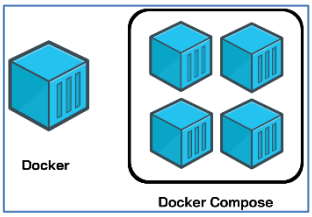
COMPRENDRE LA STRUCTURE D'UN FICHIER DE TYPE YAML



Docker Compose est particulièrement utile lorsque vous travaillez sur des applications qui **comprennent plusieurs conteneurs**, comme une application Web qui utilise un conteneur pour le serveur Web et un autre pour la base de données. Au lieu de gérer individuellement chaque conteneur, vous pouvez utiliser Docker Compose pour gérer l'ensemble de l'application d'un seul coup.

Docker Compose est une fonctionnalité permettant d'orchestrer plusieurs conteneurs qui doivent travailler ensemble. Pour ce faire, on crée un fichier « YAML » (Yet Another Markup Language) à l'intérieur duquel on spécifie les configurations nécessaires à chaque service. Grâce à Docker Compose, tous les conteneurs dont on a besoin pourront être exécutés à l'aide d'une seule commande.

Qu'est-ce qu'un fichier « yaml » ?

Les fichiers avec l'extension « .yml » sont des fichiers « YAML ». Ils **permettent de structurer les données**. C'est un équivalent du XML ou du JSON. Mais le YAML est plus lisible pour un humain. On hiérarchise les données grâce à la tabulation.

Le but du docker-compose.yml est de **gérer correctement les conteneurs en décrivant ce que nous souhaitons faire**.
Normalement, pour lancer un conteneur, il faut saisir une commande qui peut être complexe (le fameux « CLI »). Avec le fichier « docker-compose.yml », on décrit ce que l'on souhaite faire et on lance une seule commande qui exécutera tout ce que nous avons indiqué dans le fichier « docker-compose.yml ».

Exemple de fichier « docker-compose.yml » pour l'installation automatisée du CMS Wordpress :

```yaml
version: '3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
```

1. Définir la version du fichier (en fonction de la version de Docker Compose présente)
2. Définir les services (conteneurs) et les nommer (exemple « db »)
3. Indiquer l'image qui sera utilisée pour le conteneur
4. Indiquer le(les) volume(s) utilisé(s) par le conteneur
5. On peut définir la politique de redémarrage du conteneur
6. Indiquer, si nécessaire, les variables d'environnement

Ici, nous avons la structure du 2ème conteneur qui correspond, cette fois, aux sources du CMS Wordpress avec un lien vers la base de données (argument « depends_on : »).

Dans le cas présent, nous souhaitons conteneuriser Wordpress avec une instance correspondant à la base de données et une instance relative au CMS.

Plusieurs **informations** sont nécessaires pour bien **utiliser** le « docker-compose.yml ». Certaines sont **obligatoires** et d'autres sont **facultatives** car elles dépendent de ce que nous souhaitons déployer.

a) Définition de la **version** du fichier « yml »

```
1 version: '3'
```

L'argument « **version:** » permet de spécifier à Docker Compose quelle version on souhaite utiliser. Ici, la version « 3 » a été indiquée car il s'agit de la version actuellement la plus utilisée. Celle-ci permet de définir la comptabilité de notre fichier avec le moteur Docker installé sur l'hôte. A ce jour, la version la plus récente est la **'3.8'**.

Plus d'informations sur la compatibilité des versions ici.

b) Définition des **services**

```
2 services:
3   db:
```

Ici on « déclare » un service qui correspond à la création d'un conteneur qui sera nommé « db » pour database.

L'ensemble des conteneurs qui doivent être créés doivent être définis sous l'argument « **services:** ». Chaque conteneur commence avec un nom qui lui est propre. Par exemple, ici, le premier conteneur se nommera « db » (il correspondra à la base de données que l'on veut créer).

c) Description du **conteneur**

```
4     image: mysql:5.7
```

Ici on indique quelle est l'image qui servira de base à la « construction » du conteneur. Dans notre cas, l'image « mysql » dans sa version 5.7 sera choisie. Si rien n'est stipulé, la version « latest » (la dernière) sera automatiquement téléchargée.

