

# DÉPLOIEMENT D'UN SERVICE AVEC DOCKER

## ACTIVITÉ 5 – DÉPLOIEMENT D'UN SERVICE AVEC DOCKER COMPOSE

### Liens :

- Lien officiel : <https://docs.docker.com/compose/>
- Lien vers une formation sur Docker complète mais très simple : <https://blog.microlinux.fr/formation-docker/>

### Les documents suivants viennent en complément :

- récapitulatif des commandes Docker Compose ;
- récapitulatif des commandes Docker ;
- les Instructions Dockerfile ;


## SOMMAIRE

A. Brève Présentation de Docker Compose.....	1
B. Un exemple de déploiement avec phpIPAM.....	2
1. Le fichier « compose.yaml ».....	3
2. Lancement des conteneurs.....	5
3. Finalisation de l'installation.....	6
C. Les commandes essentielles de Docker Compose.....	8
D. Déploiement de l'application NetworkManager.....	9

CUB offre de nombreux services applicatifs Web qui nécessitent l'accès à des bases de données relationnelles, par exemple :

- un site Web basé sur le CMS WordPress ;
- l'application phpIPAM ;
- des applications développées en interne.

Ces applications (comme les sites Web développés avec WordPress) nécessitent l'utilisation de plusieurs conteneurs.

 Même s'il est possible d'utiliser un seul conteneur avec tous les outils, Docker recommande de séparer chaque service dans un conteneur différent. Il sera ainsi possible de rendre l'environnement plus modulaire en permettant de tester plusieurs versions de service Web, de PHP et/ou d'une base de données (MariaDB et Postgres par exemple).

Dans une architecture « micro-services », cette situation est très courante avec beaucoup de conteneurs liés à lancer et les commandes sont parfois très longues et compliquées.

**La technologie Docker Compose permet de réaliser cette tâche très facilement. Docker Compose simplifie le processus de gestion de plusieurs conteneurs en permettant de décrire les services, les réseaux et les volumes de notre application dans un seul fichier YAML.**

**Vous procédez au déploiement des applications « phpIPAM » et NetworkManager.**

## A. BRÈVE PRÉSENTATION DE DOCKER COMPOSE

Docker Compose permet la définition de l'architecture de l'application et l'orchestration simple des conteneurs en :

- définissant l'ensemble des services (appelé “**stack**”) qui composent une application dans un fichier YAML “docker-compose.yml” afin qu'ils puissent être exécutés ensemble dans un environnement isolé et sécurisé ;
- permettant de piloter en une seule commande le lancement de tous les conteneurs décrits dans ce fichier, leur arrêt, etc.

Docker Compose automatise la mise en œuvre de plusieurs conteneurs en lien les uns avec les autres.

 De nombreuses applications connues (comme GLPI, phpIPAM et WordPress) sont fournies sous forme de « stack » via Docker Compose. Il est donc intéressant d'en comprendre le fonctionnement.

L'installation de Docker a installé automatiquement le paquet nécessaire docker-compose-plugin.

Comme nous allons le voir page suivante, le fichier « compose.yaml » va permettre de créer 3 conteneurs. Il contient tous les paramètres de connexion à la base de données, de l'application web (avec le *mapping* des ports) ainsi que les éléments permettant de créer les volumes persistants. Les images se basent sur des images disponibles sur le docker hub. Il est bien évidemment possible de modifier les valeurs pour les faire correspondre à l'infrastructure voulue.

Docker Compose utilise un fichier YAML (YAML Ain't Markup Language) pour définir les services, les réseaux et les volumes de l'application. Cela permet de structurer la configuration en un seul endroit d'une manière facile à lire et à comprendre. Cela facilite grandement la gestion et le déploiement de l'application de manière cohérente et reproductible.

