Doc environnement de développement

Auteur: REICHHART Damien - BENAD Kilian
Date de création: 23/04/2025 **Dernière modification**: 23/04/2025

Documentation Technique Complète pour l'Environnement de Développement

Introduction

L'objectif de cette documentation est de fournir une vue d'ensemble exhaustive et détaillée de l'environnement de développement configuré avec Docker et orchestré à l'aide de plusieurs fichiers Docker Compose et Dockerfiles. Cet environnement comprend plusieurs services essentiels pour le développement et le déploiement d'applications web, notamment Apache, PHP-FPM, MariaDB, PhpMyAdmin, et un environnement Docker in Docker (DIND). Ces composants sont intégrés de manière fluide afin de garantir une productivité accrue, une sécurité renforcée, et une gestion efficace des ressources.

Cet environnement a été conçu avec un souci particulier pour la sécurité, la modularité et l'extensibilité, permettant une adaptation facile à différents contextes de développement ou de production. Dans ce rapport, nous allons explorer chaque composant en détail, discuter des configurations réseau, des volumes, des tests automatisés, et des pratiques recommandées pour maintenir un environnement de développement fiable et performant.

Sommaire

---

1. Configuration Réseau

1.1. Vue d'ensemble des réseaux Docker

Docker Compose permet la création de réseaux virtuels isolés, ce qui est crucial pour séparer les différents services et garantir la sécurité et l'efficacité de la communication entre eux. Dans cette configuration, deux réseaux sont définis :

- **Réseau** `front` : Ce réseau est principalement utilisé pour les services qui nécessitent une exposition au monde extérieur, comme le serveur Apache. Il permet l'accès au site web via les ports 80 (HTTP) et 443 (HTTPS).
- **Réseau** `back` : Ce réseau est utilisé pour la communication interne entre les services tels que PHP-FPM, MariaDB, DIND, et d'autres. Il permet une meilleure isolation des services, limitant l'exposition aux menaces extérieures.

1.1.1. Pourquoi utiliser plusieurs réseaux ?

L'utilisation de plusieurs réseaux dans Docker présente plusieurs avantages :

- **Sécurité** : Le réseau `back` n'est pas directement accessible depuis l'extérieur, minimisant ainsi les risques de compromission des services internes comme la base de données.
- **Contrôle précis des flux** : En isolant les services dans différents réseaux, nous pouvons contrôler exactement quelles interactions sont autorisées entre les services. Par exemple, MariaDB n'est accessible qu'à partir de PHP ou de PhpMyAdmin via le réseau `back`.
- **Gestion simplifiée** : Avec une configuration réseau segmentée, il devient plus facile de gérer les adresses IP, les règles de pare-feu, et de diagnostiquer les problèmes de connectivité.

1.2. Configuration du réseau frontal (`front`)

Le réseau `front` utilise un **driver bridge** et est configuré avec le sous-réseau `192.168.10.0/24`, avec la passerelle définie sur `192.168.10.1`. Ce réseau est principalement utilisé par Apache et PhpMyAdmin, les deux services ayant besoin d'une exposition externe.

- **Apache** est assigné à l'adresse IP `192.168.10.10` dans ce réseau. Il est configuré pour écouter sur les ports 80 et 443, ce qui permet aux utilisateurs d'accéder au site web en HTTP et en HTTPS.
- **PhpMyAdmin** utilise l'adresse IP `192.168.10.20` dans ce réseau et est exposé sur le port 8080 pour permettre une gestion facile de la base de données MariaDB via une interface web.

1.3. Configuration du réseau back(`back`)

Le réseau `back` est également un **bridge** Docker, mais son sous-réseau est `192.168.20.0/24` avec la passerelle `192.168.20.1`. Ce réseau est exclusivement utilisé pour la communication entre les services qui ne doivent pas être exposés au monde extérieur, comme PHP-FPM, MariaDB, DIND, etc.

Voici comment les services sont assignés dans ce réseau :

- **PHP-FPM** est associé à l'adresse IP `192.168.20.20`. Ce service communique avec Apache via un socket Unix pour gérer l'exécution du code PHP.
- **MariaDB** est sur l'IP `192.168.20.30`. Ce service n'est accessible qu'à partir de PHP-FPM ou de PhpMyAdmin, et ne peut être joint directement depuis l'extérieur, renforçant ainsi la sécurité de la base de données.
- **DIND** (Docker in Docker) utilise l'adresse `192.168.20.40`, facilitant les tâches liées à Docker, comme le CI/CD, tout en restant isolé du réseau frontal.

