Análisis Detallado del Proyecto Docker Compose - WordPress

Introducción

Este proyecto implementa un stack LEMP (Linux, Nginx, MariaDB, PHP) usando Docker Compose para desplegar una instalación de WordPress. El proyecto está diseñado siguiendo las mejores prácticas de contenedores Docker y utiliza certificados SSL autofirmados para comunicación HTTPS.

Estructura General del Proyecto

El proyecto se organiza en una estructura jerárquica clara:

```
.
├── Makefile                  # Automatización de tareas
├── setup-database.sh        # Script de configuración de BD
├── secrets/                  # Credenciales sensibles
│   ├── mariadb_root_passwd.txt
│   ├── mariadb_usr_passwd.txt
│   └── mycredentials.txt
└── srcs/
    ├── docker-compose.yml    # Definición de servicios
    └── requirements/         # Configuraciones de contenedores
        ├── mariadb/
        ├── nginx/
        └── wordpress/
```

1. Análisis del Makefile

El `Makefile` actúa como interfaz de automatización principal del proyecto:

Variables y Configuración

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

- **COMPOSE_CMD**: Define el comando base para Docker Compose
- **DOMAIN_NAME**: Variable de entorno exportada para uso en contenedores
- El operador `?=` permite sobrescribir el valor desde variables de entorno

Targets Principales

Target `all` y `run`

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

- `all` es el target por defecto que ejecuta `run`
- `run` depende de `build` y `up`, ejecutándolos secuencialmente

Target `build`

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

Construye todas las imágenes Docker definidas en el docker-compose.yml

Target `up`

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

Inicia los contenedores en modo detached (background)

Target `down`

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

Detiene y elimina contenedores, redes y volúmenes (`-v`)

Target `clean` (Limpieza Completa)

```
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
```

Este target realiza una limpieza exhaustiva del sistema Docker: 1. Detiene todos los contenedores (`docker ps -qa`) 2. Elimina todos los contenedores 3. Elimina todas las imágenes (`-f` fuerza la eliminación) 4. Elimina todos los volúmenes 5. Elimina todas las redes 6. Ejecuta limpieza del sistema con `docker system prune`

Target `fclean`

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

Extiende `clean` eliminando también los datos persistentes del host

Target `setup`

```
setup:
    @echo "📁 Creando directorios para volúmenes..."
    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 "🔐 Generando certificados SSL para $(DOMAIN_NAME)..."
    ./srcs/requirements/nginx/tools/generate-ssl.sh
```

Prepara el entorno: 1. Crea directorios para volúmenes persistentes 2. Establece permisos correctos 3. Genera certificados SSL autofirmados

2. Análisis del Docker Compose

Estructura de Servicios

El `docker-compose.yml` define tres servicios interconectados:

```yaml
services:
  mariadb:    # Base de datos
  wordpress:  # Aplicación PHP
  nginx:      # Servidor web/proxy reverso
```

Servicio MariaDB

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

Características Clave:

- **Volumen persistente**: `mariadb_data` montado en `/var/lib/mysql`
- **Red aislada**: Conectado a la red `network`
- **Reinicio automático**: `restart: always`

Variables de Entorno:

```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
```

Sistema de Secrets:

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

Los secrets se montan en `/run/secrets/` y se leen desde archivos en lugar de variables de entorno, mejorando la seguridad.

Servicio WordPress

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

Dependencias:

- **depends_on**: Garantiza que MariaDB se inicie antes que WordPress
- **Red compartida**: Permite comunicación con MariaDB por nombre de servicio

Configuración de Base de Datos:

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

`DB_HOST=mariadb` utiliza la resolución DNS interna de Docker Compose.

Servicio 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
```

Configuración de Red:

- **Puerto expuesto**: Solo HTTPS (443)
- **Volumen compartido**: Acceso de solo lectura a archivos de WordPress

- **Certificados SSL**: Montados desde el host

Configuración de Volúmenes

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

Los volúmenes utilizan **bind mounts** en lugar de volúmenes Docker nativos: - **Ventaja**: Acceso directo desde el host - **Ubicación fija**: Datos en `/home/luisanch/data/`

Sistema de Secrets

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

Los secrets se cargan desde archivos externos, separando las credenciales del código.

3. Análisis del Contenedor MariaDB

Dockerfile de MariaDB

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

Utiliza **Debian 12 Slim** como imagen base, priorizando seguridad y tamaño reducido.

Instalación de Paquetes:

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

- **mariadb-server**: Servidor de base de datos
- **mariadb-client**: Herramientas de cliente
- **gettext-base**: Para interpolación de variables de entorno
- **Limpieza**: `rm -rf /var/lib/apt/lists/*` reduce el tamaño de la imagen

Configuración de Directorios:

