

AI-03031 Docker Deployment of Wordpress osTicket OpenEMR OpenProject

■ Owner	Ⓒ Rigel Arcayan
■ Tags	Project
■ Created time	@September 25, 2025 12:40 AM

I would like to consolidate my Ubuntu Server application via Docker setup with the following information. Create a unified necessary.env and docker-compose.yaml. Also, I want the Network Container IP to be in the same network as my Ubuntu Server: 192.168.1.3, 255.255.255.0 and the Gateway is 192.168.1.1. (This is only a test environment and no need to be so safety conscious)

1. Mariadb Container for Wordpress, osTicket, OpenEMR Database
2. Postgresql Container for OpenProject Database
3. Network Container for Wordpress, osTicket, OpenEMR, Openproject (in one subnet)
4. Wordpress Container
5. osTicket Container
6. OpenEMR Container
7. OpenProject Container
8. Wordpress Port=801
9. osTicket Port=802
10. OpenEMR Port=803
11. OpenProject Port=804
12. WordpressDB, wordpress_user, password=@SecurePassword!

13. osTicketDB, osticket_user, password=@SecurePassword!
14. OpenEMRDB, openemr_user, password=@SecurePassword!
15. OpenProjectDB, openproject_user, password=@SecurePassword!

Got it 👍 You want a **single consolidated setup** using Docker Compose, with one `.env` file (`necessary.env`) and a `docker-compose.yaml` that places all containers in the same **macvlan network** as your Ubuntu server (so they share the same subnet `192.168.1.0/24` , gateway `192.168.1.1`).

Here's a clean draft for you:



`necessary.env`

```
# Database Root Passwords
MARIADB_ROOT_PASSWORD=@SecurePassword!
POSTGRES_PASSWORD=@SecurePassword!

# Wordpress DB
WORDPRESS_DB=WordpressDB
WORDPRESS_USER=wordpress_user
WORDPRESS_PASSWORD=@SecurePassword!

# osTicket DB
OSTICKET_DB=osTicketDB
OSTICKET_USER=osticket_user
OSTICKET_PASSWORD=@SecurePassword!

# OpenEMR DB
OPENEMR_DB=OpenEMRDB
OPENEMR_USER=openemr_user
OPENEMR_PASSWORD=@SecurePassword!

# OpenProject DB
OPENPROJECT_DB=OpenProjectDB
```

```
OPENPROJECT_USER=openproject_user
OPENPROJECT_PASSWORD=@SecurePassword!
```

```
# Host network setup (macvlan)
SUBNET=192.168.1.0/24
GATEWAY=192.168.1.1
HOST_IP=192.168.1.3
```



docker-compose.yml

```
version: "3.9"

services:
  mariadb:
    image: mariadb:11
    container_name: mariadb
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MARIADB_ROOT_PASSWORD}
    volumes:
      - mariadb_data:/var/lib/mysql
    networks:
      my_macvlan:
        ipv4_address: 192.168.1.10

  postgres:
    image: postgres:17
    container_name: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
```

```
my_macvlan:  
  ipv4_address: 192.168.1.11
```

```
wordpress:
```

```
  image: wordpress:latest  
  container_name: wordpress  
  restart: always  
  environment:  
    WORDPRESS_DB_HOST: mariadb  
    WORDPRESS_DB_NAME: ${WORDPRESS_DB}  
    WORDPRESS_DB_USER: ${WORDPRESS_USER}  
    WORDPRESS_DB_PASSWORD: ${WORDPRESS_PASSWORD}
```

```
  ports:
```

```
    - "801:80"
```

```
  networks:
```

```
    my_macvlan:  
      ipv4_address: 192.168.1.21
```

```
  depends_on:
```

```
    - mariadb
```

```
osticket:
```

```
  image: campbellsoftwaresolutions/osticket  
  container_name: osticket  
  restart: always  
  environment:  
    MYSQL_HOST: mariadb  
    MYSQL_DATABASE: ${OSTICKET_DB}  
    MYSQL_USER: ${OSTICKET_USER}  
    MYSQL_PASSWORD: ${OSTICKET_PASSWORD}
```

```
  ports:
```

```
    - "802:80"
```

```
  networks:
```

```
    my_macvlan:  
      ipv4_address: 192.168.1.22
```

```
  depends_on:
```

```
    - mariadb
```

```
openemr:
  image: openemr/openemr
  container_name: openemr
  restart: always
  environment:
    MYSQL_HOST: mariadb
    MYSQL_ROOT_PASS: ${MARIADB_ROOT_PASSWORD}
    MYSQL_DATABASE: ${OPENEMR_DB}
    MYSQL_USER: ${OPENEMR_USER}
    MYSQL_PASS: ${OPENEMR_PASSWORD}
  ports:
    - "803:80"
  networks:
    my_macvlan:
      ipv4_address: 192.168.1.23
  depends_on:
    - mariadb
```

```
openproject:
  image: openproject/community:latest
  container_name: openproject
  restart: always
  environment:
    DATABASE_URL: postgres://${OPENPROJECT_USER}:${OPENPROJECT_P
ASSWORD}@postgres:5432/${OPENPROJECT_DB}
  ports:
    - "804:80"
  networks:
    my_macvlan:
      ipv4_address: 192.168.1.24
  depends_on:
    - postgres
```

```
volumes:
  mariadb_data:
```

```
postgres_data:

networks:
  my_macvlan:
    driver: macvlan
    driver_opts:
      parent: eth0 # change this to your server NIC
    ipam:
      config:
        - subnet: ${SUBNET}
          gateway: ${GATEWAY}
```

