Nguyen Dang Raphaël
Segard Thomas
Vinadelle Corentin

Projet de Virtualisation

I - Introduction :

La virtualisation et la conteneurisation sont des outils que chaque ingénieur et futur ingénieur doivent maîtriser. C'est donc dans cette optique que nous allons réaliser ce projet afin de comprendre et d'apprendre à utiliser ses outils.

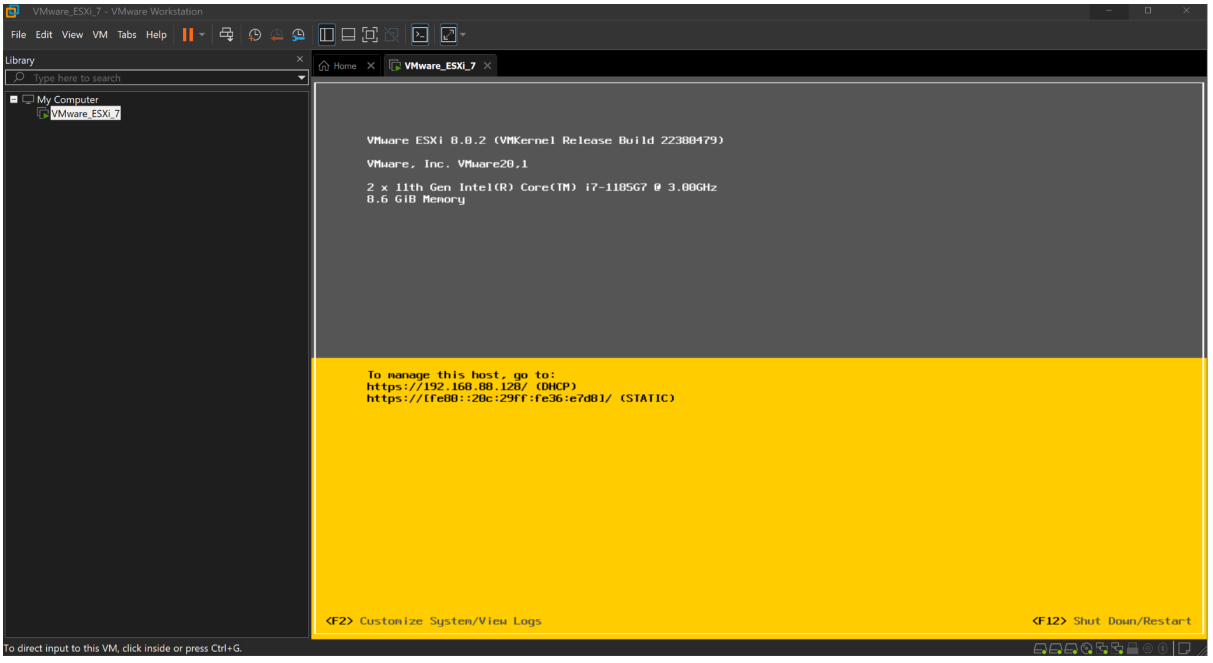
Le but est de réussir à déployer une application à l'aide de docker puis docker compose dans un premier temps et ensuite kubernetes (cluster). Mais pour commencer, il nous faut configurer notre espace de travail.

Ici le github du projet:

https://github.com/CorentinVinadelle/esiea-ressources

II-Hôte VMWare EXSI avec 3 VMs

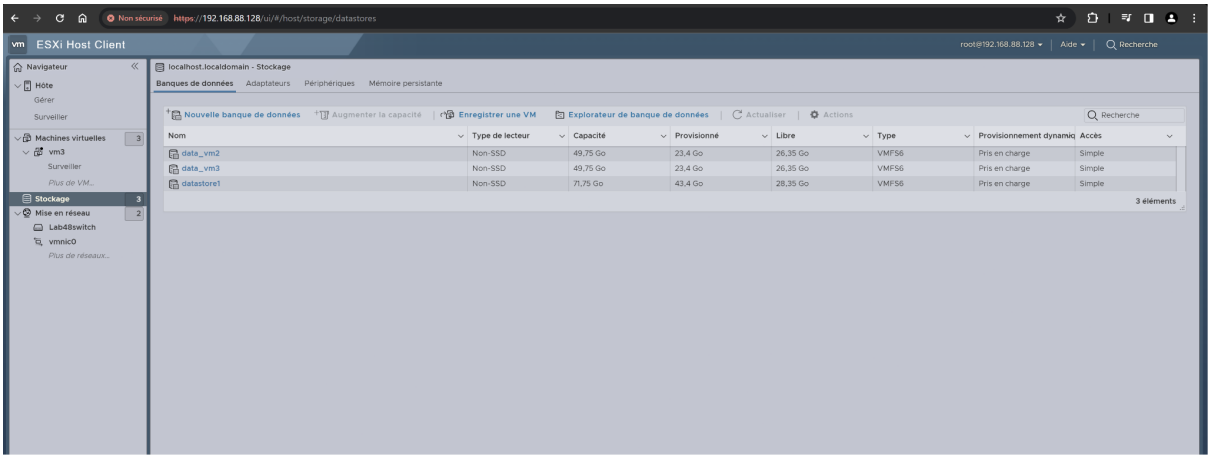
Dans un premier temps, nous commençons par installer VMware workstation ainsi qu'un hyperviseur EXSI. VMware ESXi est un hyperviseur qui s'installe facilement sur votre serveur/machine et le partitionne en plusieurs machines virtuelles. C'est cette fonctionnalité qui va nous intéresser ici.



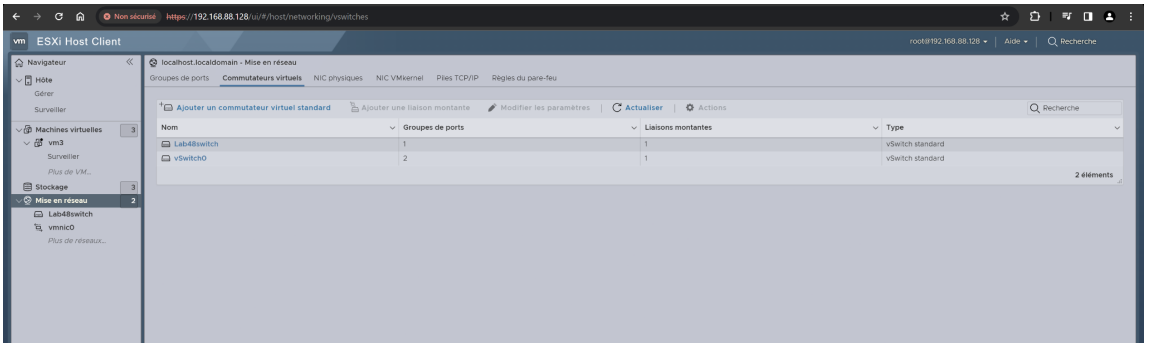
Une fois notre hyperviseur installé et bien configuré, nous pouvons ensuite accéder à son interface via l'adresse fourni par le DHCP comme visible ci-dessus. Une fois connecté, on se retrouve sur une page de configuration avec les informations générales de notre système tel que le CPU ou encore la RAM disponible.

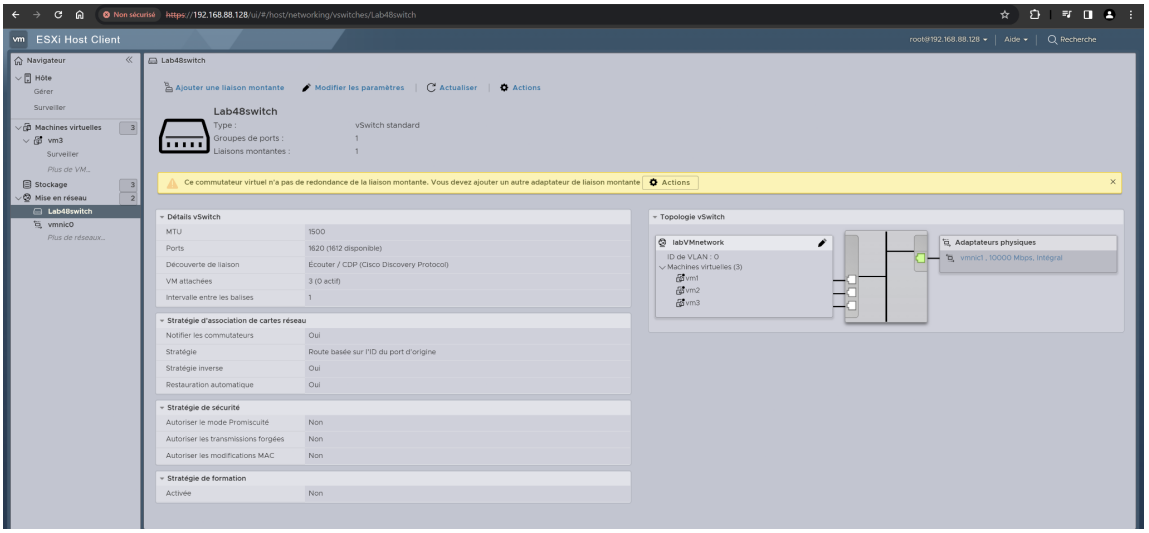
On va maintenant pouvoir rentrer dans l’étape où l'on met en place nos 3 machines virtuelles.