Ici on a indiqué que le conteneur relatif à la base de données sera créé à partir de l'image « mysql 5.7 ». On aurait pu aussi indiquer par exemple « mariadb:latest ».

d) Déclaration du(des) **volume**(s) qui permettra(ont) de conserver les données

```
5     volumes:
6       - db_data:/var/lib/mysql
```

Le volume Docker « db_data » sera « monté » dans le conteneur dans le dossier « /var/lib/mysql » du conteneur.

Les conteneurs Docker ne conservent pas les données si un volume ou un lecteur attaché de type « bind » n'est pas indiqué. Il est cependant possible d'utiliser **l'argument « volumes: »** qui vous permet de stocker l'ensemble du contenu du dossier /var/lib/mysql dans un disque persistant sur la machine hôte.

Cette description est présente grâce à la ligne « db_data:/var/lib/mysql ». « db_data » est un volume créé par Docker directement qui permet d'écrire les données sur le disque hôte sans spécifier l'emplacement exact.

Vous auriez pu aussi faire un /data/mysql:/var/lib/mysql qui serait aussi fonctionnel (lecteur attaché de type « bind »).

e) Politique de redémarrage du conteneur

```
7       restart: always
```

En indiquant « always » après « restart », on stipule que le conteneur doit automatiquement redémarrer après une erreur inattendue.

Un conteneur étant par définition monoprocessus, s'il rencontre une erreur fatale, il peut être amené à s'arrêter. Dans notre cas, si le serveur MySQL s'arrête, celui-ci redémarrera automatiquement grâce à l'argument « **restart:always** ».

f) Définition des variables d'environnement

L'image MySQL fournie dispose de **plusieurs variables d'environnement** que vous pouvez utiliser. Dans notre cas, nous allons donner au conteneur les valeurs des différents mots de passe et utilisateurs qui doivent exister sur cette base. Quand vous souhaitez donner des variables d'environnement à un conteneur, vous devez utiliser l'argument « **environment:** ».

```
 8       environment:
 9         MYSQL_ROOT_PASSWORD: somewordpress
10         MYSQL_DATABASE: wordpress
11         MYSQL_USER: wordpress
12         MYSQL_PASSWORD: wordpress
```

g) Définition d'un nouveau service (2ème conteneur)

Dans le second service, nous créons un conteneur qui contiendra le nécessaire pour faire fonctionner votre site avec **WordPress**. Cela nous permet d'introduire deux arguments supplémentaires.

```
14    wordpress:
15      depends_on:
16        - db
17      image: wordpress:latest
18      ports:
19        - "8000:80"
20      restart: always
21      environment:
22        WORDPRESS_DB_HOST: db:3306
23        WORDPRESS_DB_USER: wordpress
```

Création du 2ème conteneur nommé « wordpress » qui devra être lié au conteneur « db » contenant la base de données mysql. Ici, un mappage du port 8000 de l'hôte a été indiqué. Cela signifie que pour accéder au CMS Wordpress, il faudra ajouter dans la barre d'adresse « :8000 » pour y accéder (il faut penser à ouvrir ce port sur votre box/routeur).

Le premier argument, « **depends_on:** », permet de créer une **dépendance** entre deux conteneurs. Ainsi, Docker démarrera le service « db » avant de démarrer le service « wordpress ». Ce qui est un comportement souhaitable car WordPress dépend de la base de données pour fonctionner correctement.

Le second argument, « **ports:** », permet de dire à Docker Compose qu'on veut exposer un **port** de notre machine hôte vers notre conteneur et le rendre accessible depuis l'extérieur (pensez à ouvrir ce port dans votre routeur !).

h) Indication du(des) volume(s) utilisé(s)

```
27  volumes:
28    db_data: {}
```

L'argument « volumes: » permet de spécifier le nom du volume Docker qui sera utilisé pour les données persistantes.

CREATION AUTOMATISEE DE L'INFRASTRUCTURE « GLPI »

Lorsque vous utilisez un « docker-compose », vous pouvez, par exemple, créer un dossier sur la machine hôte avec un nom explicite. **Ensuite, vous enregistrerez, dans ce dossier, votre fichier « YAML » qui devra absolument porter le nom de « docker-compose.yml ».**

Une fois le « docker-compose.yml » défini, il faut le lancer avec la commande « **docker-compose up -d** ».

