AI-03078 DockerLab Two Compose-Gemini

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	@October 15, 2025 8:55 AM

mini

Build a Self-contained Docker Lab in my Ubuntu Server VM in Hyper-V.

Build 2 docker-compose files:
1.docker-compose-core.yml, for nginx npm, portainer, mariadb, postgres, pgadmin, phymyadmin
2.docker-compose-app.yml, for wordpress, wikijs, openemr, openproject, osticket, inventree

HyperVM Host IP = 192.168.1.2
Gateway = 192.168.1.1
DNS = 8.8.8.8, 8.8.4.4
Domain = ubuntuserver3.local
Ubuntu Server Host IP = 192.168.1.3
Ubuntu Server Ethernet Name = eth0
Ubuntu Server OS Version = Ubuntu 25.04 (plucky)
Home Directory of All .yaml, sql, env = /home/Docker

Docker Lab information and objectives:
Provide all the sudo commands I need to install and run Docker, Docker-Compose and Portainer, NGINX Proxy Manager, pgAdmin, phpMyAdmin, MariaDB, PostgreSQL with detailed functional descriptions. Assume that I am fairly new to Ubunbu and Docker but had been functionally educated through hands on with ChatGPT and actual HyperV VM, Ubuntu Server and Docker Compose and Portainer.
I want to use 1 network for the entire Docker Lab (Do not use Macvlan)
Provide commands to create all necessary folders inside my /home/docker directory.
Save all persistent volumes of MariaDB and PostgreSQL inside /home/docker. I want to use 1 Mariadb and 1 Postgresql db to simplify studying of db tables.
Create a unified necessary.env and docker-compose.yaml.
Create a db init script to initialize MariaDB and PostgreSQL on first use.
Provide instructions on how to start the Docker Lab at first use
Provide instructions on how to access the completed apps from my internal network.
Provide instructions on how to access the completed apps using the internet.
Provide instructions on how to stop the Docker Lab safely without losing data.
Provide instructions on how to stop the Docker Lab and wipe out all data to start over.
Provide instructions on how to monitor the Docker Lab using Portainer.
Provide instructions on how to monitor the Docker Lab using command lines.
Provide a summary of sudo commands for simplicity.

Folder Structure
/home/docker (home of each of the applications, persistent volumes and db)
/home/docker/portainer
/home/docker/nginx
/home/docker/mariadb
/home/docker/postgresql
/home/docker/openproject
/home/docker/osticket
/home/docker/openemr

/home/docker/inventree
/home/docker/wordpress
/home/docker/wikijs

Network Ports Assignment:
OpenProject Port=801
osTicket Port=802
OpenEMR Port=803
Inventree Port=804
Wordpress=805
Wikijs=806

DB Credentials: db name, user, passwords
openproject, openproject_user, password=SecurePassword!
osticket, osticket_user, password=SecurePassword!
openemr, openemr_user, password=SecurePassword!
inventree, inventree_user, password=SecurePassword!
wordpress, wordpress_user, password=SecurePassword!
wikijs, wikijs_user, password=SecurePassword!

This is a comprehensive, multi-step deployment plan. Since you are new to Ubuntu and Docker, I've broken down every step and provided a consolidated summary of the required setup commands.

I will generate the following essential files for your Docker Lab in the `/home/Docker` directory:

1. `necessary.env`: The central configuration file for all database credentials and environment variables.
2. `db_init.sql`: The database initialization script to create all required schemas and users for your applications (MariaDB and PostgreSQL).
3. `docker-compose-core.yml`: Defines the central infrastructure (NPM, Portainer, MariaDB, PostgreSQL, Admin UIs).
4. `docker-compose-app.yml`: Defines all six application services.
5. `create_directories.sh`: A script to quickly set up the required persistent volume folders.
6. `sudo_commands_summary.md`: A summary of all required installation and setup commands for clarity.

Phase 1: Ubuntu & Docker Installation (Use these first)

First, you need to execute these commands on your Ubuntu Server VM (192.168.1.3) to install Docker, Docker Compose, and prepare the file structure.

