MODULE 4 : ORCHESTRATION DANS DOCKER

I - Docker Compose

Concepts de base

Docker Compose repose sur plusieurs objets fondamentaux pour orchestrer des applications conteneurisées :

- **Services** : Un service correspond à un conteneur Docker défini dans le fichier YAML. Chaque service décrit un aspect de l'application, comme une base de données, une API ou une interface utilisateur.
- **Volumes** : Les volumes sont des espaces de stockage partagés entre conteneurs pour persister les données au-delà du cycle de vie des conteneurs.
- **Réseaux** : Les réseaux permettent aux services de communiquer entre eux. Docker Compose crée automatiquement un réseau par défaut, mais il est aussi possible de configurer des réseaux personnalisés.
- **Fichier docker-compose.yml** : Ce fichier est le cœur de Docker Compose. Il décrit la configuration des services, les volumes à monter, les réseaux à utiliser et d'autres options de déploiement.

Structure d'un fichier Docker Compose

```
version: '3'
services:
  service1:
    image: nom_de_l_image
    ports:
      - "port_hôte:port_conteneur"
    environment:
      - VARIABLE_1=valeur_1
    volumes:
      - /chemin/hote:/chemin/conteneur
    depends_on:
      - service2
  service2:
    # Configuration du deuxième service

networks:
  mynetwork:
    driver: bridge
```

Configuration des services

Les images

```
services:
  my-service:
    image: ubuntu:latest
  web:
```

```
    build: /path/to/dockerfile/
  db:
    build: https://gitlab.com/srobert/my-project.git
```

Avec un tag :

```
  web:
    build: https://gitlab.com/srobert/my-project.git
    image: my-project-image
```

La gestion des dépendances entre services

```
version: '3'
services:
  app:
    image: my-node-app
    ports:
      - "80:80"
    depends_on:
      - db
  db:
    image: postgres:latest
    environment:
      POSTGRES_PASSWORD: mysecretpassword
```

Configuration des réseaux

```
services:
  service-1:
    image: alpine:latest
    networks:
      - network-1
  service-2:
    image: alpine:latest
    networks:
      - network-1
  service-3:
    image: alpine:latest
    networks:
      - network-2

networks:
  network-1: {}
  network-2: {}
```

Avec driver :

```
networks:
  network-1:
    driver: bridge
```

Exposition de ports

```
services:
  service-1:
    image: alpine:latest
    ports:
      - "80:80"
  service-2:
    image: alpine:latest
    ports:
      - "8080:3000"
  service-3:
    image: alpine:latest
    ports:
      - "8081:3000"
```

Les volumes

```
services:
  service-1:
    image: alpine:latest
    volumes:
      - global-volume:/my-volumes/named-global-volume
volumes:
  global-volume:
```

Les variables d'environnement

```
services:
  database:
    image: "postgres:${POSTGRES_VERSION}"
    environment:
      DB: mydb
      USER: "${USER}"
```

La gestion des secrets

```
version: '3'
services:
  myapp:
    image: myapp:latest
    secrets:
      - my_secret
secrets:
  my_secret:
    file: ./my_secret.txt
```

Les secrets sont montés dans /run/secrets/ et sont accessibles uniquement aux conteneurs concernés.

II - Docker Swarm

1. Initialisation du Swarm
docker swarm init
Crée un cluster Swarm et transforme la machine en manager.
2. Ajouter des nœuds au Swarm
docker swarm join-token manager
docker swarm join-token worker
Puis sur une autre machine :
docker swarm join --token <TOKEN> <IP_MANAGER>:2377
3. Vérifier les nœuds du cluster
docker node ls
Liste les nœuds du cluster (uniquement pour le manager).
4. Déployer une stack
docker stack deploy -c docker-compose.yml nom_de_la_stack
Déploie les services décrits dans le fichier Compose.
5. Afficher les services d'une stack
docker stack services nom_de_la_stack
6. Visualiser les conteneurs actifs
docker stack ps nom_de_la_stack
docker service ps nom_service
7. Mettre à jour un service
docker service update --image nouvelle_image nom_du_service
8. Répliquer un service
docker service scale nom_service=3
Crée 3 instances du service.
9. Supprimer une stack
docker stack rm nom_de_la_stack
10. Quitter un nœud du Swarm
docker swarm leave
ou
docker swarm leave --force
Exemple complet de déploiement avec réplication :
# 1. Initialiser le cluster
$ docker swarm init

# 2. Déployer une stack avec réplication
$ cat > docker-compose.yml <<EOF
version: '3'
services:

```
 web:
  image: nginx:latest
  ports:
   - "8080:80"
  deploy:
   replicas: 3
   restart_policy:
    condition: on-failure
EOF

$ docker stack deploy -c docker-compose.yml mystack

# 3. Vérifier les services
$ docker stack services mystack
```

Cette approche permet d'assurer la haute disponibilité, la mise à l'échelle horizontale, et une gestion centralisée de vos conteneurs dans un cluster Docker.