

PROJET : DOCKER SWARM

PROJET : DOCKER SWARM.....	0
Introduction.....	1
Résumé du sujet : Déploiement d'un Cluster Docker Swarm pour la Continuité d'Activité.....	1
Docker VS Docker Swarm.....	2
1. Notre architecture.....	3
2. Configuration réseau de nos machines pour configurer le bridge.....	4
3. Création du Swarm.....	5
3.1 Installation de Docker sur le master et les workers.....	5
3.2 Configuration de la machine Master.....	5
3.3 Configuration des machines workers.....	5
3.4 Vérifications.....	6
4. Déploiement des services.....	7
4.1 NGINX.....	7
4.2 MARIADB.....	10
4.3 PHP.....	11
4.4 VSCode Server.....	12
4.5 Local Registry.....	13
4.5.1 Mise en place.....	13
4.5.2 Test du HA.....	15
5. Scalabilité des services.....	16
5.1 Commande pour le scaling.....	16
5.2 Script pour le scaling.....	16
6. Configuration de TrueNAS.....	19
6.1 Montage d'un RAID 5 sur notre NAS.....	19
6.2 Création d'un pool dans TrueNAS.....	19
6.3 Création de notre partage SMB sur TrueNAS.....	21
6.4 Montage de notre volume NFS.....	22
7. Installation d'un service de visualisation.....	23
8. Automatisation : update automatique des services.....	24

Sources :

- tuto : <https://docs.docker.com/engine/swarm/swarm-tutorial/#three-networked-host-machines>
- Spec machine : <https://docs.mirantis.com/msr/3.1/common/msr-system-reqs/swarm-system-reqs.html>
- création d'un swarm : <https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/>
- déploiement des services : <https://docs.docker.com/engine/swarm/swarm-tutorial/deploy-service/>
- pour le local registry : <https://github.com/liranfar/local-registry-swarm>
- pour scale les services dans le swarm : <https://docs.docker.com/engine/swarm/swarm-tutorial/scale-service/>

Introduction

Résumé du sujet : Déploiement d'un Cluster Docker Swarm pour la Continuité d'Activité

Introduction : Docker Swarm est une technologie d'orchestration de conteneurs qui permet de gérer facilement des applications conteneurisées à grande échelle. Il regroupe plusieurs hôtes Docker en un cluster, avec un nœud maître pour la gestion et des nœuds ouvriers pour l'exécution des conteneurs. Grâce à sa simplicité, son intégration avec Docker et sa mise à l'échelle facile, Docker Swarm est une solution efficace pour déployer et gérer des applications distribuées.

Objectif : Vous êtes sollicité pour construire un cluster Docker Swarm robuste, spécialement conçu pour assurer la Continuité d'Activité (PCA) et la Reprise d'Activité (PRA) dans les environnements les plus hostiles. Ce cluster doit être basé sur une architecture Debian, garantissant une infrastructure stable et fiable même dans des conditions extrêmes.

Architecture du Cluster :

1. **Maître de la Survie (Master) :**
 - Une VM Debian jouant le rôle de nœud maître, orchestrant les opérations de reprise d'activité.
2. **Esclaves de Secours (Slaves) :**
 - Plusieurs VM Debian servant de nœuds ouvriers, prêtes à exécuter les conteneurs et à restaurer les services essentiels en cas de catastrophe.
3. **Station de Résilience (NFS) :**
 - Une VM dédiée au stockage sécurisé des volumes Docker, assurant la disponibilité immédiate des données et la sauvegarde de la config du cluster.

Déploiement des Conteneurs Résilients :

1. **Forteresse des Données (Repository Local) :**
 - Conteneur pour le stockage des artefacts logiciels nécessaires.
2. **Bastion de la Base de Données (MariaDB) :**
 - Conteneur pour la base de données MariaDB, essentiel pour la restauration des opérations.
3. **Ravitaillement Rapide (PHP) :**
 - Conteneur PHP pour fournir la puissance de calcul nécessaire aux applications.
4. **Sentinelle Web (Nginx) :**
 - Conteneur Nginx pour diriger le trafic vers les services restaurés.
5. **QG de Commandement (VSCode Server) :**
 - Conteneur VSCode Server pour permettre la collaboration et l'orchestration des opérations de sauvetage en temps réel.

Mission : Votre mission consiste à établir un cluster Docker Swarm capable de garantir la continuité des opérations numériques, même en cas de catastrophes majeures. Vous devez assurer la résilience technologique et la disponibilité immédiate des services critiques.

Ensemble, nous protégerons nos opérations contre les menaces numériques, garantissant ainsi la continuité et la reprise rapide des activités en toute circonstance.

Docker VS Docker Swarm

Qu'est-ce que Docker Swarm ?

Docker Swarm est un outil de gestion de clusters de conteneurs Docker. Il permet de gérer un groupe de machines (appelées nœuds) en les unifiant pour les faire fonctionner comme un seul et même système. Avec Docker Swarm, vous pouvez orchestrer des conteneurs Docker sur plusieurs machines, en simplifiant la gestion, le déploiement et la scalabilité des applications conteneurisées.

Différences entre Docker et Docker Swarm

Docker

- **Fonctionnalité** : Docker est une plateforme de conteneurisation qui permet de créer, déployer et exécuter des applications dans des conteneurs. Il fournit les outils nécessaires pour emballer une application avec toutes ses dépendances afin qu'elle puisse fonctionner de manière cohérente dans différents environnements.
- **Utilisation** : Docker est utilisé pour créer des images de conteneurs, gérer des conteneurs sur une seule machine, et tester des applications dans un environnement isolé.
- **Composants** : Les principaux composants de Docker sont Docker Engine (le moteur de conteneurisation), Docker CLI (interface en ligne de commande) et Docker Compose (pour définir et gérer des applications multi-conteneurs).

Docker Swarm

- **Fonctionnalité** : Docker Swarm ajoute une couche d'orchestration à Docker. Il permet de gérer un cluster de nœuds Docker et de déployer des services conteneurisés sur ce cluster de manière automatisée et scalable.
- **Utilisation** : Docker Swarm est utilisé pour orchestrer des conteneurs Docker sur plusieurs machines, ce qui permet de garantir une haute disponibilité et une scalabilité horizontale des applications.
- **Composants** : Les principaux composants de Docker Swarm incluent les nœuds managers (qui gèrent le cluster) et les nœuds workers (qui exécutent les conteneurs). Il utilise également des concepts comme les services et les stacks pour déployer des applications.

Comparaison en détails

Fonctionnalité	Docker	Docker Swarm
Déploiement	Sur une seule machine	Sur un cluster de plusieurs machines
Scalabilité	Limitée à la machine hôte	Scalabilité horizontale sur plusieurs nœuds
Orchestration	Nécessite des outils externes comme Kubernetes ou Docker Compose	Intégrée nativement avec Docker Swarm
Tolérance aux pannes	Non gérée	Gérée par la redondance des services sur plusieurs nœuds
Gestion des réseaux	Basique (réseau de pont par défaut)	Avancée (overlay network pour les clusters)
Mise à jour des services	Manuelle	Automatisée avec des stratégies de déploiement

En résumé, Docker est idéal pour le développement et le test d'applications conteneurisées sur une seule machine. Docker Swarm, quant à lui, est conçu pour gérer des applications en production sur un cluster de machines, offrant des fonctionnalités avancées d'orchestration et de gestion des conteneurs.