```
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
```

Establece estructura de directorios y permisos para el usuario `mysql`.

Configuración del Servidor:

```
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
```

Configuración crítica: - **bind-address** = **0.0.0.0**: Permite conexiones desde otros contenedores - **socket**: Ubicación del socket Unix - **datadir**: Directorio de datos (montado como volumen)

Script de Entrypoint

El `entrypoint.sh` gestiona la inicialización de MariaDB:

Lectura de Secrets:

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

Inicialización Condicional:

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

Solo inicializa si no existe la estructura de datos previa.

Configuración de Usuarios:

```
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;
"
```

Proceso de configuración: 1. Establece contraseña de root 2. Crea base de datos WordPress 3. Crea usuario específico con permisos limitados 4. El patrón `@'%'` permite conexiones desde cualquier host

4. Análisis del Contenedor 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/*
```

Paquetes esenciales: - **nginx**: Servidor web - **openssl**: Para validación de certificados SSL - **gettext-base**: Para sustitución de variables de entorno

Configuración Dinámica:

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

Este comando: 1. Sustituye variables de entorno en el template 2. Inicia Nginx en modo foreground (requerido para contenedores)

Configuración de Nginx

Configuración 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;
```

Configuración de seguridad SSL robusta: - **Protocolos modernos**: Solo TLS 1.2 y 1.3 - **Ciphers seguros**: Configuración restrictiva - **Session cache**: Optimización de rendimiento

Redirección HTTP a HTTPS:

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

Redirección automática 301 (permanente) de HTTP a HTTPS.

Configuración 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**: Protocolo moderno habilitado
- **DocumentRoot**: Apunta a la instalación de WordPress
- **Certificados**: Cargados desde volumen montado

Configuración 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**: Comunicación con PHP-FPM en el contenedor WordPress
- **Puerto 9000**: Puerto estándar de PHP-FPM
- **wordpress:9000**: Resolución DNS interna de Docker

Generación de Certificados SSL

El 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"
```

Genera: 1. **Clave privada RSA**: 2048 bits 2. **Certificado autofirmado**: Válido por 365 días 3. **Subject**: Información de la organización

5. Análisis del Contenedor WordPress

Dockerfile de WordPress

Stack PHP Completo:

```
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
```

Extensiones PHP esenciales para WordPress: - **php8.2-mysql**: Conectividad con MariaDB - **php8.2-gd**: Manipulación de imágenes - **php8.2-curl**: Comunicaciones HTTP - **php8.2-mbstring**: Manejo de cadenas multibyte - **php8.2-xml**: Procesamiento XML - **mariadb-client**: Para scripts de configuración

Descarga 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
```

Descarga la versión más reciente de WordPress directamente desde el sitio oficial.

Configuración 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
```

Modificaciones críticas: 1. **Puerto TCP**: Cambia de socket Unix a puerto TCP 9000 2. **Modo foreground**: Necesario para contenedores Docker

Configuración de WordPress

wp-config-simple.php

```
// Read database credentials from secrets files
$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');
```

Lectura segura de credenciales: Lee desde archivos de secrets en lugar de variables de entorno.

Configuración de Seguridad:

```
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');
// ... más claves de seguridad
```

Salt keys únicas: Mejoran la seguridad de las sesiones y cookies.

Configuración de URLs:

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

URLs dinámicas: Se adaptan a la variable de entorno DOMAIN_NAME.

Configuración de Seguridad Avanzada:

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

- **DISALLOW_FILE_EDIT**: Desactiva editor de archivos en admin
- **DISALLOW_FILE_MODS**: Previene instalación de plugins/temas
- **WP_DEBUG**: Activa logging para desarrollo

6. Sistema de Networking

Red Bridge Personalizada

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

Ventajas de la Red Bridge:

1. **Aislamiento**: Los contenedores solo pueden comunicarse entre sí
2. **Resolución DNS**: Cada servicio es accesible por su nombre
3. **Seguridad**: No hay acceso directo desde el host excepto por puertos expuestos

Comunicación Entre Servicios:

```
Internet → Nginx:443 → WordPress:9000 → MariaDB:3306
```

