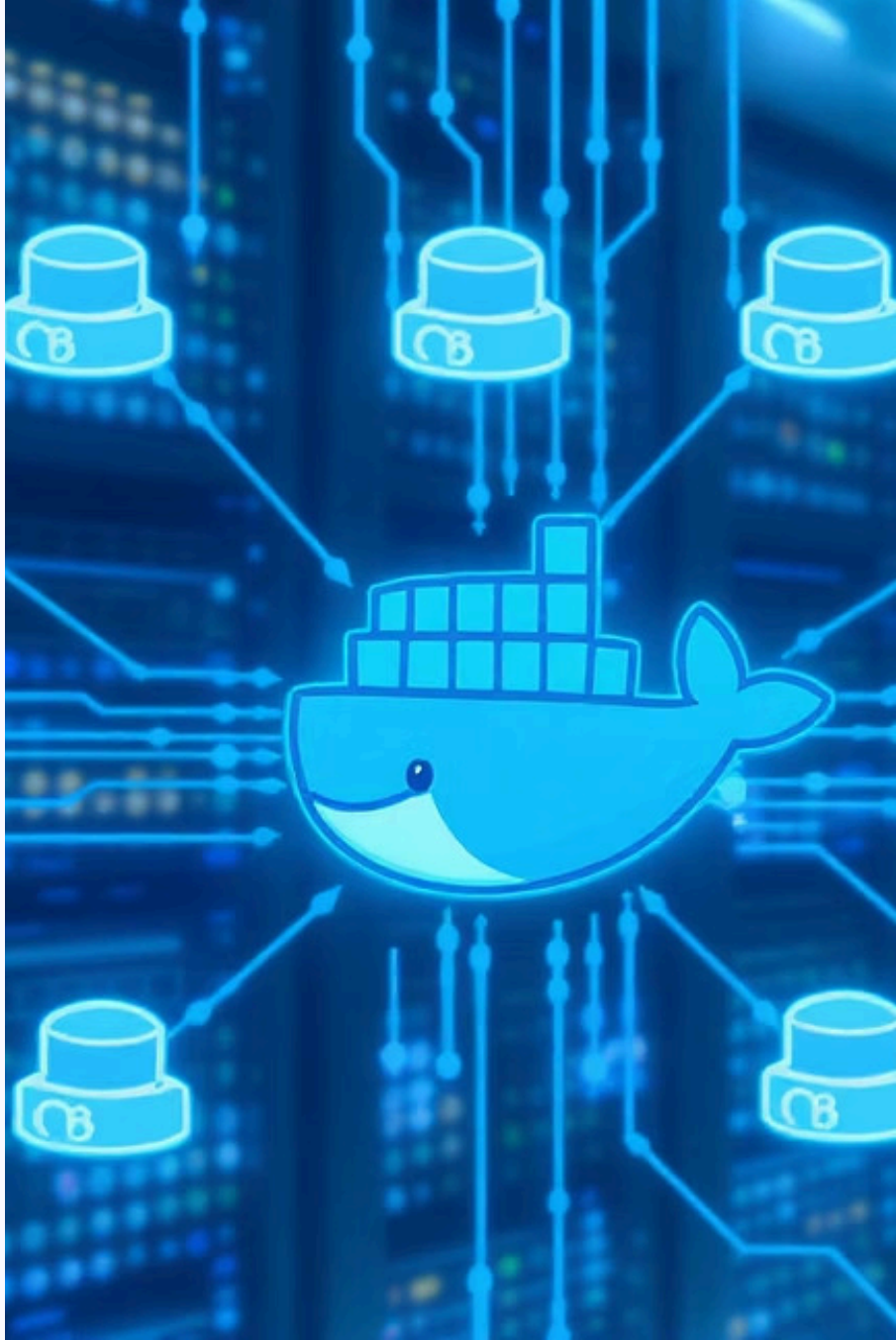
Docker Swarm pour la Résilience

Déploiement d'un Cluster Swarm pour la Continuité d'Activité

Une solution robuste pour garantir la haute disponibilité de vos applications conteneurisées dans un environnement d'entreprise



Objectif du Projet

Notre mission est de concevoir une infrastructure Docker Swarm résiliente intégrant des plans de continuité d'activité (PCA) et de reprise d'activité (PRA).

