Running | 1 (59s ago) | 2m9s |
| redis-9d95cff9-jtwc2 | 0/1 | Running | 1 (116s ago) | 2m9s |
| result-64d8bbb59-zmfqk | 0/1 | ImagePullBackOff | 0 | 2m9s |
| seed-data-5f4ffb69bd-dhhlj | 0/1 | ImagePullBackOff | 0 | 2m9s |
| vote-7cc8669b9d-92gx5 | 0/1 | ImagePullBackOff | 0 | 2m8s |
| worker-f4f464db6-xfzl2 | 0/1 | ImagePullBackOff | 0 | 2m9s |

On peut voir qu'ils comportent une erreur "ImagePullBackOff". D'après ce que nous avons pu trouver sur internet j'ai cru comprendre qu'il n'arrive pas à accéder aux images qui sont pourtant bien build :
Avec la commande docker images:



| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|---|---|---|---|---|
| esiea-ressources-worker | latest | 781e4c41716d | 51 minutes ago | 194MB |
| esiea-ressources-result | latest | 246fec921a6a | 51 minutes ago | 224MB |
| esiea-ressources-vote | latest | e10ef3a2b8bd | 51 minutes ago | 154MB |
| postgres | 15-alpine | d492c6ccb0d1 | 2 weeks ago | 240MB |
| redis | latest | e40e2763392d | 4 weeks ago | 138MB |
| moby/buildkit | buildx-stable-1 | be698b50dea4 | 5 weeks ago | 172MB |

Nous avons essayé d'uploader les images sur docker hub mais nous n'avons pas réussi..

En théorie c'est la dernière étape pour mettre notre application sur un cluster Kubernetes.