I'm putting these in a summary file for easy copy/paste.

http://googleusercontent.com/immersive_entry_chip/0

http://googleusercontent.com/immersive_entry_chip/1

Phase 2: Configuration and Compose Files

These files define the environment variables, the database initialization script, and the two main Docker Compose files.

1\. Environment Variables (`necessary.env`)

This file holds all your sensitive credentials.

http://googleusercontent.com/immersive_entry_chip/2

2\. Database Initialization Script (`db_init.sql`)

This script uses the credentials from `necessary.env` to create all required databases and users in a single run. Place this file inside `/home/Docker/init`.

http://googleusercontent.com/immersive_entry_chip/3

3\. Core Services (`docker-compose-core.yml`)

This file sets up the shared network, NGINX Proxy Manager (NPM), Portainer, and the database servers with their respective admin UIs.

http://googleusercontent.com/immersive_entry_chip/4

4\. Application Services (`docker-compose-app.yml`)

This file defines the six applications, all connecting to the databases and sharing the network defined in the core file.

Note on Ports: *Per best practice with NGINX Proxy Manager, the application containers are using their default internal ports (usually 80 or 3000) and are* ***not*** *exposing them externally. NPM (in the core file) will handle the routing to the host IP (192.168.1.3) via the standard 80/443 ports.*

http://googleusercontent.com/immersive_entry_chip/5

Phase 3: Operational Instructions

1. Start the Docker Lab (First Use)

1. **Place Files:** Ensure all `.yml`, `.env`, `.sql`, and the `.sh` script are inside `/home/Docker`.
2. **Execute Setup Script:** Make the directory script executable and run it:

```
cd /home/Docker
chmod +x create_directories.sh
./create_directories.sh
3.  **Start the Stack:** Use the combined `docker compose` command to pull the images, create the network, and s
tart all services.
```bash
sudo docker compose -f docker-compose-core.yml -f docker-compose-app.yml up -d
    * *Functional Description:* The `-f` flag tells Docker to load multiple compose files. `up` starts the services. `-d` d
etaches the process, leaving them running in the background.
```

3. **Wait for DBs:** Wait a few minutes for the MariaDB and PostgreSQL containers to finish initializing and running the `db_init.sql` script.

2. Accessing the Completed Apps (Internal Network)

Since you are using NGINX Proxy Manager (NPM), all access will be routed through your Ubuntu Server's IP (192.168.1.3) via ports 80 and 443.

A. Initial NPM Setup

1. Open your web browser on your Host machine (192.168.1.2) and navigate to: `http://192.168.1.3:81`
2. Log in with the default credentials:
   - **Email:** `admin@example.com`
   - **Password:** `changeme`
3. *CRITICAL:* Immediately change your password and email in the NPM interface.

B. Creating Reverse Proxies

For each application, you need to create a **Proxy Host** in NPM. This maps a domain name (like `wordpress.ubuntuserver3.local`) to the correct container (e.g., `wordpress` container on its internal port 80).

1. In NPM, go to **Hosts** -\> **Proxy Hosts** -\> **Add Proxy Host**.
2. **Example: OpenProject**
   - **Domain Names:** `openproject.ubuntuserver3.local`
   - **Scheme:** `http`
```

- **Forward Hostname/IP:** `openproject` (This is the service name from `docker-compose-app.yml`)
- **Forward Port:** `8080` (The internal port used by the OpenProject container)
- **Cache Assets:** On
- **SSL:** Go to the SSL tab, choose **Request a new SSL Certificate**, and ensure **Force SSL** is on. (This is for self-signed certificates for your local domain).

C. Final Internal Access

To make `openproject.ubuntuserver3.local` resolve to `192.168.1.3`, you must edit the **hosts file** on your **Hyper-V Host PC (192.168.1.2)**.