Cette architecture s'appuie sur des **machines virtuelles Debian** pour garantir la **disponibilité permanente des services numériques critiques**, même en cas de défaillance d'un nœud.

Ce système permet d'assurer:

- La redondance des services
- L'équilibrage de charge automatique
- La récupération après incident
- La persistance des données

L'architecture garantit que les applications critiques restent opérationnelles même lors d'incidents majeurs, assurant ainsi la continuité des services.

Architecture du Cluster

Manager (192.168.40.148)

Le cerveau du cluster qui:

- Orchestre les déploiements
- Surveille l'état des nœuds
- Distribue les tâches aux workers
- Maintient l'état souhaité du système

Worker 1 (192.168.40.151)

Nœud d'exécution principal qui:

- Exécute les conteneurs assignés
- Rapporte son état au manager
- Gère les ressources locales

Worker 2 (192.168.40.146)

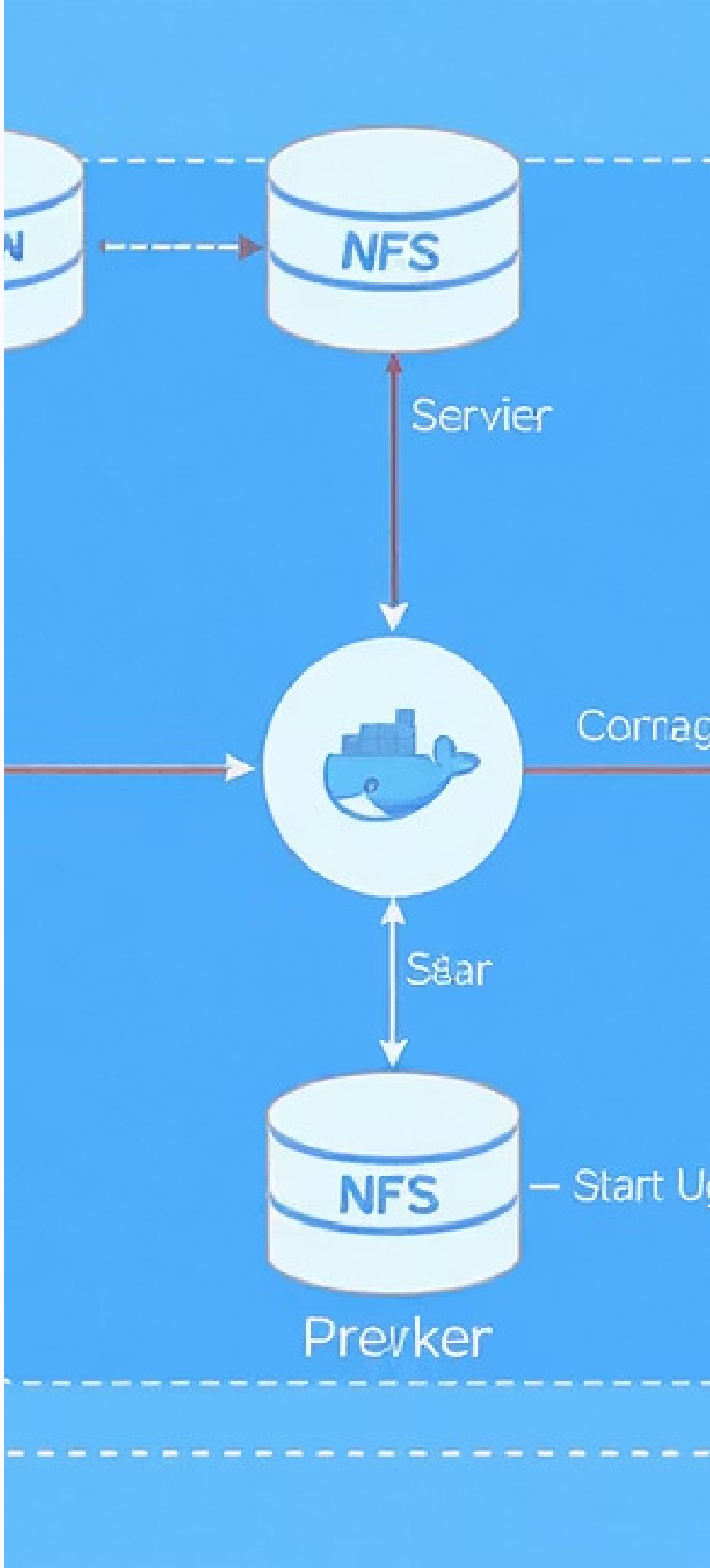
Nœud de redondance qui:

- Assure la haute disponibilité
- Prend le relais en cas de défaillance
- Participe à l'équilibrage de charge

Serveur NFS (192.168.40.147)

Stockage centralisé qui:

- Partage les volumes entre nœuds
- Garantit la persistance des données
- Facilite les sauvegardes centralisées



Services Conteneurisés Déployés

1

MariaDB en HA

Base de données hautement disponible avec:

- Réplication multi-nœuds
- Basculement automatique
- Persistance via volumes NFS

PHP

Serveur d'application qui:

- Héberge le code métier
- S'interface avec MariaDB
- Fournit l'API backend

Nginx

Serveur web et proxy inverse assurant:

- Point d'entrée HTTP unique
- Équilibrage de charge
- Mise en cache et compression

Registry Interne

Dépôt privé d'images Docker permettant:

- Stockage local des images personnalisées
- Indépendance des registres externes
- Contrôle d'accès sécurisé

VSCode Server

Environnement de développement:

- Accès distant au code
- Intégration au cluster
- Édition en temps réel

Cette architecture en couches garantit isolation, scalabilité et résilience pour tous vos services critiques.

Missions de l'Étudiant: Cas Concrets

1

Initialisation du Cluster Swarm

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

Sur les workers:

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

Résolution manuelle des conflits de ports si nécessaire.

2

Déploiement avec Docker Stack

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

Adaptation des fichiers de configuration pour la distribution multi-nœuds des services.

1

Stockage Partagé NFS

```
echo "192.168.40.147:/srv/docker-share /mnt/nfs nfs defaults 0 0" >> /etc/fstab
```

Configuration des points de montage et des droits d'accès pour les volumes partagés.

2

Tests de Reprise d'Activité

```
docker service rm web
docker service ps web
```

Validation du redéploiement automatique et observation du temps de reprise.

