TP Conteneur
Orchestration – partie 3

5 avril 2024

L'objectif de ce TP est d'utiliser les fonctionnalités de déploiement permises par Docker. Les outils vus aideront à la relance automatique des services suite à une panne, la mise à jour d'une service sans interruption de service ou le passage à l'échelle via la réplication en cas de hausse de trafic. On s'appuiera sur Docker Compose et Docker Swarm.

Nous allons nous focaliser par la suite sur une application classique, WordPress, pour démontrer comment Docker facilite son déploiement et en particulier son passage à l'échelle. Bien que l'on considère uniquement WordPress (le CMS le plus commun sur le Web), l'architecture utilisée illustre bien la plupart des applications webs.

Il peut être nécessaire de nettoyer le système utilisé.

1 Docker Compose

1.1 Découverte de l'image `wordpress`

On va commencer par expérimenter l'utilisation de l'image de base `wordpress`. En ligne de commande, lancer l'exécution d'un conteneur en mettant seulement en place la redirection des ports pour avoir accès au service sur le port 8080 de l'hôte.

Pour vérifier le bon fonctionnement, commencer la configuration en vous connectant au service. Quel serveur web est utilisé dans ce conteneur ? Avec quel système de gestion de bases de données ?

1.2 Composition avec deux conteneurs

On va repartir de l'exemple du cours avec un conteneur pour WordPress et un autre pour MySQL décrits dans un fichier `docker-compose.yml` (disponible dans l'archive liée au sujet de TP).

La documentation du format de fichier se trouvera sur https://docs.docker.com/compose/compose-file/. On utilisera principalement la commande : `docker-compose up`. Tester le bon fonctionnement en vous connectant au site web. Redémarrer l'ensemble pour vérifier la persistance des données. Parcourir la documentation pour trouver la section qui explique comment rediriger le port 8080 de la machine hôte au service `wordpress`.

À ce stade, vous devriez avoir observé que la composition facilite le déploiement d'une architecture minimaliste mais souffre de plusieurs limitations :

— Le mot de passe apparaît directement dans le fichier de configuration.
— La base de données prend beaucoup de temps à être initialisée et le serveur web peut afficher un message d'erreur à ce sujet malgré la dépendance `depends_on`.

1.3 Variables d'environnement

On va commencer par créer un fichier (à accès restreint dans l'idéal) pour mettre le mot de passe dans une variable d'environnement :

```
echo "PASSWORD=1234" > password.env
```

Il suffit ensuite de remplacer les mots de passe dans le fichier YAML par `${PASSWORD}`.

Vérifier que le déploiement fonctionne toujours en vous connectant à WordPress avec cette méthode. Il faudra préciser le fichier avec la variable d'environnement avec `--env-file password.env` (à positionner avant le `up`).

1.4 Contrôle de la dépendance

Pour être sûr que WordPress ne s'exécute que lorsque la base de données est disponible, on va rajouter une condition `service_healthy` dans l'élément `depends_on`.

Il faudra alors spécifier le test qui garantit la disponibilité de la base de données. Cela se fait avec l'élément `healthcheck` au niveau du service (voir https://docs.docker.com/compose/compose-file/05-services/#healthcheck). La commande suivante est plus précise que celle vue en cours pour vérifier que MySQL est disponible est :

```
mysqladmin ping -h localhost -u $$MYSQL_USER -p$$MYSQL_PASSWORD
```

2 Docker Swarm

L'objectif est désormais de déployer l'application sur plusieurs machines : la base de données dans un unique conteneur et le serveur web répliqué 2 fois sur 2 machines distinctes. Il faudra utiliser 2 machines distinctes pour pouvoir interpréter facilement les observations. On utilisera un *swarm* (ensemble de machines interconnectées avec Docker) mais on aurait pu choisir une solution alternative comme Kubernetes.

On peut commencer par nettoyer la configuration existante (à refaire en fin de séance également) :

```
docker swarm leave --force
```

2.1 Création du *swarm*

Les commandes de création de *swarm* et de rajout de machines sont `docker swarm init` et `docker swarm join` (commandes à compléter par rapport aux messages obtenus en ligne de commande).

On n'aura besoin que d'un seul *manager*, mais de 2 *workers* (un *manager* étant également un *worker* par défaut).

Identifier les différentes machines du *swarm* grâce à `docker node ls` et vérifier leurs statuts grâce à `docker node inspect`.

2.2 `docker service`

L'utilisation manuelle (en ligne de commande) d'un *swarm* passe par les commandes de type `docker service` (on verra les sous-commandes `create`, `ls`, `ps`, `logs`, `scale` et `rm`). La création d'un service est similaire à l'utilisation basique de Docker en ligne de commande : on exécute un conteneur basé sur une image et une configuration particulière (via les options). La différence se situe au niveau de son déploiement sur un *swarm* avec une répartition automatique sur les *workers*. Par ailleurs, le conteneur peut être répliqué, fonctionnalité requise pour le passage à l'échelle et la tolérance aux pannes.

Adapter la création suivante d'un conteneur qui *ping* le site `dept-info.univ-fcomte.fr` en service qui réalise la même chose.

```
docker container run bash ping dept-info.univ-fcomte.fr
```

Tester les commandes de diagnostique pour vérifier le bon fonctionnement du service.

Répliquer le service 3 fois. Vérifier l'état du service et du *swarm*, puis supprimer le service.

2.3 `docker stack`

L'équivalent de l'automatisation qu'apporte Docker Compose s'appelle un *stack* dans le cas d'un *swarm*. Les sous-commandes que l'on testera sont `deploy`, `ls`, `ps`, `services` et `rm`.

Il faudra adapter le ficher YAML en retirant la partie `healthcheck`, en remettant le mot de passe en clair et en rajoutant une section pour la réplication du serveur web :

```
wordpress:
  deploy:
    replicas: 2
```

Vérifier le bon fonctionnement du site web (depuis chaque machine) et la présence de 3 conteneurs en cours d'exécution.

3 Conclusion et finalisation

L'infrastructure finale consiste donc en une base de données unique et plusieurs serveurs webs WordPress. L'équilibrage de charge est réalisé par Docker Swarm : les clients se connectent à n'importe quel membre du *swarm* et chaque requête est redirigée vers l'un des réplicas. Dans une architecture plus avancée, l'équilibrage de charge pourrait être remplacé ou complété par un *reverse proxy* comme Nginx ou HAProxy. En rajoutant une couche supplémentaire entre les clients et le *back-end*, ceux-ci auraient pour but principal de répartir les requêtes aux serveurs webs.

Ce TP se finit par le nettoyage du système. Quelles sont les commandes pour cela ?