- **Windows Host File Location:** `C:\\Windows\\System32\\drivers\\etc\\hosts`
- **Add the following lines:**

```
192.168.1.3 openproject.ubuntuserver3.local
192.168.1.3 osticket.ubuntuserver3.local
192.168.1.3 openemr.ubuntuserver3.local
192.168.1.3 inventree.ubuntuserver3.local
192.168.1.3 wordpress.ubuntuserver3.local
192.168.1.3 wikijs.ubuntuserver3.local
```

Now you can access the applications using their friendly domain names (e.g., `https://wordpress.ubuntuserver3.local/`).

3. Accessing the Completed Apps (Internet/External)

To access your lab from the internet, you must configure **Port Forwarding** on your home router (Gateway: 192.168.1.1).

- **Forward External Ports 80 (HTTP) and 443 (HTTPS)** to the **Internal IP 192.168.1.3**.
- This directs all internet traffic meant for your server to NGINX Proxy Manager, which then routes it to the correct application.
- You must set up a dynamic DNS (DDNS) service or have a static public IP to assign the domain names to.
- In NPM, when setting up the Proxy Host, you would use a public domain name (e.g., `wordpress.yourdomain.com`) and then use the NPM interface to automatically generate a valid Let's Encrypt SSL certificate.

4. Monitoring the Docker Lab

A. Using Portainer (GUI)

1. Navigate to `http://192.168.1.3:9000`.
2. Set up your initial administrator user and password.
3. Select the **Local** environment.
4. In the left sidebar, you can view:
   - **Containers:** Check status (running/stopped), view logs, restart, and see resource usage.
   - **Stacks:** View the entire `core-services` and `application-services` stacks.
   - **Networks:** See the shared `lab_network`.
   - **Volumes:** Verify that persistence is working by checking the `/home/Docker/...` volumes.

B. Using Command Line (CLI)

Command	Description
`sudo docker ps`	Lists all currently **running** containers, their status, ports, and names.
`sudo docker container ls -a`	Lists **all** containers, including stopped ones.
`sudo docker logs <container_name>`	Displays the output logs for a specific container (e.g., `sudo docker logs mariadb`).

Command	Description
`sudo docker stats`	Shows a live stream of CPU, memory, and network usage for all running containers.
`sudo docker network inspect lab_network`	Shows detailed information about the shared network and which containers are attached to it.

5. Stopping and Wiping the Docker Lab

Objective	Command	Description
Stop Safely (Save Data)	`sudo docker compose -f docker-compose-core.yml -f docker-compose-app.yml down`	This stops and removes the containers and networks, but **keeps all persistent data** in the `/home/Docker` volumes.
Start Again	`sudo docker compose -f docker-compose-core.yml -f docker-compose-app.yml up -d`	Starts the containers using the existing persistent data.
Wipe All Data & Containers (Start Over)	`sudo docker compose -f docker-compose-core.yml -f docker-compose-app.yml down -v --rmi all`	This is a destructive command. It stops containers, removes them, removes all local **volumes** (`-v`), and removes all downloaded **images** (`--rmi all`). **All database data and app files will be deleted.**

I have provided all the necessary files and detailed instructions to set up, run, access, and maintain your self-contained Docker Lab. Let me know if you want to modify any of the application configurations or troubleshoot the initial setup\!

1. **INSTALLATION AND SETUP SUMMARY**

Docker Lab Setup Commands (Run on Ubuntu Server VM)

This is a comprehensive summary of all commands needed to install Docker, Docker Compose, and set up your lab environment. You can copy and paste this entire block into your terminal.

Step 1: Install Docker and Dependencies

Update your package index and install necessary dependencies

sudo apt update -y
sudo apt install -y ca-certificates curl gnupg lsb-release

Add Docker's official GPG key

sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://www.google.com/search?q=https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

Add the Docker repository to Apt sources

echo

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" |

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Install Docker Engine (adjusting for Ubuntu 25.04 'plucky')

sudo apt update -y
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Step 2: Add Current User to the 'docker' Group

This allows you to run docker commands without 'sudo' (you must log out and back in for this to take effect)

sudo usermod -aG docker $USER

Step 3: Create the Root Docker Directory

mkdir -p /home/Docker
cd /home/Docker

Step 4: Create the Required Directories for Persistence

This command executes the script provided in 'create_directories.sh'

/bin/bash create_directories.sh

2. DIRECTORY CREATION SCRIPTS

#!/bin/bash

Define the root directory

ROOT_DIR="/home/Docker"

echo "Creating required folder structure in $ROOT_DIR..."

