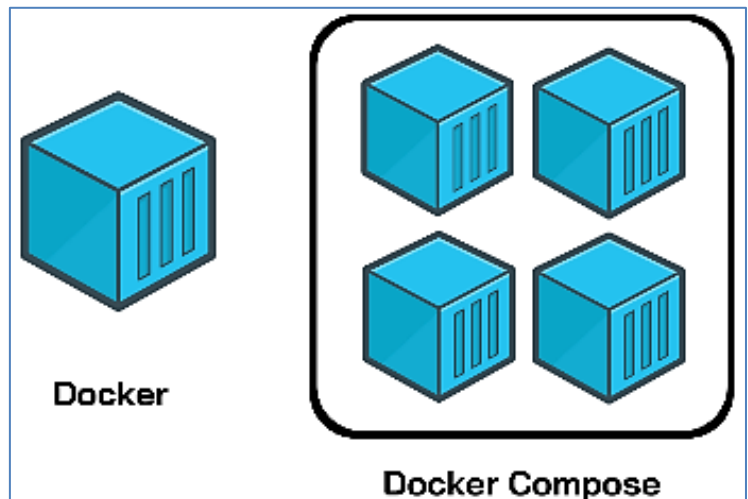


PREMIERS PAS AVEC DOCKER COMPOSE

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 « **.yaml** » 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 l'indentation (**les différents « niveaux » sont matérialisés par 2 espaces**).

Le but d'un fichier « **docker-compose.yaml** » est de **gérer correctement les conteneurs en décrivant ce que nous souhaitons faire**. Normalement, pour lancer un conteneur avec Docker CLI, il faut saisir une commande qui peut être complexe. Avec le fichier « **docker-compose.yaml** », 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.yaml** ».

Pour comprendre le fonctionnement global, nous allons déployer l'helpdesk GLPI. Pour cela, connectez-vous au Docker Hub et, dans la barre de recherche, saisissez « **glpi** » et localisez le projet suivant :



Le projet « **diouxx/glpi** » est le plus connu pour déployer GLPI. De plus, le projet est régulièrement maintenu.

- [Deploy with docker-compose](#)
 - [Deploy without persistence data \(for quickly test \)](#)
 - [Deploy a specific release](#)
 - [Deploy with persistence data](#)
 - [mariadb.env](#)
 - [docker-compose .yaml](#)

Un script au format « yml » est fourni et nous pouvons l'analyser :

```
---
services:
#MariaDB Container
  mariadb:
    image: mariadb:10.7
    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
```

Commencez votre fichier par saisir --- (il n'est plus utile de mettre une version)

Définir les services (conteneurs) et les nommer (exemple « mariadb »)

Indiquer l'image qui sera utilisée pour le conteneur

Nommer le conteneur et le « hostname » du conteneur (localhost par défaut)

Indiquer le(les) volume(s) utilisé(s) par le conteneur

Indiquer, si un fichier de variables d'environnement doit être utilisé

On peut définir la politique de redémarrage du conteneur

Ici, nous avons la structure du 2^{ème} conteneur qui correspond, cette fois, aux sources de l'helpdesk GLPI. On a mappé, ici, le port 8090 de la machine hôte vers le port 80 du conteneur. Cela signifie que pour se connecter à l'interface web de GLPI il faudra saisir l'IP de la machine hôte Debian suivie de :8090

Ici, nous avons mappé le port 8090 de la machine hôte Debian. Cela signifie que pour nous connecter à GLPI, il faudra saisir :8090 dans l'URL.

Remarque importante :

Faites très attention à l'indentation ; n'utilisez jamais la tabulation dans le fichier yml ! Les différents « niveaux » sont matérialisés par 2 espaces.

CREATION DU FICHIER « **compose.yml** » POUR LE DEPLOIEMENT DE GLPI

Sur votre machine hôte, commencez par créer votre **arborescence de travail** ; par exemple dans votre dossier « **home** », créez un dossier « **compose** » puis dans ce dossier créez un dossier « **glpi** ». C'est dans ce dossier que nous allons créer le fichier « **compose.yml** » ainsi que fichier contenant les variables d'environnement.