Dans un premier temps, il faut configurer le stockage pour chacune des différentes machines virtuelles (VMs). Pour cela il nous faut créer des nouvelles banques de données, il faut faire attention à ce qu’elle ne soit pas trop petite afin de pouvoir réaliser nos différentes installations sur ces dernières.

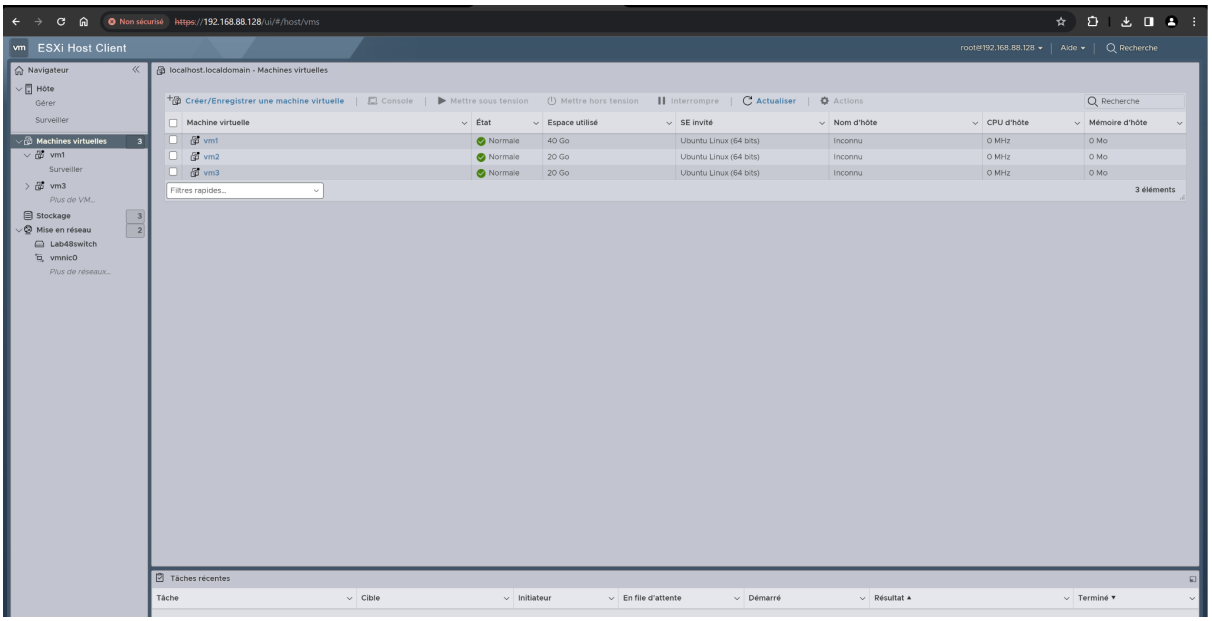


Ensuite, pour que nos VMs soient reliées au réseau, il faut que l’on ajoute un commutateur virtuel standard avec un port pour chacune de nos VMs.





