

docker
Compose

DOCKER COMPOSE

Premiers pas

SOMMAIRE

1. RAPPELS SUR L'INSTALLATION DE DOCKER (Debian 12)
2. COMPRENDRE LA STRUCTURE D'UN FICHIER « YAML »
 - a. L'argument « version »
 - b. L'argument « services »
 - c. Description du conteneur
 - d. Déclaration des volumes
 - e. Politique de redémarrage du conteneur
 - f. Définition des variables d'environnement
3. STRUCTURE D'UN FICHIER "docker-compose.yml"
4. MISE EN ŒUVRE D'UNE STACK WORDPRESS
5. MISE EN ŒUVRE D'UNE STACK GLPI
6. CREATION D'UNE STACK NEXTCLOUD
 - a. En utilisant des lecteurs attachés ("bind")
 - b. En utilisant des volumes Docker
7. CREATION ET DEPLOIEMENT AUTOMATISE D'UNE STACK « HOME ASSISTANT » DEPUIS PORTAINER-CE

© tutos-info.fr - 02/2024



DIFFICULTE



UTILISATION COMMERCIALE INTERDITE

1 – RAPPELS SUR L'INSTALLATION DE DOCKER SUR DEBIAN 12

Docker Compose est un outil permettant de **définir le comportement de vos conteneurs** et **d'exécuter des applications Docker à conteneurs multiples**. La configuration se fait à partir d'un fichier « YAML » et, avec une seule commande, vous **créez et démarrez tous vos conteneurs de votre configuration**.

1^{ère} étape : installation de Docker

En mode labo, il est possible d'installer Docker avec le script officiel. Pour cela, utilisez les commandes suivantes :

```
apt install curl -y
curl -fsSL https://get.docker.com | sh
```

Note importante (pour travail en production) :

Si vous travaillez en « mode production », **il est recommandé de créer un utilisateur pour gérer Docker et de l'inclure dans le groupe « docker »**. Ainsi, il ne sera plus nécessaire de saisir "sudo" avant toute commande. Pour cela, réalisez les étapes suivantes :

```
# 1ère solution : création d'un utilisateur « docker » qui sera ajouté au groupe « docker »
useradd -g docker docker
```

ou

```
# 2ème solution : attribution du groupe « docker » à un utilisateur existant :
usermod -aG docker nom_user_existant_à_ajouter
```

Déconnectez votre session Debian et reconnectez-vous pour valider l'opération.

Pour vérifier que Docker est bien installé sur votre machine Debian, saisissez **docker --version**

Pour vérifier la version de Docker Compose installée, saisissez : **docker compose version**

Votre environnement Docker est prêt !

En cas de problème, vous pouvez désinstaller Docker ainsi :

```
systemctl stop docker
apt purge docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

2^{ème} étape : installation de PortainerCE

Vous pouvez également installer **PortainerCE** pour gérer, via une interface graphique, l'ensemble de votre infrastructure. L'installation de PortainerCE est simple et se déroule ainsi :

1. Création d'un volume « portainer_data » :