**Le contenu du fichier docker-compose.yml utilise donc la [syntaxe yaml](#)** dont les points utiles dans cette initiation sont résumés ci-dessous :

- la syntaxe **impose une indentation** (nombre de caractères d'espacement au début de la ligne – j'utilise de manière arbitraire 2 caractères) qui marque une arborescence. **Cette indentation doit être absolument respectée pour que le fichier soit interprété correctement ;**
- les commentaires sont signalés par le signe dièse (#) (si par contre le dièse apparaît dans une chaîne, il signifie alors un nombre littéral) ;
- les éléments de listes sont dénotés par le tiret (-), suivi d'une espace ;
- les tableaux sont de la forme clé: valeur, à raison d'un couple par ligne ;
- les scalaires peuvent être entourés de guillemets doubles ("), ou simples ('), sachant qu'un guillemet s'échappe avec un antislash (\), alors qu'une apostrophe s'échappe avec une autre apostrophe.



Vous trouverez sur Internet un nombre conséquent de ressources sur Docker-Compose (avec un tiret au lieu de l'espace). Il s'agit de la version 1 de Docker Compose développée en Python qui n'est plus maintenue depuis le mois de juillet 2023.

La dernière version de Docker Desktop regroupe la plate-forme Docker Engine et Docker CLI, y compris Compose v2 a été développée en Go.

Une forte rétro-compatibilité existe. La plupart des fichiers YAML que vous trouverez sur Internet fonctionneront avec Docker Compose v2.

**La commande pour gérer les conteneurs est différente : docker compose (sans le trait d'union).**

## B. UN EXEMPLE DE DÉPLOIEMENT AVEC PHPIPAM

phpIPAM est une application de gestion d'adresses IP open source basée sur PHP et MySQL. Il est possible, via Docker Compose, de configurer et déployer phpIPAM ainsi que ses services requis, comme la base de données, le serveur Web et le gestionnaire de tâche automatique.

### 1. LE FICHIER « COMPOSE.YAML »

➤ Créer dans « contexte » un dossier « dc\_phpipam ».

➤ Créer dans le dossier « php\_ipam » le fichier compose.yaml suivant élaboré à partir de <https://hub.docker.com/r/phpipam/phpipam-www>.



Il est fortement conseillé d'utiliser Vscode pour créer le fichier compose.yaml car votre syntaxe sera automatiquement corrigée.

```
1  services:
2    phpipam-web:
3      image: phpipam/phpipam-www:latest
4      container_name: phpipam-web
5      ports:
6        - "8099:80"
7        - "44399:443"
8      environment:
9        - TZ=Europe/Paris
10       - IPAM_DATABASE_HOST=phpipam-mariadb
11       - IPAM_DATABASE_PASS=mysql&Ipam0
12       - IPAM_DATABASE_WEBHOST=%
13      restart: unless-stopped
14      volumes:
15        - phpipam-logo:/phpipam/css/images/logo
16        - phpipam-ca:/usr/local/share/ca-certificates:ro
17      depends_on:
18        - phpipam-mariadb
19
20    phpipam-cron:
21      image: phpipam/phpipam-cron:latest
22      container_name: phpipam-cron
23      environment:
24        - TZ=Europe/Paris
25        - IPAM_DATABASE_HOST=phpipam-mariadb
26        - IPAM_DATABASE_PASS=mysql&Ipam0
27        - SCAN_INTERVAL=1h
28      restart: unless-stopped
29      volumes:
30        - phpipam-ca:/usr/local/share/ca-certificates:ro
31      depends_on:
32        - phpipam-mariadb
33
34    phpipam-mariadb:
35      image: mariadb:latest
36      container_name: phpipam-db
37      environment:
38        - MYSQL_ROOT_PASSWORD=mysql&Root0
39      restart: unless-stopped
40      volumes:
41        - phpipam-db-data:/var/lib/mysql
42
43  volumes:
44    phpipam-db-data:
45    phpipam-logo:
46    phpipam-ca:
```

Le fichier compose.yaml permet de **définir les services** qui composent l'application. Dans notre cas, nous souhaitons trois services : **phpipam-web**, **phpipam-cron** et **phpipam-mariadb**. **Ces noms seront automatiquement les noms d'hôte du conteneur (et NON le nom des conteneurs) et sont utilisés pour les liens entre eux.** Chaque service représente un conteneur, dont le nom est choisi arbitrairement, qui s'exécutera dans le cadre de l'application.