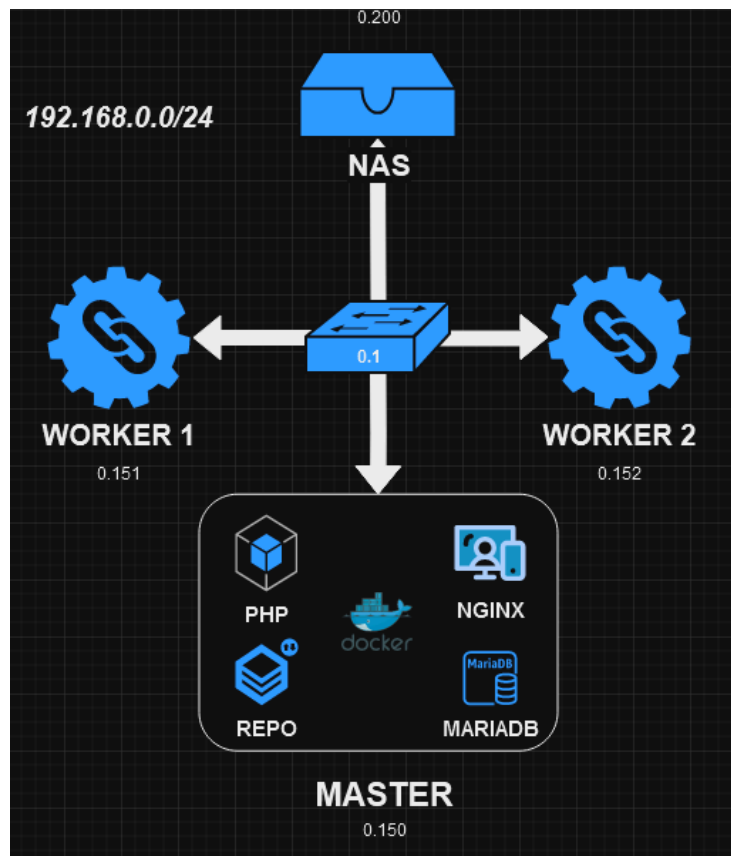
1. Notre architecture



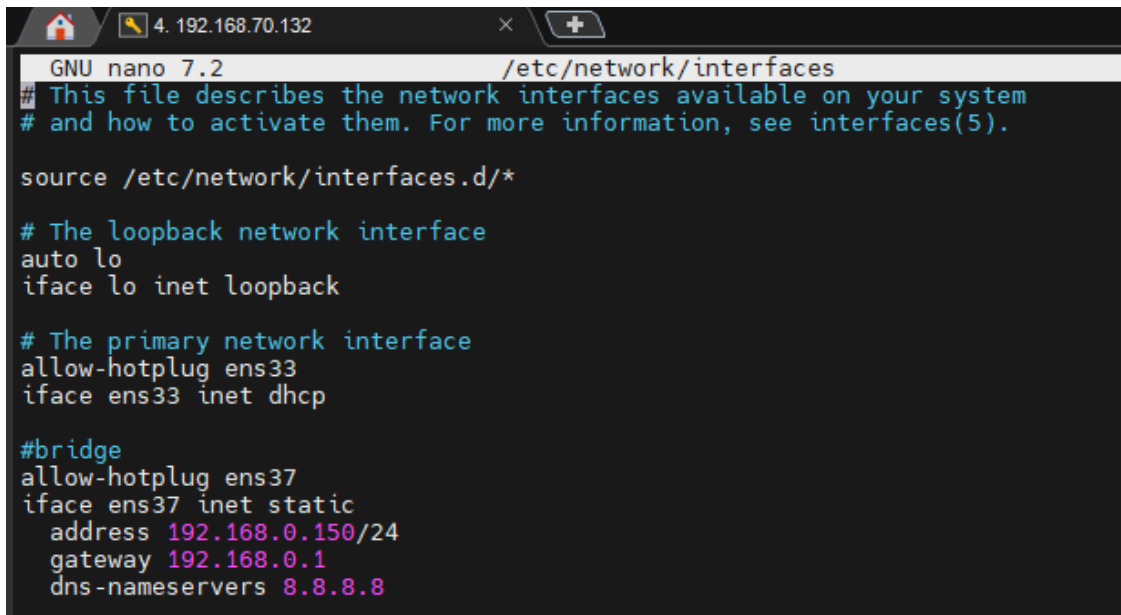
Nous avons choisi d'utiliser un switch pour créer notre réseau local, et pouvoir avoir accès à nos VM. Nous allons partager les ressources de nos PCs pour faire tourner nos différentes machines virtuelles.

Configuration IP de nos machines sur le réseau 192.168.0.0/24:

- VM Master = 192.168.0.150/24
- VM Worker1 = 192.168.0.151/24
- Vm Worker2 = 192.168.0.152/24
- VM NAS = 192.168.0.200/24
- Switch = 192.168.0.1/24 (admin / laplateforme)



2. Configuration réseau de nos machines pour configurer le bridge



```
GNU nano 7.2 /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
iface ens33 inet dhcp

#bridge
allow-hotplug ens37
iface ens37 inet static
    address 192.168.0.150/24
    gateway 192.168.0.1
    dns-nameservers 8.8.8.8
```

Ici on configure notre interface réseau qui est configurée en bridge.

On doit modifier le fichier **/etc/network/interfaces**.

puis on ajoute la configuration ci dessus, avec les IP adéquates

On lui spécifier notre réseau et aussi notre masque de sous réseau :

- réseau 192.168.0.150
- masque de sous réseau : 255.255.255.0
- gateway : 192.168.0.1
- dns-nameservers : 8.8.8.8 (Indispensable pour avoir accès à internet).

Une fois notre fichier configuré, on relance notre service networking :

sudo service networking restart

3. Création du Swarm

3.1 Installation de Docker sur le master et les workers

Sur notre master et nos workers, on installe Docker via un script :

```
bash
#!/bin/bash

# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \e
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get updat
```

Puis on utilise cette commande pour installer la dernière version de Docker :

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

3.2 Configuration de la machine Master

On met en place le Swarm avec cette commande :

```
sudo docker init --advertise-addr 192.168.0.150
```

```
master@master:~$ sudo docker swarm init --advertise-addr 192.168.0.150
Swarm initialized: current node (1sc6isz3tfd0p998tl7kk17zq) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3qzeolc2jybgkri0sk7skqxf8ldeh0r4gexeeby37qwgcptvy-27rbgx287strm3
kxtob9n9ym1 192.168.0.150:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Cela nous retourne la commande à utiliser sur un worker pour l'ajouter au swarm.

3.3 Configuration des machines workers

Depuis une machine, on utilise cette commande pour l'ajouter comme Worker au Swarm :

