AI-03075 Docker Wordpress Deployment

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	@October 14, 2025 10:03 AM

Create a Wordpress docker deployment step by step

Excellent — let's deploy **WordPress** step-by-step in **Docker** (using Docker Compose), with persistent volumes and an optional phpMyAdmin for database management.

We'll use a clean, **production-ready** layout that you can host on your **Ubuntu Server** or in your **Docker Lab setup**.

🚢 OVERVIEW

We'll create a WordPress deployment consisting of:

1. **MariaDB** — database backend
2. **WordPress** — PHP + Apache app server
3. **phpMyAdmin (optional)** — to manage MariaDB
4. **Persistent volumes** — for data durability
5. **Bridge network (`dockerlab_net`)** — for inter-container communication

---

⚙️ STEP-BY-STEP SETUP

1. Create the project folder

```
mkdir -p ~/docker/wordpress
cd ~/docker/wordpress
```

2. Create a `.env` file

This will store credentials securely.

```
nano .env
```

Add the following:

```
# WordPress environment variables
WORDPRESS_DB_HOST=mariadb:3306
WORDPRESS_DB_USER=wpuser
WORDPRESS_DB_PASSWORD=wppass
WORDPRESS_DB_NAME=wpdb

# MariaDB root password
MYSQL_ROOT_PASSWORD=rootpass
MYSQL_DATABASE=wpdb
MYSQL_USER=wpuser
MYSQL_PASSWORD=wppass
```

Save and exit (`Ctrl + O`, `Enter`, `Ctrl + X`).

3. Create `docker-compose.yml`

```
nano docker-compose.yml
```

Paste this file:

```
version: "3.9"

services:
  mariadb:
    image: mariadb:11
    container_name: mariadb
    restart: unless-stopped
```

```
    env_file: .env
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - dockerlab_net

  wordpress:
    image: wordpress:latest
    container_name: wordpress
    restart: unless-stopped
    depends_on:
      - mariadb
    ports:
      - "8080:80"
    env_file: .env
    volumes:
      - wp_data:/var/www/html
    networks:
      - dockerlab_net

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: phpmyadmin
    restart: unless-stopped
    depends_on:
      - mariadb
    ports:
      - "8081:80"
    environment:
      - PMA_HOST=mariadb
      - PMA_USER=wpuser
      - PMA_PASSWORD=wppass
    networks:
      - dockerlab_net

networks:
```

```
  dockerlab_net:
    driver: bridge

volumes:
  db_data:
  wp_data:
```

---

4. Launch the stack

Run:

```
docker compose up -d
```

This will pull the images and start all three containers.

---

5. Verify containers

```
docker ps
```

You should see:

```
wordpress      Up   0.0.0.0:8080→80/tcp
phpmyadmin     Up   0.0.0.0:8081→80/tcp
mariadb        Up   3306/tcp
```

---

6. Access the services

- 🌐 WordPress → http://YOUR_SERVER_IP:8080
- 🗄️ phpMyAdmin → http://YOUR_SERVER_IP:8081

Use credentials:

- User: `wpuser`
- Password: `wppass`

---

7. To stop or remove

Stop:

```
docker compose down
```

Remove containers + volumes:

```
docker compose down -v
```

🧩 OPTIONAL: Make it accessible via Tailscale domain

If you already have **Tailscale** on your Ubuntu Server:

```
tailscale status
```

Find your device name, e.g. `ubuntuserver3.local.ts.net`,

then you can access WordPress at:

```
http://ubuntuserver3.local.ts.net:8080
```

✅ SUMMARY

Component	Purpose	Port
WordPress	Website CMS	8080
MariaDB	Database	internal only
phpMyAdmin	DB management	8081

Would you like me to extend this with:

- **NGINX Reverse Proxy (with SSL via Let's Encrypt)** for production use, or
- **Backup strategy** for WordPress and MariaDB data?

Create a Wordpress docker deployment step by step. I have already a mariadb running and phpmyadmin in the network called dockerlab in bridge mode. Use port 805.

Direct answer

Below is a **ready-to-run, minimal Docker Compose** setup for **WordPress** that uses your existing **mariadb + phpmyadmin** already attached to the `dockerlab` bridge network. This Compose file exposes WordPress on **host port 8050 → container port 80** and attaches the container to the existing `dockerlab` network so it can reach your MariaDB.