- **Nginx**: Único punto de entrada público
- **WordPress**: Accesible solo desde Nginx
- **MariaDB**: Accesible solo desde WordPress

7. Gestión de Volúmenes y Persistencia

Estrategia de Bind Mounts

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

Ventajas de Bind Mounts:

1. **Acceso directo**: Los datos son accesibles desde el host
2. **Backup simple**: Fácil respaldo y restauración
3. **Migración**: Portabilidad entre entornos

Estructura de Datos:

```
/home/luisanch/data/
├── mariadb/          # Datos de base de datos
├── wordpress/        # Archivos de WordPress
└── ssl/             # Certificados SSL
```

8. Seguridad y Buenas Prácticas

Sistema de Secrets

El proyecto implementa Docker Secrets para gestión segura de credenciales:

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

Beneficios:

1. **Separación**: Credenciales fuera del código
2. **Montaje seguro**: En `/run/secrets/` con permisos restrictivos
3. **No exposición**: No aparecen en variables de entorno

Configuración SSL

Certificados Autofirmados:

- **Desarrollo/Testing**: Adecuados para entornos no productivos
- **Encriptación completa**: Protege tráfico entre cliente y servidor
- **Fácil generación**: Script automatizado

Hardening de WordPress

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

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

Estas configuraciones: 1. **Previenen modificaciones**: No se pueden editar archivos desde admin 2. **Seguridad adicional**: Reducen superficie de ataque

9. Proceso de Despliegue

Flujo de Inicialización

1. **Preparación del entorno**:

   ```
   make setup
   ```

   - Crea directorios
   - Genera certificados SSL

2. **Construcción de imágenes**:

   ```
   make build
   ```

   - Construye todas las imágenes Docker

3. **Inicio de servicios**:

   ```
   make up
   ```

   - Inicia contenedores en orden de dependencias

Orden de Inicio

1. **MariaDB**: Se inicializa primero
2. **WordPress**: Espera a que MariaDB esté listo
3. **Nginx**: Se inicia último, depende de WordPress

Verificación de Estado

```
make info
```

Muestra: - Estado de contenedores - URLs de acceso - Información de conectividad

10. Troubleshooting y Mantenimiento

Comandos de Diagnóstico

Verificar logs:

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

Acceder a contenedores:

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

Verificar conectividad:

```
docker exec wordpress ping mariadb
```

Limpieza y Reset

Limpieza parcial:

```
make down
```

Limpieza completa:

```
make fclean
```

Elimina: - Todos los contenedores - Todas las imágenes - Todos los volúmenes - Datos persistentes del host

11. Optimizaciones y Configuración Avanzada

Configuración de PHP-FPM

El proyecto configura PHP-FPM para: - **Comunicación TCP**: En lugar de sockets Unix - **Modo foreground**: Compatible con Docker - **Pool de procesos**: Configuración por defecto de www

Configuración de MariaDB

- **InnoDB**: Motor de almacenamiento por defecto
- **UTF8**: Charset compatible con WordPress
- **Buffer pool**: Configuración por defecto optimizada

Configuración de Nginx

- **Gzip**: Compresión habilitada por defecto
- **Static files**: Servidos directamente por Nginx
- **PHP files**: Pasados a PHP-FPM

Conclusión

Este proyecto implementa una arquitectura robusta y escalable para WordPress utilizando Docker Compose. La separación de responsabilidades, el uso de redes aisladas, la gestión segura de credenciales y la configuración SSL proporcionan una base sólida para un entorno de desarrollo o testing.

Puntos Fuertes:

1. **Arquitectura clara**: Separación de servicios bien definida
2. **Seguridad**: Uso de secrets y SSL
3. **Automatización**: Makefile para gestión simplificada
4. **Persistencia**: Datos preservados en bind mounts
5. **Escalabilidad**: Fácil modificación y extensión

Áreas de Mejora para Producción:

1. **Certificados válidos**: Reemplazar autofirmados por Let's Encrypt
2. **Proxy reverso**: Considerar Traefik o similar para múltiples servicios
3. **Monitoring**: Integrar soluciones de monitoreo
4. **Backup**: Implementar estrategias automatizadas de respaldo
5. **CI/CD**: Integrar pipelines de despliegue automatizado

El proyecto demuestra un excelente entendimiento de Docker Compose y las mejores prácticas para despliegue de aplicaciones web modernas.