

Docker Swarm

Rappel : Fonctionnement de base de Docker

Docker c'est quoi ?

Docker est une plateforme permettant de créer, déployer et exécuter des applications dans des conteneurs. Un conteneur est une unité légère, portable et isolée qui contient tout ce dont une application a besoin pour fonctionner : code, bibliothèques, dépendances, etc.

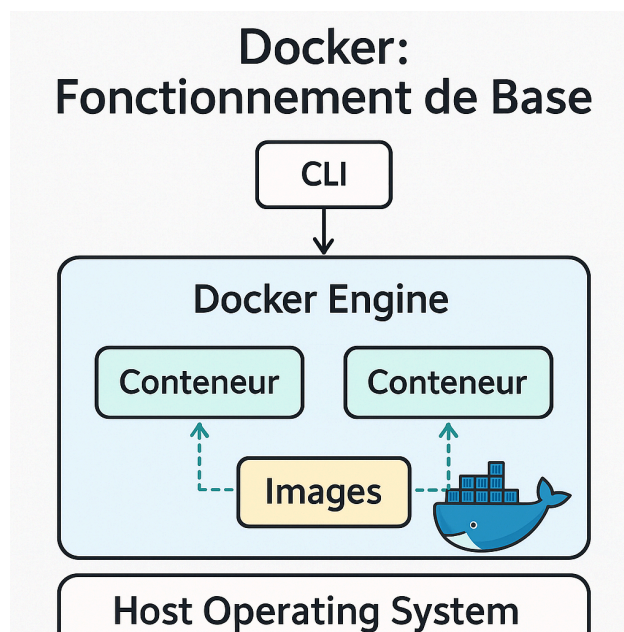
Composants clés :

- Docker Engine : moteur qui exécute les conteneurs
- Image Docker : modèle pour créer un conteneur (comme un ISO pour VM)
- Conteneur : instance d'une image, en cours d'exécution
- Dockerfile : script qui décrit comment construire une image
- Docker Hub : registre public pour partager des images Docker

📦 Cycle de vie classique :

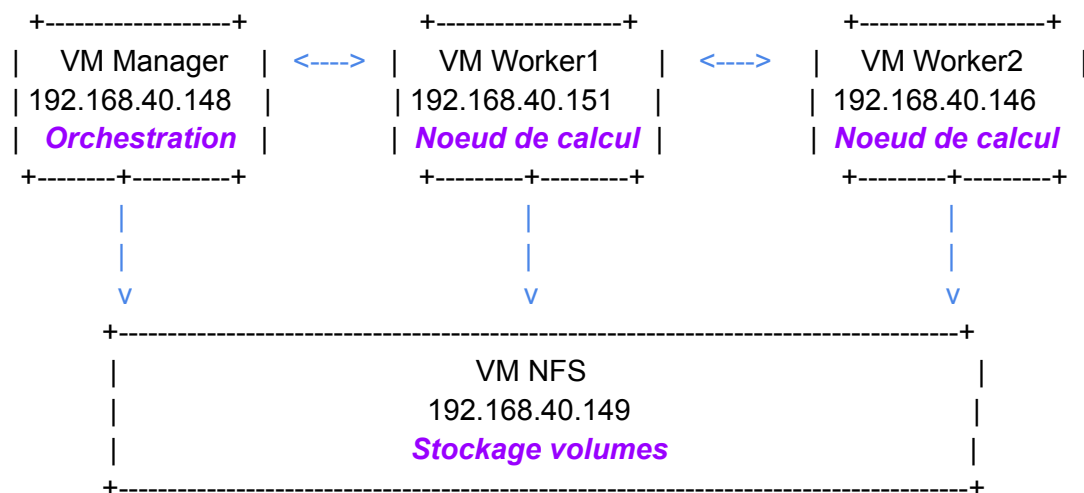
1. Créer une image avec un `Dockerfile`
2. Construire l'image : `docker build`
3. Lancer un conteneur : `docker run`
4. Superviser les conteneurs : `docker ps`, `docker logs`

Schéma du fonctionnement de Docker



Déploiement Docker Swarm pour la Continuité d'Activité

Architecture



Installation des machines virtuelles (VM)

- ◆ Répartition des rôles

Nom de la VM	Adresse IP	Rôle
manager	192.168.40.148	Chef du cluster Docker Swarm
worker1	192.168.40.151	Nœud de calcul Docker
worker2	192.168.40.146	Nœud de calcul Docker
nfs	192.168.40.149	Serveur de stockage NFS

- ◆ Configuration recommandée pour chaque VM

- **RAM** : 2 Go
- **vCPU** : 2
- **Disque** :

- 8 Go pour manager et workers
- 20 Go pour la VM NFS

Chaque VM est installée avec **Debian 12 (sans interface graphique)**.

Installation des paquets requis

Sur les VM **manager**, **worker1**, **worker2**

```
sudo apt update && sudo apt install -y \  
  curl \  
  ca-certificates \  
  gnupg \  
  lsb-release \  
  apt-transport-https \  
  vim \  
  net-tools \  
  openssh-server \  
  nfs-common
```

Installer Docker

```
curl -fsSL https://get.docker.com | sudo sh
```

Activer et démarrer Docker

```
sudo systemctl enable docker  
sudo systemctl start docker
```

Vérifier les versions

```
docker --version  
docker compose version
```

Sur la VM **nfs**

```
sudo apt update && sudo apt install -y \  
  nfs-kernel-server \  
  vim \  
  net-tools \  
  openssh-server
```

Les paquets sont installés une fois que les VMs sont temporairement configurées avec une adresse IP en **DHCP** pour avoir accès à Internet.

