AI-02080 Docker Database in Azure

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	@September 7, 2025 2:12 PM

I'm new to Azure, Ubuntu server and Docker, I need help in the following objectives.

1. **Use 1 vm for my running Ubuntu server in Azure**
2. **Create a docker container MySQL database as backend db**
3. **Create a docker container Mariadb as backend db**
4. **Create a docker container Postgres as backend db**
5. **Run docker OpenEMR, Wordpress, osTicket, BookStack, OpenProject on each container and connect to each respective backend databases**

Write a comprehensive step by step instructions on how to deploy all the applications mentioned as standalone containers connecting to the backend database

Direct answer — short summary

Yes — you can run **one Ubuntu VM in Azure** and host **three database containers (MySQL, MariaDB, Postgres)** plus **five application containers (OpenEMR, WordPress, osTicket, BookStack, OpenProject)** on that VM using Docker and Docker Compose. Below I give a clear, production-minded **step-by-step plan** (Azure VM → install Docker → prepare DB init scripts → `docker compose` file → deploy → verify & backup). I also include a ready-to-use `docker-compose.yml`, `.env` example,

and small SQL init scripts you can paste and run. Key vendor docs are linked as inline citations so you can verify specifics.

(If you prefer I can produce separate `docker-compose` files per app, add TLS + automatic Let's Encrypt via a reverse proxy (Traefik/Nginx Proxy Manager), or a one-line script to build everything — tell me and I'll add it.)

Key reasons (concise)

1. Single VM + Docker is simple and low-cost for a test/dev or small production proof-of-concept — use a B-series small VM and scale later. (Microsoft Learn)
2. Docker Compose makes wiring apps → DBs reliable and reproducible (you get volumes, restart policies, networks). (Docker Documentation)
3. Official container images provide documented environment variables for automatic install + DB init hooks (MySQL/MariaDB/Postgres images and app images). (Docker Hub, LinuxServer)

Alternatives (brief)

- Use Azure Container Instances or App Service for individual apps (no VM management) — good if you prefer managed infra. (Microsoft Learn)
- Use a reverse proxy (Traefik / Nginx Proxy Manager) + Let's Encrypt for friendly domains & TLS (recommended before opening to public).
- Run each app in its own VM / Kubernetes cluster for production-grade isolation (more complex).

Action plan — step-by-step (copy/paste commands & files)

> Assumptions: you're comfortable using the Azure Portal or az CLI, and you have an Azure subscription and an SSH key. I'll show Azure CLI steps + the

> commands to run on the Ubuntu VM. Official quickstarts referenced below. (Microsoft Learn, Docker Documentation)

---

1) Create an Ubuntu VM in Azure (example via Azure CLI)

Replace resource names and `--location` as needed.

```
# create resource group
az group create --name myResourceGroup --location eastus

# create VM (Ubuntu LTS), 2 vCPU / 4GB is a reasonable starting point (Stand
ard_B2s)
az vm create \
  --resource-group myResourceGroup \
  --name dockerVM \
  --image UbuntuLTS \
  --size Standard_B2s \
  --admin-username azureuser \
  --generate-ssh-keys

# open necessary ports (SSH, HTTP, a few test ports)
az vm open-port --resource-group myResourceGroup --name dockerVM --po
rt 22
az vm open-port --resource-group myResourceGroup --name dockerVM --po
rt 80
az vm open-port --resource-group myResourceGroup --name dockerVM --po
rt 443
# open example app ports 8080-8085 (you may prefer using a reverse-proxy
later)
az vm open-port --resource-group myResourceGroup --name dockerVM --po
rt 8080-8085
```

Docs: Azure VM quickstart and `az vm open-port`. (Microsoft Learn)

---

2) SSH into the VM and install Docker + Docker Compose

Follow the official Docker Engine install for Ubuntu (commands below follow Docker docs). After SSH, run:

```
# SSH (replace IP)
ssh azureuser@<VM_PUBLIC_IP>

# update
sudo apt update -y && sudo apt upgrade -y

# prerequisites and Docker repo keys
sudo apt-get install -y ca-certificates curl gnupg lsb-release
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearm
or -o /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
null

# install docker engine + compose plugin
sudo apt update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-pl
ugin docker-compose-plugin

# allow current user to run docker
sudo usermod -aG docker $USER
# either logout/login or run
newgrp docker
```

Official Docker install docs and Compose docs. (Docker Documentation)

Verify:

```
docker --version
docker compose version
```

---

3) Plan the DB ↔ App mapping (my recommended mapping)

- **MySQL container** → hosts DBs for **OpenEMR** and **osTicket**.
- **MariaDB container** → hosts DBs for **WordPress** and **BookStack**.
- **Postgres container** → used by **OpenProject**.

This uses three DB containers (one each) and multiple databases within them — simple and efficient.

