STRUCTURER UN FICHIER DE TYPE YAML

Exemple de fichier de type « yml » utilisé avec Docker Compose :

```yaml
version: "3.3"
services:

  mysql:
    container_name: mysqlpourwordpress
    environment:
      - MYSQL_ROOT_PASSWORD=motdepasseroot
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=monwordpress
    networks:
    - wordpress
    image: "mysql:5.7"

  wordpress:
    depends_on:
      - mysql
    container_name: wordpressavecmysql
    environment:
      - "WORDPRESS_DB_HOST=mysqlpourwordpress:3306"
      - WORDPRESS_DB_PASSWORD=monwordpress
      - WORDPRESS_DB_USER=wordpress
    networks:
    - wordpress
    ports:
      - "80:80"
    image: wordpress
    volumes:
      - wordpress_config:/var/www/html/

networks:
  wordpress:

volumes:
  wordpress_config:
```

Dans une application multi-conteneur comme Wordpress, on ne parle plus de conteneurs mais de « **services** ». L'ensemble des services déclarés forment une **STACK**.

On peut définir une liste de variables d'environnement. Ces variables sont nécessaires au bon fonctionnement du service.

Pour fonctionner, le CMS Wordpress doit utiliser la base de données liée au service (conteneur) « mysql ».

La clé de premier niveau « **networks:** » permet de déclarer les réseaux utilisés dans votre *stack*. On utilise généralement des réseaux de type « **bridge ».**

Le volume « wordpress_config » est monté dans le conteneur Wordpress.

Les volumes utilisés dans la *stack* sont déclarés par une clé de premier niveau nommée « **volumes:** ».

Syntaxe

- Alignement ! (**2 espaces** !!)
- ALIGNEMENT !! (comme en python)
- **ALIGNEMENT !!!** (le défaut du YAML, pas de correcteur syntaxique automatique, c'est bête mais vous y perdrez forcément quelques heures !
- des listes (tirets)
- des paires **clé: valeur**
- Un peu comme du JSON, avec cette grosse différence que le JSON se fiche de l'alignement et met des accolades et des points-virgules

Structure d'un fichier « yml » :

Version

Un fichier **Compose** commence par déclarer la version de la *spécification Compose* utilisée dans le fichier:

```
version: "3.8"
```

A ce jour, on utilise souvent une version de type « 3 » qui assurera un bon compromis. Attention aux versions trop récentes ou obsolètes !

https://docs.docker.com/compose/compose-file/compose-file-v3/#compose-and-docker-compatibility-matrix

Réseaux

La clé de premier niveau `networks` permet de déclarer les **réseaux** utilisés dans votre *stack*. On utilise généralement des réseaux de type **bridge**:

```
networks:
    interne:
        driver: bridge
    externe:
        driver: bridge
```

Ici `interne` et `externe` sont les noms des réseaux, à remplacer par ce que vous souhaitez.

https://docs.docker.com/compose/compose-file/compose-file-v3/#network-configuration-reference

Volumes

Comme pour les réseaux, les **volumes** utilisés dans la *stack* sont déclarés par une clé de premier niveau `volumes` :

```
volumes:
    base-de-donnees:
    cache:
```

Les volumes sont nécessaires pour la persistence des données des conteneurs.

Ici sont déclarés 2 volumes `base-de-donnees` et `cache` .

https://docs.docker.com/compose/compose-file/compose-file-v3/#volume-configuration-reference

Services

Dans un application dite **multi-conteneur** on ne parle plus de *conteneurs* mais bien de **services**. L'ensemble des **services** forment une **stack**.

On liste les **services** dans une clé de premier niveau `services` en indiquant le nom de notre choix à chaque service:

```
services:
    database:
        # ...
    app:
        # ...
```

La « stack » est composée de plusieurs conteneurs.

https://docs.docker.com/compose/compose-file/compose-file-v3/

Image

La clé `image` permet d'indiquer l'image à utiliser pour un service:

```
services:
    app:
        image: aymdev/mon-app:1.2.3
```

Il est important de « taguer » l'image afin de spécifier la version désirée sinon la version « latest » sera utilisée par défaut.

https://docs.docker.com/compose/compose-file/compose-file-v3/#image

Variables d'environnement

On peut définir une liste de variables d'environnement avec `environment`, ou un fichier de variables avec `env_file` :

```yaml
services:
    app:
        # ...
        env_file: ./config/app.env
        environment:
            APP_ENV: dev
            DATABASE_URL: pgsql:user:pass@database:5432/example
```

https://docs.docker.com/compose/compose-file/compose-file-v3/#environment
https://docs.docker.com/compose/compose-file/compose-file-v3/#env_file

Réseaux et ports

La clé `networks` liste les réseaux (déclarés dans la section `networks` de premier niveau) auxquels le service est connecté:

```yaml
services:
    app:
        # ...
        networks:
            - interne
            - externe
```

Les *mappings de ports* sont listés par la clé `ports`

```yaml
services:
    app:
        # ...
        ports:
            - 8000:80
            - 8080:443
```

Specify both ports (HOST:CONTAINER)

https://docs.docker.com/compose/compose-file/compose-file-v3/#build

Volumes

La clé `volumes` liste les volumes liés (*bind mounts*) ou nommés (déclarés dans la section `volumes` de premier niveau) utilisés par le service:

```yaml
services:
    app:
        # ...
        volumes:
            # volume nommé
            - cache:/var/cache
            # bind mount
            - ./app:/srv/app
```

Le volume nommé correspond au volume créé par Docker avec la commande « docker volume create ».

Le volume « monté » correspond au volume de type « bind » (chemin host : répertoire dans le conteneur).

https://docs.docker.com/compose/compose-file/compose-file-v3/#volumes