Une fois les installations terminées, les VMs sont repassées en **IP fixe** pour la stabilité du cluster.

Initialisation du cluster Docker Swarm

Sur la VM manager

```
sudo docker swarm init --advertise-addr 192.168.40.148
```

Cette commande initialise le cluster et fournit une commande avec un token permettant aux Workers de rejoindre le cluster. Exemple :

```
docker swarm join --token SWMTKN-1-abc123... 192.168.40.148:2377
```

Ajout des Workers

Sur chaque VM Worker, exécuter la commande `join` avec `sudo` :

```
sudo docker swarm join --token SWMTKN-1-abc123... 192.168.40.148:2377
```

Si le token est expiré ou perdu, le régénérer depuis le Manager :

```
docker swarm join-token worker
```

Vérification du cluster

Depuis le manager :

```
docker node ls
```

Sortie attendue :

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
xxxxx	manager	Ready	Active	Leader
yyyyy	worker1	Ready	Active	
zzzzz	worker2	Ready	Active	

Accès SSH aux Workers depuis le manager

Comme les Workers n'ont pas Internet, le plus simple est de se connecter à eux depuis la VM manager.

```
ssh utilisateur@192.168.40.148
```

Cela permet de copier-coller facilement la commande Swarm depuis le manager et de la lancer sur chaque Worker

Test du stockage partagé avec NFS

Objectif : Vérifier que le volume Docker `nfs-volume` fonctionne correctement et permet d'échanger des fichiers entre les conteneurs, quel que soit le nœud sur lequel ils tournent.

Commande pour créer le volume (à exécuter uniquement sur le Manager)

```
docker volume create \
  --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.40.149,rw \
  --opt device=:/srv/docker-share \
  nfs-volume
```

```
dome@Manager:~$ sudo mount -t nfs 192.168.40.149:/srv/docker-share /mnt
dome@Manager:~$ sudo docker run -it --rm -v nfs-volume:/data alpine sh
/ # cat data
cat: read error: Is a directory
/ # ls
bin      dev      home     media    opt      root     sbin     sys      usr
data     etc      lib      mnt      proc     run      srv      tmp      var
/ # cat data/test.txt
Hello Swarm!
/ # exit
```

Étape 1 : Écriture dans le volume depuis un conteneur

Lancer un conteneur Alpine sur le **Manager** :

```
docker run -it --rm -v nfs-volume:/data alpine sh
```

Dans le conteneur :

```
echo "Hello Swarm!" > /data/test.txt
exitls
```

Étape 2 : Lecture du fichier depuis un autre conteneur

Toujours depuis le manager (ou un worker si souhaité) :

```
docker run -it --rm -v nfs-volume:/data alpine sh
```

Puis dans le conteneur :

```
cat /data/test.txt
```

Sortie attendue :Hello Swarm!

```
dome@Manager:~$ sudo docker run -it --rm -v nfs-volume:/data alpine sh
/ # echo "hello from alpine" > /data/test.txt
/ # exit
dome@Manager:~$ |
dome@NFS:~$ cat /srv/docker-share/test.txt
hello from alpine
dome@NFS:~$
```

Conclusion

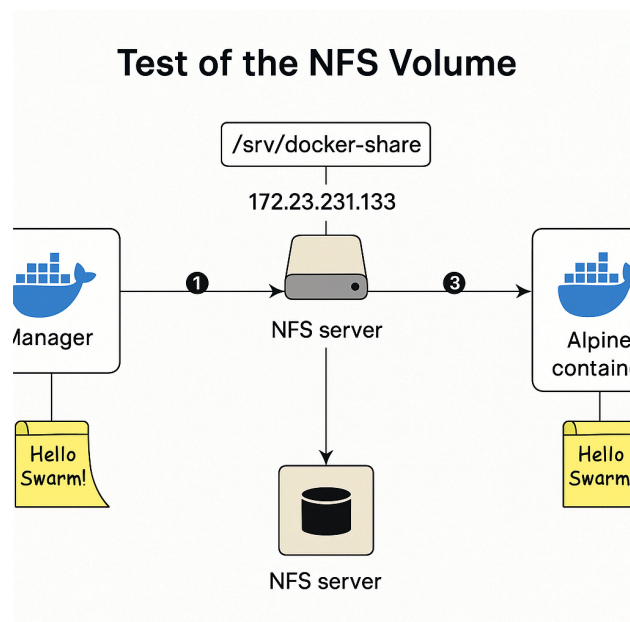
Le test prouve que :

- Le volume `nfs-volume` est monté correctement via le serveur NFS
- Les données sont persistées et partagées entre tous les conteneurs du cluster
- Le volume est prêt pour être utilisé par les services (comme Nginx ou MariaDB) dans Swarm

Schéma de fonctionnement

1. Le conteneur Alpine sur le manager écrit un fichier dans `/data`, qui correspond à `/srv/docker-share` sur le serveur NFS (`172.23.231.133`).
2. Le fichier est sauvegardé sur le serveur NFS.
3. Un autre conteneur (même nœud ou autre) relit ce fichier depuis le même volume.

Tous les conteneurs voient le même contenu.



Nginx dans Docker Swarm

Objectif

Mettre en place un service **web Nginx** au sein du **cluster Docker Swarm**, pour assurer la **haute disponibilité** et tester le **fonctionnement basique de l'orchestration** sur les nœuds Workers.