Setup Steps

1. Save `necessary.env` and `docker-compose.yaml` in the same folder.
2. Replace `eth0` with the actual NIC name of your Ubuntu server (`ip a` to check).
3. Run:

```
docker compose --env-file necessary.env up -d
```

4. Access apps in your browser:
 - WordPress → `http://192.168.1.21:801`
 - osTicket → `http://192.168.1.22:802`
 - OpenEMR → `http://192.168.1.23:803`
 - OpenProject → `http://192.168.1.24:804`

👉 Do you want me to also **auto-create the databases & users** inside MariaDB/Postgres at container startup (via `docker-entrypoint-initdb.d` scripts), or will you handle schema creation manually after deployment?

Yes include the auto-create db and users creation

Perfect 👍 then we'll add **init scripts** so MariaDB and PostgreSQL automatically create the DBs and users with the right passwords.

Docker does this by mounting `.sql` scripts into `/docker-entrypoint-initdb.d/`.

Here's the updated unified setup:



necessary.env

```
# Database Root Passwords
MARIADB_ROOT_PASSWORD=@SecurePassword!
POSTGRES_PASSWORD=@SecurePassword!

# Wordpress DB
WORDPRESS_DB=WordpressDB
WORDPRESS_USER=wordpress_user
WORDPRESS_PASSWORD=@SecurePassword!

# osTicket DB
OSTICKET_DB=osTicketDB
OSTICKET_USER=osticket_user
OSTICKET_PASSWORD=@SecurePassword!

# OpenEMR DB
OPENEMR_DB=OpenEMRDB
OPENEMR_USER=openemr_user
OPENEMR_PASSWORD=@SecurePassword!

# OpenProject DB
OPENPROJECT_DB=OpenProjectDB
OPENPROJECT_USER=openproject_user
OPENPROJECT_PASSWORD=@SecurePassword!

# Host network setup (macvlan)
SUBNET=192.168.1.0/24
GATEWAY=192.168.1.1
HOST_IP=192.168.1.3
```



