Analyse Détaillée du Projet Docker Compose - WordPress

Introduction

Ce projet implémente une pile LEMP (Linux, Nginx, MariaDB, PHP) utilisant Docker Compose pour déployer une installation WordPress. Le projet est conçu en suivant les meilleures pratiques des conteneurs Docker et utilise des certificats SSL auto-signés pour la communication HTTPS.

Structure Générale du Projet

Le projet s'organise dans une structure hiérarchique claire :

```
.
- Makefile                  # Automatisation des tâches
- setup-database.sh        # Script de configuration BD
- secrets/                  # Identifiants sensibles
  - mariadb_root_passwd.txt
  - mariadb_usr_passwd.txt
  - mycredentials.txt
- srcs/
  - docker-compose.yml    # Définition des services
  - requirements/         # Configurations des conteneurs
    - mariadb/
    - nginx/
    - wordpress/
```

1. Analyse du Makefile

Le `Makefile` agit comme interface principale d'automatisation du projet :

Variables et Configuration

```
COMPOSE_CMD = docker-compose -f srcs/docker-compose.yml
DOMAIN_NAME ?= luisanch.42.fr
export DOMAIN_NAME
```

- **COMPOSE_CMD** : Définit la commande de base pour Docker Compose
- **DOMAIN_NAME** : Variable d'environnement exportée pour utilisation dans les conteneurs
- L'opérateur `?=` permet de surcharger la valeur depuis les variables d'environnement

Targets Principaux

Target `all` et `run`

```
all: run
run: build up
```

- `all` est le target par défaut qui exécute `run`
- `run` dépend de `build` et `up`, les exécutant séquentiellement

Target `build`

```
build:
    $(COMPOSE_CMD) build
```

Construit toutes les images Docker définies dans le docker-compose.yml

Target `up`

```
up:
    $(COMPOSE_CMD) up -d
```

Démarre les conteneurs en mode détaché (arrière-plan)

Target `down`

```
down:
    $(COMPOSE_CMD) down -v
```

Arrête et supprime les conteneurs, réseaux et volumes (`-v`)

Target `clean` (Nettoyage Complet)

```
clean:
    @docker stop $$(docker ps -qa) 2>/dev/null || true; \
    docker rm $$(docker ps -qa) 2>/dev/null || true; \
    docker rmi -f $$(docker images -qa) 2>/dev/null || true; \
    docker volume rm $$(docker volume ls -q) 2>/dev/null || true; \
    docker network rm $$(docker network ls -q) 2>/dev/null || true;
        \
    docker system prune -f --volumes
```

Ce target effectue un nettoyage exhaustif du système Docker : 1. Arrête tous les conteneurs (`docker ps -qa`) 2. Supprime tous les conteneurs 3. Supprime toutes les images (`-f` force la suppression) 4. Supprime tous les volumes 5. Supprime tous les réseaux 6. Exécute le nettoyage système avec `docker system prune`

Target `fclean`

```
fclean: clean
    sudo rm -rf /home/luisanch/data/mariadb/*
    sudo rm -rf /home/luisanch/data/wordpress/*
```

Étend `clean` en supprimant également les données persistantes de l'hôte

Target `setup`

```
setup:
    @echo "🔧 Création des répertoires pour volumes..."
    sudo mkdir -p /home/luisanch/data/mariadb
    sudo mkdir -p /home/luisanch/data/wordpress
    sudo mkdir -p /home/luisanch/data/ssl
    sudo chown -R luisanch:luisanch /home/luisanch/data/
    @echo "🔐 Génération des certificats SSL pour $(DOMAIN_NAME)..."
    ./srcs/requirements/nginx/tools/generate-ssl.sh
```

Prépare l'environnement : 1. Crée les répertoires pour volumes persistants 2. Établit les permissions correctes 3. Génère les certificats SSL auto-signés

2. Analyse du Docker Compose

Structure des Services

Le `docker-compose.yml` définit trois services interconnectés :

```yaml
services:
  mariadb:    # Base de données
  wordpress:  # Application PHP
  nginx:      # Serveur web/proxy inverse
```

Service MariaDB

```yaml
mariadb:
  container_name: mariadb
  build: requirements/mariadb
  image: mariadb
  volumes:
    - mariadb_data:/var/lib/mysql
  networks:
    - network
  restart: always
```

Caractéristiques Clés :

- **Volume persistant** : `mariadb_data` monté dans `/var/lib/mysql`
- **Réseau isolé** : Connecté au réseau `network`
- **Redémarrage automatique** : `restart: always`