Étape 1 : Création du fichier `nginx-swarm.yml`

📌 À réaliser **sur la VM manager uniquement**

version: '3.8'

services:

nginx:

image: nginx:latest

ports:

- "80:80"

deploy:

replicas: 2

placement:

constraints:

- node.role == worker

Sauvegarder le fichier avec :

`nano nginx-swarm.yml`

Étape 2 : Déploiement du service

Toujours depuis la **VM manager**, lancer la commande suivante :

`sudo docker stack deploy -c nginx-swarm.yml nginx_stack`

```
dome@Manager:~$ sudo docker stack deploy -c nginx-swarm.yml nginx_stack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network nginx_stack_default
Creating service nginx_stack_nginx
dome@Manager:~$
```

Voir l'emplacement des conteneurs déployés :

`sudo docker service ps nginx_stack_nginx`

```
dome@Manager:~$ sudo docker service ps nginx_stack_nginx
ID                NAME                IMAGE                NODE                DESIRED STATE       CURRENT STATE       ERROR                PORTS
urxarlnD7tuv      nginx_stack_nginx.1  nginx:latest         Worker-1            Running              Running 12 minutes ago
vdgk37vfjbow      nginx_stack_nginx.2  nginx:latest         Worker-2            Running              Running 12 minutes ago
dome@Manager:~$
```

pour voir où on en est:

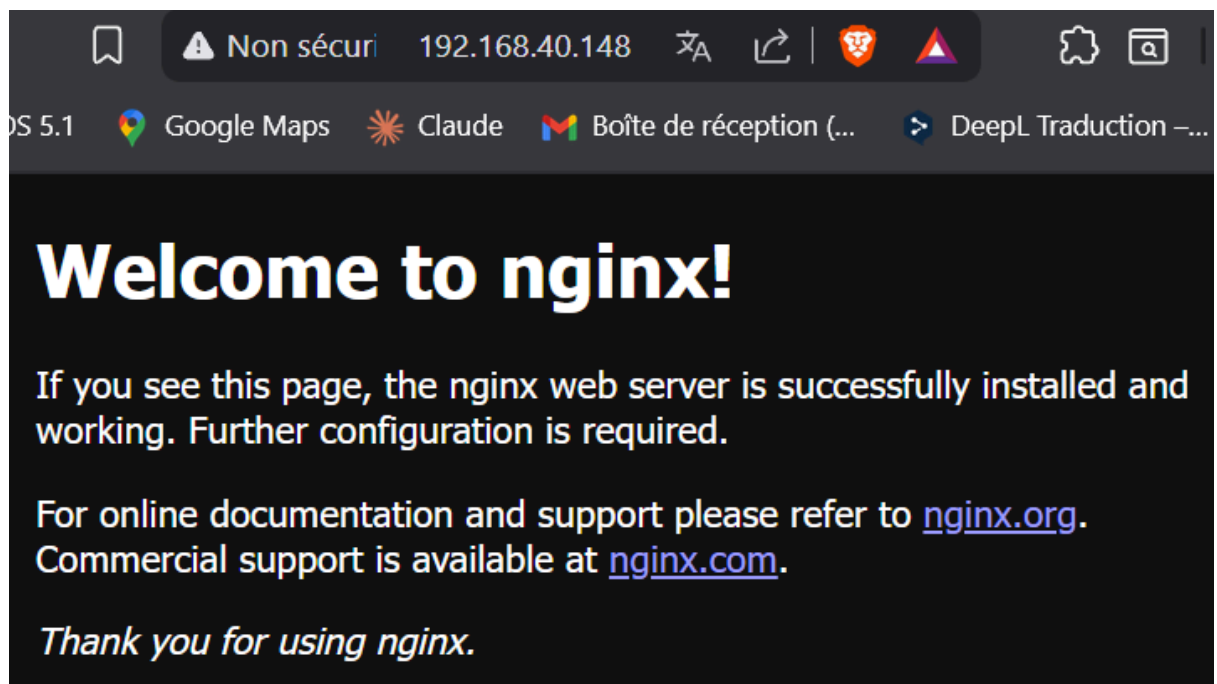
```
dome@Manager:~$ sudo docker service ls
sudo docker service ps nginx-service
ID                NAME                MODE                REPLICAS    IMAGE                PORTS
p4wikkyfkze4     nginx_stack_nginx    replicated          2/2         nginx:latest        *:80->80/tcp
no such service: nginx-service
dome@Manager:~$ sudo docker service ps nginx-service
no such service: nginx-service
```

Accéder à Nginx via un navigateur ou `curl` :

```
dome@Manager:~$ curl http://192.168.40.148
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
dome@Manager:~$ |
```



Vérifier la répartition réelle des conteneurs sur les nœuds du cluster:

Pour voir **sur quels nœuds** les services ont été déployés, utilise la commande suivante sur le **nœud manager** :

```
sudo docker stack ps nginx_stack
```

```
dome@Manager:~$ sudo docker stack ps nginx_stack
ID                NAME                IMAGE             NODE       DESIRED STATE   CURRENT STATE           ERROR           PORTS
o7gcgb299zrb     nginx_stack_nginx.1  nginx:latest     Worker-1   Running         Running 3 minutes ago
ch7gryblblex     nginx_stack_nginx.2  nginx:latest     Worker-2   Running         Running 3 minutes ago
dome@Manager:~$
```

Interprétation :

- **NAME** : Nom du service (avec index de réplica).
- **NODE** : Le **nœud Docker (VM)** sur lequel tourne le conteneur.
- **DESIRED STATE** : Ce que Swarm souhaite (Running, Shutdown, etc.).
- **CURRENT STATE** : Ce qui se passe actuellement.
- **ERROR** : S'il y a une erreur de démarrage, elle s'affiche ici.