En segmentant les réseaux de cette manière, l'environnement de développement devient non seulement plus sécurisé, mais également plus flexible en cas de besoin d'ajout de nouveaux services à l'avenir.

```
networks:
  front:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.10.0/24
          gateway: 192.168.10.1
  back:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.20.0/24
          gateway: 192.168.20.1
```

---

2. Services

L'environnement Docker compose plusieurs services essentiels à la création d'un environnement de développement web complet. Ces services incluent des serveurs web, des interpréteurs PHP, des bases de données, et des outils de gestion.

2.1. Apache

Apache est le serveur HTTP principal utilisé dans cet environnement. Il est responsable de servir les fichiers statiques et d'agir comme proxy pour les requêtes dynamiques dirigées vers PHP-FPM. Voici un aperçu plus

détaillé de sa configuration et de ses caractéristiques :

2.1.1. Base Image : httpd

L'image utilisée pour Apache est basée sur **httpd**

, une version légère d'Apache basée sur Alpine Linux, ce qui permet de réduire la taille de l'image Docker et d'améliorer les performances.

2.1.2. Modules activés

La configuration d'Apache a été optimisée en activant uniquement les modules nécessaires au bon fonctionnement du serveur. Parmi ces modules :

- **mod_deflate** : Ce module est utilisé pour compresser les réponses HTTP, réduisant ainsi la bande passante nécessaire pour servir les fichiers et améliorant les temps de chargement.
- **mod_proxy** et **mod_proxy_fcgi** : Ces modules permettent à Apache de servir de proxy pour les requêtes FastCGI dirigées vers PHP-FPM. Cela permet de séparer le serveur web du traitement PHP, une architecture courante pour améliorer les performances.
- **mod_ssl** : Ce module est essentiel pour chiffrer les communications via HTTPS.

2.1.3. Sécurisation d'Apache

La sécurité d'Apache est une priorité, et plusieurs mesures ont été prises pour durcir la configuration du serveur :

- **Utilisation d'un utilisateur non-root** : Apache est exécuté en tant qu'utilisateur `apache` avec UID 1010, ce qui réduit les risques liés à l'exécution de services avec des privilèges élevés.
- **En-têtes de sécurité HTTP** : Des en-têtes tels que `X-Content-Type-Options`, `X-XSS-Protection`, `Strict-Transport-Security` et `X-Frame-Options` sont configurés pour protéger contre diverses attaques web (e.g., injection MIME, attaques XSS, clickjacking).
- **Certificats SSL** : Les certificats SSL sont installés et configurés pour permettre une communication sécurisée via HTTPS. Le fichier de configuration inclut les chemins vers les certificats et la clé privée, avec des permissions strictes (chmod 600) pour empêcher tout accès non autorisé.
- **Utilisation de crowdsec** : Pour une sécurité accrue, une solution de pare feu applicatif à été déployée afin de garantir une sécurité optimal en bloquant les comportements suspects

2.1.4. Hôtes Virtuels et Redirection HTTPS

Le fichier de configuration `apache.chost.conf` définit deux hôtes virtuels :