Variables d'Environnement :

```yaml
environment:
  - MYSQL_ROOT_PASSWORD_FILE=/run/secrets/mariadb_root_password
  - MYSQL_DATABASE=wordpress
```

```
  - MYSQL_USER_FILE=/run/secrets/mariadb_user
  - MYSQL_PASSWORD_FILE=/run/secrets/mariadb_password
```

Système de Secrets :

```
secrets:
  - mariadb_root_password
  - mariadb_user
  - mariadb_password
```

Les secrets sont montés dans `/run/secrets/` et lus depuis des fichiers au lieu de variables d'environnement, améliorant la sécurité.

Service WordPress

```
wordpress:
  container_name: wordpress
  build: requirements/wordpress
  volumes:
    - wordpress_data:/var/www/wordpress
  depends_on:
    - mariadb
```

Dépendances :

- **depends_on** : Garantit que MariaDB démarre avant WordPress
- **Réseau partagé** : Permet la communication avec MariaDB par nom de service

Configuration de Base de Données :

```
environment:
  - DB_NAME=wordpress
  - DB_USER_FILE=/run/secrets/mariadb_user
  - DB_PASSWORD_FILE=/run/secrets/mariadb_password
  - DB_HOST=mariadb
```

`DB_HOST=mariadb` utilise la résolution DNS interne de Docker Compose.

Service Nginx

```
nginx:
  container_name: nginx
  build: requirements/nginx
  ports:
    - "443:443"
  volumes:
    - wordpress_data:/var/www/wordpress:ro
    - /home/luisanch/data/ssl:/etc/nginx/ssl:ro
```

Configuration Réseau :

- **Port exposé** : Seulement HTTPS (443)
- **Volume partagé** : Accès en lecture seule aux fichiers WordPress
- **Certificats SSL** : Montés depuis l'hôte

Configuration des Volumes

```yaml
volumes:
  mariadb_data:
    driver: local
    driver_opts:
      type: none
      device: /home/luisanch/data/mariadb
      o: bind
```

Les volumes utilisent des **bind mounts** au lieu de volumes Docker natifs : - **Avantage** : Accès direct depuis l'hôte - **Emplacement fixe** : Données dans `/home/luisanch/data/`

Système de Secrets

```yaml
secrets:
  mariadb_root_password:
    file: ../secrets/mariadb_root_passwd.txt
  mariadb_user:
    file: ../secrets/mariadb_usr_passwd.txt
  mariadb_password:
    file: ../secrets/mycredentials.txt
```

Les secrets sont chargés depuis des fichiers externes, séparant les identifiants du code.

3. Analyse du Conteneur MariaDB

Dockerfile de MariaDB

```dockerfile
FROM debian:12-slim
```

Utilise **Debian 12 Slim** comme image de base, priorisant la sécurité et la taille réduite.

Installation des Paquets :

```dockerfile
RUN apt-get update && apt-get install -y \
    mariadb-server \
    mariadb-client \
    gettext-base \
    && rm -rf /var/lib/apt/lists/*
```

- **mariadb-server** : Serveur de base de données
- **mariadb-client** : Outils client

- **gettext-base** : Pour l'interpolation des variables d'environnement
- **Nettoyage** : `rm -rf /var/lib/apt/lists/*` réduit la taille de l'image

Configuration des Répertoires :

```
RUN mkdir -p /var/lib/mysql /var/run/mysqld /var/log/mysql
RUN chown -R mysql:mysql /var/lib/mysql /var/run/mysqld
        /var/log/mysql
```

Établit la structure des répertoires et les permissions pour l'utilisateur `mysql`.

Configuration du Serveur :

```
RUN echo "[mysqld]" > /etc/mysql/mariadb.conf.d/50-server.cnf && \
    echo "datadir = /var/lib/mysql" >> /etc/mysql/mariadb.conf.d/50-
        server.cnf && \
    echo "socket = /var/run/mysqld/mysqld.sock" >>
        /etc/mysql/mariadb.conf.d/50-server.cnf && \
    echo "bind-address = 0.0.0.0" >> /etc/mysql/mariadb.conf.d/50-
        server.cnf
```

Configuration critique : - **bind-address** = **0.0.0.0** : Permet les connexions depuis d'autres conteneurs - **socket** : Emplacement du socket Unix - **datadir** : Répertoire de données (monté comme volume)