Core Services & Volumes

mkdir -p $ROOT_DIR/portainer
mkdir -p $ROOT_DIR/nginx
mkdir -p $ROOT_DIR/mariadb/data
mkdir -p $ROOT_DIR/postgresql/data
mkdir -p $ROOT_DIR/init # For DB initialization scripts

Application Volumes

mkdir -p $ROOT_DIR/openproject
mkdir -p $ROOT_DIR/osticket
mkdir -p $ROOT_DIR/openemr
mkdir -p $ROOT_DIR/inventree
mkdir -p $ROOT_DIR/wordpress
mkdir -p $ROOT_DIR/wikijs

echo "Setting permissions for persistent data volumes..."

Set correct permissions for the database volumes (critical for Linux host)

sudo chown -R 999:999 $ROOT_DIR/postgresql/data
sudo chown -R 999:999 $ROOT_DIR/mariadb/data

```
sudo chmod -R 775 $ROOT_DIR/mariadb/data
sudo chmod -R 775 $ROOT_DIR/postgresql/data

echo "Directory setup complete."
```

3. ENV File

```
# -- GENERAL SETTINGS ---
TZ=America/New_York
LAB_NETWORK_NAME=lab_network

# -- DATABASE ROOT CREDENTIALS (Used by Admin UIs and for management) ---

# MariaDB (for Wordpress, osTicket, OpenEMR)
MARIADB_ROOT_PASSWORD=MySecureRootPassword123!
DB_HOST_MARIA=mariadb
DB_PORT_MARIA=3306
PGADMIN_EMAIL=admin@ubuntuserver3.local
PGADMIN_PASSWORD=SecurePassword!
PHPMYADMIN_USERNAME=phpmyadmin
PHPMYADMIN_PASSWORD=SecurePassword!

# PostgreSQL (for WikiJS, OpenProject, Inventree)
POSTGRES_USER_ROOT=postgres_root
POSTGRES_PASSWORD_ROOT=MySecureRootPassword456!
DB_HOST_POSTGRES=postgresql
DB_PORT_POSTGRES=5432

# -- APPLICATION DATABASE CREDENTIALS ---

# Database names, users, and password (SecurePassword! as requested)

# Wordpress (MariaDB)
WORDPRESS_DB_NAME=wordpress
WORDPRESS_DB_USER=wordpress_user
WORDPRESS_DB_PASSWORD=SecurePassword!

# OpenEMR (MariaDB)
OPENEMR_DB_NAME=openemr
OPENEMR_DB_USER=openemr_user
OPENEMR_DB_PASSWORD=SecurePassword!

# OS Ticket (MariaDB)
```

OSTICKET_DB_NAME=osticket
OSTICKET_DB_USER=osticket_user
OSTICKET_DB_PASSWORD=SecurePassword!

WikiJS (PostgreSQL)

WIKIJS_DB_NAME=wikijs
WIKIJS_DB_USER=wikijs_user
WIKIJS_DB_PASSWORD=SecurePassword!

OpenProject (PostgreSQL)

OPENPROJECT_DB_NAME=openproject
OPENPROJECT_DB_USER=openproject_user
OPENPROJECT_DB_PASSWORD=SecurePassword!

Inventree (PostgreSQL)

INVENTREE_DB_NAME=inventree
INVENTREE_DB_USER=inventree_user
INVENTREE_DB_PASSWORD=SecurePassword!

4. DB INIT SCRIPTS

-- DATABASE INITIALIZATION SCRIPT ---

This script is executed on first run by MariaDB and PostgreSQL containers.

