Docker Environnement

Le but étant de préparer un environnement facile d'utilisation en simplifiant la gestion des droits.

- Préparation Environnement Docker
- Exemple Setup Postgres Container

Préparation Environnement Docker

Arborescence standard

Nous allons mettre en place une arborescence cohérente pour gérer les différents fichiers nécessaire pour mettre en place des services avec Docker.

Les fichiers docker-composes.yml se trouveront dans le répertoire `/docker` puis dans un sous répertoire correspondant au nom du service. Par exemple, pour le service Nextcloud : `/docker/nextcloud/docker-compose.yml`

Les volumes qui devront être montés pour garder la persistance de certain fichiers par les images docker seront monté dans un dossier `/volumes` puis dans un sous répertoire correspondant au nom du service. Par exemple, pour le service Nextcloud : `/volumes/nextcloud/config`, `/volumes/nextcloud/logs`, etc...

```
# arborescence "infra"
sudo mkdir -p /docker/{nextcloud,immich,jellyfin,vault,bookstack,stump}
sudo mkdir -p /mnt/volumes/{nextcloud,immich,jellyfin,vault,bookstack,stump}

# exemples de sous-dossiers de données (ajustez selon le besoin)
sudo mkdir -p /mnt/volumes/nextcloud/{config,data,db,redis,logs}
sudo mkdir -p /mnt/volumes/jellyfin/{config,cache,media}
sudo mkdir -p /mnt/volumes/bookstack/{config,uploads,db}
sudo mkdir -p /mnt/volumes/stump/{config,cache,media}
sudo mkdir -p /mnt/volumes/vault/{data,config}
sudo mkdir -p /mnt/volumes/immich/{config,photos,thumbs,db,redis}
```

Créer un unique utilisateur/groupe « technique »

Maintenant nous allons créer un utilisateur sans home ni SSH, qui servira d'utilisateur technique pour les services docker.

```
# 10050 est un exemple stable (en dehors de la plage "classique" des UIDs système)
sudo groupadd --system --gid 10050 dockersvc
sudo useradd  --system --uid 10050 --gid 10050 \
  --no-create-home --home-dir /nonexistent \
```

```
  --shell /usr/sbin/nologin dockersvc

# par sécurité, verrouille le mot de passe
sudo passwd -l dockersvc
```

Cela permettra de s'assurer que, tous les droits de lecture et d'écriture, seront uniquement pour cet utilisateur. Ainsi les autres utilisateurs du serveur ne pourront pas lire, modifier ou exécuter les fichiers appartenant à notre utilisateur dockersvc.

Propriété et ACL par défaut sur les volumes

Deux approches possibles pour les permissions :

- **A. Standardiser les UID/GID des conteneurs** sur `PUID=10050` / `PGID=10050` (idéal avec les images linuxserver.io qui supportent ces variables).
- **B. Si une image n'accepte pas PUID/PGID**, nous pourrons soit forcer `user: "10050:10050"` dans Compose (si l'image le supporte), soit laisser l'image avec son UID (ex: `33` pour `www-data`) et **ajouter une ACL** pour que `dockersvc` ait toujours les droits.

Je propose de faire les deux :

- on met `dockersvc` propriétaire des volumes
- **et** on pose des ACL par défaut pour éviter toute surprise si un service réécrit des permissions.

```
# propriété "dockersvc"
sudo chown -R dockersvc:dockersvc /mnt/volumes

# Enlever les droits pour les autres
sudo chmod o= -R /mnt/volumes

# ACL: dockersvc = rwx et droits par défaut hérités (rwX)
sudo apt-get update && sudo apt-get install -y acl
sudo setfacl -R -m u:dockersvc:rwx,g:dockersvc:rwx,o::- /mnt/volumes
sudo setfacl -R -d -m u:dockersvc:rwx,g:dockersvc:rwx,o::- /mnt/volumes
```

Explications :

- `u:dockersvc:rwx` → permet de définir les permissions sur l'utilisateur, donne lecture/écriture/exécution
- `g:dockersvc:rwx` → permet de définir les permissions sur le groupe, donne lecture/écriture/exécution

- `o::-` → permet de définir les permissions sur les autres, en l'occurrence mettre aucun droit
- `-m` → modifier les ACL
- `-d` → ACL **par défaut** → appliquées automatiquement aux **nouveaux fichiers/dossiers**
- `-R` → applique récursivement sur tout `/mnt/volumes`

Ainsi, **un seul compte** (dockersvc) maîtrise tous les dossiers, et les nouveaux fichiers héritent des bons droits.
Pour les images ne supportant pas `user`/`PUID`, on pourra ajouter au besoin des ACL ciblées sur leur UID (ex : `setfacl -m u:33:rwx /mnt/volumes/nextcloud/data`).