- **VirtualHost sur le port 80** : Ce hôte redirige automatiquement tout le trafic HTTP vers HTTPS.
- **VirtualHost sur le port 443** : Ce hôte gère le trafic HTTPS et utilise les certificats SSL pour sécuriser les communications. Il est également configuré pour fonctionner avec PHP-FPM, en utilisant un socket Unix pour proxy les requêtes PHP.

```
apache:
    build:
      context: ./docker
      dockerfile: ./apache.dockerfile
```

```
    container_name: apache
    ports:
      - 80:80
      - 443:443
    restart: unless-stopped
    volumes:
      - .:/var/www/html
      - php-socket:/var/run
    depends_on:
      - db
      - php
      - dind
    networks:
      front:
        ipv4_address: 192.168.10.10
      back:
        ipv4_address: 192.168.20.10
```

```
# Base image with Alpine (minimal, lightweight)
FROM httpd:alpine

ARG UID=1010
ARG GID=1010

RUN apk add nano

# Install the necessary packages
RUN apk add --no-cache shadow

# Security: Avoid running as root; create a non-root user and group for Apache to
run under
RUN addgroup -g 1010 apache && adduser -u 1010 -G apache -S apache

# Create the socket right group and assign the user
RUN addgroup --gid 1020 socket
RUN usermod -a -G socket apache

# Copy Apache virtual host configuration to the container
COPY apache/config/apache.vhost.conf
/usr/local/apache2/conf/extra/apache.vhost.conf

# Enable Apache modules to ensure proper functionality
RUN sed -i \
    # Uncomment the configuration for mod_deflate to enable compression
    -e '/#LoadModule deflate_module/s/^#//g' \
    # Uncomment the configuration for mod_proxy to enable proxying capabilities
    -e '/#LoadModule proxy_module/s/^#//g' \
    # Uncomment the configuration for mod_proxy_fcgi to enable FastCGI proxy
module
    -e '/#LoadModule proxy_fcgi_module/s/^#//g' \
    # Uncomment the configuration for mod_proxy_unix to enable UNIX domain sockets
```

```
    -e '/#LoadModule proxy_unix_module/s/^#//g' \
    # Uncomment the configuration for mod_rewrite to enable URL rewriting
    -e '/#LoadModule rewrite_module/s/^#//g' \
    /usr/local/apache2/conf/httpd.conf

# Enable mod_ssl and other required modules
RUN sed -i '/#LoadModule ssl_module/s/^#//g' /usr/local/apache2/conf/httpd.conf \
    && sed -i '/#LoadModule socache_shmcb_module/s/^#//g'
/usr/local/apache2/conf/httpd.conf

# Security: Enable security-related headers and protection mechanisms
RUN echo "LoadModule headers_module modules/mod_headers.so" >>
/usr/local/apache2/conf/httpd.conf \
    && echo "Header always set X-Content-Type-Options \"nosniff\"" >>
/usr/local/apache2/conf/httpd.conf \
    && echo "Header always set X-XSS-Protection \"1; mode=block\"" >>
/usr/local/apache2/conf/httpd.conf \
    && echo "Header always set Strict-Transport-Security \"max-age=31536000;
includeSubDomains\"" >> /usr/local/apache2/conf/httpd.conf \
    && echo "Header always set X-Frame-Options \"SAMEORIGIN\"" >>
/usr/local/apache2/conf/httpd.conf

# Copy SSL certificates and keys
COPY ./apache/certs/server.crt /usr/local/apache2/conf/server.crt
COPY ./apache/certs/server.key /usr/local/apache2/conf/server.key

# Include the virtual host configuration in Apache's main config
RUN echo "Include /usr/local/apache2/conf/extra/apache.vhost.conf" >>
/usr/local/apache2/conf/httpd.conf

# Set correct permissions for the certificates and key
RUN chmod 600 /usr/local/apache2/conf/server.key
/usr/local/apache2/conf/server.crt

# Change the Apache user and group in httpd.conf
RUN sed -i 's/User daemon/User apache/g' /usr/local/apache2/conf/httpd.conf && \
    sed -i 's/Group daemon/Group apache/g' /usr/local/apache2/conf/httpd.conf

# Security: Change ownership of necessary files to the non-root apache user
RUN mkdir /var/www
RUN chown -R apache:apache /var/www
RUN chown -R apache:apache /usr/local/apache2
RUN chmod -R 777 /var/run

# Security : install crowdsec agent and bouncer

RUN cd /tmp && \
    wget
https://github.com/crowdsecurity/crowdsec/releases/download/v1.6.3/crowdsec-
release.tgz && \
    tar xzf crowdsec-release* && \
    rm *.tgz && \
    apk add bash newt envsubst && \
    cd /tmp/crowdsec-v* && \
```

```
    # Docker mode skips configuring systemd
    ./wizard.sh --docker-mode && \
    cscli hub update && \
    # A collection is just a bunch of parsers and scenarios bundled together for
convienence
    apk add nftables && \
    echo "http://dl-cdn.alpinelinux.org/alpine/edge/testing" >>
/etc/apk/repositories && \
    apk update && \
    apk add cs-firewall-bouncer&& \
    cscli collections install crowdsecurity/linux && \
    cscli parsers install crowdsecurity/whitelists && \
    cscli collections install crowdsecurity/apache2 && \
    cscli collections install crowdsecurity/apiscp && \
    cscli collections install crowdsecurity/appsec-crs && \
    cscli collections install crowdsecurity/appsec-generic-rules && \
    cscli collections install crowdsecurity/appsec-virtual-patching && \
    cscli appsec-configs install crowdsecurity/appsec-default && \
    cscli collections install crowdsecurity/linux && \
    cscli collections install crowdsecurity/linux-lpe

# Set working directory
WORKDIR /var/www/html

# Expose port 80 (HTTP) and 443 (HTTPS)
EXPOSE 80 443

# Run Apache as the non-root user
USER apache
```

```
# Set the ServerName to localhost
ServerName localhost

LoadModule deflate_module /usr/local/apache2/modules/mod_deflate.so
LoadModule proxy_module /usr/local/apache2/modules/mod_proxy.so
LoadModule proxy_fcgi_module /usr/local/apache2/modules/mod_proxy_fcgi.so
LoadModule ssl_module /usr/local/apache2/modules/mod_ssl.so

Include conf/extra/httpd-ssl.conf

# Ensure required modules are loaded (already handled in the Dockerfile)

# Enable the rewrite engine
RewriteEngine on

# Redirect HTTP to HTTPS
<VirtualHost *:80>
    ServerName localhost
    Redirect permanent / https://localhost
</VirtualHost>
```

```
# Configure HTTPS VirtualHost
<VirtualHost *:443>
    # Enable SSL
    SSLEngine on

    # Specify the paths to your SSL certificate and key files
    SSLCertificateFile /usr/local/apache2/conf/server.crt
    SSLCertificateKeyFile /usr/local/apache2/conf/server.key

    # Proxy PHP requests to the PHP-FPM socket
    ProxyPassMatch ^/(.*\.php(/.*)?)$
unix:/var/run/php.sock|fcgi://localhost/var/www/html/$1

    # Set the DocumentRoot for the virtual host
    DocumentRoot /var/www/html/

    # Directory configuration for the DocumentRoot
    <Directory /var/www/html/>
        DirectoryIndex public/index.php
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Define the CustomLog and ErrorLog destinations
    CustomLog /proc/self/fd/1 common
    ErrorLog /proc/self/fd/2
</VirtualHost>
```