==========

MARIA DB Initialization (Uses 'MYSQL' variables from .env)

==========

Create databases and users for MariaDB-based apps

CREATE DATABASE IF NOT EXISTS ${WORDPRESS_DB_NAME};
CREATE USER IF NOT EXISTS '${WORDPRESS_DB_USER}'@'%' IDENTIFIED BY '${WORDPRESS_DB_PASSWORD}';
GRANT ALL PRIVILEGES ON ${WORDPRESS_DB_NAME}.* TO '${WORDPRESS_DB_USER}'@'%';

CREATE DATABASE IF NOT EXISTS ${OPENEMR_DB_NAME};
CREATE USER IF NOT EXISTS '${OPENEMR_DB_USER}'@'%' IDENTIFIED BY '${OPENEMR_DB_PASSWORD}';
GRANT ALL PRIVILEGES ON ${OPENEMR_DB_NAME}.* TO '${OPENEMR_DB_USER}'@'%';

CREATE DATABASE IF NOT EXISTS ${OSTICKET_DB_NAME};
CREATE USER IF NOT EXISTS '${OSTICKET_DB_USER}'@'%' IDENTIFIED BY '${OSTICKET_DB_PASSWORD}';
GRANT ALL PRIVILEGES ON ${OSTICKET_DB_NAME}.* TO '${OSTICKET_DB_USER}'@'%';

FLUSH PRIVILEGES;

==========

POSTGRESQL Initialization (Uses 'POSTGRES' variables from .env)

==========

PostgreSQL requires these commands to be run externally or through the entrypoint.

The standard PostgreSQL Docker image entrypoint will process the following SQL.

- - Create users and databases for PostgreSQL-based apps
  CREATE USER ${WIKIJS_DB_USER} WITH PASSWORD '${WIKIJS_DB_PASSWORD}';
  CREATE DATABASE ${WIKIJS_DB_NAME} OWNER ${WIKIJS_DB_USER};

CREATE USER ${OPENPROJECT_DB_USER} WITH PASSWORD '${OPENPROJECT_DB_PASSWORD}';
CREATE DATABASE ${OPENPROJECT_DB_NAME} OWNER ${OPENPROJECT_DB_USER};

CREATE USER ${INVENTREE_DB_USER} WITH PASSWORD '${INVENTREE_DB_PASSWORD}';
CREATE DATABASE ${INVENTREE_DB_NAME} OWNER ${INVENTREE_DB_USER};

- - Note: The PostgreSQL entrypoint requires specific file naming (.sql, .sh)
  -- and proper user authentication for execution. The above uses standard
  -- substitution variables from the main postgres container.