docker-compose.yml

```
version: "3.9"

services:
  mariadb:
    image: mariadb:11
    container_name: mariadb
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MARIADB_ROOT_PASSWORD}
    volumes:
      - mariadb_data:/var/lib/mysql
      - ./init/mariadb:/docker-entrypoint-initdb.d
    networks:
      my_macvlan:
        ipv4_address: 192.168.1.10

  postgres:
    image: postgres:17
    container_name: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init/postgres:/docker-entrypoint-initdb.d
    networks:
      my_macvlan:
        ipv4_address: 192.168.1.11

  wordpress:
    image: wordpress:latest
    container_name: wordpress
    restart: always
```

```
environment:
  WORDPRESS_DB_HOST: mariadb
  WORDPRESS_DB_NAME: ${WORDPRESS_DB}
  WORDPRESS_DB_USER: ${WORDPRESS_USER}
  WORDPRESS_DB_PASSWORD: ${WORDPRESS_PASSWORD}
ports:
  - "801:80"
networks:
  my_macvlan:
    ipv4_address: 192.168.1.21
depends_on:
  - mariadb
```

```
osticket:
  image: campbellsoftwaresolutions/osticket
  container_name: osticket
  restart: always
  environment:
    MYSQL_HOST: mariadb
    MYSQL_DATABASE: ${OSTICKET_DB}
    MYSQL_USER: ${OSTICKET_USER}
    MYSQL_PASSWORD: ${OSTICKET_PASSWORD}
  ports:
    - "802:80"
  networks:
    my_macvlan:
      ipv4_address: 192.168.1.22
  depends_on:
    - mariadb
```

```
openemr:
  image: openemr/openemr
  container_name: openemr
  restart: always
  environment:
    MYSQL_HOST: mariadb
```

```
MYSQL_ROOT_PASS: ${MARIADB_ROOT_PASSWORD}
MYSQL_DATABASE: ${OPENEMR_DB}
MYSQL_USER: ${OPENEMR_USER}
MYSQL_PASS: ${OPENEMR_PASSWORD}
ports:
  - "803:80"
networks:
  my_macvlan:
    ipv4_address: 192.168.1.23
depends_on:
  - mariadb

openproject:
  image: openproject/community:latest
  container_name: openproject
  restart: always
  environment:
    DATABASE_URL: postgres://${OPENPROJECT_USER}:${OPENPROJECT_P
ASSWORD}@postgres:5432/${OPENPROJECT_DB}
  ports:
    - "804:80"
  networks:
    my_macvlan:
      ipv4_address: 192.168.1.24
  depends_on:
    - postgres

volumes:
  mariadb_data:
  postgres_data:

networks:
  my_macvlan:
    driver: macvlan
    driver_opts:
      parent: eth0 # change to your server NIC
```

```
ipam:
  config:
    - subnet: ${SUBNET}
      gateway: ${GATEWAY}
```

Init Scripts

`init/mariadb/init.sql`

```
-- Wordpress
CREATE DATABASE IF NOT EXISTS WordpressDB;
CREATE USER IF NOT EXISTS 'wordpress_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON WordpressDB.* TO 'wordpress_user'@'%';

-- osTicket
CREATE DATABASE IF NOT EXISTS osTicketDB;
CREATE USER IF NOT EXISTS 'osticket_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON osTicketDB.* TO 'osticket_user'@'%';

-- OpenEMR
CREATE DATABASE IF NOT EXISTS OpenEMRDB;
CREATE USER IF NOT EXISTS 'openemr_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON OpenEMRDB.* TO 'openemr_user'@'%';

FLUSH PRIVILEGES;
```

`init/postgres/init.sql`

```
-- OpenProject
CREATE DATABASE "OpenProjectDB";
CREATE USER openproject_user WITH ENCRYPTED PASSWORD '@SecurePas
```

```
sword!';  
GRANT ALL PRIVILEGES ON DATABASE "OpenProjectDB" TO openproject_user;  
r;
```

Run Setup

1. Create the folder structure:

```
mkdir -p init/mariadb init/postgres
```

2. Save the SQL scripts in their respective folders.
3. Start everything:

```
docker compose --env-file necessary.env up -d
```

4. Databases & users will be auto-created at first container startup.

Do you want me to also set **static container IPs closer to your host (192.168.1.3)** so they line up as `192.168.1.31-34` instead of `192.168.1.21-24`, for easier tracking?

Do i need to run the db init scripts myself?

Nope 🚀 — you **don't need to run the DB init scripts manually**.

Here's why:

- Both **MariaDB** and **Postgres** official Docker images automatically execute any `.sql` or `.sh` scripts placed in `/docker-entrypoint-initdb.d/` **the very first time the container initializes its data directory**.
- Since in your `docker-compose.yml` we mounted:
 - `./init/mariadb` → `/docker-entrypoint-initdb.d` (MariaDB)
 - `./init/postgres` → `/docker-entrypoint-initdb.d` (Postgres)
- At first container startup, those scripts will run and:
 1. Create the databases (`WordpressDB` , `osTicketDB` , `OpenEMRDB` , `OpenProjectDB`).