- Depuis le dossier « **glpi** » ouvrez l'éditeur « **nano** » en saisissant « **nano compose.yml** » et copiez-collez le contenu du fichier yml donné en page précédente.
- Quittez et sauvegardez le fichier.
- Toujours 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

Copiez le contenu suivant :

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

Vous pouvez, bien entendu, modifier ces variables mais notez-les car vous en aurez besoin par la suite !

- Quittez le fichier en le sauvegardant sous le nom « **mariadb.env** ».

Depuis le dossier « **glpi** » dans lequel vous avez vos fichiers « **compose.yml** » et « **mariadb.env** », lancez le déploiement de GLPI avec la commande suivante :

docker compose up -d

Le déploiement est lancé ; patientez ! Après quelques minutes, votre environnement est créé et une instance GLPI est prête !

Vous pouvez vérifier, depuis une machine de votre réseau local, que le déploiement a réussi en ouvrant un navigateur web et en saisissant dans l'URL ceci : http://ip_Debian-Docker:8090

L'assistant d'installation de GLPI doit s'afficher ; il ne reste plus qu'à terminer l'installation !

TEST DE DEPLOIEMENT DU CMS WORDPRESS

WordPress est l'un des CMS les plus utilisés au monde. Son architecture repose sur une base de données MySQL/MariaDB et un serveur PHP. C'est un excellent cas d'usage pour Compose : **deux services distincts qui communiquent**.

Objectifs du TP

- Créer un fichier « compose.yml ».
- Déployer WordPress avec MariaDB en mode multi-conteneurs.
- Vérifier le bon fonctionnement de l'application.
- Comprendre la persistance des données via les volumes.
- Savoir arrêter, redémarrer et nettoyer un déploiement Compose.

Architecture cible

Schema réseau - WordPress

Navigateur		Réseau Docker interne	
Port 8095 ----->		wordpress:80 ----->	db:3306
(Hôte)		(PHP/Apache)	(MariaDB)
wp_data (volume)		db_data (volume)	

Préparation - Créer l'arborescence de travail

```
mkdir ~/wordpress
cd ~/wordpress
```

Création du fichier « compose.yml »

```
nano compose.yml
```

Contenu du fichier « compose.yml » pour le déploiement de WordPress

```
---
services:

# ---- Service 1 - Conteneur Base de données MariaDB
db:
  image: mariadb:10.11
  container_name: wp-db
  restart: unless-stopped
  environment:
    MARIADB_ROOT_PASSWORD: rootpass
    MARIADB_DATABASE: wordpress
    MARIADB_USER: wpuser
    MARIADB_PASSWORD: wppass
  volumes:
    - db_data:/var/lib/mysql
  networks:
    - wp-net
```

```
# ---- Service 2 - Application WordPress
wordpress:
  image: wordpress:latest
  container_name: wp-app
  restart: unless-stopped
  ports:
    - "8095:80"
  environment:
    WORDPRESS_DB_HOST: db
    WORDPRESS_DB_NAME: wordpress
    WORDPRESS_DB_USER: wpuser
    WORDPRESS_DB_PASSWORD: wppass
  volumes:
    - wp_data:/var/www/html
  depends_on:
    - db
  networks:
    - wp-net

# ---- Volumes persistants ----
volumes:
  db_data:
  wp_data:

# ---- Réseau interne dédié à Wordpress ----
networks:
  wp-net:
```

Quittez et sauvegardez le fichier.

Démarrage, déploiement et vérification

1. Lancer les services

```
docker compose up -d
```

Lors du premier lancement, Docker télécharge les images. Attendez que la commande se termine.

2. Vérifier l'état des conteneurs

```
docker compose ps
```

Tous les services doivent afficher le statut running (ou Up).

3. Accéder à l'assistant d'installation de WordPress, depuis une machine de votre réseau local, en ouvrant un navigateur et en saisissant l'URL suivante : http://IP_hôte:8095

```
>> Assistant d'installation WordPress
```

Au premier accès, WordPress affiche son assistant de configuration.
 Choisissez la langue, définissez le titre du site et le compte administrateur.
 Vous n'avez PAS à reconfigurer la base de données : les paramètres sont déjà injectés.