---

4) Create directories and DB init scripts on the VM

We'll use `docker-entrypoint-initdb.d` to create additional DBs/users at first-db-start.

```
mkdir -p ~/docker/{mysql,mariadb}
cd ~/docker

# MySQL init (creates openemr + osticket DB + users)
cat > mysql/initdb.sql <<'SQL'
CREATE DATABASE IF NOT EXISTS openemr CHARACTER SET utf8mb4 COLL
ATE utf8mb4_general_ci;
CREATE DATABASE IF NOT EXISTS osticket CHARACTER SET utf8mb4 COLLA
TE utf8mb4_general_ci;
CREATE USER IF NOT EXISTS 'openemr'@'%' IDENTIFIED BY 'openemr_pas
s';
GRANT ALL PRIVILEGES ON openemr.* TO 'openemr'@'%';
CREATE USER IF NOT EXISTS 'osticket'@'%' IDENTIFIED BY 'osticket_pass';
GRANT ALL PRIVILEGES ON osticket.* TO 'osticket'@'%';
FLUSH PRIVILEGES;
SQL
```

```
# MariaDB init (wordpress + bookstack DBs & users)
cat > mariadb/initdb.sql <<'SQL'
CREATE DATABASE IF NOT EXISTS wordpress CHARACTER SET utf8mb4 COL
LATE utf8mb4_unicode_ci;
CREATE DATABASE IF NOT EXISTS bookstack CHARACTER SET utf8mb4 COL
LATE utf8mb4_unicode_ci;
CREATE USER IF NOT EXISTS 'wp_user'@'%' IDENTIFIED BY 'wp_pass';
GRANT ALL PRIVILEGES ON wordpress.* TO 'wp_user'@'%';
CREATE USER IF NOT EXISTS 'bookstack'@'%' IDENTIFIED BY 'bookstack_pa
ss';
GRANT ALL PRIVILEGES ON bookstack.* TO 'bookstack'@'%';
FLUSH PRIVILEGES;
SQL
```

(You should **replace** passwords above with secure random secrets — see step 6.)

---

5) Create a `.env` (secrets) file

Create `~/docker/.env` with your passwords (this avoids hardcoding in `docker-compose.yml`). Example:

```
# mysql root password (for mysql container)
MYSQL_ROOT_PASSWORD=ReplaceRootMySQL

# mariadb root password
MARIADB_ROOT_PASSWORD=ReplaceRootMaria

# postgres
POSTGRES_PASSWORD=ReplacePostgres

# app-specific DB passwords (must match init SQL if you used those)
OPENEMR_DB_PASS=openemr_pass
OSTICKET_DB_PASS=osticket_pass
WP_DB_PASS=wp_pass
BOOKSTACK_DB_PASS=bookstack_pass
```

```
# OpenProject host & secret
OPENPROJECT_HOST=your.vm.ip.or.domain:8084
OPENPROJECT_SECRET_KEY_BASE=replace_with_generated_secret
```

Generate secure values for the passwords, e.g.:

```
# short random password
openssl rand -base64 18
# OpenProject secret key base
openssl rand -hex 64
```

Docker image env variable behavior: official MySQL/MariaDB/Postgres images accept these standard env variables. (Docker Hub, MariaDB)