2.2. PHP-FPM

PHP-FPM (FastCGI Process Manager) est un gestionnaire de processus PHP hautes performances, spécialement conçu pour traiter les requêtes FastCGI. Dans cet environnement, PHP-FPM est utilisé pour traiter toutes les requêtes PHP envoyées par Apache.

2.2.1. Base Image : php:8.3-fpm

Le fichier `php.dockerfile` utilise l'image officielle `php:8.3-fpm`, qui contient PHP 8.3 avec PHP-FPM. Cette image est configurée pour traiter les scripts PHP via des sockets Unix, plutôt qu'en utilisant un port TCP, ce qui améliore les performances et la sécurité.

2.2.2. Gestion des utilisateurs et permissions

La gestion des utilisateurs est cruciale pour garantir un environnement sécurisé. Le fichier `php.dockerfile` crée deux groupes d'utilisateurs :

- **apache** : Ce groupe est utilisé par le serveur web Apache pour se connecter au socket PHP-FPM.
- **php** : Ce groupe est utilisé par PHP-FPM pour exécuter les scripts PHP.

Les fichiers de configuration PHP-FPM sont modifiés pour s'assurer que le socket Unix est accessible uniquement par l'utilisateur et le groupe `apache`, avec des permissions strictes (`0660`), ce qui réduit les risques d'accès non autorisé aux sockets.

2.2.3. Extensions PHP

Plusieurs extensions PHP importantes sont installées via le gestionnaire `install-php-extensions` :

- **intl** : Utilisé pour la gestion des formats internationaux et des localisations.
- **pdo_mysql** : Permet à PHP de communiquer avec la base de données MariaDB.
- **gd** : Bibliothèque graphique utilisée pour la manipulation d'images, souvent nécessaire dans les applications web.

2.2.4. Optimisations PHP

Plusieurs optimisations sont appliquées à la configuration PHP pour améliorer les performances :

- **Limite de mémoire** : La limite de mémoire PHP est augmentée à 1000M pour permettre le traitement de scripts complexes.
- **Limites de téléchargement et de publication** : Les paramètres `post_max_size` et `upload_max_filesize` sont définis à 500M pour permettre l'upload de fichiers volumineux, ce qui peut être utile dans un environnement de développement avec des données lourdes.