Cette commande est **essentielle** pour vérifier que l'orchestration fonctionne bien et que les conteneurs sont **répartis sur différents workers**.

```
dome@Manager:~$ sudo docker stack deploy -c mariadb-swarm.yml mariadb_stack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network mariadb_stack_default
Creating service mariadb_stack_mariadb
dome@Manager:~$
```

MariaDB avec NFS

Étape 1 : Créer le répertoire partagé sur la VM NFS

Connecte-toi à la VM NFS (celle avec l'IP 192.168.40.149 normalement).

Puis exécute les commandes suivantes :

```
sudo mkdir -p /srv/nfs/mariadb
```

```
sudo chown -R nobody:nogroup /srv/nfs/mariadb
```

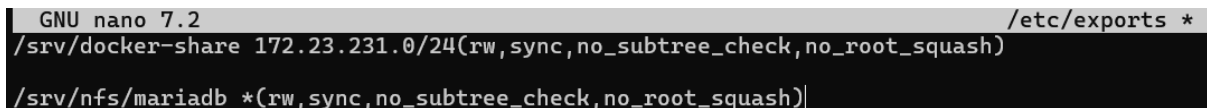
```
sudo chmod -R 777 /srv/nfs/mariadb
```

Ensuite, ajouter le partage NFS dans le fichier `/etc/exports` (si ce n'est pas déjà fait) :

```
sudo nano /etc/exports
```

Ajoute cette ligne :

```
/srv/nfs/mariadb *(rw,sync,no_subtree_check,no_root_squash)
```

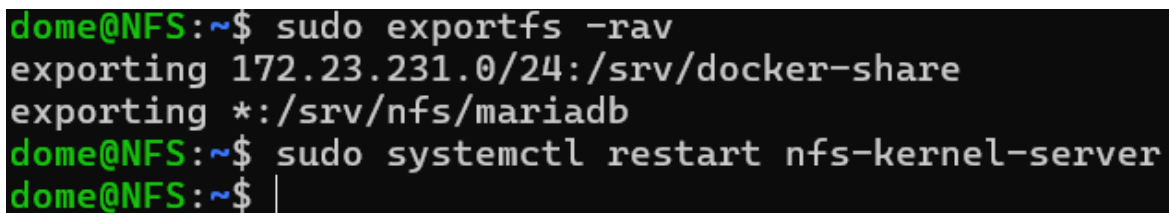


```
GNU nano 7.2 /etc/exports *  
/srv/docker-share 172.23.231.0/24(rw,sync,no_subtree_check,no_root_squash)  
/srv/nfs/mariadb *(rw,sync,no_subtree_check,no_root_squash)|
```

Puis redémarrer le serveur NFS :

```
sudo exportfs -rav
```

```
sudo systemctl restart nfs-kernel-server
```



```
dome@NFS:~$ sudo exportfs -rav  
exporting 172.23.231.0/24:/srv/docker-share  
exporting */srv/nfs/mariadb  
dome@NFS:~$ sudo systemctl restart nfs-kernel-server  
dome@NFS:~$ |
```

Étape 2 : Vérifier l'accès NFS depuis tous les nœuds (manager et workers)

Sur chaque nœud Docker (manager + workers) :

1. Installer le client NFS (si ce n'est pas déjà fait) :

```
sudo apt update
```

```
sudo apt install nfs-common -y
```

2. Monter temporairement le répertoire pour tester :

```
sudo mount -t nfs 172.23.231.133:/srv/nfs/mariadb /mnt
```

Tester l'écriture :

```
sudo touch /mnt/test.txt
```

Si le fichier est créé sans erreur → les droits sont OK 

Démonter le dossier :

```
sudo umount /mnt
```

```
dome@Manager:~$ sudo mount -t nfs 172.23.231.133:/srv/nfs/mariadb /mnt
[sudo] Mot de passe de dome :
dome@Manager:~$ sudo touch /mnt/test.txt
dome@Manager:~$ sudo umount /mnt
dome@Manager:~$
```

```
dome@Worker-1:~$ sudo mount -t nfs 172.23.231.133:/srv/nfs/mariadb /mnt
dome@Worker-1:~$ sudo touch /mnt/test.txt
dome@Worker-1:~$ sudo umount /mnt
dome@Worker-1:~$
```

```
dome@Worker-2:~$ sudo mount -t nfs 172.23.231.133:/srv/nfs/mariadb /mnt
[sudo] Mot de passe de dome :
dome@Worker-2:~$ sudo touch /mnt/test.txt
dome@Worker-2:~$ sudo umount /mnt
dome@Worker-2:~$
```

Étape 3 : Préparer le volume partagé sur chaque nœud

À exécuter sur le manager ET tous les workers :

```
sudo docker volume create \
```

```
--driver local \
```

```
--opt type=nfs \
```

```
--opt o=addr=172.23.231.133,rw \
```

```
--opt device=:/srv/nfs/mariadb \
```

```
mariadb_data
```

Cela crée un volume nommé `mariadb_data` pointant vers le dossier `/srv/nfs/mariadb` sur le serveur NFS.

```
dome@Worker-1:~$ sudo docker volume create \
--driver local \
--opt type=nfs \
--opt o=addr=172.23.231.133,rw \
--opt device=:/srv/nfs/mariadb \
mariadb_data
[sudo] Mot de passe de dome :
mariadb_data
dome@Worker-1:~$ sudo mount -t nfs 172.23.231.133:/srv/nfs/mariadb /mnt
dome@Worker-1:~$ sudo touch /mnt/test.txt
dome@Worker-1:~$ sudo umount /mnt
```


```
dome@Worker-1:~$ sudo docker volume create \
--driver local \
--opt type=nfs \
--opt o=addr=172.23.231.133,rw \
--opt device=:/srv/nfs/mariadb \
mariadb_data
[sudo] Mot de passe de dome :
mariadb_data
dome@Worker-1:~$
```

```
dome@Worker-2:~$ sudo docker volume create \
--driver local \
--opt type=nfs \
--opt o=addr=172.23.231.133,rw \
--opt device=:/srv/nfs/mariadb \
mariadb_data
mariadb_data
dome@Worker-2:~$
```

⚠ À savoir :

Le volume `mariadb_data` doit être créé **manuellement sur chaque nœud Docker** (manager et tous les workers), même si le service MariaDB ne s'exécute que sur un seul.