```
docker volume create portainer_data
```

2. Création du conteneur « portainer-ce » :

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

3. Accéder à Portainer :

Pour accéder à Portainer, ouvrez votre navigateur et saisissez dans la barre d'adresse soit votre IP Wan, soit votre domaine et précisez le port 9443 : <https://votrewan:9443> (pensez à ouvrir le port 9443 sur votre routeur !).

La fenêtre suivante s'affiche :

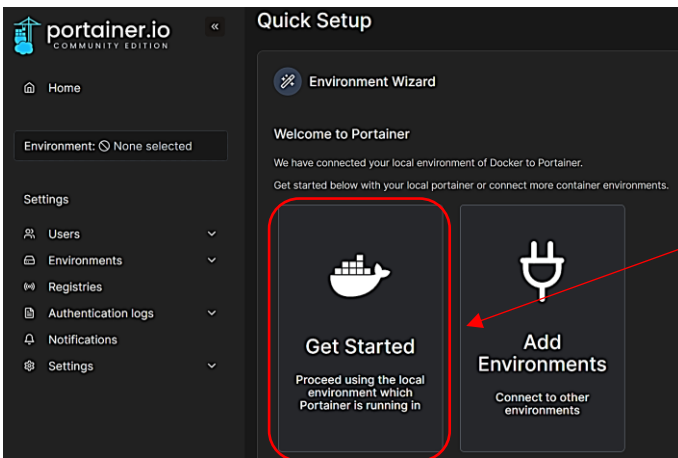


Lors de la 1^{ère} connexion à l'interface de Portainer, vous devez créer un utilisateur « administrateur » et un mot de passe.

Si vous avez trop attendu après l'installation de PortainerCE, il est possible que la fenêtre de PortainerCE ne s'ouvre pas lors de la première connexion et qu'un message vous demande de relancer votre conteneur. Dans ce cas, saisissez sur votre serveur :

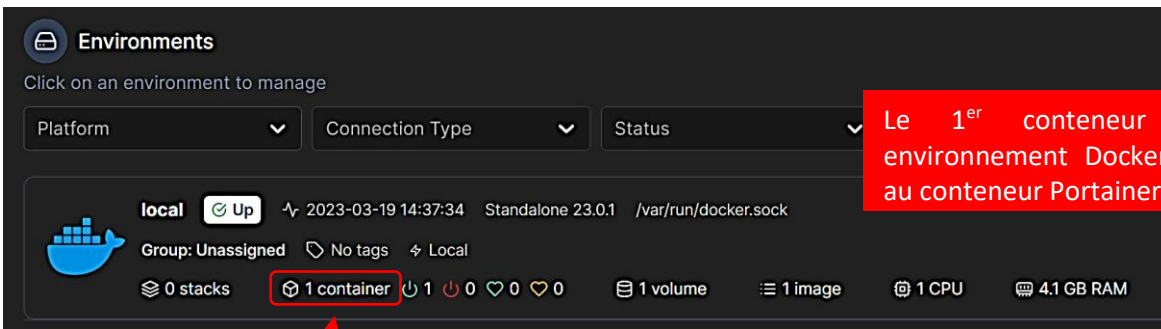
```
docker stop portainer
docker start portainer
```

Dans la fenêtre suivante, cliquez sur « Get started » :



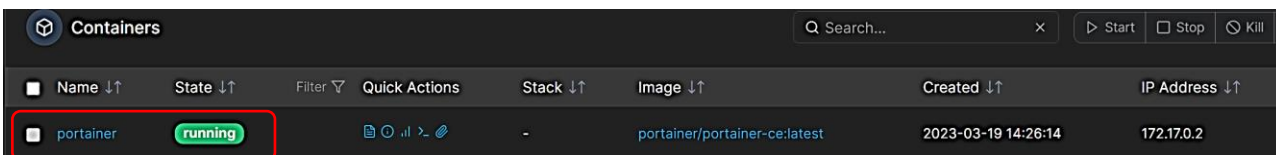
Lors de la première connexion à l'interface de Portainer-CE, cliquez sur « Get started » pour connecter Portainer à votre environnement local Docker.

Votre environnement local s'affiche :



Le 1^{er} conteneur de votre environnement Docker correspond au conteneur Portainer-CE

Si vous cliquez sur « 1 container » vous obtenez un détail de votre environnement Docker :



Attention, l'interface intuitive de PortainerCE ne dispense pas d'utiliser Docker en mode « cli » (lignes de commandes).

En effet, il est parfois nécessaire de maîtriser les commandes pour certaines opérations plus complexes.

2 – COMPRENDRE LA STRUCTURE D'UN FICHER 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 : on parle alors de "**STACK**" (ou de « pile »).

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. **Le nom du fichier sera "docker-compose.yml"**.

Qu'est-ce qu'un fichier « yaml » ?

Les fichiers avec l'extension « .yaml » 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. Attention, **YAML** est sensible à la casse (minuscules/majuscules) !

Le but du fichier "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** » :

Lorsque vous rédigez votre fichier "yaml", respectez l'espacement (par exemple, "service1" est situé sous "services" **avec une indentation (2 espaces de plus par exemple ou une tabulation) ; cette indentation symbolise une arborescence**).

version: "3" # Version de Docker Compose à utiliser (voir signification plus bas)

services:

nom_service1: # Nom du premier service ; c'est-à-dire le 1^{er} conteneur de la stack

image: nom_de_l_image # Image Docker utilisée pour ce service

ports:

- "port_hôte:port_conteneur" # Mappage des ports

environment:

- VARIABLE_1=valeur_1 # Variables d'environnement

volumes:

- /chemin/hôte:/chemin/conteneur # Mappage de volumes

depends_on:

- nom_service2 # Dépendances entre les services

nom_service2: # Nom du deuxième service ; c'est-à-dire le 2^{ème} conteneur de la stack

image: # Configuration du deuxième service

etc...

networks:

mynetwork: # Définition d'un réseau personnalisé (si nécessaire ; par défaut Docker utilisera Docker0)

driver: bridge # Type de réseau si utilisation d'un réseau personnalisé

3 – MISE EN PLACE D'UNE STACK WORDPRESS – CREATION DU FICHIER YAML

Dans le cas présent, nous souhaitons créer une stack Wordpress avec une instance correspondant à la base de données MariaDB et une instance relative au CMS. Pour cela, nous procédons ainsi :

1^{ère} étape : création d'un dossier qui recevra notre fichier "docker-compose.yml" sur la machine hôte Debian

```
mkdir stack_wordpress
```

2^{ème} étape : réalisation du fichier "docker-compose.yml" dans le dossier de la stack