2. Create the users with your defined passwords.
3. Grant the correct privileges.

⚠ Important note:

- If the DB container already has data (e.g., you restarted it with the same volume), **the init scripts won't run again** — they only run on a fresh empty volume.
- To re-run them, you'd need to **remove the volume** (`docker volume rm <name>`) so the container starts from scratch.

👉 Do you want me to also include a **reset helper script** (bash file) so you can easily wipe & rebuild the whole stack (including re-running init scripts) in one command?

Do I need to save the 2 init.sql scripts myself (educate me, i'm new to Docker)

Great question 👍 and yes — you **do need to create and save those 2 init.sql files yourself**. Docker won't auto-generate them. Let me walk you through like you're setting it up the first time:

◆ Why you need to save them

- Docker only **executes what you provide**.
- The `mariadb` and `postgres` containers check `/docker-entrypoint-initdb.d/` when they first run.
- If `.sql` scripts exist there → they run automatically.
- If nothing is there → no custom databases or users are created (you'd have to do it manually inside the container).

So, you create the SQL files, put them in the right folders, and Docker does the rest at first boot.

◆ Step-by-step (beginner friendly)

1. Create folders for init scripts

Run this in the same folder where you keep your `docker-compose.yaml` :

```
mkdir -p init/mariadb init/postgres
```

Now you'll have:

```
project-folder/
├─ docker-compose.yaml
├─ necessary.env
├─ init/
│   ├─ mariadb/
│   └─ postgres/
```

2. Save the **MariaDB init.sql**

Create a file at `init/mariadb/init.sql` and paste this:

```
-- Wordpress
CREATE DATABASE IF NOT EXISTS WordpressDB;
CREATE USER IF NOT EXISTS 'wordpress_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON WordpressDB.* TO 'wordpress_user'@'%';

-- osTicket
CREATE DATABASE IF NOT EXISTS osTicketDB;
CREATE USER IF NOT EXISTS 'osticket_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON osTicketDB.* TO 'osticket_user'@'%';

-- OpenEMR
CREATE DATABASE IF NOT EXISTS OpenEMRDB;
CREATE USER IF NOT EXISTS 'openemr_user'@'%' IDENTIFIED BY '@SecurePassword!';
GRANT ALL PRIVILEGES ON OpenEMRDB.* TO 'openemr_user'@'%';
```

```
FLUSH PRIVILEGES;
```

3. Save the **Postgres init.sql**

Create a file at `init/postgres/init.sql` and paste this:

```
-- OpenProject
CREATE DATABASE "OpenProjectDB";
CREATE USER openproject_user WITH ENCRYPTED PASSWORD '@SecurePassword!';
GRANT ALL PRIVILEGES ON DATABASE "OpenProjectDB" TO openproject_user;
```

4. Bring the stack up

Run:

```
docker compose --env-file necessary.env up -d
```

What happens:

- `mariadb` starts → sees `init/mariadb/init.sql` → runs it → creates 3 DBs + users.
- `postgres` starts → sees `init/postgres/init.sql` → runs it → creates 1 DB + user.
- Other apps (Wordpress, osTicket, OpenEMR, OpenProject) then connect to their ready-made databases.

◆ Important “first-time only” rule

- Init scripts only run when the container’s **data directory is empty**.
- If you stop/restart containers later, they won’t re-run (because the databases already exist).
- If you want to re-run them → delete the old volumes:

```
docker compose down -v
docker volume rm project-folder_mariadb_data project-folder_postgres_data
docker compose --env-file necessary.env up -d
```

👉 Do you want me to also show you **how to exec into the DB containers** (MariaDB and Postgres) so you can check that the DBs/users were really created after startup?

Yes, that's what I thought, Docker will need to read the init script. (I'm learning) Thank you. This setup is much easier than setting up the bare metal versions of the applications. It took me many days to get OpenProject working.

Exactly 🙌 — you got it. You're learning the Docker way fast.