Key reasons (very short)

- Reuses your existing MariaDB and phpMyAdmin (no duplicate DB service).
- Uses an `external` Docker network `dockerlab` so all containers share the same bridge.
- Persistent volume for WordPress files (uploads, plugins, themes).
- Simple `.env` to keep credentials out of the compose file.

Alternatives (short)

- If you prefer host-level folder instead of named volume, replace `wp_data:` with a bind mount.
- For production: add NGINX reverse proxy + Let's Encrypt for TLS on top of this.
- If DB container has a different name than `mariadb`, set `WORDPRESS_DB_HOST` to that container name or IP.

Action plan — step-by-step

1. **Create project folder & enter it**

```
mkdir -p ~/docker/wordpress
cd ~/docker/wordpress
```

1. **Create a** `.env` **file** (edit values to match your MariaDB user/password/database)

```
cat > .env <<EOF
# WordPress → MariaDB connection
WORDPRESS_DB_HOST=mariadb:3306
WORDPRESS_DB_USER=wpuser
WORDPRESS_DB_PASSWORD=wppass
WORDPRESS_DB_NAME=wpdb

# (optional) WordPress admin suggested values for automated setup - not use
d by the image automatically
# WP_ADMIN_USER=admin
# WP_ADMIN_PASSWORD=strongpassword
# WP_ADMIN_EMAIL=your-email@example.com
EOF
```

> Note: I set WORDPRESS_DB_HOST=mariadb:3306 assuming your MariaDB container is named mariadb. If it has a different container name, replace mariadb with that name (see step 4 to verify).

1. **Create** `docker-compose.yml` (this file contains only the WordPress service and uses the external `dockerlab` network)

```
version: "3.9"

services:
  wordpress:
    image: wordpress:latest
    container_name: wordpress
    restart: unless-stopped
    env_file:
```

```
    - .env
  ports:
    - "8050:80"            # host port 8050 → container 80
  volumes:
    - wp_data:/var/www/html
  networks:
    - dockerlab

volumes:
  wp_data:

networks:
  dockerlab:
    external: true
```

1. **(Optional but recommended) Verify MariaDB container name and that the `dockerlab` network exists**

```
# list containers (look for your mariadb container name)
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Status}}"

# check networks and confirm `dockerlab` exists
docker network ls

# inspect the dockerlab network to see connected containers
docker network inspect dockerlab
```

If your MariaDB container is named something else (for example `mariadb-db`), open `.env` and set:

```
WORDPRESS_DB_HOST=mariadb-db:3306
```

1. **Start WordPress**

```
docker compose up -d
```

1. **Verify WordPress container is running**

```
docker ps --filter name=wordpress
# or
docker logs -f wordpress   # watch logs for DB connection errors
```

1. **Open WordPress install in your browser**

   Visit: `http://YOUR_SERVER_IP:8050`

   Complete the WordPress setup (site title, admin user, password, email).

   If DB connection fails, check logs and confirm credentials in `.env` and the MariaDB container name.

2. **If you want to test DB connectivity from inside the WordPress container**

```
# open a shell inside wordpress container
docker exec -it wordpress bash

# from inside container test connecting with mysql client (mysql-client may no
t be present)
# if mysql client not installed, you can curl php info or use php -r to test socke
t or install mariadb-client in a debug container:
exit

# alternate: run a temporary mariadb client container on the same network:
docker run --rm -it --network dockerlab mariadb:11 sh -c 'mysql -h mariadb -
uwpuser -pwppass -e "SHOW DATABASES;"'
```

Replace `mariadb`, `wpuser`, `wppass` with your actual values.

1. **Use your existing phpMyAdmin**

   Since phpMyAdmin already exists on `dockerlab`, point it to your DB as you already do — no changes needed for phpMyAdmin. You can manage the DB from there.

2. **Stop / Remove**

```
# stop:
docker compose down

# stop and remove volumes (data loss):
docker compose down -v
```

Extra production notes (short)

- Set strong DB and admin passwords in `.env`. Consider using Docker secrets for more security.
- Add a reverse proxy (NGINX / Traefik) for TLS termination and to expose WordPress on standard ports (80/443).
- Back up `wp_data` and your MariaDB data regularly (mysqldump + store off-host).
- Limit direct exposure of ports; prefer proxy + firewall rules.

If you want, I can:

- produce a version of the Compose file that also auto-creates the WP admin user (via WP-CLI container or init script),
- add an NGINX reverse-proxy + Let's Encrypt example for port 80/443, or
- convert the `wp_data` volume to a host bind mount and show recommended file permissions.

Which one shall I generate next?