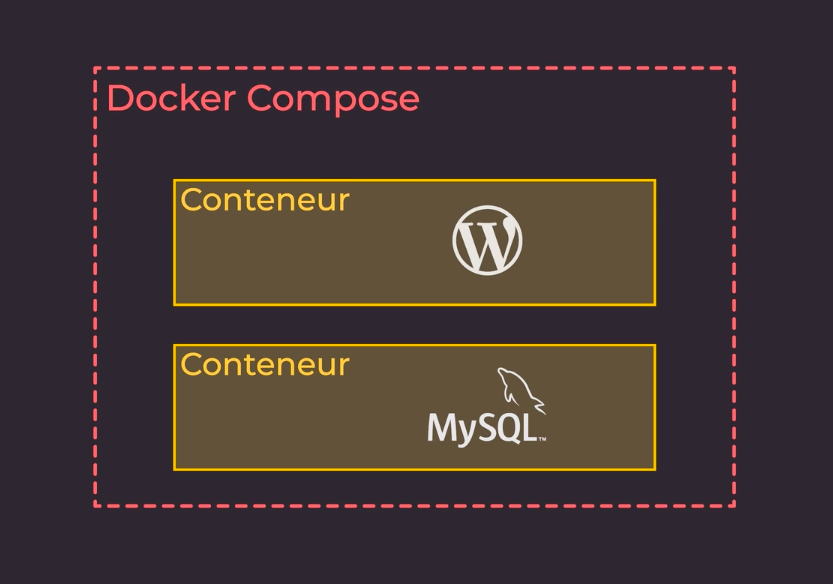
vous avez un nouveau projet de site avec Wordpress, et vous souhaitez simplifier la gestion de l'infrastructure.

Vous devez maintenant réaliser un déploiement en production, où l'ensemble des composants sont dans des conteneurs Docker.

Pour cela, vous allez avoir besoin de plusieurs composants :

* une base de données**MySQL** ;
* le système de fichiers**Wordpress**.

Structure de notre Docker Compose

Vous devez commencer par créer un fichier docker-compose.yml à la racine de votre projet. Dans celui-ci, nous allons décrire l'ensemble des ressources et services nécessaires à la réalisation de votre POC.

**Décrivez votre premier service : db**

**Définissez la version de Docker Compose**

Un fichier docker-compose.yml commence toujours par les informations suivantes :

version: '3'

L'argument version permet de spécifier à Docker Compose quelle **version** on souhaite utiliser, et donc d'utiliser ou pas certaine version. Dans notre cas, nous utiliserons la version 3, qui est actuellement la version la plus utilisée.

**Déclarez le premier service et son image**

Nous allons maintenant déclarer notre premier service, et donc créer notre stack Wordpress !

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 ; dans notre cas, notre premier conteneur se nommera db.

services:

db:

image: mysql:5.7

Puis, vous devez **décrire votre conteneur**; dans notre cas, nous utilisons l’argument image qui nous permet de définir l'image Docker que nous souhaitons utiliser.

Nous aurions pu aussi utiliser l’argument build en lui spécifiant le chemin vers notre fichier Dockerfile ; ainsi, lors de l’exécution de Docker Compose, il aurait construit le conteneur via le Dockerfile avant de l’exécuter.

**Définissez le volume pour faire persister vos données**

services:

db:

image: mysql:5.7

volumes:

- mabase:/var/lib/mysql

Pour rappel, **les conteneurs Docker ne sont pas faits pour faire fonctionner des services stateful**, et une base de données est par définition un service stateful. Cependant, vous pouvez utiliser l'argument volumes qui vous permet de stocker l'ensemble du contenu du dossier /var/lib/mysql dans un disque persistant. Et donc, de pouvoir garder les données en local sur notre host.

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.

**Définissez la politique de redémarrage du conteneur**

services:

db:

image: mysql:5.7

volumes:

- mabase:/var/lib/mysql

restart: always

Les conteneurs étant par définition monoprocessus, si celui-ci 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.

**Définissez les variables d'environnement**

services:

db:

image: mysql:5.7

volumes:

- mabase:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: m2iformation

MYSQL\_DATABASE: formation

MYSQL\_USER: formation

MYSQL\_PASSWORD: formation

L'image MySQL fournie dispose de **plusieurs variables d'environnement** que vous pouvez utiliser ; dans votre 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, comme nous avons utilisé dans le fichier docker-compose.yml ci-dessus.

**Décrivez votre second service : Wordpress**

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.

services:

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: formation

WORDPRESS\_DB\_PASSWORD: formation

WORDPRESS\_DB\_NAME: formation

Le premier argument, depends\_on, nous 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 ainsi le rendre accessible depuis l'extérieur.

Voici le fichier **docker-compose.yml** dans sa version finale :

version: '3'

services:

db:

image: mysql:5.7

volumes:

- mabase:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: m2iformation

MYSQL\_DATABASE: formation

MYSQL\_USER: formation

MYSQL\_PASSWORD: formation

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: formation

WORDPRESS\_DB\_PASSWORD: formation

WORDPRESS\_DB\_NAME: formation

volumes:

mabase: {}

**Lancez votre stack Docker Compose**

Quand vous lancerez vos conteneurs avec la commande docker-compose up -d, vous devriez avoir le résultat suivant :

$ 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 depuis une registry, ou les build via un docker build.

Puis celui-ci lance les deux conteneurs sur votre système ; dans notre cas, vous pourrez voir le résultat en vous ouvrant l'URL suivante dans votre navigateur :

**http://127.0.0.1:8000**

Votre site Wordpress fonctionnel !

**(((OOOOOOOPS localhost ou 127.0.0.1 sur votre machine !!!!!!!!!!!)))**

**Résumé**

Vous savez maintenant utiliser les commandes de base de Docker Compose, et créer un fichier docker-compose.yml pour orchestrer vos conteneurs Docker.

Pour rappel, voici les arguments que nous avons pu voir dans ce chapitre :

* image qui permet de spécifier l'**image source** pour le conteneur ;
* build qui permet de spécifier le **Dockerfile source** pour créer l'image du conteneur ;
* volume qui permet de spécifier les **points de montage** entre le système hôte et les conteneurs ;
* restart qui permet de définir le comportement du conteneur **en cas d'arrêt** du processus ;
* environment qui permet de définir les **variables d’environnement** ;
* depends\_on qui permet de dire que le conteneur**dépend** d'un autre conteneur ;
* ports qui permet de définir les **ports** disponibles entre la machine host et le conteneur.