```
cd stack_wordpress  
nano docker-compose.yml
```

Il convient, ensuite, de rédiger le fichier YAML en respectant certaines règles (indentation et casse des caractères).

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 »

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 qui offre une compatibilité optimale avec les systèmes actuels.

La version saisie 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**'.

Il est conseillé de ne pas mettre la dernière version car les versions Docker installées sur les serveurs ne sont pas forcément les dernières ; dans ce cas votre fichier ".yml" serait incompatible !

Vous trouverez plus d'informations sur la compatibilité des versions [ici](#) :

Compose file format	Docker Engine release
Compose specification	19.03.0+
3.8	19.03.0+
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+

Une version « 3.2 » est une bonne alternative puisqu'elle assure une compatibilité avec le Docker Engine 17.04 et + (à ce jour, nous en sommes à la version 25.0.3 du Docker Engine).

On commence par indiquer, **à la marge de gauche et sans indentation**, le numéro de la version du docker-compose qui sera utilisée :

```
version: "3"
```

b) Définition des SERVICES

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 « **bdd** » (il correspondra à la base de données que l'on veut créer).

On commence à la marge de gauche pour l'argument "services:" puis on décale pour le nom du conteneur :

```
version: "3"
```

```
services:
```

```
  bdd:
```

c) Description du conteneur

Ici on indique que le conteneur relatif à la base de données sera créé à partir de l'image « mysql:latest ». On passe à la ligne **et on décale "image:" de 2 espaces supplémentaires pour créer une arborescence** (soit 4 espaces depuis la marge de gauche) :

```
version: "3"
```

```
services:
```

```
  bdd:
```

```
    image: mysql:latest
```

d) Déclaration du(des) VOLUME(S) qui permettra(ont) de conserver les données

Les conteneurs Docker ne conservent pas les données si un volume Docker 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.

On indique ici "volumes:" (aligné avec "image:") et on spécifie le nom du volume à créer et le dossier du conteneur :

```
version: "3"
```

```
services:
```

```
  bdd:
```

```
    image: mysql:latest
```

```
    volumes:
```

```
      - db_data:/var/lib/mysql
```

Cette description **permet de créer un volume « db_data » qui est un volume créé par Docker**. Ainsi, les données du conteneur seront inscrites sur le disque hôte (équivalent de "docker volume create") dans l'arborescence "/var/lib/docker/volumes/..."

Nous aurions aussi pu attacher un dossier de la machine hôte, sans nous servir des volumes Docker, en saisissant "/data/mysql:/var/lib/mysql" (**lecteur attaché de type « bind » qui est à créer au préalable !**).

e) Politique de redémarrage du conteneur avec la directive « RESTART »

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:** ». Pour cela, on ajoute l'argument, aligné avec le précédent ("volumes") :

```
version: "3"
services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    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:** ». L'argument sera aligné avec le précédent ("restart") :

```
version: "3"
services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

Attention, **il est possible également de créer un fichier caché de variables sur l'hôte pour éviter d'écrire les mots de passe en clair dans le fichier docker-compose.yml** ; on peut, par exemple, créer un fichier nommé ".env" pour plus de confidentialité (voir plus loin dans ce tutoriel).

g) Définition d'un nouveau service (2^{ème} 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 : « **depends_on:** » et « **ports:** ».

Le premier argument, « **depends_on:** », permet de **créer une dépendance entre deux conteneurs**. Ainsi, Docker démarrera le service « bdd » avant de démarrer le service « wordpress ».

Ce qui est un comportement souhaitable car **WordPress dépend de la base de données** pour fonctionner correctement. Pour information, on peut aussi utiliser l'argument « **links:** ».

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

On ajoute donc un 2^{ème} service qui correspondra au conteneur Wordpress de la stack ; attention, alignez "wordpress:" avec le nom du 1^{er} conteneur "bdd" :

```
version: "3"
services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - bdd
    image: wordpress:latest
    ports:
      - "8088:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: bdd:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
```

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

On termine le fichier avec l'argument « **volumes:** » qui permet de spécifier le nom des volumes Docker utilisés pour cette stack. L'argument « **volumes:** » est saisi en fin de fichier, à la marge de gauche (aligné avec « **version:** » et « **services:** ») ; ce qui donne ceci :

```
version: "3"
services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - bdd
    image: wordpress:latest
    ports:
      - "8088:80"
```



```
restart: always
environment:
  WORDPRESS_DB_HOST: bdd:3306
  WORDPRESS_DB_USER: wordpress
  WORDPRESS_DB_PASSWORD: wordpress
  WORDPRESS_DB_NAME: wordpress
```

```
volumes:
  db_data: {}
```

4 – LANCEMENT DE LA STACK WORDPRESS

Lorsque vous utilisez Docker Compose, il est conseillé de créer **un fichier** nommé « **docker-compose.yml** » dans **un dossier** déterminé (**1 stack = 1 dossier**). Vous pouvez créer ce dossier et ce fichier où vous le voulez, l'essentiel étant que vous arriviez à vous y retrouver dans votre arborescence (utilisez des noms de dossiers explicites !).

Ici, nous avons créé un dossier « stack_wordpress » et le fichier « docker-compose.yml » a été placé dans ce dossier.

Notre fichier "docker-compose.yml" intégral relatif à notre stack Wordpress se présente donc ainsi :