5.1 Utiliser un fichier « .env » pour les mots de passe

Mettre des mots de passe en clair dans « compose.yml » n'est pas une bonne pratique. Docker Compose permet d'externaliser ces valeurs dans un fichier « .env ».

Création d'un fichier « mariadb.env »

```
# Fichier « mariadb.env » (même dossier que compose.yml)

MARIADB_ROOT_PASSWORD=rootpass
MARIADB_USER=wpuser
MARIADB_PASSWORD=wppass
```

Dans le « compose.yml », on référence ces variables avec la syntaxe **`${VARIABLE}`** :

Modification du fichier « compose.yml » avec les variables « .env »

```
environment:
  MARIADB_ROOT_PASSWORD: ${MARIADB_ROOT_PASSWORD}
  MARIADB_USER: ${MARIADB_USER}
  MARIADB_PASSWORD: ${MARIADB_PASSWORD}
```

! Sécurité - .gitignore

Le fichier « .env » ne doit JAMAIS être versionné dans Git.
Ajoutez « .env » dans votre fichier « .gitignore » pour l'exclure des commits.

À propos du « Healthcheck » (attendre que le service soit prêt)

« depends_on » ne vérifie que le démarrage du conteneur, pas que le service soit opérationnel. Le « **healthcheck** » permet de tester la disponibilité réelle :

Exemple de fichier « compose.yml » avec healthcheck

```
db:
  image: mariadb:10.11
  healthcheck:
    test: ['CMD', 'mysqladmin', 'ping', '-h', 'localhost']
    interval: 10s
    timeout: 5s
    retries: 5
```

INSTALLATION DE PORTAINER CE

Vous pouvez installer **Portainer** sur votre machine hôte Debian pour gérer, via une interface graphique, l'ensemble de votre infrastructure conteneurisée.

L'installation de Portainer est simple et se déroule ainsi :

1. Création d'un volume « portainer_data » sur votre machine hôte :

`docker volume create portainer_data`

2. Création du conteneur « portainer-ce » :

Attention, pour un accès depuis l'extérieur, vous devrez ouvrir sur votre pare-feu le port « 9443 » et cibler votre machine Debian qui contient le moteur Docker.