**Au sein de chaque service**, il est nécessaire de définir les différentes propriétés qui spécifient comment le conteneur doit être configuré et se comporter.

#### Description du service « **phpipam-web** » qui fournit l'interface graphique grâce au serveur web Apache2 :

- **utilisation de la dernière image officielle** (phpipam/phpipam-www:latest) présente sur le « docker hub » avec un nom de conteneur « **phpipam-web** ».
- **mappage des ports** (équivalent à l'option -p de la commande « docker » qui permet une correspondance des ports entre le conteneur et la machine hôte. Les ports 80 (http) et 443 (https) sont utilisés à l'intérieur du conteneur Docker et sont également exposés. Ici, les ports 8099 et 44399 doivent donc être utilisés pour se connecter à l'application Web.
- **utilisation de 4 variables d'environnement** nécessaires pour le fonctionnement de l'application, telles que le fuseau horaire, l'hôte de la base de données, le mot de passe de la base de données et l'hôte web ;
- **restart: unless-stopped** définit un service qui redémarrera automatiquement sauf s'il est explicitement arrêté ;
- **déclaration de 2 volumes** tels que le logo de phipam et les certificats CA permettant l'utilisation de données persistantes (équivalent à l'option -v de la commande « docker »). Ils sont créés (lorsqu'ils sont déclarés dans la section « volumes ») dans « /var/lib/docker/volumes/ ».
- **depends\_on** précise les dépendances entre services. Cela garantit que les conteneurs démarrent dans le bon ordre. Dans notre cas, phipam-web dépend de phipam-mariadb, donc le conteneur de base de données démarrera avant.

#### Description du service « **phpipam-cron** » qui est un service supplémentaire utilisé pour exécuter des tâches planifiées (la découverte de réseau) dans l'application :

- **utilisation de la dernière image officielle** (phpipam/phpipam-cron:latest) présente sur le « docker hub » avec un nom de conteneur « **phpipam-cron** » ;
- **utilisation de 4 variables d'environnement** nécessaires pour le fonctionnement de l'application, telles que le fuseau horaire, l'hôte de la base de données, le mot de passe de la base de données et l'intervalle entre deux scan du réseau ;
- **restart: unless-stopped** définit un service qui redémarrera automatiquement sauf s'il est explicitement arrêté ;
- **déclaration d'un volume** utilisé pour stocker les fichiers de certificat ;
- **depends\_on** précise les dépendances entre services. Cela garantit que les conteneurs démarrent dans le bon ordre. Dans notre cas, phipam-cron dépend de phipam-mariadb, donc le conteneur de base de données démarrera avant.

#### Description du service « **mariadb** » :

- **utilisation de la dernière image officielle** (mariadb:latest) présente sur le « docker hub » avec un nom de conteneur « **phpipam-db** » (nous choisissons arbitrairement ce nom) ;
- **utilisation d'une seule variable d'environnement** qui configure le mot de passe de l'utilisateur root qui est l'administrateur de la base de données ;
- **restart: unless-stopped** définit un service qui redémarrera automatiquement sauf s'il est explicitement arrêté ;
- **déclaration d'un volume** utilisé pour stocker les données de la base de données.
- **Les volumes** créés dans « /var/lib/docker/volumes/ » doivent être déclarés dans une section à part.