```
version: "3"
services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - bdd
    image: wordpress:latest
    ports:
      - "8088:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: bdd:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
```

Fichier « docker-compose.yml » à obtenir :

```
GNU nano 7.2
version: "3"

services:
  bdd:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - bdd
    image: wordpress:latest
    ports:
      - "8088:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: bdd:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
```

Compatibilité du fichier

1^{er} service correspondant au processus mysql (dernière version)

Création du volume Docker "db_data" pour ce processus

On stipule, ici, les variables d'environnement pour la base de données

Création du 2^{ème} service avec sa dépendance avec le 1^{er} service

On se base sur la dernière version de Wordpress et on mappe le port "8088" de l'hôte tout en exposant le port "80" du conteneur" Wordpress.

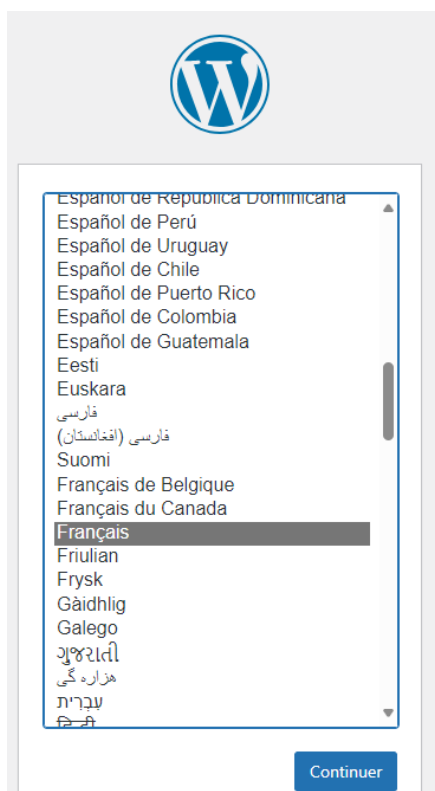
On indique à Docker qu'il devra utiliser le volume Docker "db_data"

LANCEMENT DE LA CREATION AUTOMATISEE DE LA STACK

Une fois le « docker-compose.yml » enregistré, 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 services. Dans le cas contraire, il les télécharge. Puis il lance les deux conteneurs sur votre système ; votre stack est prête !

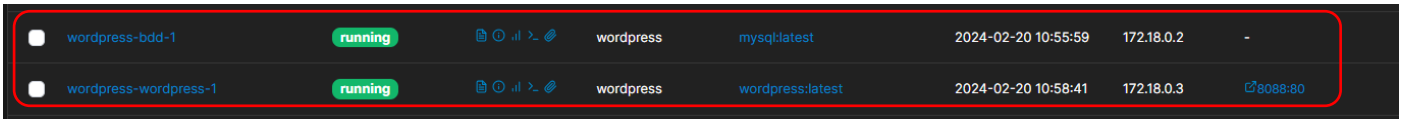
Lancez votre navigateur en saisissant, dans la barre d'adresses, votre wan:8088 ; l'assistant d'installation de Wordpress s'affiche (n'oubliez pas d'ouvrir le port "8088" sur votre routeur !) :



Votre Wordpress est installé et l'assistant d'installation est affiché ! Il n'y a plus qu'à suivre les étapes !

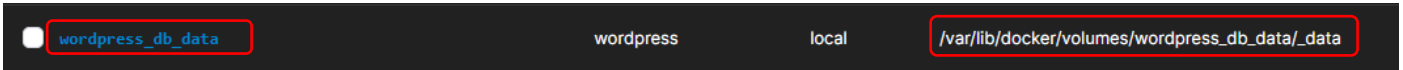
PortainerCE affiche la stack et le volume créé :

La stack, composée du serveur mysql et de wordpress :



Service	Status	Image	MySQL Image	Created	IP	Port
wordpress-bdd-1	running	wordpress	mysql:latest	2024-02-20 10:55:59	172.18.0.2	-
wordpress-wordpress-1	running	wordpress	wordpress:latest	2024-02-20 10:58:41	172.18.0.3	8088:80

Le volume Docker généré :



Volume Name	Mount Path
wordpress_db_data	/var/lib/docker/volumes/wordpress_db_data/_data

5 – CREATION AUTOMATISEE D'UNE STACK GLPI

Dans cet exemple, nous allons mettre en place un serveur MYSQL de type MariaDB et l'helpdesk GLPI. Sur le docker hub, nous pouvons trouver une multitude d'images et de fichiers « docker-compose.yml ». Nous allons également utiliser un fichier de variables d'environnement (voir argument "env_file:").

Ici, nous avons sélectionné l'image « **diouxx/glpi** » qui est très connue et mise à jour très régulièrement avec le fichier « yml » correspondant. La page Github est ici : [GitHub - DiouXX/docker-glpi: Project to deploy GLPI with docker](https://github.com/DiouXX/docker-glpi)

Mise en place des conteneurs MariaDB et GLPI (avec un fichier contenant des variables d'environnement)

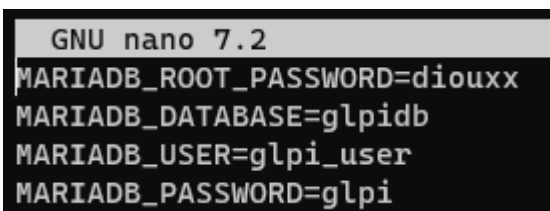
1 - Sur la machine Debian, créez un dossier « **glpi** » (ici nous l'avons créé dans le dossier ~ du root (car nous ne sommes pas en production) avec la commande « **mkdir glpi** »

2 - 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