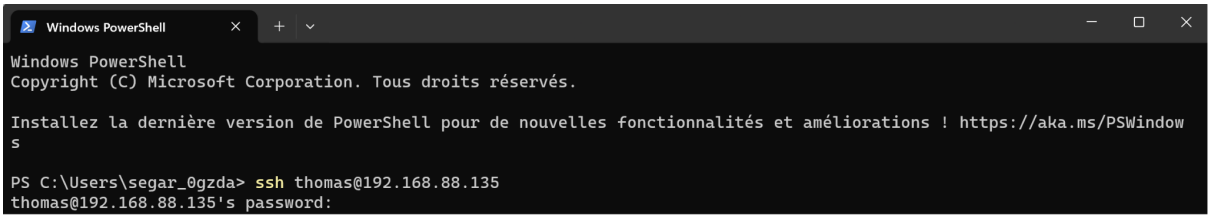
Maintenant que tout cela est configuré, on peut créer nos VMs. Nous avons décidé de leurs installer l'OS Ubuntu Server (Linux).



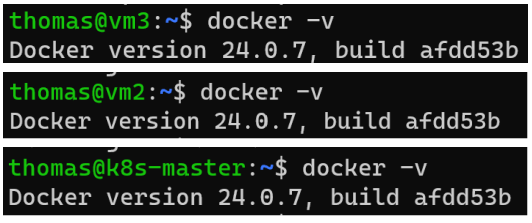
III-Installation VMs

Dans cette partie, nous allons installer/configurer les différentes dépendances et plateformes dont nos VMs ont besoin tel que Dockers.

Après avoir lancer nos VMs, on se connecte à ses dernières vis SSH pour nous faciliter la tâche (connexion vm3).



Ensuite, on installe dockers sur les 3 VMs. (on vérifie que dockers est bien installé avec la commande "docker -v").



IV-Docker compose

Pour cette partie, nous allons donc faire en sorte d'utiliser Docker Compose afin de mettre en place notre application. On commence par installer docker compose.

```
thomas@vm3:~$ docker compose version
Docker Compose version v2.21.0
```

Ensuite, on clone le dépôt github du projet afin de pouvoir utiliser ce dernier. (avec la commande "git clone [lien depot]")

```
thomas@vm3:~$ ls
esiea-ressources  snap
thomas@vm3:~$ cd esiea-ressources/
thomas@vm3:~/esiea-ressources$ ls
architecture.png  compose.yml  docker-compose.build.yml  healthchecks  README.md  result  seed-data  vote  worker
```

Afin de faire fonctionner docker compose pour ce projet, il faut créer deux fichiers, un premier compose.yml puis un fichier docker-compose.build.yml (le premier sert à décrire notre application et le deuxième va nous permettre de la lancer).

Le fichier YAML définit tous les services à déployer. Ces services s'appuient sur DockerFile ou sur une image conteneur existante.

```
[+] Running 7/5
 ✓ Network esiea-ressources_back-tier   Created                                                                    0.1
 ✓ Network esiea-ressources_front-tier  Created                                                                    0.2
 ✓ Container esiea-ressources-vote-1    Created                                                                    0.2
 ✓ Container esiea-ressources-db-1      Created                                                                    0.2
 ✓ Container esiea-ressources-redis-1   Created                                                                    0.2
 ✓ Container esiea-ressources-result-1  Created                                                                    0.2
 ✓ Container esiea-ressources-worker-1  Created                                                                    0.2
Attaching to esiea-ressources-db-1, esiea-ressources-redis-1, esiea-ressources-result-1, esiea-ressources-vote-1, esiea-
ressources-worker-1
esiea-ressources-redis-1   | 1:C 06 Jan 2024 15:42:17.390 # WARNING Memory overcommit must be enabled! Without it, a background save or replication may fail under low memory 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.
esiea-ressources-redis-1   | 1:C 06 Jan 2024 15:42:17.391 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
esiea-ressources-redis-1   | 1:C 06 Jan 2024 15:42:17.392 * Redis version=7.2.3, bits=64, commit=00000000, modified=0, pid=1, just started
esiea-ressources-redis-1   | 1:C 06 Jan 2024 15:42:17.392 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
esiea-ressources-redis-1   | 1:M 06 Jan 2024 15:42:17.393 * monotonic clock: POSIX clock_gettime
esiea-ressources-redis-1   | 1:M 06 Jan 2024 15:42:17.395 * Running mode=standalone, port=6379.
esiea-ressources-redis-1   | 1:M 06 Jan 2024 15:42:17.396 * Server initialized
esiea-ressources-redis-1   | 1:M 06 Jan 2024 15:42:17.404 * Ready to accept connections tcp
esiea-ressources-db-1      |
```

