Chapitre 3 - La stack

Docker Compose

Introduction

Docker Compose est une commande docker permettant de définir et de gérer des applications multi-conteneurs.

La commande utilise un fichier YAML (`docker-compose.yml)` pour décrire les services, les réseaux, les volumes, et les dépendances entre les conteneurs.

Ci-dessous l'exemple d'un fichier `docker-compose.yml`. Il s'agit d'un projet web avec une API Node.js, une base de données PostgreSQL, et un reverse proxy Nginx.

```
services:
  nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - web

  web:
    build: ./web
    environment:
      - NODE_ENV=production
    depends_on:
      - db

  db:
    image: postgres
    environment:
      - POSTGRES_PASSWORD=secret
    volumes:
      - pgdata:/var/lib/postgresql/data
```

```
volumes:
  pgdata:
```

La "traduction" en ligne de commandes `docker` ressemblerait à ceci :

```
# création du volume pour PostgreSQL
docker volume create pgdata

# création de l'image pour le serveur web
docker build -t web-image ./web

# lancement des conteneurs
docker run -d --name db -e POSTGRES_PASSWORD=secret -v pgdata:/var/lib/postgresql/data
postgres
docker run -d --name web -e NODE_ENV=production web-image
docker run -d --name nginx --network frontnet -p 80:80 -v ./nginx.conf:/etc/nginx/nginx.conf
nginx
```

Orchestration

L'orchestration d'une stack docker compose s'effectue avec la commande `docker compose`.

```
docker compose up # démarre les conteneurs
docker compose down # arrête et supprime les conteneurs, réseaux et volumes
docker compose build # construit les images des conteneurs
docker compose logs # affiche les logs des conteneurs
```

Syntaxe

Services

La clé `services` définit les conteneurs à lancer. Chaque service représente un conteneur Docker.

```
services:
  web:
    image: nginx:latest
```

Pour chaque service, les instructions suivantes sont disponibles.

Instruction	Description	Exemple
`image`	Image de départ	`image: ubuntu:24:04`
`container_name`	Nom du conteneur	`container_name: mon_web`
`ports`	Mappe les ports de la machine hôte vers le conteneur	`ports:` `  - 8080:80`
`volumes`	Monte des volumes dans le conteneur. Volume mappés ou managés.	`volumes:` `  - ./mon_dossier:/app`
`environment`	Variables d'environnement	`environment:` `  - DEBUG=1`
`command`	Commande au démarrage du conteneur	`command: ["python", "app.py"]`
`depends_on`	Ordre de création des conteneurs	`depends_on: ["db"]`
`restart`	Politique de redémarrage (`no`, `always`, `on-failure`, `unless-stopped`)	`restart: always`
`networks`	Réseaux du conteneur	`networks:` `  - frontend`

Volumes

La clé `volumes` permet d'automatiser la création et le montage des volumes managés.

```
volumes:
  db_data # volume sans paramètres spécifiques
  external_volume:
    external: true
    name: mon_volume_externe
```

Instruction	Description	Exemple
`external`	Indique si le volume a été créé en dehors de la stack docker compose.	`external: true`
`name`	Nom du volume, utile avec `external`	`name: mon_volume_externe`

Networks

La clé `networks` définit les différents réseaux docker de la stack.

```
networks:
  reseau1:
    driver: bridge
  reseau2:
    driver: bridge
    ipam:
      config:
        - subnet: "172.16.0.0/16"
  reseau_externe:
    external: true
    name: mon_reseau_externe
```

Instruction	Description	Exemple
`driver`	Type de réseau (`bridge`, `host`, etc.)	`driver: bridge`
`external`	Indique si le réseau a été créé en dehors de la stack docker compose.	`external: true`
`name`	Nom du réseau, utile avec `external`	`name: mon_reseau_externe`
`ipam`	Configuration IP du réseau	Voir exemple ci-dessus.