Lors de l'exécution de cette commande, Docker Compose commence par vérifier si nous disposons bien en local des images nécessaires au lancement des stacks. Dans le cas contraire, il les télécharge. Puis il lance les deux conteneurs sur votre système ; votre stack est prête !

Exemple – Création d'un fichier « docker-compose.yml » pour l'installation de GLPI (avec mariaDB)

Dans cet exemple, nous allons mettre en place un serveur SQL MariaDB et l'helpdesk GLPI en version 10.0.6. Sur le docker hub, nous pouvons trouver une multitude d'images et de fichiers « docker-compose.yml ». Ici, nous avons sélectionné l'image « diouxx/glpi » et le fichier « yml » correspondant :

Fichier « docker-compose.yml » :

```
version: "3.2"
services:
# Conteneur MARIADB
  mariadb:
    image: mariadb:latest
    hostname: mariadb
    volumes:
      - /var/lib/mysql:/var/lib/mysql
    env_file:
      - ./mariadb.env
    restart: always
```

Création du 1er conteneur nommé « mariadb » avec l'attachement d'un volume et le lien avec le fichier contenant les variables d'environnement. Le « hostname » mariadb permet de remplacer le hostname par défaut du conteneur qui est « localhost » si rien n'est stipulé.

```
# Conteneur GLPI 10
  glpi:
    image: diouxx/glpi
    container_name: glpi
    hostname: glpi
    ports:
      - "8081:80"
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
      - /var/www/html/glpi/:/var/www/html/glpi
    environment:
      - TIMEZONE=Europe/Paris
    restart: always
```

Création du 2ème conteneur « glpi » basé sur une image du Docker hub avec mappage du port 8081 (on aurait pu choisir un autre port ici) et attachement des volumes utiles à la persistence des données du conteneur.

- Sur la machine Debian, créez un dossier « glpi » (ici nous l'avons créé dans le dossier ~ du root car nous ne sommes pas en production) avec la commande « **mkdir glpi** »

- Dans le dossier « glpi », créez le fichier « **mariadb.env** » qui contient les **variables d'initialisation** de GLPI à l'aide de l'éditeur « nano » :

nano mariadb.env

```
  GNU nano 5.4
MARIADB_ROOT_PASSWORD=diouxx
MARIADB_DATABASE=glpidb
MARIADB_USER=glpi_user
MARIADB_PASSWORD=glpi
```

Ici nous avons laissé les variables d'environnement par défaut mais vous pouvez, bien entendu, les modifier (notez-les !).

- Quittez le fichier en le sauvegardant (CTRL + X + Yes) sous le nom « mariadb.env »
- Créez le fichier « docker-compose.yml » dans le dossier « glpi » à l'aide de l'éditeur « nano » et copiez le contenu du fichier trouvé sur le docker hub (voir page précédente) :

nano docker-compose.yml

```
  GNU nano 5.4
version: "3.2"

services:
# Conteneur MARIADB
  mariadb:
    image: mariadb:latest
    container_name: mariadb
    hostname: mariadb
    volumes:
      - /var/lib/mysql:/var/lib/mysql
    env_file:
      - ./mariadb.env
    restart: always

# Conteneur GLPI 10
  glpi:
    image: diouxx/glpi
    container_name: glpi
    hostname: glpi
    ports:
      - "8081:80"
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
      - /var/www/html/glpi/:/var/www/html/glpi
    environment:
      - TIMEZONE=Europe/Paris
    restart: always
```

Résumé du fichier « docker-compose.yml » qui permettra de créer votre stack « GLPI/MariaDB ».

- Quittez et sauvegardez ce fichier (CTRL + X + YES) **en veillant à ce qu'il soit bien nommé « docker-compose.yml »** sinon vous ne pourrez pas lancer la création de votre infrastructure puis lancez la commande d'exécution **depuis le dossier « glpi » contenant le fichier « docker-compose.yml »** :

docker-compose up -d

Votre environnement est créé et instance GLPI est prête !