```
MARIADB_ROOT_PASSWORD=diouxx
MARIADB_DATABASE=glpidb
MARIADB_USER=glpi_user
MARIADB_PASSWORD=glpi
```

Ce qui donne :



```
GNU nano 7.2
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 et sauvegardez le fichier (sous le nom « **mariadb.env** »)
- Créez le fichier « **docker-compose.yml** » dans le dossier « glpi » à l'aide de l'éditeur « nano » :

nano docker-compose.yml

Contenu du fichier « docker-compose.yml » à créer :

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

Création du 1^{er} conteneur nommé « mariadb » avec l'attachement d'un volume et le lien vers un 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.

```
glpi:
  image: diouxx/glpi
  container_name: glpi
  hostname: glpi
  ports:
    - "8090:80"
  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 2^{ème} conteneur « glpi » basé sur une image du Docker hub avec mappage du port 8090 (on aurait pu choisir un autre port ici) et attachement des volumes utiles à la persistance des données du conteneur.

On obtient ceci :

```
GNU nano 7.2
version: "3.2"

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

#GLPI Container
  glpi:
    image: diouxx/glpi
    container_name : glpi
    hostname: glpi
    ports:
      - "8090:80"
    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
```

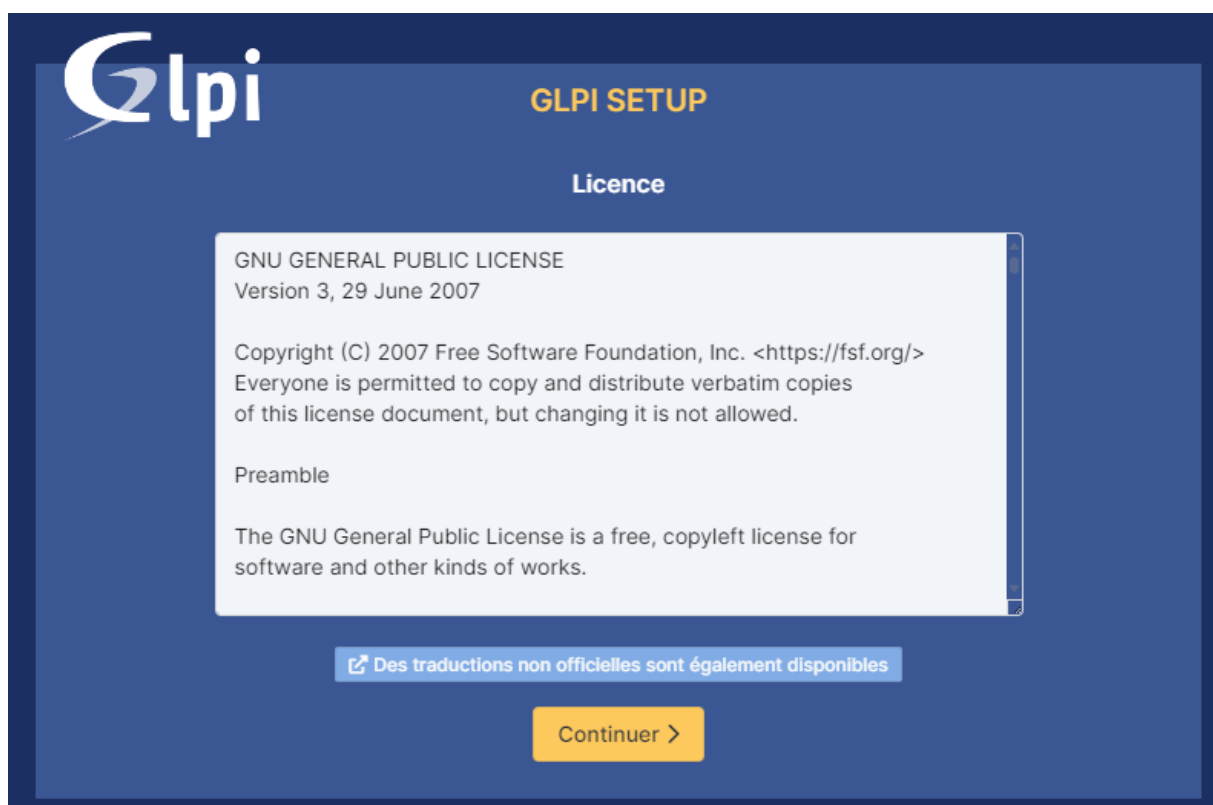
- Quittez et sauvegardez ce fichier, **en veillant à ce qu'il soit bien nommé « docker-compose.yml »** sinon vous ne pourrez pas lancer la création de votre stack.
- Exécutez, **depuis le dossier « glpi » contenant le fichier « docker-compose.yml »**, la commande suivante :

docker compose up -d

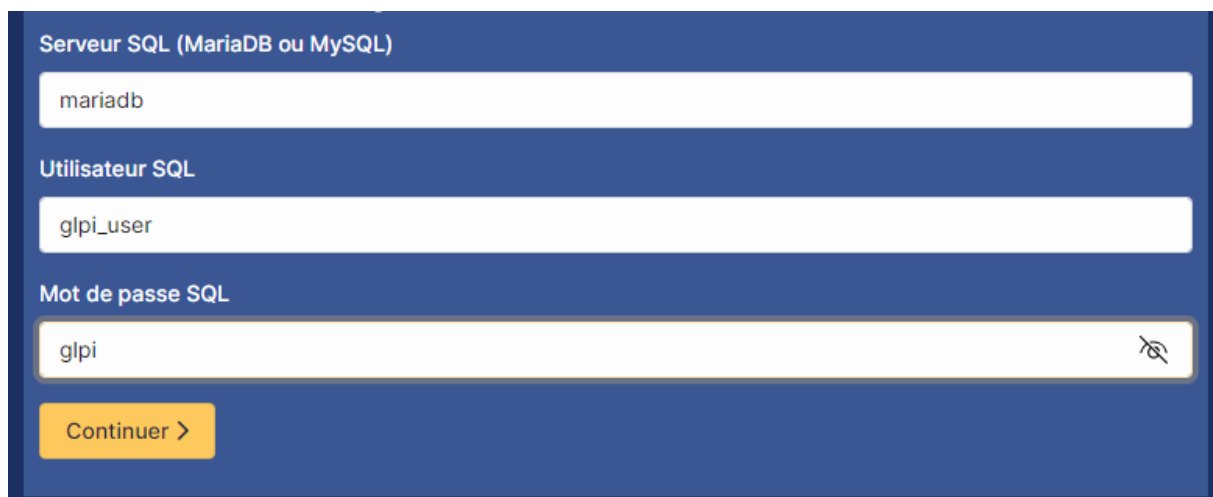
Votre environnement se crée automatiquement.