## 2. LANCEMENT DES CONTENEURS

À partir du répertoire contenant le fichier « compose.yaml », il faut lancer la commande :

```
docker compose up -d
```

L'option « -d » permet de reprendre la main sur la console.

```
user@servDockerAR:~/contexte/dc_phpipam$ docker compose up -d
[+] Running 7/16
  ⚙️ phpipam-mariadb 8 layers [#####] 48.05MB/92.83MB Pulling
    ✓ 44ba2882f8eb Pull complete
    ✓ 08b8223d0cb6 Pull complete
    ⚙️ ef2696fb09d6 Extracting [=====] 5.593MB/5.593MB
    ✓ 6ae32c298a0d Download complete
    ✓ 5dc97cb97b44 Download complete
    ⚙️ a3e4bee69a58 Downloading [=====] 42.46MB/87.24MB
    ✓ b29c582204c9 Download complete
    ✓ f05405b8aaed Download complete
  ⚙️ phpipam-cron 1 layers [ ] 0B/0B Pulling
    ⚙️ 4f4fb70ef54 Waiting
  ⚙️ phpipam-web 4 layers [###] 17.05MB/29.49MB Pulling
    ✓ 3695f6c099b8 Pull complete
    ⚙️ fa97c8bbab15 Downloading [=====] 17.05MB/29.49MB
    ⚙️ 39faeed0bd5e Waiting
```

Au premier lancement, il faut attendre que les images soient téléchargées. Docker Compose a un comportement par défaut. Nous voyons à la dernière étape que :

- Docker crée un réseau par défaut (dc\_phpipam\_default) ;
- les volumes sont créés (avec le nom prévu dans le fichier préfixé par le nom du dossier dans lequel se trouve le fichier « yaml »).
- les conteneurs sont créés.

```
[+] Running 7/7
✓ Network dc_phpipam_default Created 0.3s
✓ Volume "dc_phpipam_phpipam-db-data" Created 0.0s
✓ Volume "dc_phpipam_phpipam-logo" Created 0.0s
✓ Volume "dc_phpipam_phpipam-ca" Created 0.0s
✓ Container phpipam-db Started 6.2s
✓ Container phpipam-cron Started 2.0s
✓ Container phpipam-web Started 2.2s
```

Les commandes « docker ps » et « docker compose ps » le confirment :

```
user@servDockerAR:~/contexte/dc_phpipam$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
66344d0d42fb   phpipam/phpipam-www:latest         "/sbin/tini -- /bin/..." 7 minutes ago Up 7 minutes  0.0.0.0:8099->80/tcp, :::8099->80/tcp, 0.0.0.0:44399->443/tcp, :::44399->443/tcp
ebb47c6bb11    phpipam/phpipam-cron:latest        "/sbin/tini -- /bin/..." 7 minutes ago Up 7 minutes  80/tcp
e40fed8c38e    mariadb:latest                     "docker-entrypoint.s..." 7 minutes ago Up 7 minutes  3306/tcp
c5a07eaa2d16   aporaf/kalilinux:core              "/bin/bash /startkal..." 4 days ago    Up 4 days     0.0.0.0:13389->13389/tcp, :::13389->13389/tcp, 0.0.0.0:20022->20022/tcp, :::20022->20022/tcp
```

```
user@servDockerAR:~/contexte/dc_phpipam$ docker compose ps
NAME                IMAGE                                SERVICE      CREATED        STATUS        PORTS
phpipam-cron        phpipam/phpipam-cron:latest        phpipam-cron 19 minutes ago Up 19 minutes 80/tcp
phpipam-db          mariadb:latest                     phpipam-mariadb 19 minutes ago Up 19 minutes 3306/tcp
phpipam-web         phpipam/phpipam-www:latest         phpipam-web   19 minutes ago Up 19 minutes 0.0.0.0:8099->80/tcp, :::8099->80/tcp, 0.0.0.0:44399->443/tcp, :::44399->443/tcp
```

**i** Docker Compose accepte de nombreuses commandes permettant de gérer les conteneurs (redémarrer les services, stopper/supprimer les conteneurs, etc.). Ces commandes doivent être exécutées dans le dossier contenant le fichier « yaml ».