2.2.5. Gestion des volumes pour PHP

Plusieurs volumes sont créés à la configuration PHP pour améliorer les performances et la sécurité :

- **php-socket : /var/run** : Ce volume est un volume de docker en local permettant de stocker temporairement le socket de PHP-FPM afin de faire communiquer PHP et apache sans connexion distante afin d'éviter les sniffer de réseau
- **db-socket:/var/run/mysqld** :Ce volume est au même titre que celui du dessus à but de sécurité et de vitesse car il permet de faire communiquer php et mariadb en socket afin d'ameliorer la vitesse et la sécurité du projet
- **./docker/php/dev.php.ini:/usr/local/etc/php/php.ini** : Ce volume est à but de configuration, car il nous permet de spécifier le fichier de configuration qui sera utilisé par php

```
  php:
    build:
      context: ./docker
      dockerfile: ./php.dockerfile
    volumes:
      - .:/var/www/html
      - php-socket:/var/run
      - ./docker/php/dev.php.ini:/usr/local/etc/php/php.ini
      - db-socket:/var/run/mysqld
    depends_on:
      - db
    restart: unless-stopped
    networks:
      back:
```

```
        ipv4_address: 192.168.20.20
    dns:
      - 8.8.8.8
      - 8.8.4.4
```

```
# Use official PHP 8.3 FPM image
FROM php:8.3-fpm

ARG UID=1015
ARG GID=1015

# Create the user and group for Apache rights on the socket
# Create the group with GID 1010
RUN addgroup --gid 1010 apache
# Create the user with UID 1010 and assign to the apache group
RUN adduser --uid 1010 --ingroup apache --system apache

# Create the group with specified GID
RUN addgroup --gid ${GID} php
# Create the user with specified UID and assign to the php group
RUN adduser --uid ${UID} --ingroup php --system php

# Add the PHP extension installer from GitHub
ADD https://github.com/mlocati/docker-php-extension-
installer/releases/latest/download/install-php-extensions /usr/local/bin/install-
php-extensions

# Make the PHP extension installer executable and install required extensions
RUN chmod +x /usr/local/bin/install-php-extensions && \
    install-php-extensions intl pdo_mysql gd

# Update and upgrade system packages, clean cache after to reduce image size
RUN apt-get update && apt-get upgrade -y && apt-get clean && rm -rf
/var/lib/apt/lists/*

RUN echo 'listen=/var/run/php.sock' >> /usr/local/etc/php-fpm.conf

# Configure PHP-FPM to listen on a Unix socket, with secure permissions to permit
apache to connect
RUN sed -i 's|listen = .*|listen = /var/run/php/php-fpm.sock|g'
/usr/local/etc/php-fpm.d/www.conf && \
    sed -i 's|;listen.owner = www-data|listen.owner = apache|g'
/usr/local/etc/php-fpm.d/www.conf && \
    sed -i 's|;listen.group = www-data|listen.group = apache|g'
/usr/local/etc/php-fpm.d/www.conf && \
    sed -i 's|;listen.mode = 0660|listen.mode = 0660|g' /usr/local/etc/php-
fpm.d/www.conf

# Set the working directory
WORKDIR /var/www/html
```

2.3. MariaDB

MariaDB est un système de gestion de bases de données relationnelles open-source, utilisé comme alternative à MySQL. Dans cet environnement de développement, MariaDB est configuré pour fournir des services de bases de données à PHP et PhpMyAdmin. Il est déployé en tant que conteneur distinct, isolé dans le réseau `back` pour des raisons de sécurité.

2.3.1. Base Image : mariadb

Le fichier Docker Compose utilise l'image officielle `mariadb:latest`, qui inclut la version la plus récente et stable de MariaDB. Cette image est fiable et maintenue par la communauté MariaDB, garantissant des mises à jour régulières et la compatibilité avec les versions récentes de MySQL.

2.3.2. Configuration via les Variables d'Environnement

La configuration de MariaDB se fait principalement via des **variables d'environnement** dans le fichier `dev.compose.yaml`. Ces variables permettent de définir les principaux paramètres de la base de données, comme les informations d'identification, la base de données à créer par défaut, et d'autres aspects essentiels du système.