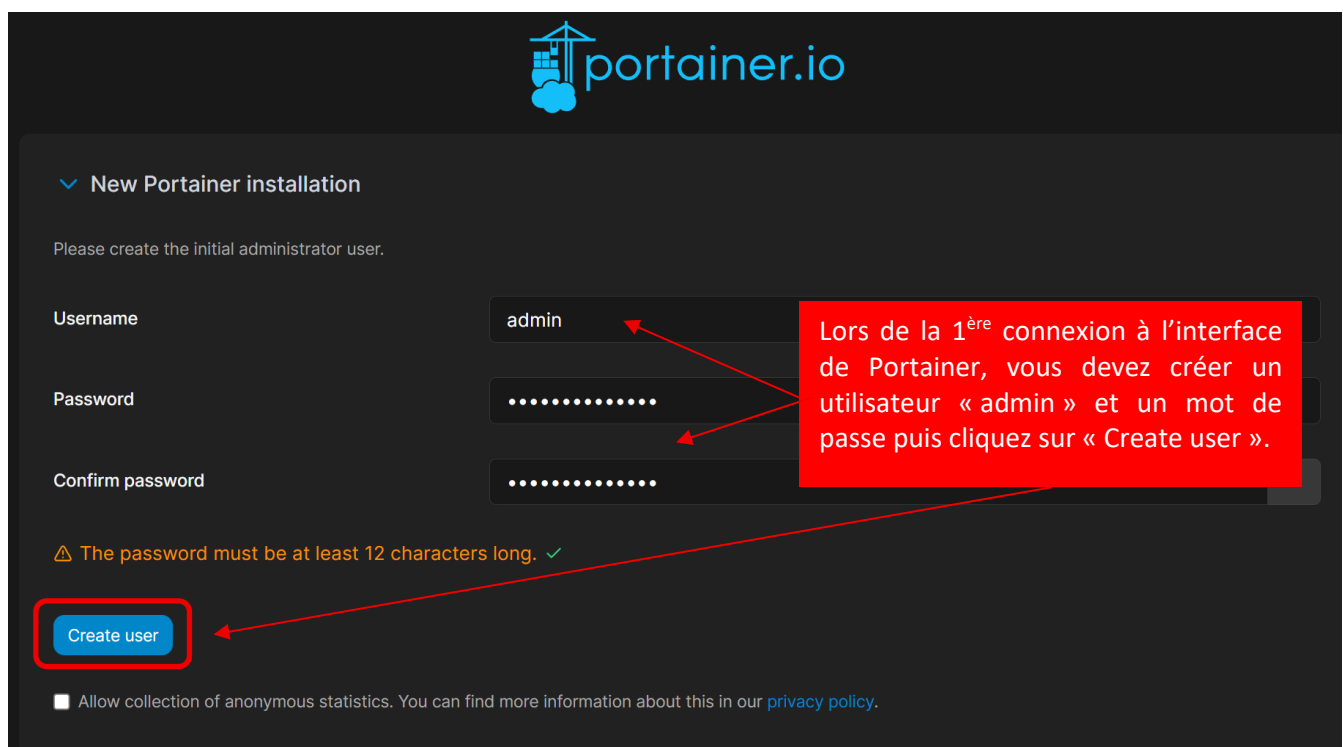
La commande à saisir pour installer PortainerCE est la suivante (la commande est à saisir sur 1 seule ligne !) :

```
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 :

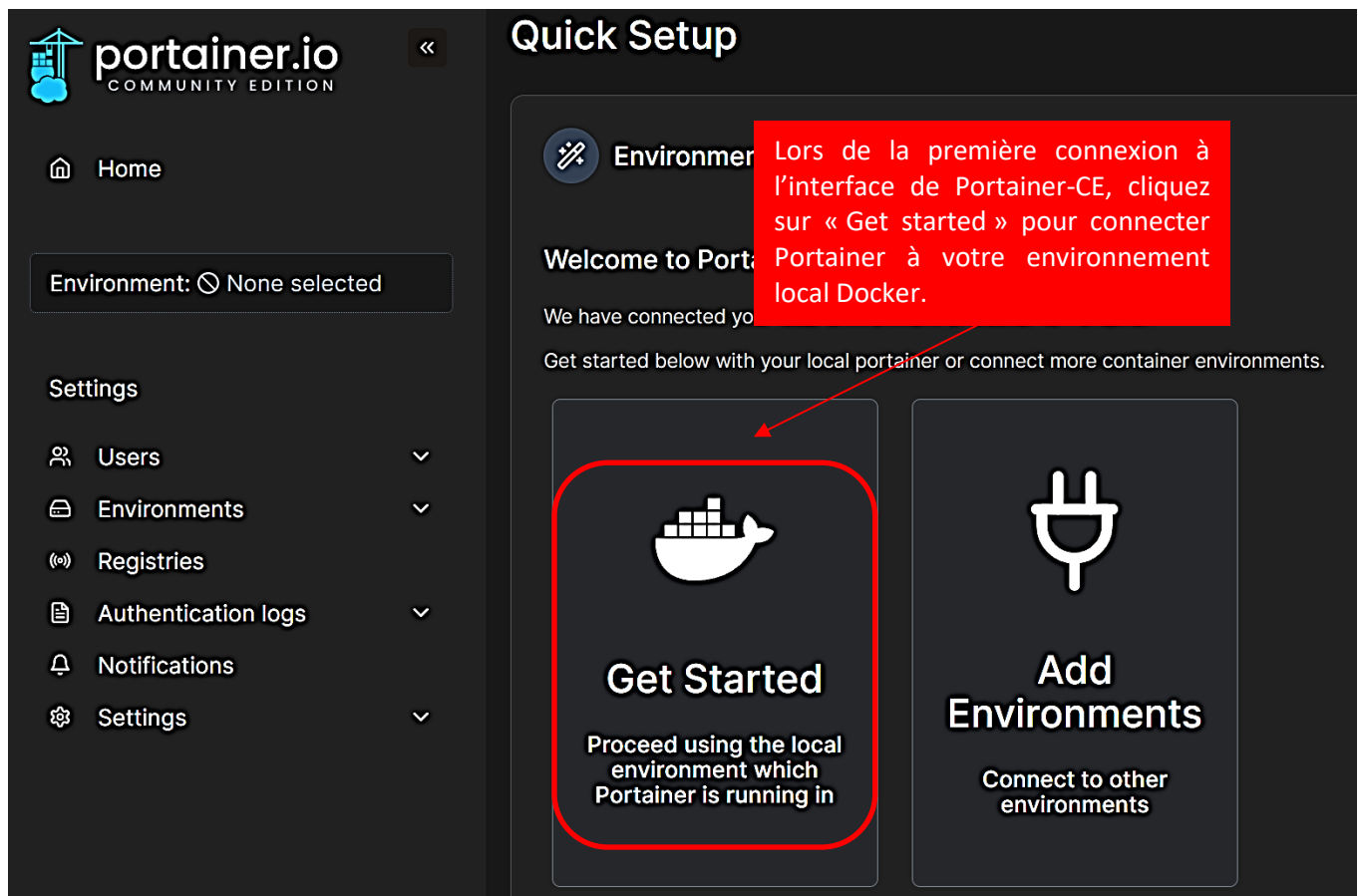
Pour accéder à Portainer, ouvrez, depuis une machine de votre réseau local, un navigateur et saisissez dans la barre d'adresse l'URL suivante : https://ip_debian_docker:9443

Un assistant s'ouvre :

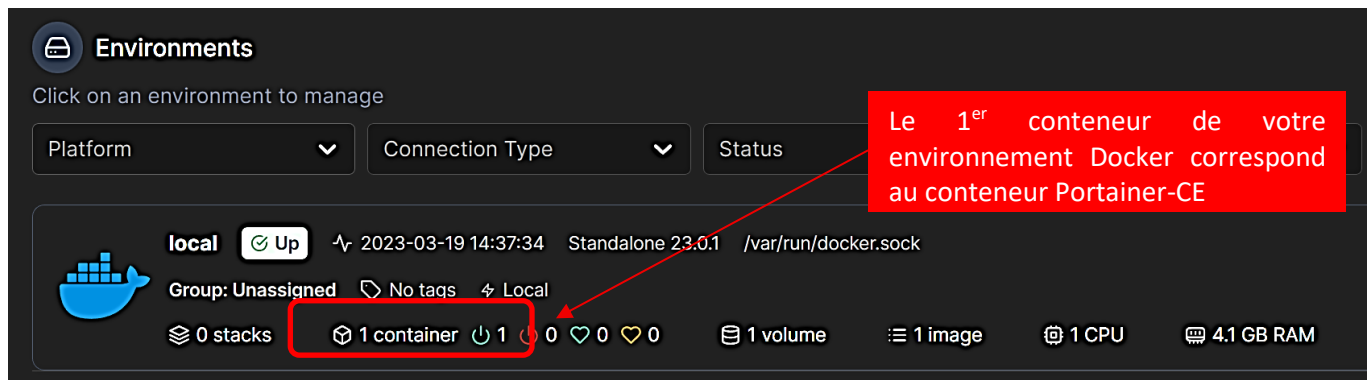


Attention, si vous ne voyez pas cette fenêtre s'afficher, il faudra supprimer et relancer votre conteneur Portainer CE car au-delà d'un certain délai, le conteneur n'est plus utilisable !

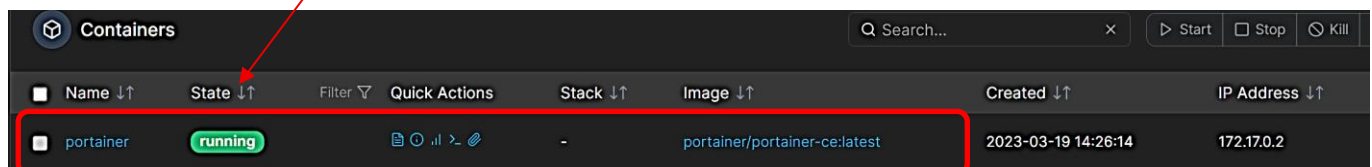
Dans la fenêtre suivante, cliquez sur « **Get started** » :



Votre environnement local s'affiche :



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). Il reste parfois nécessaire de maîtriser les commandes pour certaines opérations plus complexes.