3

Sécurisation

```
docker secret create registry_passwd -
```

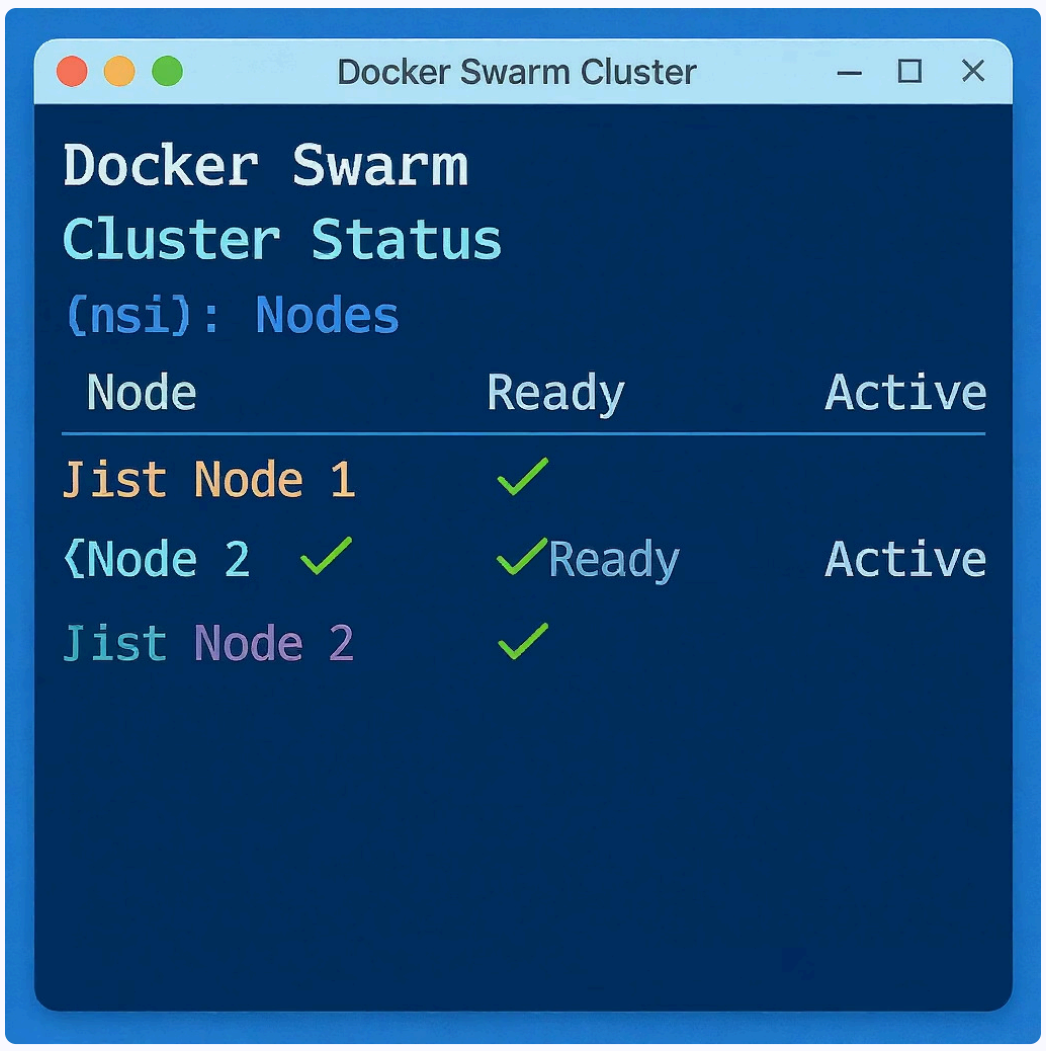
Mise en place d'authentification pour le registre interne et restriction des accès réseau.

Démonstration et Guide Pratique

🎥 Présentation en direct des composants clés du système

Vérification du Cluster

```
docker node ls
```



Simulation de Panne

```
Verifier les services en route:
sudo docker service ls
```

```
Forcer un rééquilibrage, mettre temporairement le Manager en drain :

sudo docker node update --availability drain Manager
```

```
Observation de la reprise automatique via:

sudo docker service ps moncluster_phpmyadmin
```

```
Puis le remettre en active :

sudo docker node update --availability active Manager
```

Structure de la Stack Déployée

- MariaDB: base de données avec persistance
- PHP: serveur applicatif avec PDO activé
- Nginx: serveur web et équilibreur de charge
- phpMyAdmin: interface d'administration

Persistance des Données

Volumes NFS partagés entre tous les nœuds:

```
volumes:
  php-volume:
    driver: local
    driver_opts:
      type: "nfs"
      device: ":/srv/docker-share/php"
```

Accès à l'Environnement de Développement

VSCode Server accessible via: http://192.168.40.148:PORT

Synthèse : L'Intérêt Crucial d'un Registry Local

Performance Accrue

Téléchargements d'images ultra-rapides sur votre réseau interne, optimisant les déploiements.