Voici les principales variables configurées :

- **MYSQL_ROOT_PASSWORD** : Définit le mot de passe root de MariaDB, essentiel pour accéder aux privilèges administratifs.
- **MYSQL_DATABASE** : Permet de spécifier une base de données par défaut qui sera automatiquement créée à l'initialisation du conteneur.
- **MYSQL_USER** et **MYSQL_PASSWORD** : Créent un utilisateur MariaDB supplémentaire avec des privilèges sur la base de données spécifiée, utile pour éviter d'utiliser directement l'utilisateur root, qui est une mauvaise pratique en matière de sécurité.

Ces variables sont externalisées via un fichier `.env` (non spécifié ici mais recommandé), ce qui permet de changer facilement les informations d'identification sans toucher à la configuration Docker Compose.

2.3.3. Gestion des Volumes

MariaDB utilise des volumes pour persister les données de la base de données au-delà de la durée de vie du conteneur. Ce mécanisme est crucial pour les environnements de développement et de production, car il permet de ne pas perdre les données en cas de suppression ou de redémarrage du conteneur.

Le volume configuré dans le fichier Docker Compose est :

- **./docker/db:/var/lib/mysql** : Ce volume monte un répertoire local dans le système de fichiers du conteneur à l'emplacement où MariaDB stocke ses données. Cela permet de préserver les données même si le conteneur MariaDB est recréé.

2.3.4. Sécurité de la Base de Données

Le service MariaDB est isolé dans le réseau `back`, ce qui signifie qu'il n'est accessible qu'à partir des autres services du réseau interne, comme PHP et PhpMyAdmin. Cela empêche tout accès direct depuis Internet,

limitant les vecteurs d'attaque potentiels.

En outre, le conteneur MariaDB est configuré pour redémarrer automatiquement en cas de problème, grâce à la directive `restart: unless-stopped` dans le fichier Docker Compose. Cela garantit que le service reste disponible même en cas d'incident mineur ou de mise à jour du système.

```
db:
  image: mariadb:latest
  container_name: mysql
  environment:
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    MYSQL_DATABASE: ${MYSQL_DATABASE}
    MYSQL_USER: ${MYSQL_USER}
    MYSQL_PASSWORD: ${MYSQL_PASSWORD}
  volumes:
    - ./docker/db:/var/lib/mysql
    - db-socket:/var/run/mysqld
  restart: unless-stopped
  networks:
    back:
      ipv4_address: 192.168.20.30
```

2.4. PhpMyAdmin

PhpMyAdmin est une application web open-source permettant de gérer facilement une base de données MySQL ou MariaDB via une interface graphique. Dans cet environnement de développement, PhpMyAdmin est déployé en tant que service Docker distinct, accessible depuis le navigateur pour une gestion rapide de la base de données.

2.4.1. Base Image : phpmyadmin

Le service PhpMyAdmin est configuré en utilisant l'image officielle `phpmyadmin:latest`. Cette image est maintenue par les développeurs de PhpMyAdmin et garantit la compatibilité avec MariaDB et MySQL.

2.4.2. Configuration via les Variables d'Environnement

Comme pour MariaDB, PhpMyAdmin est configuré principalement via des variables d'environnement, ce qui simplifie le processus de connexion et de gestion de la base de données.

Voici les principales variables d'environnement configurées dans `dev.compose.yaml` :

- **PMA_HOST** : Spécifie l'hôte MariaDB auquel PhpMyAdmin doit se connecter. Dans ce cas, il s'agit de l'adresse IP ou du nom du conteneur MariaDB dans le réseau Docker.
- **PMA_PORT** : Spécifie le port utilisé par MariaDB. Par défaut, MariaDB utilise le port 3306, mais cette variable peut être utilisée pour modifier le port si nécessaire.
- **PMA_USER et PMA_PASSWORD** : Permettent à PhpMyAdmin de se connecter à MariaDB avec un utilisateur et un mot de passe spécifiques. Ces variables doivent correspondre à un utilisateur existant dans MariaDB, défini lors de l'initialisation du conteneur.