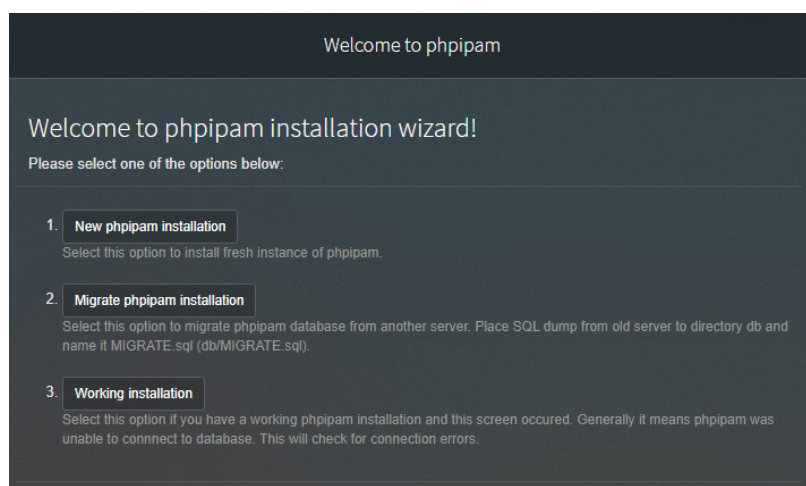
### 3. FINALISATION DE L'INSTALLATION

- Écrire ci-dessous les URL permettant d'accéder en HTTP et en HTTPS à l'application phpIPAM.



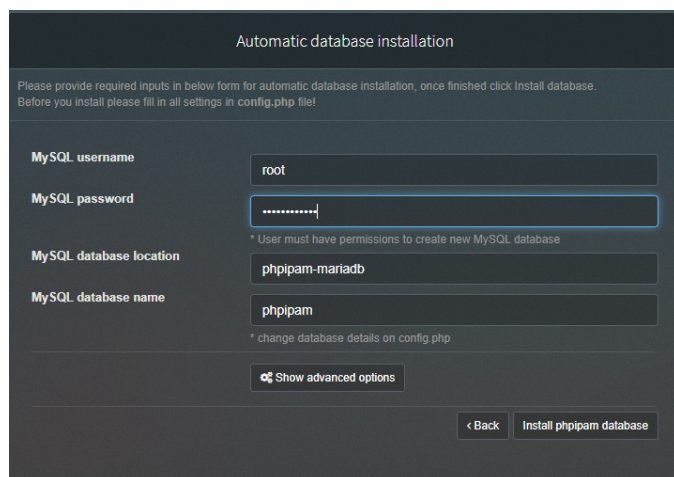
Une fois les containers lancés, se rendre en HTTPS sur l'interface Web.

- Cliquer sur « New phipam installation »



- Cliquer sur « automatic database installation » et saisir les valeurs demandées.

Comme précisé, il s'agit ici du compte root qui a l'autorisation de créer une base de données.



Automatic database installation

Please provide required inputs in below form for automatic database installation, once finished click Install database. Before you install please fill in all settings in config.php file!

MySQL username: root

MySQL password: .....

MySQL database location: phpipam-mariadb

MySQL database name: phpipam

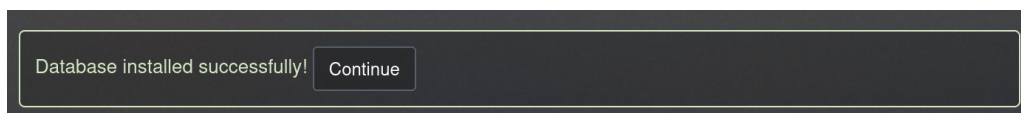
\* User must have permissions to create new MySQL database