Script d'Entrypoint

Le `entrypoint.sh` gère l'initialisation de MariaDB :

Lecture des Secrets :

```
DB_ROOT_PASSWORD=$(cat /run/secrets/mariadb_root_password)
DB_USER=$(cat /run/secrets/mariadb_user)
DB_PASSWORD=$(cat /run/secrets/mariadb_password)
```

Initialisation Conditionnelle :

```
if [ ! -d "/var/lib/mysql/mysql" ]; then
    echo "📦 Initialisation de la base de données MariaDB..."
    mysql_install_db --user=mysql --datadir=/var/lib/mysql --skip-
        test-db
```

N'initialise que si la structure de données précédente n'existe pas.

Configuration des Utilisateurs :

```
mysql -e "
    ALTER USER 'root'@'localhost' IDENTIFIED BY
        '${DB_ROOT_PASSWORD}';
    CREATE DATABASE IF NOT EXISTS wordpress;
    CREATE USER IF NOT EXISTS '${DB_USER}'@'%' IDENTIFIED BY
        '${DB_PASSWORD}';
    GRANT ALL PRIVILEGES ON wordpress.* TO '${DB_USER}'@'%';
    FLUSH PRIVILEGES;
```

```
"
```

Processus de configuration : 1. Établit le mot de passe root 2. Crée la base de données WordPress 3. Crée un utilisateur spécifique avec permissions limitées 4. Le motif `@'%'` permet les connexions depuis n'importe quel hôte

4. Analyse du Conteneur Nginx

Dockerfile de Nginx

```
FROM debian:12-slim

RUN apt-get update && apt-get install -y \
    nginx \
    openssl \
    gettext-base \
    && rm -rf /var/lib/apt/lists/*
```

Paquets essentiels : - **nginx** : Serveur web - **openssl** : Pour la validation des certificats SSL - **gettext-base** : Pour la substitution des variables d'environnement

Configuration Dynamique :

```
CMD ["/bin/bash", "-c", "envsubst '$$DOMAIN_NAME' <
        /etc/nginx/nginx.conf.template > /etc/nginx/nginx.conf &&
        nginx -g 'daemon off;'"]
```

Cette commande : 1. Substitue les variables d'environnement dans le template 2. Démarre Nginx en mode foreground (requis pour les conteneurs)

Configuration de Nginx

Configuration SSL :

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384;
ssl_prefer_server_ciphers on;
ssl_session_cache shared:SSL:10m;
```

Configuration de sécurité SSL robuste : - **Protocoles modernes** : Seulement TLS 1.2 et 1.3 - **Ciphers sécurisés** : Configuration restrictive - **Session cache** : Optimisation de performance

Redirection HTTP vers HTTPS :

```
server {
    listen 80;
    server_name ${DOMAIN_NAME} localhost;
    return 301 https://$server_name$request_uri;
```

```
}
```

Redirection automatique 301 (permanente) de HTTP vers HTTPS.

Configuration HTTPS :

```
server {
    listen 443 ssl http2;
    server_name ${DOMAIN_NAME} localhost;

    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    root /var/www/wordpress;
    index index.php index.html index.htm;
```

- **HTTP/2** : Protocole moderne activé
- **DocumentRoot** : Pointe vers l'installation WordPress
- **Certificats** : Chargés depuis le volume monté

Configuration PHP-FPM :

```
location ~ \.php$ {
    fastcgi_pass wordpress:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
    include fastcgi_params;
}
```

- **FastCGI** : Communication avec PHP-FPM dans le conteneur WordPress
- **Port 9000** : Port standard de PHP-FPM
- **wordpress:9000** : Résolution DNS interne de Docker

Génération de Certificats SSL

Le script `generate-ssl.sh` :

```
openssl genrsa -out $SSL_DIR/nginx.key 2048
openssl req -new -x509 -key $SSL_DIR/nginx.key -out
        $SSL_DIR/nginx.crt -days 365 \
  -subj "/C=ES/ST=Madrid/L=Madrid/O=42/OU=IT/CN=$DOMAIN"
```

Génère : 1. **Clé privée RSA** : 2048 bits 2. **Certificat auto-signé** : Valide 365 jours 3. **Subject** : Informations de l'organisation

5. Analyse du Conteneur WordPress

Dockerfile de WordPress

Stack PHP Complet :