```
sudo docker swarm join --token  
SWMTKN-1-3qzeolc2jybgkri0sk7skqxf8ldeh0r4gexeeby37qwgcvty-27rbgx287strm3kxtob9n9ym1  
192.168.0.150:2377
```

```
debian@debian:~$ sudo docker swarm join --token SWMTKN-1-3qzeolc2jybgkri0sk  
7skqxf8ldeh0r4gexeeby37qwgcvty-27rbgx287strm3kxtob9n9ym1 192.168.0.150:23  
77
```

On reproduit cette étape sur une autre machine afin d'avoir 2 workers.

3.4 Vérifications

On va lister tous les nœuds du cluster Swarm :

docker node ls

```
master@master:~$ sudo docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
1sc6isz3tfd0p998tl7kkl7zq *	master	Ready	Active	Leader	26.1.4
kgc6np1njzc6q3ut9tzfytz02	worker1	Ready	Active		26.1.4
oan361oq5rt4308hneo7suxbn	worker2	Ready	Active		26.1.4

4. Déploiement des services

4.1 NGINX

Pour déployer le service nginx, nous allons construire une commande pour qu'elle prenne en compte nos 3 nœuds, ainsi que le montage distant pour le volume. Le master va donc rentrer cette commande :

```
sudo docker service create --name my_nginx_service --replicas 3 --mount  
type=bind,src=/mnt/share/nginx,dst=/usr/share/nginx/html --publish published=80,target=80 nginx
```

--replicas 3 : spécifie que le service doit avoir 3 répliques. Docker Swarm va donc maintenir les 3 instances de ce service en cours d'exécution en tout temps pour assurer le HA et la répartition de la charge.

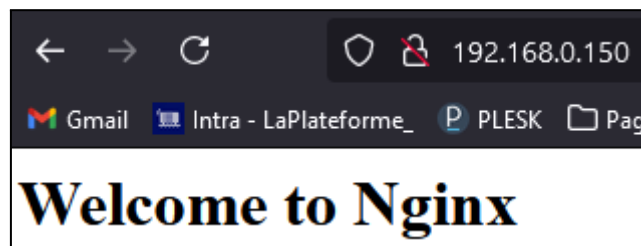
--mount type=bind,src=/mnt/share/nginx,dst=/usr/share/nginx/html : va signifier que les fichiers web seront stockés dans ce répertoire, en l'occurrence notre serveur de stockage distant

--publish published=80,target=80 : publie le port 80 de l'hôte sur le port 80 du conteneur

On définit les droits pour l'utilisateur nginx:

```
sudo chmod -R 755 /mnt/share/nginx  
sudo chown -R www-data:www-data /mnt/share/nginx
```

Nous allons nous connecter en utilisant l'IP du master, bien que les IP des workers 1 et 2 puissent également être utilisées pour vérifier le bon fonctionnement du service :



En effectuant la commande `sudo docker service ls`, on voit notre conteneur actif:

```
master@master:~$ sudo docker service ls
ID                NAME                MODE                REPLICAS    IMAGE                PORTS
oh1xd7vm8edv     my_nginx_service    replicated          3/3         nginx:latest        *:80->80/tcp
master@master:~$
```

On affiche l'état des tâches associés à notre service avec la commande:

```
sudo docker service ps my_nginx_service
```



```
master@master:~$ sudo docker service ps my_nginx_service
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ERROR		PORTS			
mraht0rg8o92	my_nginx_service.1	nginx:latest	master	Running	Running 12 minutes ago
sknzicnfjp4d	_ my_nginx_service.1	nginx:latest	master	Shutdown	Shutdown 12 minutes ago
f14y0n7wy3t3	_ my_nginx_service.1	nginx:latest	worker2	Shutdown	Shutdown 13 minutes ago
oi2tsjal69xg	_ my_nginx_service.1	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
xgw2vf4w68hl	_ my_nginx_service.1	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
"invalid mount config for type..."					
ke00smywhckx	my_nginx_service.2	nginx:latest	worker2	Running	Running 12 minutes ago
27767j18eie6	_ my_nginx_service.2	nginx:latest	worker1	Shutdown	Shutdown 12 minutes ago
vs1i3g0kgtzb	_ my_nginx_service.2	nginx:latest	worker2	Shutdown	Shutdown 13 minutes ago
lsdp93ib3d6l	_ my_nginx_service.2	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
"invalid mount config for type..."					
znn8chxelu7o	_ my_nginx_service.2	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
"invalid mount config for type..."					
wwy8lmaghivg	my_nginx_service.3	nginx:latest	worker1	Running	Running 12 minutes ago
l2ju3qtxovmm	_ my_nginx_service.3	nginx:latest	worker1	Shutdown	Shutdown 12 minutes ago
t1igjfsdowky	_ my_nginx_service.3	nginx:latest	worker2	Shutdown	Shutdown 13 minutes ago
8hrquavr0eqs	_ my_nginx_service.3	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
"invalid mount config for type..."					
u4b8mm8r6khf	_ my_nginx_service.3	nginx:latest	worker1	Shutdown	Rejected 2 hours ago
"invalid mount config for type..."					

On voit bien ici (outre les services shutdown dûs à nos tests) que les 3 nœuds sont en train d'effectuer les tâches relatives à votre service nginx.

Nous pouvons effectuer un test pour vérifier le fonctionnement du HA. Nous allons arrêter en premier lieu le noeud master, voir si les workers ont pris le relais, puis réactiver le master et désactiver un des deux workers.

Nous désactivons donc le master avec la commande `sudo docker node update --availability drain master`, puis afficher les l'état des tâches associés à notre service (en filtrant les services shutdown qui s'affichent lors de notre commande sans filtre) **sudo docker service ps --filter "desired-state=running" my_nginx_service** :

```
master@master:~$ sudo docker node update --availability drain master
```

```
master
```

```
master@master:~$ sudo docker service ps --filter "desired-state=running" my_nginx_service
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ERROR		PORTS			
1l2v493kpia2	my_nginx_service.1	nginx:latest	worker1	Running	Running about a minute ago
ke00smywhckx	my_nginx_service.2	nginx:latest	worker2	Running	Running 19 minutes ago
wwy8lmaghivg	my_nginx_service.3	nginx:latest	worker1	Running	Running 18 minutes ago

On va repasser le master en actif avec la commande **sudo docker node update --availability active master**, puis je vais passer le worker1 en inactif et afficher l'état des tâches:

```
master@master:~$ sudo docker node update --availability drain worker1
```

```
worker1
```

```
master@master:~$ sudo docker service ps --filter "desired-state=running" my_nginx_service
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ERROR		PORTS			
1l5y66yfnc3j	my_nginx_service.1	nginx:latest	master	Running	Running 11 seconds ago
ke00smywhckx	my_nginx_service.2	nginx:latest	worker2	Running	Running 22 minutes ago
amt49hmlfsf8	my_nginx_service.3	nginx:latest	master	Running	Running 12 seconds ago

On peut voir que c'est maintenant le master qui a pris le relais sur les tâches du worker1.

Je vais pour finir réactiver mon worker1, puis mettre à jour les tâches du service pour que nos 3 noeuds se partagent de manière homogène nginx avec la commande **sudo docker service update --force my_nginx_service** :