\* change database details on config.php

Show advanced options

< Back Install phpipam database

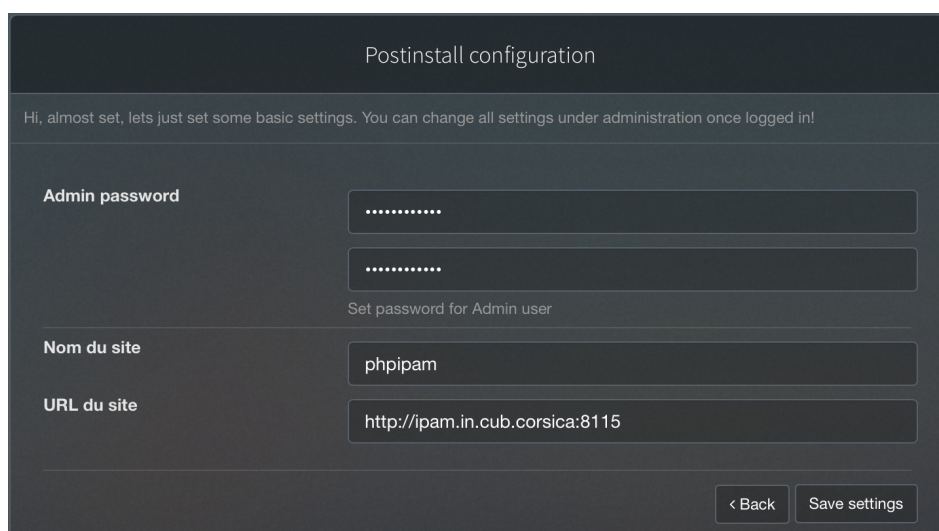
➤ Cliquer sur « Install phpipam database ».



Database installed successfully! Continue

➤ Cliquer sur « Continue ».

➤ Définir le mot de passe (par exemple phpipamAdmin0) pour le compte administrateur (admin) de l'interface ainsi que l'URL de l'application (à adapter) :



Postinstall configuration

Hi, almost set, lets just set some basic settings. You can change all settings under administration once logged in!

Admin password: .....