2.4.3. Accès via le Port 8080

PhpMyAdmin est exposé via le port **8080**, ce qui signifie qu'il est accessible via l'URL `http://localhost:8080` ou `http://<IP_DU_SERVEUR>:8080`. Cela permet aux développeurs de gérer facilement la base de données à partir d'un navigateur web, sans avoir à utiliser des outils en ligne de commande ou des connexions SSH.

```
phpmyadmin:
    image: phpmyadmin:latest
    container_name: phpmyadmin
    environment:
      PMA_HOST: ${PMA_HOST}
      PMA_PORT: ${PMA_PORT}
      PMA_USER: root
      PMA_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      UPLOAD_LIMIT: 100M
    ports:
      - "8080:80"
    restart: unless-stopped
    depends_on:
      - db
    networks:
      front:
        ipv4_address: 192.168.10.20
      back:
        ipv4_address: 192.168.20.50
```

2.5. Docker in Docker (DIND)

Docker in Docker, ou **DIND**, est une configuration avancée qui permet d'exécuter des instances Docker à l'intérieur d'un conteneur Docker. Cette fonctionnalité est particulièrement utile pour les pipelines d'intégration continue (CI/CD), les environnements de test, ou tout autre cas où il est nécessaire de lancer des conteneurs dynamiquement dans un environnement déjà conteneurisé.

2.5.1. Base Image : ubuntu

Le service DIND utilise l'image de base `ubuntu:latest`, qui est une image minimale d'Ubuntu, une distribution Linux populaire. Ubuntu est choisi ici pour sa large compatibilité avec Docker et ses outils, ainsi que pour sa fiabilité dans les environnements de développement et de production.

2.5.2. Installation de Docker

Le fichier `dind.dockerfile` installe Docker dans le conteneur DIND en suivant ces étapes :

- **Ajout de la clé GPG Docker** : Cette étape garantit que les paquets installés proviennent bien du dépôt officiel de Docker, ce qui est une mesure de sécurité importante.
- **Ajout du dépôt Docker** : Docker n'est pas inclus dans les dépôts par défaut d'Ubuntu, il est donc nécessaire d'ajouter un dépôt spécifique pour installer Docker et ses composants.

- **Installation des outils Docker** : Les paquets installés incluent `docker-ce` (le moteur Docker), `docker-compose-plugin`, et `containerd`. Ces outils permettent de gérer des conteneurs directement à partir du conteneur DIND.

2.5.3. Configuration d'un Utilisateur Non-Root

L'exécution de Docker en tant qu'utilisateur root présente des risques de sécurité. Pour éviter cela, le fichier `dind.dockerfile` crée un utilisateur non-root nommé `dockeruser`, avec UID 1025 et GID 1025, qui est ajouté au groupe `docker`. Cet utilisateur peut exécuter des commandes Docker sans privilèges root, ce qui améliore la sécurité globale de l'environnement.

2.5.4. Utilisation de SSH dans DIND

Le service DIND est également configuré pour permettre des connexions SSH, ce qui peut être utile pour des tâches de gestion à distance ou pour exécuter des commandes Docker à distance. Le serveur SSH est installé via `openssh-server`, et un script d'entrée (`start.sh`) est configuré pour démarrer le serveur SSH automatiquement lors du lancement du conteneur.

```
  dind:
    build:
      context: ./docker
      dockerfile: ./dind.dockerfile
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    privileged: true
    networks:
      back:
        ipv4_address: 192.168.20.40
```

```
# Use the latest Ubuntu image
FROM ubuntu:latest

ARG UID=1025
ARG GID=1025

ENV DOCKERUSER=dockeruser

# Install dependencies and Docker
RUN apt-get update && apt-get install -y \
    curl \
    lsb-release \
    gnupg \
    sudo \
    openssh-server \
    docker.io

# Add Docker's official GPG key
RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
# Add Docker repository
RUN echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release
-cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list

RUN apt-get update && apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin -y

# Set up a user to log in through SSH
RUN useradd -m -d /home/$DOCKERUSER -s /bin/bash $DOCKERUSER && echo
"$DOCKERUSER:dockerpassword" | chpasswd && \
    usermod -aG sudo $DOCKERUSER && \
    echo '$USER ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

RUN usermod -aG docker $DOCKERUSER

# Set up SSH
RUN mkdir /var/run/sshd

# Define entrypoint and command
RUN echo "#!/bin/sh \n \
    /usr/sbin/sshd -D" > /home/$DOCKERUSER/start.sh && \
    chmod +x /home/$DOCKERUSER/start.sh

CMD ["/home/dockeruser/start.sh"]
```