Sécurité Maîtrisée

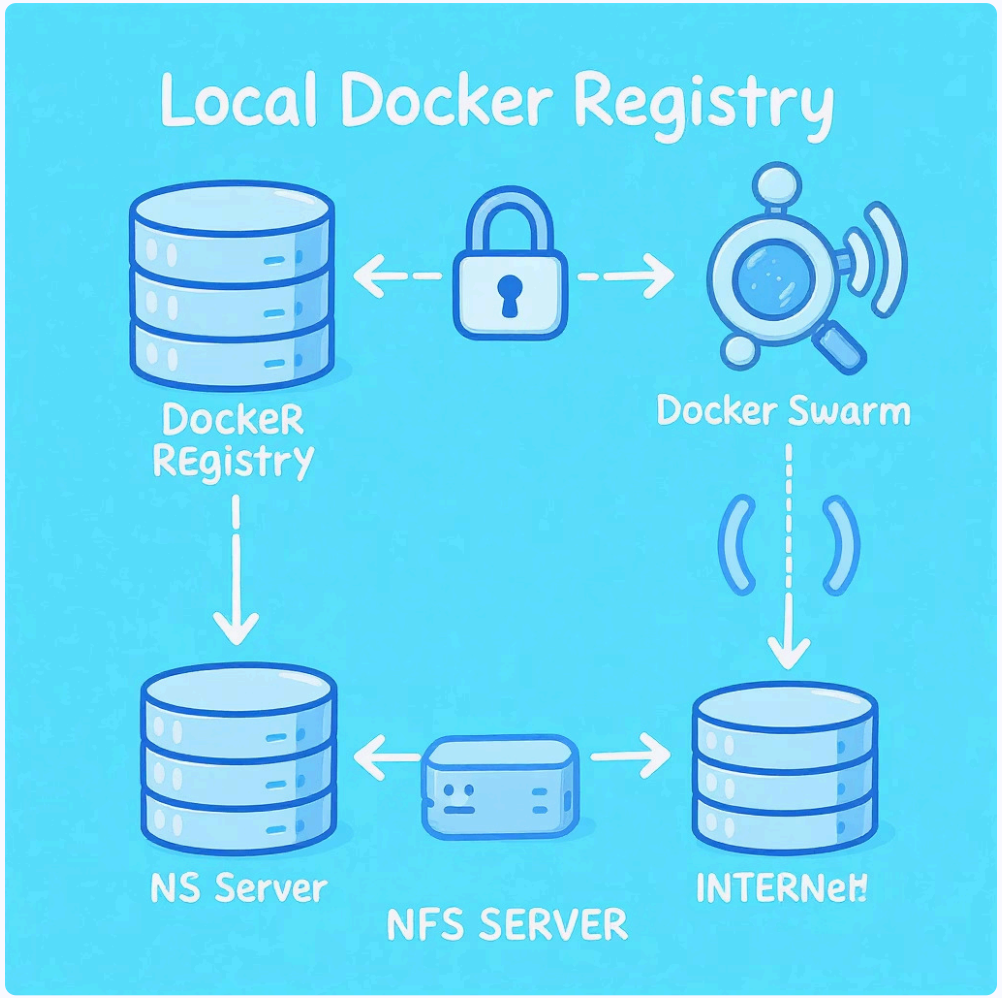
Contrôle total sur vos images, avec authentification personnalisée et scans de vulnérabilités.

Indépendance Opérationnelle

Accès garanti à vos images même sans connectivité externe, essentiel pour les environnements isolés.

Conformité Facilitée

Respect des réglementations internes et des exigences de traçabilité des images.



Un registre Docker local est essentiel pour une infrastructure Swarm robuste, offrant des avantages stratégiques qui vont au-delà de la simple gestion d'images.

Vérification du Registry Local

L'objectif est de prouver l'utilisation du registry Docker privé local pour le stockage et le pull des images, sans dépendance externe.