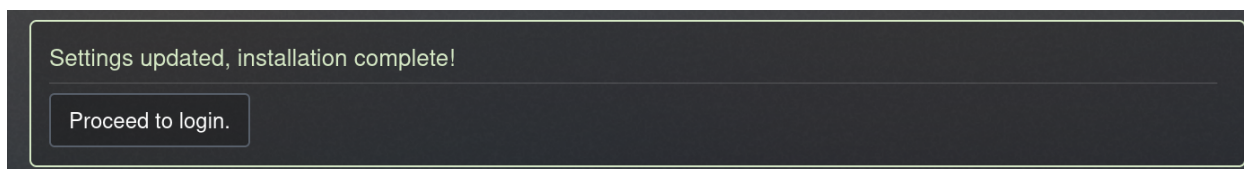
Set password for Admin user

Nom du site: phpipam

URL du site: http://ipam.in.cub.corsica:8115

< Back Save settings

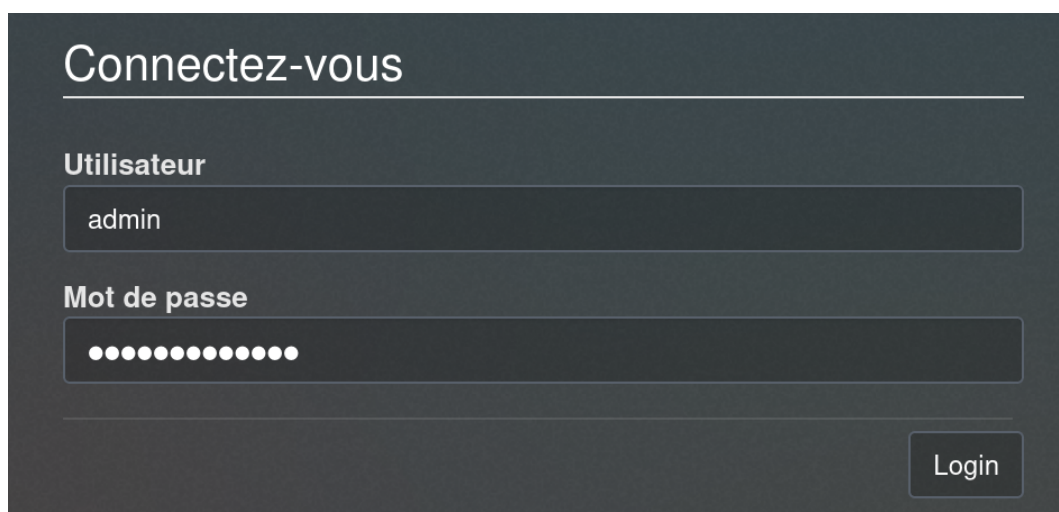
➤ Cliquer sur « Save setting »



Settings updated, installation complete!

Proceed to login.

L'installation est terminée et l'authentification est maintenant disponible. Elle se fait à partir du compte local « admin » dont le mot de passe est défini à l'installation de phpIPAM.



 Nous n'irons pas plus loin dans la configuration de phpIPAM, car il est prévu de configurer cet outil dans une autre activité.

## C. LES COMMANDES ESSENTIELLES DE DOCKER COMPOSE

Pour supprimer les conteneurs et le réseau associé



```
root@servDockerAR:/home/user/contexte/dc_phpipam# docker compose down
[+] Running 4/4
✓ Container phpipam-web      Removed
✓ Container phpipam-cron     Removed
✓ Container phpipam-db       Removed
✓ Network dc_phpipam_default Removed
```



Les volumes ne sont pas supprimés. Si on recrée l'application (docker compose up -d), nous retrouvons les données.

**docker compose <stop | start | restart >** arrête ou démarre ou redémarre l'ensemble des conteneurs.

**docker compose <stop | start | restart > <nom\_d'un\_service>** arrête ou démarre ou redémarre le conteneur correspondant.

**docker compose logs -f** affiche les logs des services et continue à les « écouter » sans rendre la main.

**docker compose logs -f <nom\_d'un\_service>** retourne les logs du service correspondant et continue à les « écouter » sans rendre la main.



Vous trouverez les autres commandes dans le fichier mis à disposition.

## D. DÉPLOIEMENT DE L'APPLICATION NETWORKMANAGER

L'application Web NetworkManager est en cours de développement. Elle permettra dans sa première version, via une interface graphique, la création, la suppression et la modification d'entrées DNS sur le domaine de CUB. Elle a également vocation de simplifier l'administration de tous les autres services offerts (comme le VPN).

Les développeurs qui travaillent sur cette application doivent pouvoir la déployer rapidement dans leur environnement de développement et peuvent être amenés à tester (tout en conservant les données) plusieurs environnements (comme une version différente de PHP).

Il a donc été décidé de déployer cette application avec Docker et Docker Compose.