Docker Compose : Exemple avec MariaDB

MariaDB et Adminer

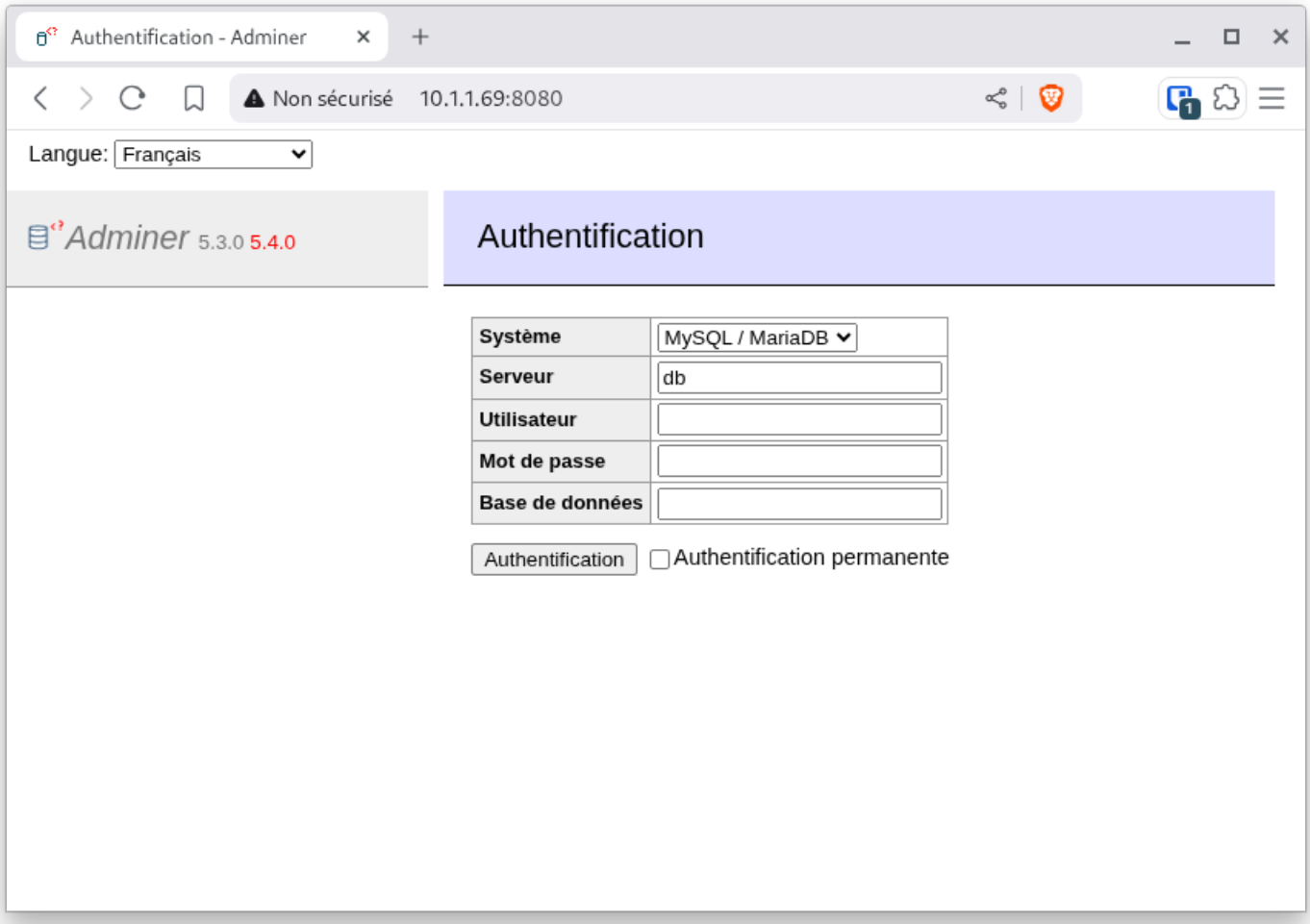
L'image officielle de `MariaDB` indique dans sa documentation un fichier `docker-compose.yml` pour le faire fonctionner avec l'interface Adminer.

https://hub.docker.com/_/mariadb#-via-docker-compose

```
# Use root/example as user/password credentials

services:

  db:
    image: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

Exécution