Cela garantit que si le conteneur MariaDB est déplacé sur un autre nœud (bascule PRA), le nouveau nœud saura monter le volume partagé NFS correctement. Sinon, le service échouera avec l'erreur : `volume not found`.

 **Important** : Tous les nœuds doivent pouvoir accéder au dossier NFS avec les droits d'écriture.

Étape 4 : Créer le fichier `mariadb-swarm.yml`

Sur la **VM manager** uniquement :

```
nano mariadb-swarm.yml
```

Colle le contenu suivant :

```
version: '3.8'
```

```
services:
```

```
  mariadb:
```

```
    image: mariadb:10.5
```

```
    environment:
```

```
      MYSQL_ROOT_PASSWORD: rootpass
```

```
      MYSQL_DATABASE: mydb
```

```
      MYSQL_USER: user
```

```
      MYSQL_PASSWORD: userpass
```

```
volumes:
```

```
  - mariadb_data:/var/lib/mysql
```

```
deploy:
```

```
  replicas: 1
```

```
  placement:
```

```
    constraints:
```

```
      - node.role == worker
```

```
volumes:
```

```
  mariadb_data:
```

external: true

Détails :

- replicas: 1 : pour éviter les conflits d'écriture dans MySQL (on verra la haute dispo ensuite).
- external: true : car le volume a déjà été créé manuellement (liens NFS).
- Variables d'environnement pour initialiser MySQL.

Étape 5 : Déployer la stack MariaDb

Sur le **manager** :

`docker stack deploy -c mariadb-swarm.yml mariadb_stack`

```
dome@Manager:~$ sudo docker stack deploy -c mariadb-swarm.yml mariadb_stack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network mariadb_stack_default
Creating service mariadb_stack_mariadb
```

Étape 6 : Vérifier le bon fonctionnement

Vérifier les services :

`docker service ls`

```
dome@Manager:~$ sudo docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
3g85ignhaedi	mariadb_stack_mariadb	replicated	1/1	mariadb:10.5	
qjjot66pjl9f	nginx_stack_nginx	replicated	2/2	nginx:latest	*:80->80

`docker stack ps mariadb_stack`

```
dome@Manager:~$ sudo docker stack ps mariadb_stack
```

ID	NAME	IMAGE	NODE	DESIRED STATE
3uu4jsg74fd8	mariadb_stack_mariadb.1	mariadb:10.5	Worker-2	Running
xofrje6lq62g	_ mariadb_stack_mariadb.1	mariadb:10.5	Worker-1	Shutdown

Running 30 hours ago
Failed 30 hours ago "task: non-zero exit (255)"
dome@Manager:~\$ |

Sur la VM NFS :

`ls -l /srv/nfs/mariadb`

```
dome@NFS:~$ ls -l /srv/nfs/mariadb
total 140288
-rw-rw---- 1 999 systemd-journal 17801216 13 juin 17:36 aria_log.00000001
-rw-rw---- 1 999 systemd-journal      52 13 juin 17:36 aria_log_control
-rw-rw---- 1 999 systemd-journal    898 13 juin 17:36 ib_buffer_pool
-rw-rw---- 1 999 systemd-journal 12582912 13 juin 17:36 ibdata1
-rw-rw---- 1 999 systemd-journal 100663296 13 juin 17:36 ib_logfile0
-rw-rw---- 1 999 systemd-journal 12582912 13 juin 17:36 ibtmp1
-rw-rw---- 1 999 systemd-journal      0 13 juin 17:36 multi-master.info
drwx----- 2 999 systemd-journal  4096 13 juin 17:36 mydb
drwx----- 2 999 systemd-journal  4096 13 juin 17:36 mysql
-rw-r--r-- 1 999 systemd-journal    15 13 juin 17:36 mysql_upgrade_info
drwx----- 2 999 systemd-journal  4096 13 juin 17:36 performance_schema
-rw-r--r-- 1 999 systemd-journal      0 13 juin 17:16 test.txt
```

Des fichiers comme ibdata1, mysql/, mydb/ devraient apparaître si tout fonctionne.

`sudo docker ps --filter name=mariadb_stack_mariadb`

`sudo docker exec -it <ID_contener> bash`

`mysql -uuser -p`

Mot de passe : userpass

```
dome@Worker-2:~$ sudo docker ps
[sudo] Mot de passe de dome :
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
6048b7b5b37d   mariadb:10.5   "docker-entrypoint.s..." 31 hours ago   Up 31 hours   3306/tcp       mariadb_stack_mariadb.1.3uu4jsg74fd845tp9bmw879ou
0259a7fb6c3c   nginx:latest   "/docker-entrypoint..." 31 hours ago   Up 31 hours   80/tcp         nginx_stack_nginx.2.s0qesav10rid9jafq05ctes0u
73c040fd91af   nginx:latest   "/docker-entrypoint..." 31 hours ago   Up 31 hours   80/tcp         nginx_stack_nginx.1.k287084ueotr6byun8nnfzfg0

dome@Worker-2:~$ sudo docker exec -it 6048b7b5b37d bash
root@6048b7b5b37d:/# mysql -uuser -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.5.29-MariaDB-ubu2004 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> |
```

Déploiement du serveur applicatif PHP

Étape 1 : Préparer le répertoire des fichiers PHP sur le serveur NFS

📌 Sur la **VM NFS (192.168.40.149)** :