Accessibilité et Contenu

Vérification de l'accès et des images stockées dans le registry.

```
curl http://192.168.40.148:5000/v2/_catalog
# Résultat: {"repositories":["monphp","alpine"]}
```

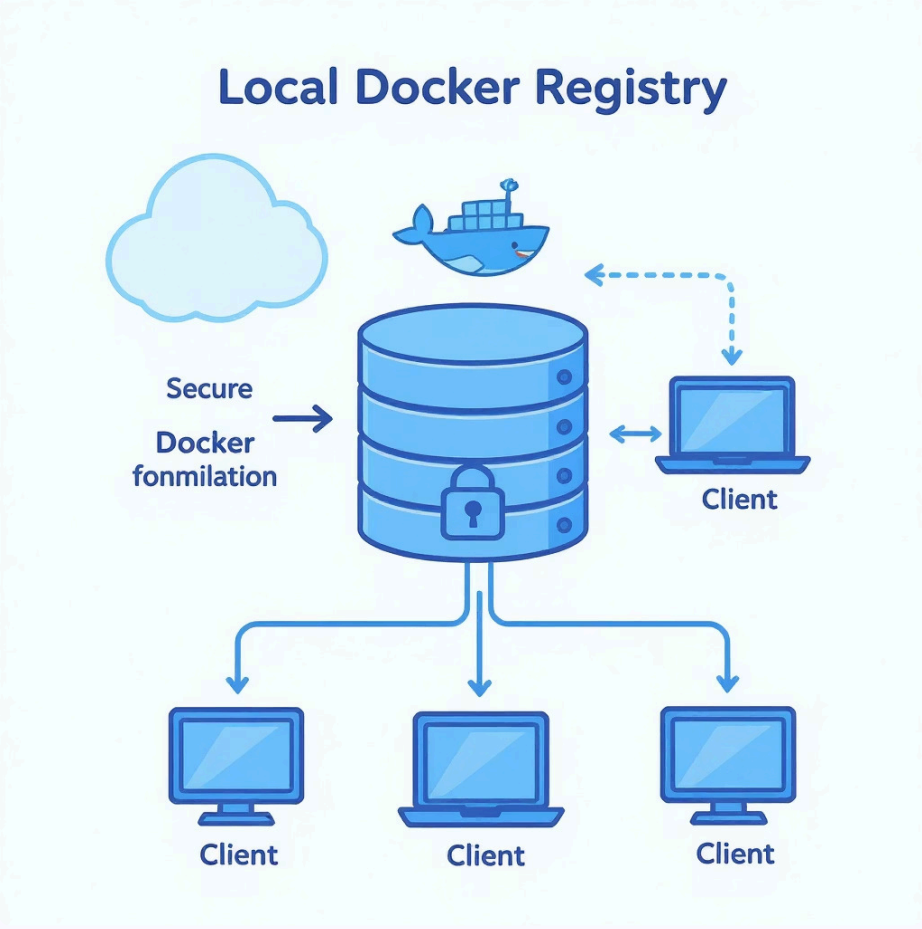
Liste des tags pour une image spécifique.

```
curl http://192.168.40.148:5000/v2/monphp/tags/list
# Résultat: {"name":"monphp","tags":["latest"]}
```

Utilisation des Images

Pull et lancement d'un conteneur à partir du registry interne.

```
docker pull 192.168.40.148:5000/monphp:latest
docker run -d -p 8888:80 192.168.40.148:5000/monphp:latest
```



Ces étapes confirment que le cluster utilise bien son propre registre d'images, assurant autonomie et rapidité pour les déploiements internes.

Extension du Registry Local

Pour exploiter pleinement votre registry local, vous pouvez y publier d'autres images Docker, assurant ainsi une indépendance totale vis-à-vis des dépôts externes et optimisant les déploiements.

Exemple : Publication de Nginx

Sur le Manager (ou un nœud avec nginx en cache) :