Lancez votre navigateur pour finaliser l'installation de GLPI en saisissant, dans la barre d'adresse, votre wan:8090 et suivez l'assistant en complétant les fenêtres successives.

Servez-vous des variables d'environnement préalablement déclarées dans le fichier « mariadb.env » pour finaliser l'installation de votre helpdesk !



Saisissez les bonnes valeurs (voir dans votre fichier ".env") :

The image shows the 'Serveur SQL' configuration screen. It has three input fields: 'Serveur SQL (MariaDB ou MySQL)' containing 'mariadb', 'Utilisateur SQL' containing 'glpi_user', and 'Mot de passe SQL' containing 'glpi'. A 'Continuer >' button is at the bottom left.

Votre stack GLPI est prête en quelques secondes !

Une fois connecté à l'interface de GLPI, des messages d'alertes s'affichent et sont relatifs aux mots de passe des identifiants par défaut et à la présence du fichier "**install.php**" dans le dossier `/var/www/html/glpi/install/install.php`

Pour corriger le message lié au fichier "install.php", nous procédons ainsi sur la machine hôte :

```
cd /var/www/html/glpi/install
rm install.php
```

En actualisant la page GLPI, on constate que le message ne s'affiche plus !

6 – CREATION AUTOMATISEE D'UNE STACK NEXTCLOUD

Dans cet exemple, nous allons mettre en œuvre un cloud privé de type "Nextcloud". **Nous allons utiliser des dossiers "attachés" (bind) pour voir le principe.** Nous avons besoin de 2 dossiers "data" et "db" pour cette stack.

1^{ère} étape : création des dossiers locaux nécessaires (dans notre /home)

```
mkdir nextcloud
cd nextcloud
mkdir data
mkdir db
```

2^{ème} étape : création du fichier "docker-compose.yml" dans le dossier "nextcloud"