```
sudo mkdir -p /srv/docker-share/php  
sudo chmod -R 777 /srv/docker-share/php
```

Étape 2 : Créer un fichier `index.php` de test

Toujours sur la **VM NFS** :

```
sudo nano /srv/docker-share/php/index.php
```

Colle ce contenu simple :

Copier le code

```
<?php  
phpinfo();  
?>
```

Cela te permettra de tester que le serveur PHP fonctionne bien.

Étape 3 : Créer le volume Docker pour PHP

📌 Sur le **Manager** :

Copier le code


```
sudo docker volume create \  
  --driver local \  
  --opt type=nfs \  
  --opt o=addr=192.168.40.149,rw \  
  --opt device=/srv/docker-share/php \  
  php-volume
```

Étape 4 : Déployer le service PHP dans Swarm

 Toujours sur le **Manager** :

Copier le code

```
sudo docker service create \
  --name php-service \
  --replicas 1 \
  --network app_net \
  --mount type=volume,source=php-volume,target=/var/www/html \
  php:8.2-cli \
  php -S 0.0.0.0:80 -t /var/www/html
```

 Cette commande :

- Lance un petit serveur HTTP avec PHP 8.2
 - Expose le fichier `index.php`
 - Monte ton dossier partagé NFS
-

Étape 5 : Vérifier l'état du service

Copier le code

```
sudo docker service ps php-service
```

Tu dois voir **Running** sur un nœud.

Étape 6 : Tester dans un navigateur

Accède à l'adresse IP du nœud qui héberge `php-service`, **sur le port 80**

Exemple :

Copier le code

```
http://192.168.40.151
```

Tu dois voir **la page de `phpinfo()`** 

Test PRA — Forcer le redéploiement d'un service sur un autre nœud

Docker Swarm permet de tester la reprise d'activité (PRA) en redéployant un service critique comme MariaDB sur un autre nœud en cas de panne.

Voici les étapes pour simuler cette situation et observer la bascule automatique :

1. Depuis le manager, exécuter la commande suivante pour forcer le redéploiement :

```
sudo docker service update --force mariadb_stack_mariadb
```

2. Vérifier ensuite l'état des conteneurs :

```
sudo docker stack ps mariadb_stack
```

Vous devriez voir :

- Un ancien conteneur "Shutdown" (sur l'ancien nœud)
- Un nouveau conteneur "Running" (sur un autre nœud)

Exemple :

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
zyeoyrtuofa8	mariadb_stack_mariadb.1	mariadb:10.5	Worker-1	Running	Running 10 seconds ago
3uu4jsg74fd8	_ mariadb_stack_mariadb.1	mariadb:10.5	Worker-2	Shutdown	Shutdown
xofrje6lq62g	_ mariadb_stack_mariadb.1	mariadb:10.5	Worker-1	Shutdown	Failed

Cela confirme que le service est bien redéployé sur un autre nœud, ce qui démontre la capacité de Docker Swarm à assurer la continuité de service (PRA).

```
dome@Manager:~$ sudo docker service update --force mariadb_stack_mariadb
mariadb_stack_mariadb
overall progress: 1 out of 1 tasks
1/1: running [=====>]
verify: Service mariadb_stack_mariadb converged
dome@Manager:~$ sudo docker stack ps mariadb_stack
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
zyeoyrtuofa8	mariadb_stack_mariadb.1	mariadb:10.5	Worker-1	Running	Running 11 seconds ago		
3uu4jsg74fd8	_ mariadb_stack_mariadb.1	mariadb:10.5	Worker-2	Shutdown	Shutdown 7 seconds ago		
xofrje6lq62g	_ mariadb_stack_mariadb.1	mariadb:10.5	Worker-1	Shutdown	Failed 31 hours ago	"task: non-zero exit (255)"	

```
dome@Manager:~$
```

Mise en place des services critiques avec Docker Compose & Dockerfile

Objectif

Utiliser **Docker Compose** pour automatiser le déploiement des services suivants sur notre cluster Docker Swarm :

- MariaDB (base de données)
- PHP (serveur web Apache avec connecteur PDO)
- Nginx (proxy inverse statique)
- phpMyAdmin (interface de gestion)

Dockerfile personnalisé : **monphp**

Rôle

Préparer une image PHP avec Apache et les extensions nécessaires pour se connecter à MariaDB (**pdo_mysql**, **mysqli**...).

Contenu du fichier **Dockerfile** :

```
FROM php:8.2-apache
```

```
RUN apt-get update && apt-get install -y \  
    libzip-dev zip unzip libonig-dev libxml2-dev mariadb-client \  
    && docker-php-ext-install pdo pdo_mysql mysqli
```

Docker Compose : **docker-compose.yml**

Objectif

Définir tous les services nécessaires avec leurs volumes, ports, réseaux et dépendances.