```
master@master:~$ sudo docker node update --availability active worker1
worker1
master@master:~$ sudo docker service update --force my_nginx_service
my_nginx_service
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service my_nginx_service converged
master@master:~$ sudo docker service ps --filter "desired-state=running" my_nginx_service
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ROR	PORTS				
96bb1nbgbbjj	my_nginx_service.1	nginx:latest	worker1	Running	Running 22 seconds ago
qmpxkgu3vdfr	my_nginx_service.2	nginx:latest	worker2	Running	Running 17 seconds ago
g5id6jjcszhs	my_nginx_service.3	nginx:latest	master	Running	Running 12 seconds ago

4.2 MARIADB

De la même manière que pour déployer le service nginx, nous allons construire une commande pour déployer le service MariaDB pour qu'elle prenne en compte nos 3 nœuds, ainsi que le montage distant pour le volume. On saisi cette commande depuis notre master :

```
master@master:~$ sudo docker service create --name my_mariadb_service --replicas 3 --mount type=bind,src=/mnt/share/mariadb,dst=/mnt/share/mariadb/mysql --env MYSQL_ROOT_PASSWORD=123 --publish published=3306,target=3306 mariadb --log-bin=off
image mariadb:latest could not be accessed on a registry to record
its digest. Each node will access mariadb:latest independently,
possibly leading to different nodes running different
versions of the image.

cxrkcs2fjae2cb6lbi072r0ap
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service cxrkcs2fjae2cb6lbi072r0ap converged
```

On vérifie que notre conteneur est bien actif avec la commande **sudo docker service ls** :

```
master@master:~$ sudo docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
cxrkcs2fjae2	my_mariadb_service	replicated	3/3	mariadb:latest	*:3306->3306/tcp
oh1xd7vm8edv	my_nginx_service	replicated	3/3	nginx:latest	*:80->80/tcp

On affiche l'état des tâches associés à notre service avec **sudo docker service ps my_mariadb_service** :

```
master@master:~$ sudo docker service ps my_mariadb_service
```

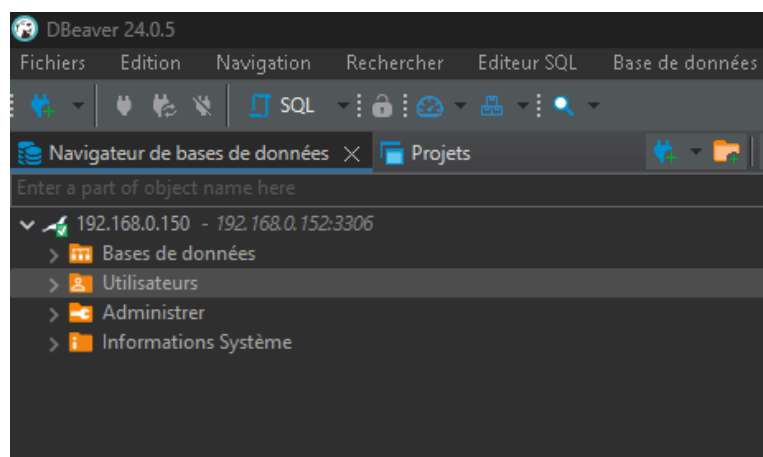
ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
s1n4u014a2cf	my_mariadb_service.1	mariadb:latest	worker1	Running	Running 2 minutes ago
0kituz5i6wbt	my_mariadb_service.2	mariadb:latest	worker2	Running	Running 2 minutes ago
733ytl4cf9u	my_mariadb_service.3	mariadb:latest	master	Running	Running about a minute ago

On voit bien ici que les 3 nœuds sont en train d'effectuer les tâches relatives à votre service MariaDB.

On définit les droits pour l'utilisateur mysql:

```
sudo chown -R 999:999 /mnt/share/mariadb # (999 est l'UID par défaut de l'utilisateur mysql dans le conteneur officiel mariadb)
sudo chmod -R 755 /mnt/share/mariadb
```

On se connecte à notre MariaDB à l'aide d'une interface graphique (en l'occurrence ici DBeaver) :



4.3 PHP

On déploie le service PHP sur nos 3 noeuds, ainsi que sur le montage distant pour le volume avec cette commande :

```
master@master:~$ sudo docker service create \
  --name my_php_service \
  --replicas 3 \
  --mount type=bind,src=/mnt/share/php,dst=/mnt/share/php \
  --publish published=81,target=80 \
  php:apache
image php:apache could not be accessed on a registry to record
its digest. Each node will access php:apache independently,
possibly leading to different nodes running different
versions of the image.

i3vgtmktcbus48ddd3yq33zx
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service i3vgtmktcbus48ddd3yq33zx converged
```

Vérification de l'état de notre conteneur avec la commande **sudo docker service ls** :

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
cxrkcs2fjae2	my_mariadb_service	replicated	3/3	mariadb:latest	*:3306->3306/tcp
ph1xd7vm8edv	my_nginx_service	replicated	3/3	nginx:latest	*:80->80/tcp
i3vgtmktcbus	my_php_service	replicated	3/3	php:apache	*:81->80/tcp

Affichons l'état des tâches associés à notre service avec **sudo docker service ps my_php_service** :

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
odbqmtof1xf4	my_php_service.1	php:apache	worker1	Running	Running 57 seconds ago
sfv1g5k77jnh	my_php_service.2	php:apache	worker2	Running	Running 52 seconds ago
p782tca0ch2q	my_php_service.3	php:apache	master	Running	Preparing 12 seconds ago

On voit bien que les 3 nœuds sont en train d'effectuer les tâches relatives au service PHP.

On définit les droits pour l'utilisateur php :

```
sudo chown -R 1000:1000 /mnt/share/php
```

```
sudo chmod -R 755 /mnt/share/php
```

On se connecte par le navigateur pour vérifier le bon fonctionnement du service :

PHP Version 8.3.8

System	Linux 7e2bbb299f4a 6.1.0-21-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-03) x86_64
Build Date	Jun 13 2024 02:02:16
Build System	Linux - Docker
Build Provider	https://github.com/docker-library/php
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-mysqld' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini

4.4 VSCode Server

Je crée le répertoire de montage pour vscode:

```
sudo mkdir -p /mnt/share/vscode
```

On définit les droits pour l'utilisateur vscode:

```
sudo chown -R 1000:1000 /mnt/share/vscode
```

```
sudo chmod -R 755 /mnt/share/vscode
```

On déploie le service VS Code Server sur nos 3 nœuds, ainsi que sur le montage distant pour le volume :

```
master@master:~$ sudo docker service create \
  --name my_vscode_service \
  --replicas 3 \
  --mount type=bind,src=/mnt/share/vscode,dst=/mnt/share/vscode/project \
  --publish published=8443,target=8443 \
  codercom/code-server:latest \
  --auth none \
  --bind-addr 0.0.0.0:8443
image codercom/code-server:latest could not be accessed on a registry to record
its digest. Each node will access codercom/code-server:latest independently,
possibly leading to different nodes running different
versions of the image.

xswonh241zj0b1hmcnec65rl3
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service xswonh241zj0b1hmcnec65rl3 converged
```

Vérifions l'état de notre conteneur avec **sudo docker service ls** :