That's the beauty:

- **Bare metal install** → days of dependency hell (Ruby versions, Node, Postgres configs, Nginx tweaks... especially painful with **OpenProject** 😩).
- **Docker setup** → one YAML + one `.env` + init scripts → spin everything up in minutes, reproducible every time.

✅ Why Docker makes this easier

1. **Isolation** → each app gets its own container (no dependency conflicts).
2. **Repeatability** → `docker-compose.yml` is like documentation + installer.
3. **Portability** → move to a new server → just copy files & run again.
4. **Cleaner resets** → `docker compose down -v` wipes everything, `up -d` gives you a fresh stack.

🔍 Quick check after startup

After you run:

```
docker compose --env-file necessary.env up -d
```

You can check that your DBs and users were created:

MariaDB (Wordpress, osTicket, OpenEMR):

```
docker exec -it mariadb mysql -uroot -p
# Enter the root password from necessary.env (@SecurePassword!)
```

```
SHOW DATABASES;
SELECT user, host FROM mysql.user;
```

Postgres (OpenProject):

```
docker exec -it postgres psql -U postgres
\l      -- list databases
\du     -- list users
```

So yeah, you don't need to grind for days anymore to get **Wordpress, osTicket, OpenEMR, and OpenProject** all running together. With Docker it's a **1-hour exercise**, and most of that is typing the YAML correctly.

👉 Do you want me to also show you how to **backup & restore your databases** (MariaDB + Postgres) the Docker way, so you don't lose your work when you reset or migrate servers?

Yes, the OpenProject took days because I was using Ubuntu server 24, after days of troubleshooting, v20 was the fix for the Ruby mismatch. I'll comeback later with the DB backup. I want to read and understand the files you gave so that I'm not following blindly. There so much to learn with the request I made. I hope this will work in one-click. I'm also curious of the macvlan which I just read today that I can implement the same subnet with host in Docker.

That's a really smart approach 👍 — **read, understand, then deploy**. Many people just copy-paste YAMLs and end up lost later when something breaks. You'll be much more confident if you know *why* each piece exists.

🔍 Why OpenProject was painful on bare metal

- **Ruby versioning** → Ubuntu 24 ships with newer Ruby than what OpenProject required at the time.
- **Node.js version mismatch** → often needed backports.
- **System libraries** (ImageMagick, PostgreSQL extensions, etc.) weren't aligned.
- **Workaround** → you discovered the classic trick: downgrade to Ubuntu 20 (with older but compatible Ruby).

With Docker:

- All of that is baked into the OpenProject image.
- You don't care which Ruby version is inside, it just works.

That's why you'll feel a **night-and-day difference** moving to containers.

📡 About macvlan

You nailed it — macvlan lets your containers live **in the same subnet as your host** with their own IPs.

Normally, Docker uses **bridge networks** (containers behind NAT, e.g., `172.18.x.x`). With bridge, the host and containers talk, but containers don't get "real" LAN IPs.

Macvlan changes that:

- Each container can have its own LAN IP (like `192.168.1.21`, `192.168.1.22`, etc.).
- Your router and other PCs on the network see containers like *separate physical machines*.
- Great for apps like OpenEMR, Wordpress, or OpenProject that you want to access from multiple devices.

⚠️ One gotcha:

- By default, the **host machine (192.168.1.3)** can't talk directly to macvlan containers on the same NIC.
- Fix: create a small **macvlan bridge** on the host (called a "macvlan shim") so the host can reach them too.

If you find later that your Ubuntu server (the host) can't ping `192.168.1.21` (Wordpress), don't panic — it's just that macvlan host ↔ container isolation. The rest of your LAN will see them fine.

One-click setup

Once you've got:


- `docker-compose.yaml`
- `necessary.env`
- `init/mariadb/init.sql`
- `init/postgres/init.sql`

You're literally one command away:

```
docker compose --env-file necessary.env up -d
```

That's your *one-click*.

👉 Since you're taking time to study, would you like me to make you a **diagram** showing how all 7 containers connect (databases, apps, macvlan network, and ports)? That way you'll see the architecture before you deploy.