```
debian@debian:~/dc$ cat docker-compose.yml
version: '3.8'
services:

  db:
    image: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
debian@debian:~/dc$ docker compose up
WARN[0000] /home/debian/dc/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 3/3
 ✔ Network dc_default      Created
 ✔ Container dc-adminer-1  Created
 ✔ Container dc-db-1       Created
Attaching to adminer-1, db-1
w Enable Watch
w Enable Watch
w Enable Watch
```




Sélectionner la base de donné
Non sécurisé
10.1.1.69:8080/?server=db&username=root
Langue: Français
MariaDB » db
root
Déconnexion
Adminer 5.3.0 5.4.0
Sélectionner la base de données
BD:
Requête SQL
Importer
Exporter
Créer une base de données
Privilèges
Liste des processus
Variables
Statut
Version de MariaDB : 12.0.2-MariaDB-ubu2404 via l'extension PHP MySQLi
Authentifié en tant que : root@172.18.0.2
Base de données - Rafraîchir
Interclassement
Tables
Taille - Calcul
information_schema
utf8mb3_general_ci
mysql
utf8mb4_uca1400_ai_ci
performance_schema
utf8mb3_general_ci
sys
utf8mb3_general_ci
Sélectionnée(s) (0)
Supprimer
Loaded plugins

Docker Swarm

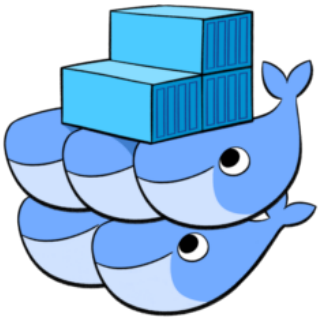
Introduction

Docker Swarm est une surcouche native de Docker pour **orchestrer** un cluster de machines exécutant des conteneurs Docker.

Contrairement à une **exécution locale** de conteneurs, Swarm permet de déployer des conteneurs avec quelques avantages :

- Distribution automatique
- Haute disponibilité
- Scaling automatique
- Redondances
- Etc.

> Docker Swarm transforme un ensemble de nœuds Docker en un **cluster** unique.



Chaque nœud peut être un **manager** (coordonne le cluster) ou un **worker** (exécute les tâches).
Les services sont définis avec des règles de réplication, de placement et de mise à jour.

Exemple

- Nous partons d'une application web hébergée sur **Nginx** avec une exigence de **haute disponibilité** et de **répartition de charge**.

- Nous disposons de plusieurs **instances Docker** (machines hôtes) et nous souhaitons que l'application soit toujours accessible, même si une machine tombe en panne.

```
# Sur la machine principale (noeud principal)
docker swarm init

# Sur chaque noeud
docker swarm join IP_NOEUD_PRINCIPAL:2377

# Création du service avec 3 instances
docker service create --name web --replicas 3 -p 80:80 nginx

# Vérifier le déploiement
docker service ls
docker service ps web
```

> Le service Nginx est maintenant réparti sur plusieurs nœuds, avec équilibrage de charge.

Différence avec Docker Compose

Docker Compose est principalement utilisé pour le développement local ou le déploiement d'application sur une machine unique.
L'ensemble des services, réseaux et volumes nécessaires sont détaillés dans le fichier `docker-compose.yml`.

Docker Swarm est conçu pour le déploiement en production sur plusieurs machines.
Il offre des fonctionnalités d'orchestration qui ne sont pas disponibles avec **Docker Compose**.

Fonctionnalité	Docker Compose	Docker Swarm
Scope	Machine locale	Cluster
Orchestration	Basique	Avancée
Scaling	Non	Oui

Fonctionnalité	Docker Compose	Docker Swarm
Haute dispo	Non	Oui
Load balancing	Non	Oui

Docker Secret

Docker Swarm propose une fonctionnalité **Docker Secrets** pour stocker et distribuer les données sensibles de manière sécurisée aux services qui en ont besoin.

Exemple :

```
# Création d'un secret
printf "my super secret password" | docker secret create my_secret -

# Utilisation avec un service
docker service create --name app --secret my_secret alpine:latest sh -c "cat
/run/secrets/my_secret"
```

Dans cet exemple, le conteneur `alpine` accède au mot de passe via le fichier `/run/secrets/my_secret`.

Ce fichier est monté automatiquement par **Swarm** et n'est pas visible en dehors du conteneur.

> La fonctionnalité Docker Secret n'est disponible qu'avec Docker Swarm.
> Les volumes sont le meilleur moyen de gérer des informations sensible en dehors d'un cluster Docker Swarm.