Contenu simplifié du fichier `docker-compose.yml`

```
version: "3.8"

services:
  mariadb:
    image: mariadb:10.7
    environment:
      MYSQL_ROOT_PASSWORD: SuperPass123
    volumes:
      - mariadb-volume:/var/lib/mysql
    networks:
      - app_net

  php:
    image: monphp:latest
    build: ./php
    volumes:
      - php-volume:/var/www/html
    ports:
      - "8080:80"
    networks:
      - app_net

  nginx:
    image: nginx:latest
    volumes:
      - nginx-volume:/usr/share/nginx/html
    ports:
      - "80:80"
    networks:
      - app_net

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "8081:80"
    environment:
      PMA_HOST: mariadb
      PMA_USER: root
      PMA_PASSWORD: SuperPass123
```

```
    networks:
      - app_net

volumes:
  php-volume:
    driver: local
    driver_opts:
      type: "nfs"
      o: "addr=192.168.40.149,rw"
      device: ":/srv/docker-share/php"

  mariadb-volume:
    driver: local
    driver_opts:
      type: "nfs"
      o: "addr=192.168.40.149,rw"
      device: ":/srv/docker-share/mariadb/data"

  nginx-volume:
    driver: local
    driver_opts:
      type: "nfs"
      o: "addr=192.168.40.149,rw"
      device: ":/srv/docker-share/nginx/html"

networks:
  app_net:
    driver: overlay
```

Commandes de déploiement

1. Lancer le stack :

```
sudo docker stack deploy -c docker-compose.yml app_stack
```

2. Vérifier l'état :

```
sudo docker stack ps app_stack
```

3. Forcer une mise à jour :

```
sudo docker service update --force app_stack_php
```

4. Supprimer proprement :

```
sudo docker stack rm app_stack
```

Conseils

- Tous les volumes NFS doivent être **accessibles depuis tous les nœuds**.
- Ne pas utiliser `build` : dans Docker Compose si tu ne lances pas `docker stack deploy` depuis le même dossier.
- S'assurer que les services MariaDB et PHP soient dans le **même réseau overlay**.
- Pour tester localement une image : `sudo docker run -it monphp:latest bash`.

En résumé

Service	Image utilisée	Volumes persistants	Exposé sur le port
PHP	monphp:latest	<code>/var/www/html</code> (NFS)	8080
MariaDB	mariadb:10.7	<code>/var/lib/mysql</code> (NFS)	3306 (interne)
phpMyAdmin	phpmyadmin/phpmyadmin	aucun	8081
Nginx	nginx:latest	<code>/usr/share/nginx/html</code>	80

Vérification de la haute disponibilité (PRA)

1. Vérifier que le service est actif :

Copier le code

```
sudo docker service ps mariadb-service
```

2. Simuler la perte du nœud qui héberge MariaDB (arrêt ou déconnexion réseau)
3. Observer que Docker Swarm redéploie automatiquement MariaDB sur un autre nœud (**Running**)
4. Se reconnecter à MariaDB et vérifier que les données sont intactes :

Copier le code

```
sudo docker run -it --rm --network app_net mariadb:10.7 mysql -h  
mariadb-service -u root -p
```

Résultat attendu

- MariaDB est **déployée dynamiquement** selon les nœuds disponibles
- Les données sont **conservées** grâce au volume NFS
- Le **cluster est tolérant aux pannes**, validant le **PRA**

Déployer service Registry Docker dans Swarm

Pourquoi ?

Un registre interne permet d'héberger tes propres images en local, idéal en environnement fermé ou hors ligne.

1. Créer le service Registry Docker dans Swarm

Dans Ménager

Commandes :

```
sudo docker service create \
  --name registry \
  --publish published=5000,target=5000 \
  --mount type=volume,source=registry-data,target=/var/lib/registry \
  registry:2
```

```
dome@Manager:~$ sudo docker service create \
  --name registry \
  --publish published=5000,target=5000 \
  --mount type=volume,source=registry-data,target=/var/lib/registry \
  registry:2
[sudo] Mot de passe de dome :
16keb3zkxfo88mzh7lps1htl7
overall progress: 1 out of 1 tasks
1/1: running
verify: Service 16keb3zkxfo88mzh7lps1htl7 converged
dome@Manager:~$ |
```

Mais ici on veut utiliser NFS :

Version avec NFS :

1. Sur le serveur NFS (172.23.231.133), crée un dossier exporté :

```
sudo mkdir -p /srv/nfs/registry
sudo chown -R 1000:1000 /srv/nfs/registry
```

2. Dans /etc/exports :

```
/srv/nfs/registry 172.23.231.0/24(rw,sync,no_subtree_check)
```

3. Puis :

```
sudo exportfs -ra
sudo systemctl restart nfs-kernel-server
```

```
dome@NFS:~$ sudo mkdir -p /srv/nfs/registry
[sudo] Mot de passe de dome :

dome@NFS:~$ sudo exportfs -ra
dome@NFS:~$ sudo systemctl restart nfs-kernel-server
```


4. Sur le manager Docker, crée un volume Docker monté sur NFS :

```
sudo docker volume create \  
  --driver local \  
  --opt type=nfs \  
  --opt o=addr=172.23.231.133,nolock,soft,rw \  
  --opt device=:/srv/nfs/registry \  
registry-nfs
```

```
dome@Manager:~$ sudo docker volume create \  
  --driver local \  
  --opt type=nfs \  
  --opt o=addr=172.23.231.133,nolock,soft,rw \  
  --opt device=:/srv/nfs/registry \  
registry-nfs  
registry-nfs
```

5. Puis lance le service :

```
sudo docker service create \  
  --name registry \  
  --publish published=5000,target=5000 \  
  --mount type=volume,source=registry-nfs,target=/var/lib/registry \  
registry:2
```

```
dome@Manager:~$ sudo docker service create \  
  --name registry \  
  --publish published=5000,target=5000 \  
  --mount type=volume,source=registry-nfs,target=/var/lib/registry \  
registry:2  
ju9pgo8qcfiwm9ae8poshvx9q  
overall progress: 1 out of 1 tasks  
1/1: running [=====>]  
verify: Service ju9pgo8qcfiwm9ae8poshvx9q converged  
dome@Manager:~$ |
```