Cas particuliers à surveiller

Il existe deux exceptions potentielles :

A. Certaines images Docker imposent un UID spécifique

Exemples :

- L'image officielle **Nextcloud** utilise souvent `www-data` (UID 33)
- L'image **PostgreSQL** utilise souvent `postgres` (UID 999)

Deux options si tu rencontres un problème de droits :

- **Option 1** *(recommandée)* : on continue de laisser `dockersvc:dockersvc` propriétaire mais active des **ACL ciblées** :

```
sudo setfacl -R -m u:33:rwx /mnt/volumes/nextcloud/data
```

- **Option 2** *(moins propre)* : changer le `user:` dans le `docker-compose.yml` pour exécuter le conteneur en `dockersvc`. Ça marche avec les images LinuxServer, mais pas toujours avec les images officielles.

B. Bases de données sensibles (Postgres, MariaDB)

Pour MariaDB/Postgres, il vaut mieux **laisser les dossiers** `dockersvc:dockersvc` mais **ne jamais les exécuter en root**.
On s'assure que le conteneur tourne bien avec un UID compatible (souvent `999` ou `1001`).
Si ce n'est pas possible, alors on ajoutera une ACL spécifique.

Exemple Setup Postgres Container

Nous allons mettre en place le servie Postgres pour la première fois. Voici les démarches à suivre pour que le nouveau service fonctionne avec notre environnement :

Etape 1 : Création de la structure des volumes

```
sudo mkdir -p /mnt/volumes/postgres/{data,logs,conf,init}
sudo chown -R dockersvc:dockersvc /mnt/volumes/postgres
sudo setfacl -R -m u:dockersvc:rwx,g:dockersvc:rwx /mnt/volumes/postgres
sudo setfacl -R -d -m u:dockersvc:rwx,g:dockersvc:rwx /mnt/volumes/postgres
```

Cette arborescence permet :

- **data/** → stockage des bases
- **logs/** → logs PostgreSQL hors conteneur
- **conf/** → fichiers de configuration personnalisés (`postgresql.conf`, `pg_hba.conf`)
- **init/** → scripts SQL ou shell exécutés au premier lancement

Etape 2 : Création du `docker-compose.yml` PostgreSQL

```
version: "3.9"
services:
  postgres:
    image: postgres:17-alpine
    container_name: postgres
    restart: unless-stopped
    user: ${PUID}:${PGID}
    networks:
      - bddnetwork
    ports:
      - 5432:5432
    volumes:
      - ${VOLROOT}/postgres/data:/var/lib/postgresql/data
      - ${VOLROOT}/postgres/logs:/var/log/postgresql
      - ${VOLROOT}/postgres/conf:/etc/postgresql/custom
      - ${VOLROOT}/postgres/init:/docker-entrypoint-initdb.d
```

```
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=OYsA4X8N0xC0AN
      - TZ=${TZ}
    healthcheck:
      test:
        - CMD-SHELL
        - pg_isready -U ${POSTGRES_USER}
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
  postgres: null
networks:
  bddnetwork:
    external: true
```

Créer le network

Notre stack docker compose utilise un network "bddnetwork" externe. Pour que l'exécution docker compose fonctionne il est nécessaire de créer le network avant de la démarrer :

```
sudo docker network create bddnetwork
```

Etape 3 : Démarrer et Vérifier les UID/GID

Pour vérifier les UID/GID après le premier démarrage :

```
docker exec -it postgres id
```

Résultat attendu :

```
uid=10050(dockersvc) gid=10050(dockersvc) groups=10050(dockersvc)
```

Si l'image officielle PostgreSQL refuse `user: "${PUID}:${PGID}"`, alors **on garde dockersvc propriétaire des volumes** mais **on lance le conteneur sous avec l'utilisateur par défaut mais on va ajouter des ACL spécifique.**

Pour ajouter une ACL spécifique :

```
sudo setfacl -R -m u:999:rwx /mnt/volumes/postgres
```

Tester la connexion à la base de donnée

Se connecter dans le conteneur :

```
sudo docker exec -it postgres bash
```

Puis se connecter à la BDD avec l'user et le nom de la base de donnée définit dans la stack docker compose :

```
psql -U <username> -d <dbname>
```