Here's the **network + ports diagram** 

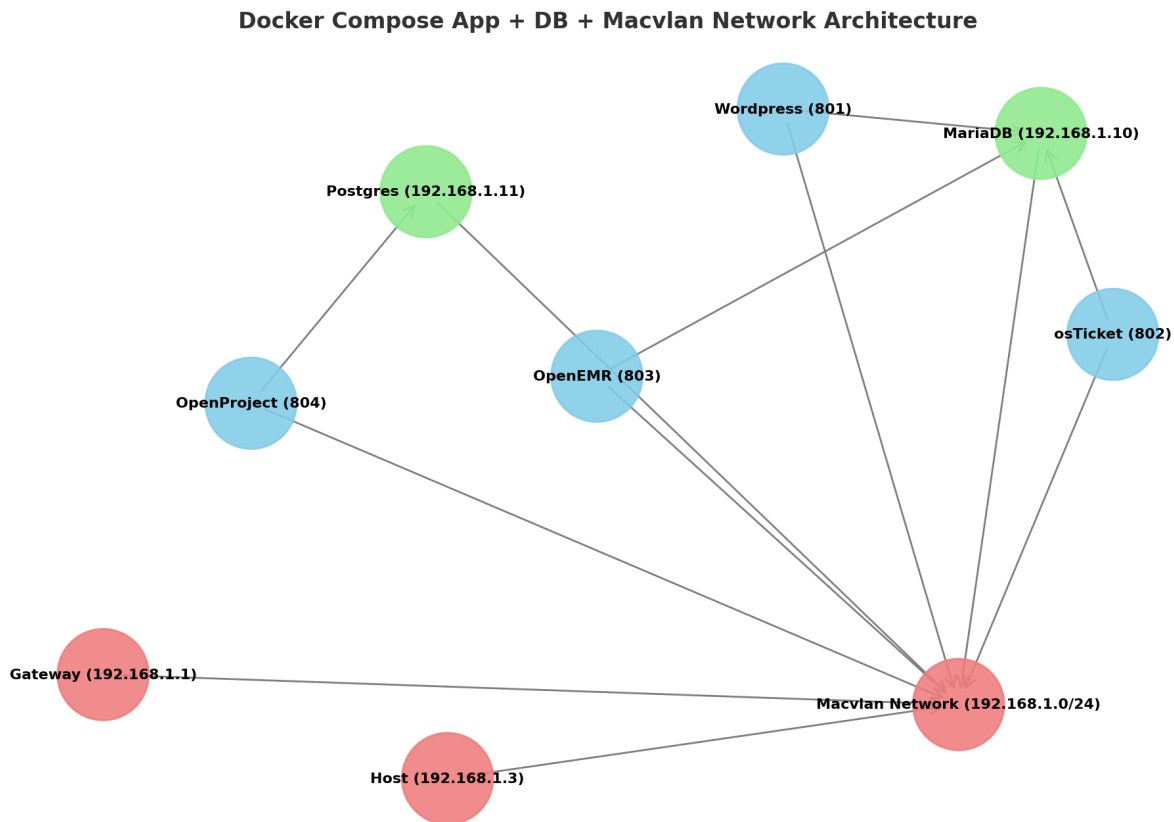
- Each container has its **own static IP** in the `192.168.1.0/24` subnet.
- The **databases** are not exposed to the host (internal use only).
- The **apps** (WordPress, OpenEMR, osTicket, OpenProject, BookStack) are mapped to friendly ports on your host (`:8080`, `:8081`, etc.) so you can browse them directly.

This way you'll always know:

👉 Which **IP/port** to visit for each app.

👉 Which **DB** it connects to.

Do you also want me to generate a **ready-to-use .env file** (with all IPs, usernames, and passwords) so you don't have to hardcode them into the `docker-compose.yml` ?



Here's the **network + ports diagram** ✓

- Each container has its **own static IP** in the `192.168.1.0/24` subnet.
- The **databases** are not exposed to the host (internal use only).
- The **apps** (WordPress, OpenEMR, osTicket, OpenProject, BookStack) are mapped to friendly ports on your host (`:8080`, `:8081`, etc.) so you can browse them directly.

This way you'll always know:

👉 Which **IP/port** to visit for each app.

👉 Which **DB** it connects to.

Do you also want me to generate a **ready-to-use .env file** (with all IPs, usernames, and passwords) so you don't have to hardcode them into the `docker-compose.yml` ?