```
version: '3'
services:
  nextcloud_db:
    image: mariadb:latest
    container_name: nextcloud_db
    restart: always
    volumes:
      - ./db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=$NEXTCLOUD_MYSQL_ROOT_PASSWORD # Mot de passe de l'utilisateur root de mariadb
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données à créer à l'initialisation du conteneur
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données créée
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur créé

  nextcloud_app:
    image: nextcloud:latest
    restart: always
    ports:
      - "8091:80"
    links:
      - nextcloud_db
    volumes:
      - ./data:/var/www/html
    environment:
      - MYSQL_HOST=nextcloud_db # Nom du conteneur de la base de données
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur de la base de données
```

Le fichier "docker-compose.yml" se présente ainsi :

```
GNU nano 7.2                                docker-compose.yml *
version: '3'
services:
  nextcloud_db:
    image: mariadb:latest
    container_name: nextcloud_db
    restart: always
    volumes:
      - ./db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=$NEXTCLOUD_MYSQL_ROOT_PASSWORD # Mot de passe de l'utilisateur root de mariadb
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données à créer à l'initialisation du conteneur
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données créée
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur créé

  nextcloud_app:
    image: nextcloud:latest
    restart: always
    ports:
      - "8091:80"
    links:
      - nextcloud_db
    volumes:
      - ./data:/var/www/html
    environment:
      - MYSQL_HOST=nextcloud_db # Nom du conteneur de la base de données
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur de la base de données
```

3^{ème} étape : création du fichier ".env" dans le dossier "nextcloud" (le point avant le nom permet de le cacher) :

```
nano .env
```

```
NEXTCLOUD_MYSQL_DATABASE=nextcloud
NEXTCLOUD_MYSQL_USER=nextcloud
NEXTCLOUD_MYSQL_ROOT_PASSWORD=root_mysql_password
NEXTCLOUD_MYSQL_PASSWORD=nextcloud_password
```

4^{ème} et dernière étape : création de la stack

docker compose up -d

Pour finaliser l'installation de votre Nextcloud, lancez votre navigateur et saisissez votre wan:8091 ; votre Nextcloud est prêt en quelques secondes !

Stack Nextcloud avec utilisation des volumes Docker

Il est possible d'utiliser les **volumes Docker**. Dans ce cas, il n'est pas utile de créer les dossiers "data_nextcloud" et "db_data" sur l'hôte avant. Docker créera les volumes directement et les placera dans "/var/lib/docker/volumes/...". On modifie le fichier "docker-compose.yml" ainsi :

```
version: '3'
services:
  nextcloud_db:
    image: mariadb:latest
    container_name: nextcloud_db
    restart: always
    volumes:
      - db_data:/var/lib/mysql
```

```

environment:
  - MYSQL_ROOT_PASSWORD=$NEXTCLOUD_MYSQL_ROOT_PASSWORD # Mot de passe de l'utilisateur root de mariadb
  - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données à créer à l'initialisation du conteneur
  - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données créée
  - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur créé

nextcloud_app:
  depends_on:
    - nextcloud_db
  image: nextcloud:latest
  restart: always
  ports:
    - "8092:80"
  volumes:
    - data_nextcloud:/var/www/html
  environment:
    - MYSQL_HOST=nextcloud_db # Nom du conteneur de la base de données
    - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données
    - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données
    - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur de la base de données
  volumes:
    db_data: {}
    data_nextcloud: {}

```

Le fichier « docker-compose.yml » se présente ainsi :

```

GNU nano 7.2                                docker-compose.yml *
version: '3'
services:
  nextcloud_db:
    image: mariadb:latest
    container_name: nextcloud_db
    restart: always
    volumes:
      - db_data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=$NEXTCLOUD_MYSQL_ROOT_PASSWORD # Mot de passe de l'utilisateur root de mariadb
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données à créer à l'initialisation du conteneur
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données créée
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur créé

  nextcloud_app:
    depends_on:
      - nextcloud_db
    image: nextcloud:latest
    restart: always
    ports:
      - "8092:80"
    volumes:
      - data_nextcloud:/var/www/html
    environment:
      - MYSQL_HOST=nextcloud_db # Nom du conteneur de la base de données
      - MYSQL_DATABASE=$NEXTCLOUD_MYSQL_DATABASE # Nom de la base de données
      - MYSQL_USER=$NEXTCLOUD_MYSQL_USER # Nom de l'utilisateur de la base de données
      - MYSQL_PASSWORD=$NEXTCLOUD_MYSQL_PASSWORD # Mot de passe de l'utilisateur de la base de données
  volumes:
    db_data: {}
    data_nextcloud: {}

```


Différences entre dossiers attachés ("bind") et volumes Docker :

Dossier attaché :

```
volumes:  
- ./db:/var/lib/mysql
```

Ici, on attache les dossiers "db" et "data" qui sont situés dans le répertoire courant (symbolisé par "."). Attention, il faut avoir créé ses dossiers au préalable !