```
master@master:~$ sudo docker service ls
```

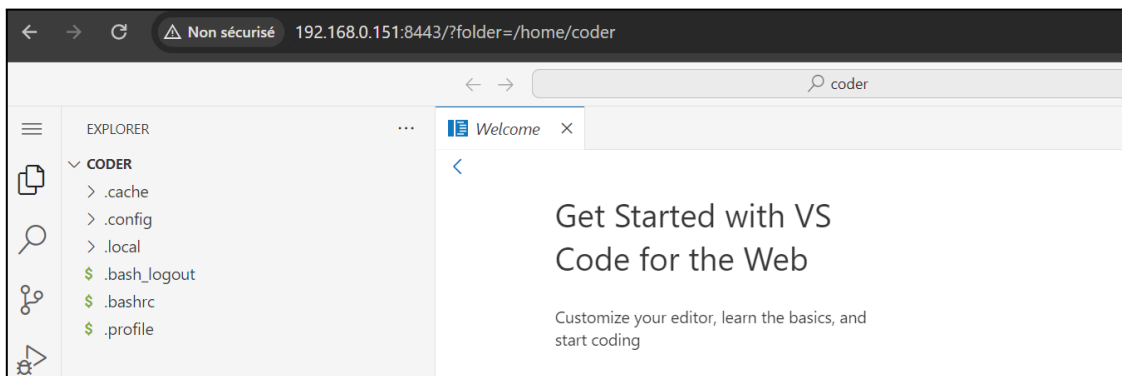
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
cxrkcs2fjae2	my_mariadb_service	replicated	3/3	mariadb:latest	*:3306->3306/tcp
cp					
oh1xd7vm8edv	my_nginx_service	replicated	3/3	nginx:latest	*:80->80/tcp
e30g39kw22dx	my_php_service	replicated	3/3	php:apache	*:81->80/tcp
xswonh241zj0	my_vscode_service	replicated	3/3	codercom/code-server:latest	*:8443->8443/tcp
cp					

Vérifions l'état des tâches du service avec **sudo docker service ps my_vscode_service** :

```
master@master:~$ sudo docker service ps --filter "desired-state=running" my_vscode_service
```

ID	NAME	PORTS	IMAGE	NODE	DESIRED STATE	CURRENT STATE
0secge0pv80n	my_vscode_service.1		codercom/code-server:latest	worker2	Running	Running 37 seconds ago
m7qdvj5p2gpb	my_vscode_service.2		codercom/code-server:latest	worker1	Running	Running 41 seconds ago
1a21mzn9b72w	my_vscode_service.3		codercom/code-server:latest	master	Running	Running 46 seconds ago

On vérifie le bon fonctionnement en se connectant depuis le navigateur à <http://192.168.0.151:8443/>



4.5 Local Registry

4.5.1 Mise en place

Déploiement du service Local Registry sur 1 seul noeud ainsi que sur le montage distant pour le volume :

`sudo docker service create --name myregistry --publish 5001:5000 --replicas 1 --mount type=bind,src=/mnt/share/myregistry,dst=/var/lib/registry registry:latest`

```
master@master:~$ sudo docker service create --name myregistry --publish 5001:5000 --replicas 1 --mount type=bind,src=/mnt/share/myregistry,dst=/var/lib/registry registry:latest
wa8j38t21b3004j8ldurmtqyf
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
```

Vérification de l'état de notre conteneur avec :

`sudo docker service ls`

```
master@master:~$ sudo docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
cxrkcs2fjae2     my_mariadb_service  replicated          3/3                 mariadb:latest      *:3306->3306/tcp
oh1xd7vm8edv     my_nginx_service    replicated          4/4                 nginx:latest         *:80->80/tcp
e30g39kw22dx     my_php_service      replicated          3/3                 php:apache           *:81->80/tcp
xswonh241zj0     my_vscode_service   replicated          3/3                 codercom/code-server:latest *:8443->8443/tcp
wa8j38t21b30     myregistry          replicated          1/1                 registry:latest      *:5001->5000/tcp
09h2o5hwedxk     registry            replicated          1/1                 registry:2           *:5000->5000/tcp
ynsmfp2dog16     viz                 replicated          1/1                 dockersamples/visualizer:latest *:8080->8080/tcp
```

Vérification de l'état des tâches du service avec:

`sudo docker service ps registry`

```
master@master:~$ sudo docker service ps myregistry
ID                NAME                IMAGE                NODE    DESIRED STATE    CURRENT STATE            ERROR    PORTS
r2uchs13ja67     myregistry.1        registry:latest      master  Running          Running 5 minutes ago
734ossxpugw6     \ myregistry.1      registry:latest      worker1 Shutdown          Shutdown 5 minutes ago
```

Nous allons choisir une image a push sur notre registry. Dans cet exemple, nous avons choisis de push l'image **"hello-world"**:

`sudo docker images`

```
master@master:~$ sudo docker images
REPOSITORY          TAG                IMAGE ID            CREATED             SIZE
codercom/code-server  latest            8e87b53420f7       6 days ago         683MB
nginx                <none>            e0c9858e10ed       7 days ago         188MB
alpine               latest            a606584aa9aa       7 days ago         7.8MB
php                  apache            405449117b3b       2 weeks ago        507MB
mariadb              latest            4486d64c9c3b       2 weeks ago        406MB
redis                alpine            38a44d796822       5 weeks ago        40.7MB
registry             <none>            6a3edb1d5eb6       8 months ago       25.4MB
hello-world          latest            d2c94e258dcb       14 months ago      13.3kB
dockersamples/visualizer <none>            43ce62428b8c       3 years ago        185MB
```


On tag dans un premier temps notre image **hello-world** avec la commande :

sudo docker tag hello-world:latest 192.168.0.150:5001/hello-world

```
master@master:~$ sudo docker tag hello-world:latest 192.168.0.150:5001/hello-world
master@master:~$
```

Une fois notre image tag, on peut maintenant la push vers notre repository avec la commande :

sudo docker push 192.168.0.150:5001/hello-world

```
master@master:~$ sudo docker push 192.168.0.150:5001/hello-world
Using default tag: latest
The push refers to repository [192.168.0.150:5001/hello-world]
ac2880ec8bb: Pushed
latest: digest: sha256:d37ada95d47ad12224c205a938129df7a3e52345828b4fa27b03a98825d1e2e7 size: 524
master@master:~$
```

Je me connecte sur mon **worker1** pour faire le test, et tente de récupérer une image depuis mon repository:

sudo docker pull 192.168.0.150:5001/hello-world