sudo docker-compose -f docker-compose.yml build //cmd pour build fichier ymal

```
thomas@vm3:~/esiea-ressources$ sudo docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS                   PORTS      NAMES
b03e94532ecf   postgres:15-alpine       "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes (unhealthy)   5432/tcp   esiea-ressources-db-1
8cb3d4a1de6e   esiea-ressources-vote    "gunicorn app:app -b…"   4 minutes ago   Up 4 minutes               80/tcp     esiea-ressources-vote-1
8623de2024d6   esiea-ressources-result  "/usr/bin/tini -- no…"   4 minutes ago   Up 4 minutes               80/tcp     esiea-ressources-result-1
943eaf43a3ef   redis                    "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes               6379/tcp   esiea-ressources-redis-1
e900cfb8bd8e   esiea-ressources-worker  "dotnet Worker.dll"      4 minutes ago   Up 4 minutes                          esiea-ressources-worker-1
```

```
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:
```

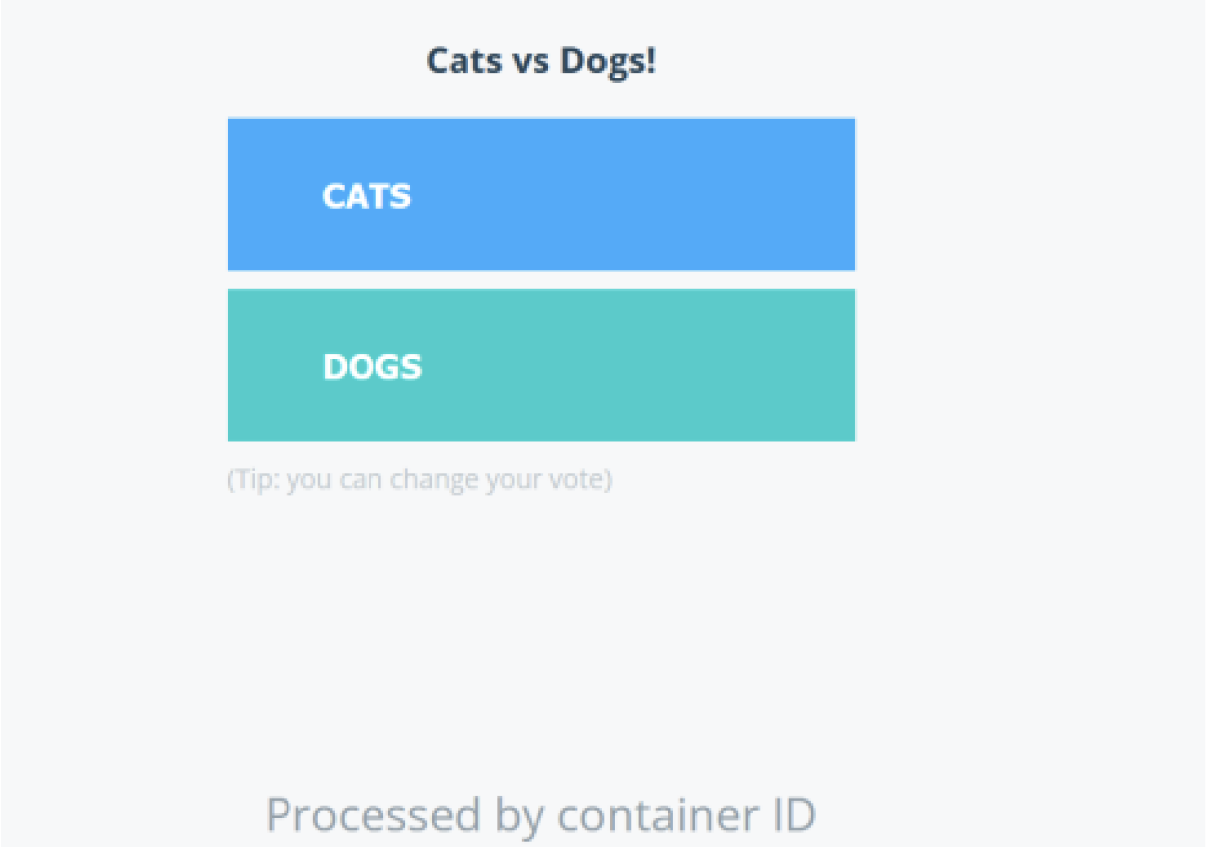
compose.yml

```yaml
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

    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

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"

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