```
volumes:  
- ./data:/var/www/html
```

Volumes Docker :

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

Ici, demande la création des volumes Docker "db_data" et "data_nextcloud" au moment de la création de la stack. Il n'est pas utile de créer les dossiers au préalable car Docker va générer des volumes qui seront placés dans "/var/lib/docker/volumes/..."

```
volumes:  
- data_nextcloud:/var/www/html
```

```
volumes:  
  db_data: {}  
  data_nextcloud: {}
```

Ici, on indique à Docker qu'il devra utiliser les volumes "db_data" et "data_nextcloud" dans la stack.

7 – CREATION ET DEPLOIEMENT AUTOMATISE D'UNE STACK « HOME ASSISTANT » DEPUIS PORTAINER-CE

Dans cet exemple, nous allons mettre en œuvre une stack depuis l'interface de PortainerCE. Nous allons, ici, préparer une stack basée sur l'**application open-source Home Assistant**.

Home Assistant est une **plateforme de domotique open-source**. Home Assistant permet de contrôler et d'automatiser de nombreux appareils et systèmes de la maison. En outre elle vous permet de contrôler vos lumières, thermostats, serrures, caméras de sécurité, enceintes connectées et bien plus encore.

Il est possible de déployer une stack directement avec Portainer-CE de la manière suivante :

- Connectez-vous à Portainer
- Cliquez, dans le volet de gauche, sur "**Stacks**"
- Cliquez, en haut à droite, sur le bouton bleu "**Add stack**"
- Copiez, dans le "**Web editor**", les lignes de la page suivante qui correspondent au fichier "docker-compose.yml" permettant de générer le déploiement de Home Assistant.

Le fichier « docker-compose.yml » se présente ainsi :

```
version: '3'
```

```
services:
```

```
  homeassistant:
```

```
    image: homeassistant/home-assistant:stable
```

```
    container_name: homeassistant
```

```
    environment:
```

```
      - TZ=Europe/Paris
```

```
    ports:
```

```
      - 8123:8123
```

```
    volumes:
```

```
      - config_homeassistant:/config
```

```
      - /etc/localtime:/etc/localtime:ro
```

```
    restart: always
```

```
volumes:
```

```
  config_homeassistant: {}
```

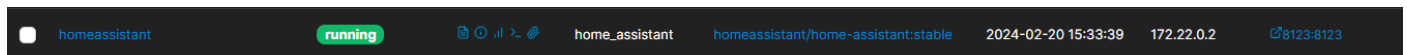
Création du conteneur "homeassistant" basé sur la dernière image "stable".

Ici, on mappe le port "8123" de l'hôte et du conteneur.

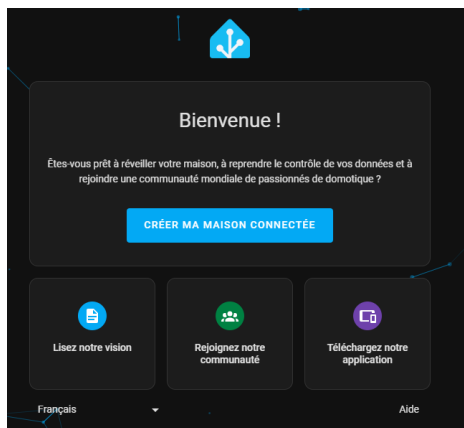
On crée un volume Docker nommé "config_homeassistant" qui sera monté dans le dossier "config" du conteneur.

On indique à Docker d'utiliser le volume créé "config_homeassistant" dans cette stack.

Une fois le fichier préparé, on lance le déploiement en cliquant le bouton "Deploy the stack" (en bas de la fenêtre). Il faut patienter le temps que la stack soit déployée et, si tout est correct, Portainer affiche le conteneur en mode "running" une fois ce dernier prêt :



Il ne reste plus qu'à ouvrir un navigateur et saisir l'adresse wan:8123 (pensez à ouvrir le port sur le routeur !) :



Votre home assistant est prêt ! Il ne reste plus qu'à le configurer pour faire de votre maison une "maison connectée" !

Attention, lorsque la stack est déployée depuis PortainerCE, le fichier "docker-compose.yml" se trouve dans le volume "portainer_data" de base et dans le sous-dossier "compose" :

```
/var/lib/docker/volumes/portainer_data/_data/compose
```

PortainerCE crée, ensuite, des dossiers numérotés pour chaque stack déployée (ici "1") :

```
drwx----- 2 root root 4096 20 févr. 15:31 1
root@debian:/var/lib/docker/volumes/portainer_data/_data/compose# cd 1
root@debian:/var/lib/docker/volumes/portainer_data/_data/compose/1# ls -la
total 12
drwx----- 2 root root 4096 20 févr. 15:31 .
drwx----- 3 root root 4096 20 févr. 15:31 ..
-rw----- 1 root root 337 20 févr. 15:31 docker-compose.yml
```