```
debian@worker1:~$ sudo docker pull 192.168.0.150:5001/hello-world
Using default tag: latest
latest: Pulling from hello-world
c1ec31eb5944: Pull complete
Digest: sha256:d37ada95d47ad12224c205a938129df7a3e52345828b4fa27b03a98825d1e2e7
Successfully downloaded newer image for 192.168.0.150:5001/hello-world:latest
192.168.0.150:5001/hello-world:latest
debian@worker1:~$
```

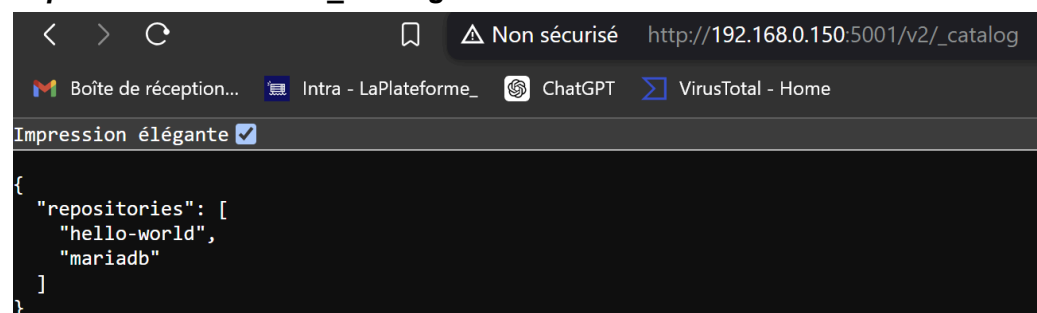
On peut maintenant vérifier les images présentes en local sur worker1:

sudo docker images

```
debian@worker1:~$ sudo docker images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
codercom/code-server latest       8e87b53420f7  6 days ago  683MB
nginx                <none>       e0c9858e10ed  7 days ago  188MB
192.168.0.150:5000/php latest       405449117b3b  2 weeks ago  507MB
php                 apache       405449117b3b  2 weeks ago  507MB
mariadb             latest       4486d64c9c3b  2 weeks ago  406MB
redis               <none>       38a44d796822  5 weeks ago  40.7MB
registry            <none>       6a3edb1d5eb6  8 months ago  25.4MB
192.168.0.150:5001/hello-world latest       d2c94e258dcb  14 months ago  13.3kB
dockersamples/visualizer <none>       43ce62428b8c  3 years ago  185MB
debian@worker1:~$
```

Voici le résultat en passant par le navigateur:

http://192.168.0.150/v2/_catalog



The screenshot shows a web browser window with the address bar displaying `http://192.168.0.150:5001/v2/_catalog`. The browser's address bar also shows a warning icon and the text "Non sécurisé". The browser's tabs include "Boîte de réception...", "Intra - LaPlateforme_", "ChatGPT", and "VirusTotal - Home". The main content area of the browser displays a JSON response from the Docker registry catalog, which lists the repositories "hello-world" and "mariadb".

```
{
  "repositories": [
    "hello-world",
    "mariadb"
  ]
}
```

4.5.2 Test du HA

Nous allons maintenant tester le HA sur notre service, le registry ne pouvant pas fonctionner en même temps sur plusieurs nœuds, nous avons choisis de le placer sur un seul nœud, qui change d'hôte à chaque fois que ce dernier tombe.

On peut vérifier sur quel nœud le service fonctionne actuellement :

sudo docker service ps myregistry

```
master@master:~$ sudo docker service ps myregistry
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
r2uchsl3ja67	myregistry.1	registry:latest	master	Running	Running 23 minutes ago		
734ossxpugw6	_ myregistry.1	registry:latest	worker1	Shutdown	Shutdown 23 minutes ago		

```
master@master:~$
```

On voit ici que le service fonctionne sur le nœud **master**. Que se passe-t-il si je stoppe le nœud ?

sudo docker node update --availability drain master

```
master@master:~$ sudo docker node update --availability drain master
master
```

On peut maintenant constater que le service run sur le **worker2**:

sudo docker service ps myregistry

```
master@master:~$ sudo docker service ps myregistry
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
jdllqpbm1v02	myregistry.1	registry:latest	worker2	Running	Running 40 seconds ago		
r2uchsl3ja67	_ myregistry.1	registry:latest	master	Shutdown	Shutdown 43 seconds ago		
734ossxpugw6	_ myregistry.1	registry:latest	worker1	Shutdown	Shutdown 26 minutes ago		

```
master@master:~$
```

Et que notre registry est toujours up !

< > ↺ 🔖 ⚠ Non sécurisé http://192.168.0.150:5001/v2/_catalog

Impression élégante ☐

```
{"repositories":["hello-world","mariadb"]}
```


5. Scalabilité des services

5.1 Commande pour le scaling

La scalabilité des services est déjà en place avec le “replicas 3”.

On peut modifier le nombre de tâche pour un service avec la commande:

`sudo docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>`

```
master@master:~$ sudo docker service scale my_nginx_service=4
my_nginx_service scaled to 4
overall progress: 4 out of 4 tasks
1/4: running [=====>]
2/4: running [=====>]
3/4: running [=====>]
4/4: running [=====>]
verify: Service converged
```

On peut voir que les tâches sont passées de 3 à 4, dont 2 tournent actuellement sur worker2:

```
master@master:~$ sudo docker service ps --filter "desired-state=running" my_nginx_service


| ID           | NAME               | IMAGE        | NODE    | DESIRED STATE | CURRENT STATE          | ERROR | PORTS |
|--------------|--------------------|--------------|---------|---------------|------------------------|-------|-------|
| t1ypuhcwfohd | my_nginx_service.1 | nginx:latest | worker2 | Running       | Running 38 seconds ago |       |       |
| qdm19udgfhdn | my_nginx_service.2 | nginx:latest | worker2 | Running       | Running 33 seconds ago |       |       |
| q9lqk6s7yao9 | my_nginx_service.3 | nginx:latest | master  | Running       | Running 28 seconds ago |       |       |
| go8h3pncckyu | my_nginx_service.4 | nginx:latest | worker1 | Running       | Running 42 seconds ago |       |       |


master@master:~$
```

5.2 Script pour le scaling

```
#!/bin/bash
continuer="True"
continuer1="True"

nombre() {
    [[ $1 =~ ^[0-9]+$ ]]
}

while [ $continuer = "True" ];
do
    clear
    echo ""
    echo "-----Choix de votre service-----"
    echo ""
    echo "Choisissez un service à scale :"
    echo ""
    echo "1 = Nginx"
    echo "2 = Mariadb"
    echo "3 = VsCode Server"
    echo "4 = PHP"
    echo "5 = Quitter"
    echo ""
    echo "Entrez le chiffre de votre choix :"
    read service

    if [[ $service = "1" ]]; then
        continuer="False"
        service="my_nginx_service"
        clear
        while [ $continuer1 = "True" ];do
            echo "De combien voulez-vous scale le service :"
            echo "Pour quitter entrer : q"
            echo "Pour revenir en arrière entrer : r"
            read scale
            if nombre "$scale"; then
                continuer1="False"
                sudo docker service scale $service=$scale
            elif [[ $scale = "q" ]]; then
                exit
            elif [[ $scale = "r" ]]; then
                continuer1="True"
            else
                clear
                echo "-----Entrée invalide, veuillez entrer uniquement des chiffres.-----"
            fi
        done
    fi
done
```