4. DOCKER LAB CORE

```
version: '3.8'

# Load environment variables from necessary.env
name: core-services

services:
  # -----------------------------------------------------
  # 1. NGINX Proxy Manager (NPM) - Handles all external traffic
  # Description: Provides a web interface to easily create reverse proxies,
  #              manage SSL certificates (Let's Encrypt), and handle routing
  #              for all applications. Access via 192.168.1.3:81
  # -----------------------------------------------------
  nginx:
    image: jc21/nginx-proxy-manager:latest
    container_name: nginx-proxy-manager
    restart: unless-stopped
    ports:
      - "80:80"     # HTTP for forwarding and challenges
      - "443:443"   # HTTPS for secure forwarding
      - "81:81"     # Admin web interface (192.168.1.3:81)
    volumes:
      - /home/Docker/nginx/data:/data
      - /home/Docker/nginx/letsencrypt:/etc/letsencrypt
```

```
  environment:
    # Default admin user: admin@example.com / changeme
    # You MUST change this immediately upon first login.
    - TZ=${TZ}
  networks:
    - ${LAB_NETWORK_NAME}

# ------------------------------------------------------
# 2. Portainer - Docker Management GUI
# Description: A user-friendly web UI for managing all your Docker resources,
#              including containers, images, volumes, and networks. Access via 192.168.1.3:9000
# ------------------------------------------------------
portainer:
  image: portainer/portainer-ce:latest
  container_name: portainer
  restart: unless-stopped
  ports:
    - "9000:9000" # Portainer web interface
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - /home/Docker/portainer/data:/data
  networks:
    - ${LAB_NETWORK_NAME}

# ------------------------------------------------------
# 3. MariaDB - Relational Database Server
# Description: Primary database for Wordpress, osTicket, OpenEMR.
# ------------------------------------------------------
mariadb:
  image: mariadb:10.11
  container_name: mariadb
  restart: unless-stopped
  environment:
    - MARIADB_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
    - TZ=${TZ}
  volumes:
    - /home/Docker/mariadb/data:/var/lib/mysql
    - /home/Docker/init/db_init.sql:/docker-entrypoint-initdb.d/db_init.sql:ro
  # Only exposing the port for external access (e.g., local SQL client)
  ports:
    - "3306:3306"
  networks:
    - ${LAB_NETWORK_NAME}

# ------------------------------------------------------
# 4. PostgreSQL - Relational Database Server
# Description: Primary database for WikiJS, OpenProject, Inventree.
# ------------------------------------------------------
postgresql:
  image: postgres:15-alpine
  container_name: postgresql
  restart: unless-stopped
  environment:
    - POSTGRES_USER=${POSTGRES_USER_ROOT}
```

```
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD_ROOT}
      - TZ=${TZ}
    volumes:
      - /home/Docker/postgresql/data:/var/lib/postgresql/data
      - /home/Docker/init/db_init.sql:/docker-entrypoint-initdb.d/db_init.sql:ro
    # Only exposing the port for external access (e.g., local SQL client)
    ports:
      - "5432:5432"
    networks:
      - ${LAB_NETWORK_NAME}

  # ------------------------------------------------------
  # 5. phpMyAdmin - MariaDB GUI
  # Description: Web interface to manage MariaDB. Access via 192.168.1.3:8080
  # ------------------------------------------------------
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: phpmyadmin
    restart: unless-stopped
    ports:
      - "8080:80" # Access port
    environment:
      - PMA_HOST=${DB_HOST_MARIA}
      - PMA_PORT=${DB_PORT_MARIA}
      - MYSQL_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
      - TZ=${TZ}
    depends_on:
      - mariadb
    networks:
      - ${LAB_NETWORK_NAME}

  # ------------------------------------------------------
  # 6. pgAdmin - PostgreSQL GUI
  # Description: Web interface to manage PostgreSQL. Access via 192.168.1.3:8081
  # ------------------------------------------------------
  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: pgadmin
    restart: unless-stopped
    ports:
      - "8081:80" # Access port
    environment:
      - PGADMIN_DEFAULT_EMAIL=${PGADMIN_EMAIL}
      - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_PASSWORD}
      - TZ=${TZ}
    depends_on:
      - postgresql
    networks:
      - ${LAB_NETWORK_NAME}

networks:
  # Define the single, common network for all services
  ${LAB_NETWORK_NAME}:
```

```
    driver: bridge
    external: false
```