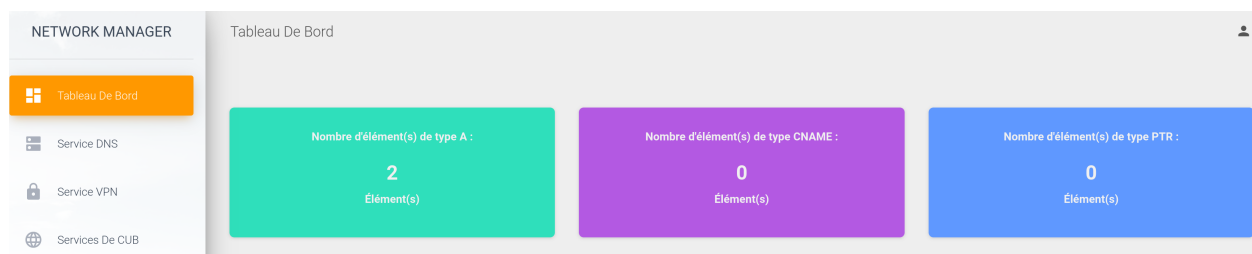
### Devoir à rendre

**En vous appuyant sur les éléments présentés dans le fichier « installNetworkManager.pdf » et sur les éléments présentés ci-dessous et page suivante vous devez déployer l'application NetworManager accessible en HTTPS.**

**Interface avant de s'authentifier :**



**Interface après l'authentification :**



**Le principe du déploiement :** deux services à déployer, un pour l'interface web et un pour la base de données MySQL.

**Le service pour l'interface Web** nécessite une **image personnalisée** via un Dockerfile (nom de l'image : ID\_Docker/nm-debian12-web:1.0) basée sur l'image officielle de debian 12 (debian12:latest) et qui intègre :

- tous les paquets nécessaires ;
- la configuration du VirtualHost permettant l'accès au site ;
- la mise en place des sources. Les sources de l'application doivent être persistantes (si l'on supprime le conteneur et qu'on lance une nouvelle instance, on doit pouvoir retrouver les sources et les modifications que l'on aurait faites sur ces sources) ;
- le fichier « .env » nécessaire à l'application NetworkManager (avec les valeurs des variables DB\_HOST et DB\_PASSWORD mises à jour) ;
- la possibilité de se connecter en SSH et la création de l'utilisateur correspondant.

**Le service pour la base de données** nécessite également une **image personnalisée** (nom de l'image : ID\_Docker/nm-mariadb:1.0) basée sur l'image officielle de mariadb (mariadb:latest) qui intègre les données de l'application. L'initialisation de la base de données fonctionne de la façon suivante :

- le script SQL fourni « network\_manager.sql » contient tous les éléments nécessaires (création de la base de données, des tables, des contraintes ainsi que les commandes SQL d'insertion des données) ;
- ce script SQL doit être copié dans /docker-entrypoint-initdb.d : l'image « mariadb » prévoit, à la création du conteneur, le lancement du fichier « entrypoint.sh » (présent à la racine du conteneur) qui, lui-même, charge tous les scripts du répertoire /docker-entrypoint-initdb.d ;
- les fichiers correspondants aux bases de données MySQL sont dans le dossier /var/lib/mysql du conteneur. De la même façon que pour le code source, ces données doivent être persistantes (si l'on supprime le conteneur et qu'on lance une nouvelle instance, on doit pouvoir retrouver les données et les modifications que l'on aurait faites) ;
- la possibilité de se connecter en SSH et la création de l'utilisateur correspondant.

### Remarques générales :

- le mot de passe de l'utilisateur qui aura accès au serveur ssh de chaque service doit pouvoir être changé au niveau du Docker Compose ;
- toutes les variables d'environnement de l'application Web sont définies via le fichier « .env » à la racine du projet : la section « environment » au niveau du service pour l'interface Web dans Docker Compose ne contiendra que la variable d'environnement nécessaire compte tenu de la remarque précédente ;
- Bonus :) : les Dockerfile et le fichier nécessaire à Docker Compose peuvent ne pas intégrer de mots de passes (voir « Utilisation des secrets dans Docker Compose » page suivante).

### Débogage :

Ne pas oublier que des volumes pour les données persistantes ont été créés. Si l'application n'est pas fonctionnelle et que vous devez à nouveau refaire les images pour déployer l'application, il vaut mieux ne pas se contenter de la commande « docker compose down » mais aussi de supprimer les deux volumes créés.

### Productions attendues :

- L'accès à l'application NetworkManager opérationnelle.
- La documentation technique intégrant les deux fichiers Dockerfile et le fichier Docker Compose.