```
RUN apt-get update && apt-get install -y \
    php8.2-fpm \
    php8.2-mysql \
    php8.2-curl \
    php8.2-gd \
    php8.2-intl \
    php8.2-mbstring \
    php8.2-soap \
    php8.2-xml \
    php8.2-xmlrpc \
    php8.2-zip \
    php8.2-cli \
    wget \
    unzip \
    gettext-base \
    mariadb-client
```

Extensions PHP essentielles pour WordPress : - **php8.2-mysql** : Connectivité avec MariaDB - **php8.2-gd** : Manipulation d'images - **php8.2-curl** : Communications HTTP - **php8.2-mbstring** : Gestion des chaînes multibyte - **php8.2-xml** : Traitement XML - **mariadb-client** : Pour les scripts de configuration

Téléchargement de WordPress :

```
RUN wget https://wordpress.org/latest.tar.gz \
    && tar -xzf latest.tar.gz \
    && mv wordpress/* /var/www/wordpress/ \
    && rm -rf wordpress latest.tar.gz
```

Télécharge la version la plus récente de WordPress directement depuis le site officiel.

Configuration PHP-FPM :

```
RUN sed -i 's/listen = \/run\/php\/php8.2-fpm.sock/listen = 9000/'
        /etc/php/8.2/fpm/pool.d/www.conf
RUN sed -i 's/;daemonize = yes/daemonize = no/'
        /etc/php/8.2/fpm/php-fpm.conf
```

Modifications critiques : 1. **Port TCP** : Change du socket Unix au port TCP 9000 2. **Mode foreground** : Nécessaire pour les conteneurs Docker

Configuration de WordPress

wp-config-simple.php

```
// Lecture des identifiants de base de données depuis les fichiers
        secrets
$db_user = trim(file_get_contents('/run/secrets/mariadb_user'));
$db_password =
        trim(file_get_contents('/run/secrets/mariadb_password'));
```

```
define('DB_USER', $db_user);
define('DB_PASSWORD', $db_password);
define('DB_HOST', 'mariadb');
```

Lecture sécurisée des identifiants : Lit depuis les fichiers secrets au lieu des variables d'environnement.

Configuration de Sécurité :

```
define('AUTH_KEY',         '~=%kPnn+AtJtuHcktHv??pK,
        [<bD+mDa,08rFI`,.^:h?5(u;7p{+!phtF5~N(%<');
define('SECURE_AUTH_KEY',  '`#JJXFV}2fr;}O@|/y^OL&;1&,(%7
        &y>n&u^(hK3~WG1FsE]Vox6bqp>L2#.XrB');
// ... plus de clés de sécurité
```

Clés salt uniques : Améliorent la sécurité des sessions et cookies.

Configuration des URLs :

```
$domain_name = getenv('DOMAIN_NAME') ?: 'luisanch.42.fr';
define('WP_HOME', 'https://' . $domain_name);
define('WP_SITEURL', 'https://' . $domain_name);
```

URLs dynamiques : S'adaptent à la variable d'environnement DOMAIN_NAME.

Configuration de Sécurité Avancée :

```
define('DISALLOW_FILE_EDIT', true);
define('DISALLOW_FILE_MODS', true);
define('WP_DEBUG', true);
define('WP_DEBUG_LOG', true);
```

- **DISALLOW_FILE_EDIT** : Désactive l'éditeur de fichiers dans l'admin
- **DISALLOW_FILE_MODS** : Empêche l'installation de plugins/thèmes
- **WP_DEBUG** : Active le logging pour le développement

6. Système de Mise en Réseau

Réseau Bridge Personnalisé

```
networks:
  network:
    driver: bridge
```

Avantages du Réseau Bridge :

1. **Isolation** : Les conteneurs ne peuvent communiquer qu'entre eux
2. **Résolution DNS** : Chaque service est accessible par son nom
3. **Sécurité** : Pas d'accès direct depuis l'hôte sauf par les ports exposés

Communication Entre Services :

```
Internet --&gt; Nginx:443 --&gt; WordPress:9000 --&gt; MariaDB:3306
```

- **Nginx** : Seul point d'entrée public
- **WordPress** : Accessible seulement depuis Nginx
- **MariaDB** : Accessible seulement depuis WordPress

7. Gestion des Volumes et Persistance

Stratégie de Bind Mounts

```yaml
volumes:
  mariadb_data:
    driver_opts:
      type: none
      device: /home/luisanch/data/mariadb
      o: bind
```

Avantages des Bind Mounts :

1. **Accès direct** : Les données sont accessibles depuis l'hôte
2. **Sauvegarde simple** : Sauvegarde et restauration faciles
3. **Migration** : Portabilité entre environnements

Structure des Données :