Image custom HTML dans Docker Swarm avec registry local

Objectif :

- Créer une image personnalisée HTML
 - La stocker dans ton **registry local Docker** (sur le Manager : `192.168.40.148:5000`)
 - Déployer le service dans le cluster
 - Vérifier que le déploiement fonctionne **même hors ligne**
-

Étape 1 — Créer le projet d'image personnalisée HTML

Sur la VM `Manager` (192.168.40.148)

Arborescence :

```
mkdir html-app && cd html-app
```

```
sudo nano index.html
```

```
<html><body><h1>Hello depuis le cluster Swarm</h1></body></html>
```

Dockerfile

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
```

Étape 2 — Construire et pousser l'image dans le registry local

Machine : Manager uniquement

Construction :

```
sudo docker build -t 192.168.40.148:5000/html-app:v1 .
```

Push vers le registry :

```
sudo docker push 192.168.40.148:5000/html-app:v1
```

Étape 3 — Autoriser le registry HTTP sur tous les nœuds

Docker n'accepte pas les registries en HTTP par défaut. Il faut les déclarer comme "insecure".

Machine : Manager + Worker-1 (192.168.40.151) + Worker-2 (192.168.40.146)

Modifier ce fichier :

```
sudo nano /etc/docker/daemon.json
```

Et ajouter (ou remplacer) le contenu par :

json

```
{  
  "insecure-registries": ["192.168.40.148:5000"]  
}
```

Redémarrer proprement Docker :

```
sudo systemctl daemon-reexec  
sudo systemctl restart docker
```

 Répéter ces deux étapes sur chaque VM du cluster

Étape 4 — Déployer l'image dans Docker Swarm

Machine : Manager uniquement

Commande :

```
sudo docker service create \  
  --name test-html \  
  --publish published=8083,target=80 \  
  192.168.40.148:5000/html-app:v1
```

 Le port **8083** est utilisé pour éviter les conflits avec **monphp** sur **8080**.

Vérifie que le service tourne :

```
sudo docker service ls
```

Étape 5 — Tester l'accès au service

Depuis ton **navigateur web sur ton PC** (connecté au même réseau que les VMs) :

`http://192.168.40.148:8083`

 Tu dois voir : **Hello depuis le cluster Swarm**

Étape 6 – Tester le déploiement sans connexion Internet

Objectif :

Prouver que ton cluster Docker Swarm peut :

- déployer une nouvelle instance d'un service
 - sans se connecter à Docker Hub
 - grâce à ton registry local hébergé sur le Manager (192.168.40.148:5000)
-

Pourquoi c'est important ?

En situation réelle de PRA (Plan de Reprise d'Activité), ton entreprise peut perdre l'accès à Internet.

Si tu n'as pas préparé ton infrastructure avec un registry privé, tu ne pourras plus créer de conteneurs (puisqu'ils viennent souvent de Docker Hub).

Donc, tu montres ici que :

- ✓ Ton registry local contient déjà l'image
 - ✓ Tous les nœuds du cluster peuvent tirer cette image
 - ✓ Même hors ligne, le service peut être redéployé
-

Étapes concrètes

À faire uniquement sur la machine Manager (192.168.40.148)

Simuler une coupure d'accès à Internet, tout en gardant le réseau local (entre Manager et Workers) fonctionnel pour que le cluster continue de tourner.

Bloquer Docker Hub avec /etc/hosts (propre, sans couper le réseau)

1. Édite le fichier hosts :

```
sudo nano /etc/hosts
```

2. Ajoute à la fin :

```
127.0.0.1 registry-1.docker.io
127.0.0.1 auth.docker.io
127.0.0.1 hub.docker.com
```

3. Teste que Docker Hub est bloqué :

```
curl https://registry-1.docker.io
```

→ Résultat attendu : connection refused ou bloqué

Le réseau local entre tes VM reste actif, mais Docker Hub est inaccessible

Résultat de la commande :

```
curl https://registry-1.docker.io  
→ curl: (7) Failed to connect
```

Cela signifie que Docker ne pourra plus aller chercher d'image sur Internet.
Et c'est parfait pour tester que ton image HTML (html-app:v1) est bien tirée depuis le registry local, même sans Docker Hub.

Finaliser l'étape 6 :

Sur la VM Manager :

```
sudo docker service scale test-html=2
```

```
dome@Manager:~/html-app$ sudo docker service scale test-html=2  
test-html scaled to 2  
overall progress: 2 out of 2 tasks  
1/2: running [=====]  
2/2: running [=====]  
verify: Service test-html converged  
dome@Manager:~/html-app$ sudo docker service ps test-html  
ID            NAME          IMAGE                                NODE     DESIRED STATE  CURRENT STATE          ERROR          PORTS  
v7dirny8vu5u  test-html.1   192.168.40.148:5000/html-app:v1     Worker-2  Running        Running 19 minutes ago  
krv86rd5perg  test-html.2   192.168.40.148:5000/html-app:v1     Worker-1  Running        Running 6 seconds ago  
dome@Manager:~/html-app$
```

Puis vérifie :

```
sudo docker service ps test-html
```

→ Si les 2 conteneurs sont bien lancés, c'est que tout est autonome grâce à ton registry local (192.168.40.148:5000)

✓ Tu viens de valider :

Test	Résultat attendu	Ton résultat
Accès à Docker Hub bloqué	✗ KO	✓ KO
Scaling sans Docker Hub	✓ OK	à tester
Registry local utilisé	✓ OK	prêt