```
volumes:
  db-data:
```

docker-compose.build.yaml



On peut voir que sur le port 5001 avec l'adresse de notre vm (ici, 192.168.88.130:5001) on retrouve l'application de vote et que sur la

V-Kubernetes

Nous allons par la suite essayer de déployer un cluster kubernetes:

Pour commencer on installe les outils kubernetes qui sont: kubeadm, kubelet et kubectl:

```
corentin@ubuntuserver1:~$ sudo apt-get update
transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sou
rces.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo systemctl enable kubelet
sudo systemctl start kubelet
```

Ensuite on doit configurer le noeud maître (Master):

On initialise le cluster et on configure kubectl pour rejoindre les travailleurs (workers):

```
corentin@ubuntuserver1:~$ sudo kubeadm init --pod-network-cidr=192.168.1.0/24
```

Malheureusement on a l'erreur suivante:

```
corentin@ubuntuserver1:~$ sudo kubeadm init --pod-network-cidr=192.168.1.0/24
I0107 12:02:21.504108  109634 version.go:256] remote version is much newer: v1.29.0
; falling back to: stable-1.28
[init] Using Kubernetes version: v1.28.5
[preflight] Running pre-flight checks
        [WARNING Swap]: swap is enabled; production deployments should disable swap
 unless testing the NodeSwap feature gate of the kubelet
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR Port-10259]: Port 10259 is in use
        [ERROR Port-10257]: Port 10257 is in use
        [ERROR Port-10250]: Port 10250 is in use
[preflight] If you know what you are doing, you can make a check non-fatal with `--
ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
```

On essaye de voir quel processus utilise ces ports pour pouvoir le kill:

```
corentin@ubuntuserver1:~$ sudo netstat -lnp | grep 1025
tcp        0      0 127.0.0.1:10256         0.0.0.0:*               LISTEN      3367/kubelite
tcp6       0      0 :::10250                :::*                    LISTEN      3367/kubelite
tcp6       0      0 :::10259                :::*                    LISTEN      3367/kubelite
tcp6       0      0 :::10257                :::*                    LISTEN      3367/kubelite
```

On doit donc kill le processus 3367:

```
corentin@ubuntuserver1:~$ sudo kill 3367
```

Le problème a bien été résolu mais un autre problème est arrivé.

En effet, lorsque kubeadm contrôle le bon fonctionnement de kubelet, il indique que kubelet n'est pas activé:

```
[kubelet-check] It seems like the kubelet isn't running or healthy.
```

Ainsi on vérifie le statut de kubelet:

```
janv. 07 18:49:42 ubuntuserver1 systemd[1]: kubelet.service: Failed with result 'exit-code'.
```

Il y a bien un problème au niveau de kubelet, mais après plusieurs tentatives de redémarrage du processus, on a toujours la même erreur.

La suite sont les grandes étapes dans le cas où nous serions allés plus loin.

On configure de la même façon les nœuds travailleurs (workers).

Une fois le cluster initialisé, on peut utiliser des fichiers YAML pour déployer l'application.

A la fin on a un cluster kubernetes fonctionnel et on retrouve l'application précédente mais avec un numéro de port différents.

VI-Conclusion

Ce projet nous a permis de découvrir et de mieux comprendre la virtualisation ainsi que la conteneurisation à l'aide de différents outils notamment Docker. Tout cela a clairement mis en évidence l'importance de ses technologies dans le monde de l'ingénieur

Docker et les outils de conteneurisation ont profondément transformé le développement technologique en offrant des solutions efficaces pour la portabilité, l'isolation des applications, la gestion des dépendances et l'évolutivité.

Cette approche a simplifié le déploiement, amélioré la collaboration entre équipes, favorisé l'adoption d'architectures basées sur des microservices, et accéléré le cycle de développement. En résumé, la conteneurisation a considérablement amélioré la flexibilité, la stabilité et l'efficacité du processus de développement d'applications.