```

        fi
    done
elif [[ $service = "2" ]]; then
    continuer="False"
    service="my_mariadb_service"
    clear
    while [ $continuer1 = "True" ];do
        echo "De combien voulez-vous scale le service :"
        echo "Pour quitter entrer : q"
        echo "Pour revenir en arriere entrer : r"
        read scale
        if nombre "$scale"; then
            continuer1="False"
            sudo docker service scale $service=$scale
        elif [[ $scale = "q" ]]; then
            exit
        elif [[ $scale = "r" ]]; then
            continuer="True"
        else
            clear
            echo "-----Entrée invalide, veuillez entrer uniquement des
chiffres.----->"
        fi
    done
elif [[ $service = "3" ]]; then
    continuer="False"
    service="my_vscode_service"
    clear
    while [ $continuer1 = "True" ];do
        echo "De combien voulez-vous scale le service :"
        echo "Pour quitter entrer : q"
        echo "Pour revenir en arriere entrer : r"
        read scale
        if nombre "$scale"; then
            continuer1="False"
            sudo docker service scale $service=$scale
        elif [[ $scale = "q" ]]; then
            exit
        elif [[ $scale = "r" ]]; then
            continuer="True"
        else
            clear
            echo "-----Entrée invalide, veuillez entrer uniquement des
chiffres.----->"
        fi
    done
elif [[ $service = "4" ]]; then
    continuer="False"
    service="my_php_service"
    clear
    while [ $continuer1 = "True" ];do
        echo "De combien voulez-vous scale le service :"
        echo "Pour quitter entrer : q"
        echo "Pour revenir en arriere entrer : r"
        read scale
        if nombre "$scale"; then
            continuer1="False"
            sudo docker service scale $service=$scale
        elif [[ $scale = "q" ]]; then
            exit
        elif [[ $scale = "r" ]]; then
            continuer="True"
        else
            clear
            echo "-----Entrée invalide, veuillez entrer uniquement des
chiffres.----->"
        fi
    done
elif [[ $service = "5" ]]; then
    continuer="False"
    echo "Vous avez choisi de quitter. Au revoir!"
    exit
else
    echo "Choix invalide, veuillez entrer un chiffre entre 1 et 5."
fi
done

```

6. Configuration de TrueNAS

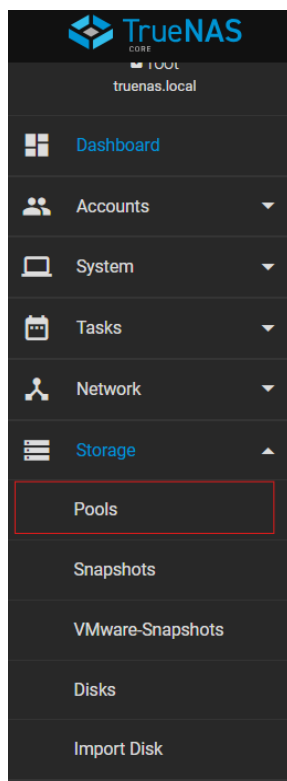
6.1 Montage d'un RAID 5 sur notre NAS

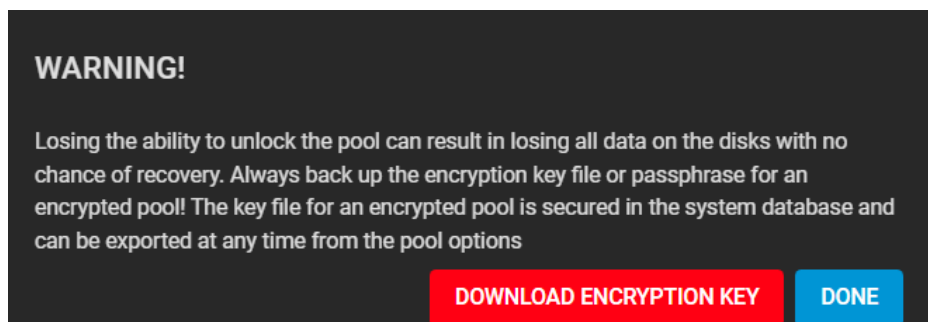
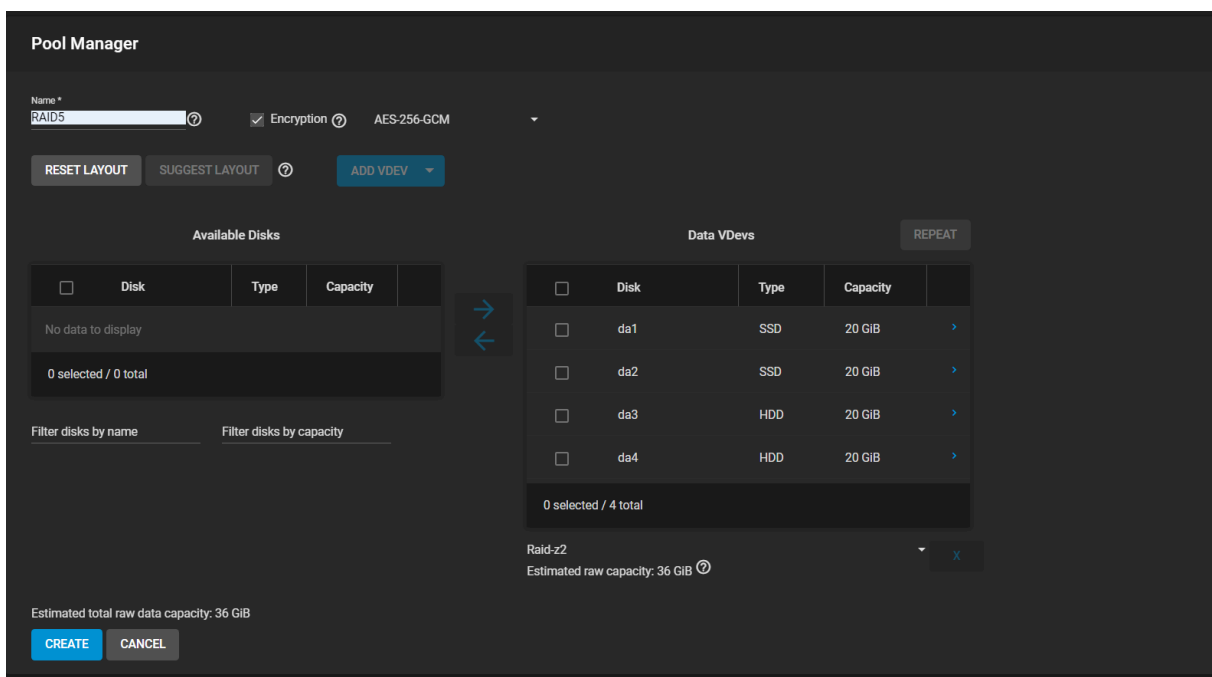
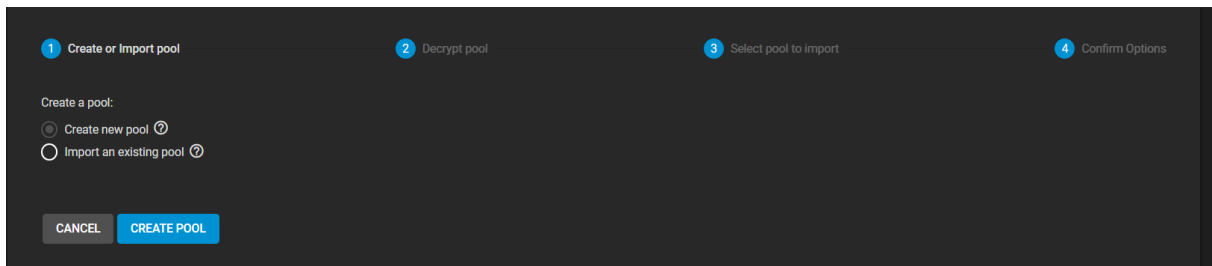
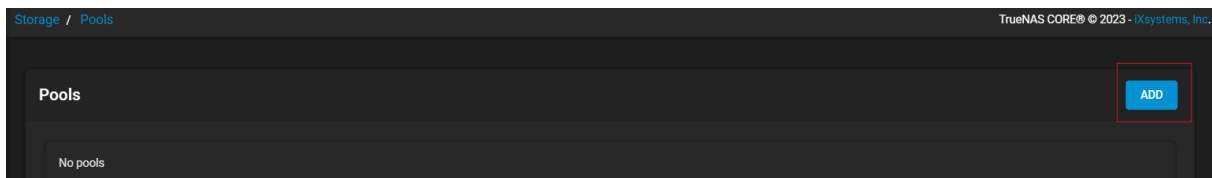
On a dans un premier temps créer une machine tournant sous TrueNAS, avec 5 disque de 20GO dessus qu'on va monter en RAID 5 :

Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	30 GB
Hard Disk 2 (SCSI)	20 GB
Hard Disk 3 (SCSI)	20 GB
Hard Disk 4 (SCSI)	20 GB
Hard Disk 5 (SCSI)	20 GB
CD/DVD (IDE)	Using file E:\iso\TrueNAS.iso
Network Adapter	Bridged (Automatic)
Network Adapter 2	NAT
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

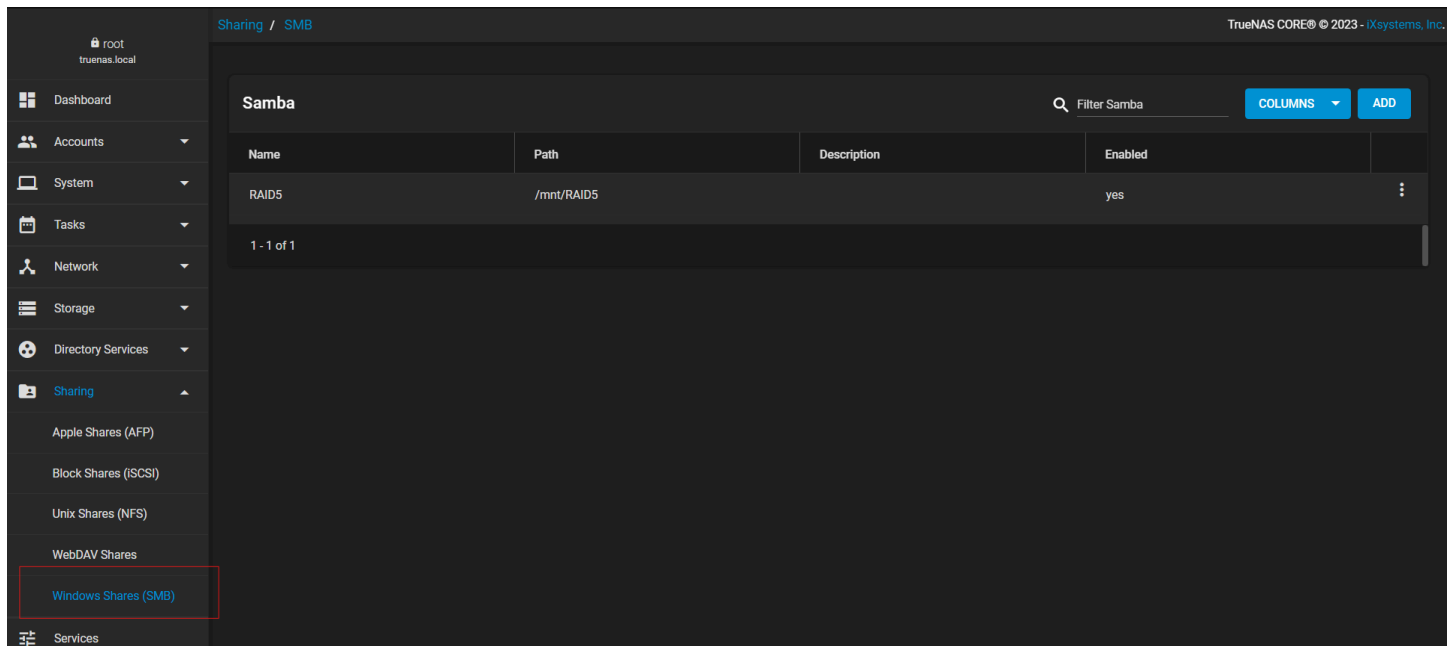
6.2 Création d'un pool dans TrueNAS

On va maintenant créer notre volume raid 5 sur notre serveur TrueNAS:





6.3 Création de notre partage SMB sur TrueNAS



6.4 Montage de notre volume NFS