```
/home/luisanch/data/
- mariadb/          # Données de base de données
- wordpress/        # Fichiers WordPress
- ssl/             # Certificats SSL
```

8. Sécurité et Bonnes Pratiques

Système de Secrets

Le projet implémente Docker Secrets pour la gestion sécurisée des identifiants :

```yaml
secrets:
  mariadb_root_password:
    file: ../secrets/mariadb_root_passwd.txt
```

Bénéfices :

1. **Séparation** : Identifiants en dehors du code
2. **Montage sécurisé** : Dans `/run/secrets/` avec permissions restrictives
3. **Pas d'exposition** : N'apparaissent pas dans les variables d'environnement

Configuration SSL

Certificats Auto-signés :

- **Développement/Test** : Adaptés aux environnements non-productifs
- **Chiffrement complet** : Protège le trafic entre client et serveur
- **Génération facile** : Script automatisé

Durcissement de WordPress

```
define('DISALLOW_FILE_EDIT', true);
define('DISALLOW_FILE_MODS', true);
```

Ces configurations : 1. **Empêchent les modifications** : Pas d'édition de fichiers depuis l'admin 2. **Sécurité additionnelle** : Réduisent la surface d'attaque

9. Processus de Déploiement

Flux d'Initialisation

1. **Préparation de l'environnement** :

   ```
   make setup
   ```

   - Crée les répertoires
   - Génère les certificats SSL

2. **Construction des images** :

   ```
   make build
   ```

   - Construit toutes les images Docker

3. **Démarrage des services** :

   ```
   make up
   ```

   - Démarre les conteneurs dans l'ordre des dépendances

Ordre de Démarrage

1. **MariaDB** : S'initialise en premier
2. **WordPress** : Attend que MariaDB soit prêt
3. **Nginx** : Démarre en dernier, dépend de WordPress

Vérification de l'État

```
make info
```

Affiche : - État des conteneurs - URLs d'accès - Informations de connectivité

10. Dépannage et Maintenance

Commandes de Diagnostic

Vérifier les logs :

```
docker-compose -f srcs/docker-compose.yml logs [service]
```

Accéder aux conteneurs :

```
docker exec -it [conteneur] /bin/bash
```

Vérifier la connectivité :

```
docker exec wordpress ping mariadb
```

Nettoyage et Reset

Nettoyage partiel :

```
make down
```

Nettoyage complet :

```
make fclean
```

Supprime : - Tous les conteneurs - Toutes les images - Tous les volumes - Données persistantes de l'hôte

11. Optimisations et Configuration Avancée

Configuration de PHP-FPM

Le projet configure PHP-FPM pour : - **Communication TCP** : Au lieu de sockets Unix - **Mode foreground** : Compatible avec Docker - **Pool de processus** : Configuration par défaut de www

Configuration de MariaDB

- **InnoDB** : Moteur de stockage par défaut
- **UTF8** : Charset compatible avec WordPress
- **Buffer pool** : Configuration par défaut optimisée

Configuration de Nginx

- **Gzip** : Compression activée par défaut
- **Fichiers statiques** : Servis directement par Nginx
- **Fichiers PHP** : Passés à PHP-FPM

Conclusion

Ce projet implémente une architecture robuste et évolutive pour WordPress utilisant Docker Compose. La séparation des responsabilités, l'utilisation de réseaux isolés, la gestion sécurisée des identifiants et la configuration SSL fournissent une base solide pour un environnement de développement ou de test.

Points Forts :

1. **Architecture claire** : Séparation des services bien définie
2. **Sécurité** : Utilisation de secrets et SSL
3. **Automatisation** : Makefile pour gestion simplifiée
4. **Persistance** : Données préservées dans les bind mounts
5. **Évolutivité** : Modification et extension faciles

Domaines d'Amélioration pour la Production :

1. **Certificats valides** : Remplacer les auto-signés par Let's Encrypt
2. **Proxy inverse** : Considérer Traefik ou similaire pour multiples services
3. **Monitoring** : Intégrer des solutions de surveillance
4. **Sauvegarde** : Implémenter des stratégies automatisées de sauvegarde
5. **CI/CD** : Intégrer des pipelines de déploiement automatisé

Le projet démontre une excellente compréhension de Docker Compose et des meilleures pratiques pour le déploiement d'applications web modernes.