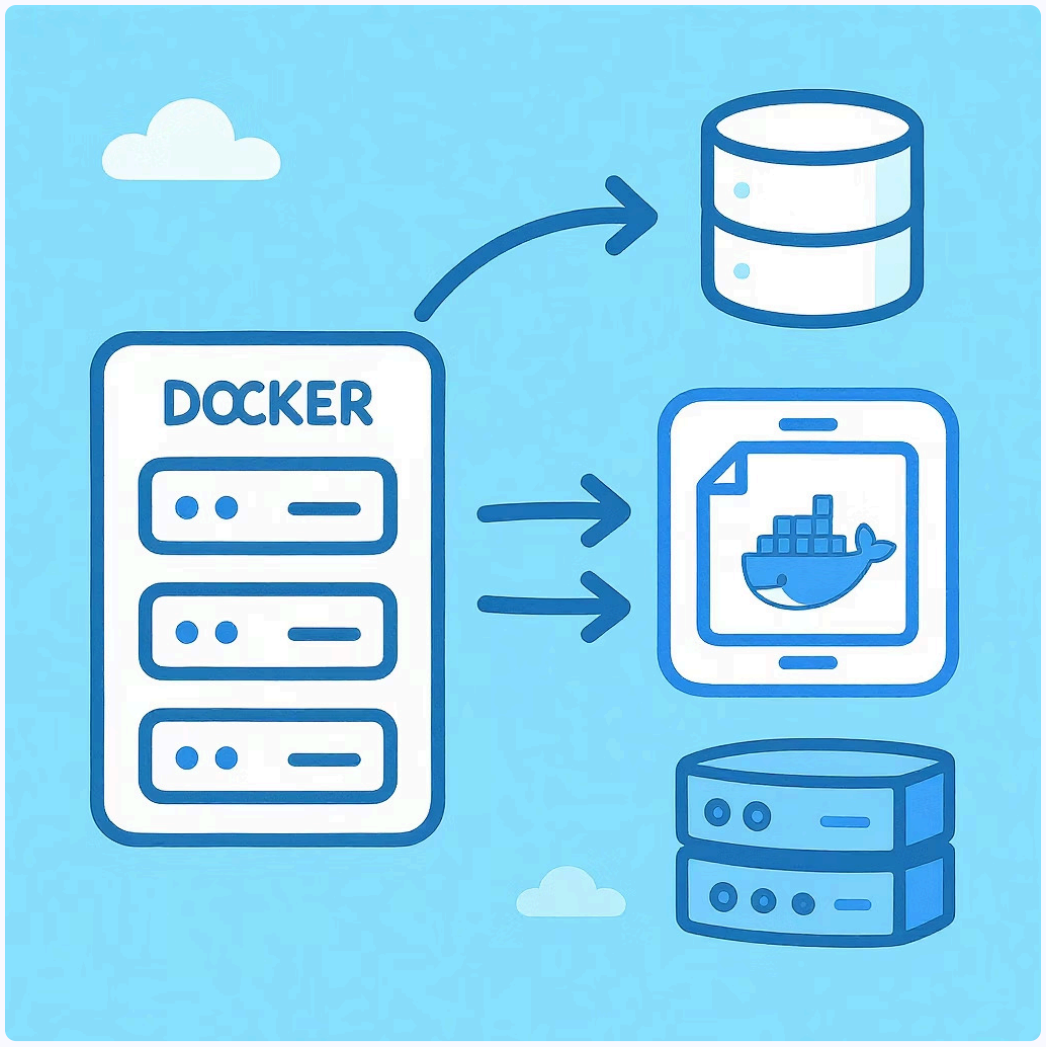
```
sudo docker pull nginx:latest
sudo docker tag nginx:latest 192.168.40.148:5000/nginx
sudo docker push 192.168.40.148:5000/nginx
```

Après publication, mettez à jour votre fichier `docker-compose.yml` en remplaçant l'image officielle par celle de votre registry :

```
image: 192.168.40.148:5000/nginx
```

Puis, redeployez votre stack Docker :

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



Ce processus vous permet de centraliser et de sécuriser toutes vos images Docker au sein de votre infrastructure Swarm.

Images Recommandées pour Votre Registry

Voici une liste d'images que vous pouvez préparer pour votre registry interne, assurant ainsi la résilience de votre environnement :

```
nginx:latest => 192.168.40.148:5000/nginx
mariadb:10.7 =>   192.168.40.148:5000/mariadb:10.7
phpmyadmin/phpmyadmin=> 192.168.40.148:5000/phpmyadmin
codercom/code-server=> 192.168.40.148:5000/code-server
```