DOCKER LAB APPLICATION

```
version: '3.8'

# Load environment variables from necessary.env
name: application-services

services:
  # ------------------------------------------------------
  # 1. OpenProject (PostgreSQL)
  # Internal Port: 8080.
  # ------------------------------------------------------
  openproject:
    image: openproject/community:13
    container_name: openproject
    restart: unless-stopped
    depends_on:
      - postgresql
    environment:
      # Database connection (using database service name and app-specific user)
      DATABASE_URL: "postgresql://${OPENPROJECT_DB_USER}:${OPENPROJECT_DB_PASSWORD}@${DB_HOST_POS
TGRES}:${DB_PORT_POSTGRES}/${OPENPROJECT_DB_NAME}"
      APP_HOST: "openproject.ubuntuserver3.local" # Used for internal redirects
      RAILS_CACHE_STORE: "redis"
      # This service will be accessed via NPM on its internal port 8080
      PORT: 8080
    volumes:
      - /home/Docker/openproject:/var/openproject
    networks:
      - ${LAB_NETWORK_NAME}

  # ------------------------------------------------------
  # 2. osTicket (MariaDB)
  # Internal Port: 80.
  # ------------------------------------------------------
  osticket:
    image: osixia/php:8.2
    container_name: osticket
    restart: unless-stopped
    depends_on:
      - mariadb
    environment:
      # PHP setup, osTicket is installed via the web installer on first visit
      TZ: ${TZ}
      OSTICKET_ENV: "production"
    volumes:
      - /home/Docker/osticket:/var/www/html
    networks:
      - ${LAB_NETWORK_NAME}
```

```
# ------------------------------------------------------
# 3. OpenEMR (MariaDB)
# Internal Port: 80.
# ------------------------------------------------------
openemr:
  image: openemr/openemr:7.0.2
  container_name: openemr
  restart: unless-stopped
  depends_on:
    - mariadb
  environment:
    DB_HOST: ${DB_HOST_MARIA}
    DB_USER: ${OPENEMR_DB_USER}
    DB_PASS: ${OPENEMR_DB_PASSWORD}
    DB_DATABASE: ${OPENEMR_DB_NAME}
  volumes:
    - /home/Docker/openemr:/var/www/localhost/htdocs/openemr/sites/default
  networks:
    - ${LAB_NETWORK_NAME}

# ------------------------------------------------------
# 4. Inventree (PostgreSQL)
# Internal Port: 8000.
# ------------------------------------------------------
inventree:
  image: inventree/inventree:latest
  container_name: inventree
  restart: unless-stopped
  depends_on:
    - postgresql
  environment:
    # Inventree settings
    INVENTREE_DB_NAME: ${INVENTREE_DB_NAME}
    INVENTREE_DB_USER: ${INVENTREE_DB_USER}
    INVENTREE_DB_PASSWORD: ${INVENTREE_DB_PASSWORD}
    INVENTREE_DB_HOST: ${DB_HOST_POSTGRES}
    INVENTREE_DB_PORT: ${DB_PORT_POSTGRES}
    # Required for first run setup
    INVENTREE_ADMIN_USER: "admin"
    INVENTREE_ADMIN_PASSWORD: "adminpassword" # CHANGE THIS LATER!
  volumes:
    - /home/Docker/inventree:/home/inventree/data
  networks:
    - ${LAB_NETWORK_NAME}

# ------------------------------------------------------
# 5. Wordpress (MariaDB)
# Internal Port: 80.
# ------------------------------------------------------
wordpress:
  image: wordpress:latest
  container_name: wordpress
  restart: unless-stopped
```

```
    depends_on:
      - mariadb
    environment:
      WORDPRESS_DB_HOST: ${DB_HOST_MARIA}:${DB_PORT_MARIA}
      WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
      WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
      WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}
    volumes:
      - /home/Docker/wordpress:/var/www/html
    networks:
      - ${LAB_NETWORK_NAME}

  # ------------------------------------------------------
  # 6. WikiJS (PostgreSQL)
  # Internal Port: 3000.
  # ------------------------------------------------------
  wikijs:
    image: ghcr.io/requarks/wiki:2
    container_name: wikijs
    restart: unless-stopped
    depends_on:
      - postgresql
    environment:
      DB_TYPE: postgres
      DB_HOST: ${DB_HOST_POSTGRES}
      DB_PORT: ${DB_PORT_POSTGRES}
      DB_USER: ${WIKIJS_DB_USER}
      DB_PASS: ${WIKIJS_DB_PASSWORD}
      DB_NAME: ${WIKIJS_DB_NAME}
    volumes:
      - /home/Docker/wikijs:/data/wiki
    networks:
      - ${LAB_NETWORK_NAME}

networks:
  # The network must be declared as 'external' here so the app stack can attach
  # to the network created by the core-services stack.
  ${LAB_NETWORK_NAME}:
    external: true
```