```
sudo apt install cifs-utils
```

```
sudo mkdir /mnt/share
```

```
sudo mount.cifs //192.168.0.200/RAID5/agent /mnt/share -o username=agent,password=123
```

```
cd /home/user
```

On créer ensuite un fichier credentials dans notre home user :

```
nano credentials
```

```
username=agent
```

```
password=123
```

ensuite on fait la commande :

```
sudo chmod 600 credentials
```

ensuite on modifier le fichier **/etc/fstab** avec la ligne suivante :

```
//192.168.0.200/partage /mnt/share cifs credentials=/home/raph/credentials,icharset=utf8,vers=3.0 0 0
```

Une fois le fichier modifier on peut faire la commande :

```
sudo mount -a
```

puis ensuite on peut faire la commande :

```
df -h
```

7. Installation d'un service de visualisation

Swarm Visualizer est un outil qui permet de visualiser la configuration du cluster Swarm. Il affiche les conteneurs exécutés sur chaque nœud sous forme de visuels.

```
sudo docker service create \
  --name=viz \
  --publish=8080:8080/tcp \
  --constraint=node.role==manager \
  --mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  dockersamples/visualizer
```

On y accède via <http://192.168.0.150:8080/>



8. Automatisation : update automatique des services

Pour mettre à jour les services, nous avons fait un script:

sudo nano /Documents/docker/update-replica.sh

```
GNU nano 7.2 update-replica.sh
#!/bin/bash

sudo docker service update --force my_nginx_service
sudo docker service update --force my_php_service
sudo docker service update --force my_mariadb_service
sudo docker service update --force my_vscode_service
sudo docker service update --force viz
```

Script que nous avons automatisé toutes les heures avec un cron dans la crontab:

sudo crontab -e

```
GNU nano 7.2
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
# :
# m h dom mon dow  command
@hourly /home/master/Documents/docker/update-replica.sh
```