---

6) Create the `docker-compose.yml`

Below is a functional, **single** `docker-compose.yml` that creates the 3 DB containers and the 5 app containers, wires them on an internal network, persists data to named volumes, and **does not publish DB ports to the public host** (safer). Save it as `~/docker/docker-compose.yml`.

```
version: '3.8'
services:

 # ---------- MySQL for OpenEMR + osTicket ----------
 mysql:
  image: mysql:8.0
  container_name: mysql
  restart: unless-stopped
  env_file: .env
  environment:
   - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
  volumes:
   - mysql_data:/var/lib/mysql
   - ./mysql/initdb.sql:/docker-entrypoint-initdb.d/init.sql:ro
  networks:
```

```
    - backend

  openemr:
    image: openemr/openemr:latest
    container_name: openemr
    depends_on:
      - mysql
    environment:
      - MYSQL_HOST=mysql
      - MYSQL_ROOT_PASS=${MYSQL_ROOT_PASSWORD}
      - MYSQL_USER=openemr
      - MYSQL_PASS=${OPENEMR_DB_PASS}
      - OE_USER=admin
      - OE_PASS=adminpass
    ports:
      - "8081:80"
    volumes:
      - openemr_sites:/var/www/localhost/htdocs/openemr
    networks:
      - backend

  osticket:
    image: docker.io/tiredofit/osticket:latest
    container_name: osticket
    depends_on:
      - mysql
    env_file: .env
    environment:
      - DB_HOST=mysql
      - DB_PORT=3306
      - DB_NAME=osticket
      - DB_USER=osticket
      - DB_PASS=${OSTICKET_DB_PASS}
      - ADMIN_USER=admin
      - ADMIN_PASS=adminpass
    ports:
```

```
    - "8082:80"
  volumes:
    - osticket_data:/data
  networks:
    - backend

# ---------- MariaDB for WordPress + BookStack ----------
mariadb:
  image: mariadb:10.6
  container_name: mariadb
  restart: unless-stopped
  env_file: .env
  environment:
    - MYSQL_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
  volumes:
    - mariadb_data:/var/lib/mysql
    - ./mariadb/initdb.sql:/docker-entrypoint-initdb.d/init.sql:ro
  networks:
    - backend

wordpress:
  image: wordpress:latest
  container_name: wordpress
  depends_on:
    - mariadb
  env_file: .env
  environment:
    - WORDPRESS_DB_HOST=mariadb:3306
    - WORDPRESS_DB_USER=wp_user
    - WORDPRESS_DB_PASSWORD=${WP_DB_PASS}
    - WORDPRESS_DB_NAME=wordpress
  ports:
    - "8080:80"
  volumes:
    - wordpress_data:/var/www/html
  networks:
```

```
    - backend

bookstack:
  image: lscr.io/linuxserver/bookstack:latest
  container_name: bookstack
  depends_on:
    - mariadb
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=Etc/UTC
    - APP_URL=http://${OPENPROJECT_HOST:-localhost}:8083
    - DB_HOST=mariadb
    - DB_PORT=3306
    - DB_USERNAME=bookstack
    - DB_PASSWORD=${BOOKSTACK_DB_PASS}
    - DB_DATABASE=bookstack
  ports:
    - "8083:80"
  volumes:
    - bookstack_config:/config
  networks:
    - backend

# ---------- Postgres for OpenProject ----------
postgres:
  image: postgres:15
  container_name: postgres
  restart: unless-stopped
  env_file: .env
  environment:
    - POSTGRES_USER=openproject
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_DB=openproject
  volumes:
    - postgres_data:/var/lib/postgresql/data
```

```
    networks:
      - backend

  openproject:
    image: openproject/openproject:16
    container_name: openproject
    depends_on:
      - postgres
    environment:
      - DATABASE_URL=postgres://openproject:${POSTGRES_PASSWORD}@po
stgres:5432/openproject
      - OPENPROJECT_HOST__NAME=${OPENPROJECT_HOST}
      - OPENPROJECT_HTTPS=false
      - OPENPROJECT_SECRET_KEY_BASE=${OPENPROJECT_SECRET_KEY_B
ASE}
    ports:
      - "8084:80"
    volumes:
      - openproject_data:/var/db/openproject
    networks:
      - backend

volumes:
  mysql_data:
  mariadb_data:
  postgres_data:
  openemr_sites:
  wordpress_data:
  bookstack_config:
  osticket_data:
  openproject_data:

networks:
  backend:
    driver: bridge
```

Notes & references:

- BookStack uses the linuxserver image with `DB_*` env vars; see linuxserver BookStack docs. (LinuxServer)
- osTicket image (tiredofit) accepts `DB_HOST`, `DB_NAME`, `DB_USER`, `DB_PASS` etc. See its README. (GitHub)
- OpenEMR official image requires `MYSQL_HOST` and `MYSQL_ROOT_PASS` for auto-install; see OpenEMR docker docs. (Docker Hub)
- OpenProject supports `DATABASE_URL` (postgres connection URL). (OpenProject.org)

---

7) Start everything

From `~/docker`:

```
# ensure .env, init sql files, and docker-compose.yml exist in ~/docker
docker compose up -d
docker compose ps
```

Check logs if a service is not healthy:

```
docker compose logs -f openemr
docker compose logs -f wordpress
# or for any service
docker logs -f <container_name>
```

---

8) First-access URLs (example)

- WordPress → `http://<VM_PUBLIC_IP>:8080`
- OpenEMR → `http://<VM_PUBLIC_IP>:8081`
- osTicket → `http://<VM_PUBLIC_IP>:8082`
- BookStack → `http://<VM_PUBLIC_IP>:8083`
- OpenProject → `http://<VM_PUBLIC_IP>:8084`

(Replace with your domain + reverse proxy when you add TLS.)

9) Backups & upgrades (brief)

- MySQL dump example:

```
docker exec -i mysql mysqldump -uroot -p"${MYSQL_ROOT_PASSWORD}" o
penemr > openemr_backup.sql
```

- Postgres backup:

```
docker exec -t postgres pg_dump -U openproject openproject > openproject_
backup.sql
```

- To update images: `docker compose pull && docker compose up -d` then verify. Use volumes and keep backups before upgrades. See Docker update docs. (Docker Documentation)

10) Security & production notes (must-read)

- **Do not** expose DB ports (3306/5432) to the public internet — this compose does not publish them. Instead apps connect over Docker network (good). (Docker Documentation)
- Use **strong, unique passwords** (use `openssl rand -base64 ...`).
- Put a **reverse proxy** (Traefik / Nginx Proxy Manager) in front of the apps and terminate TLS there (Let's Encrypt) before opening to users. Traefik/NGINX setup is a short follow-up I can add.
- Monitor VM CPU / RAM; if load grows, resize the VM (Azure allows scaling up/down). (Microsoft Learn)

Useful references (quick)

- Create Linux VM in Azure (CLI + Portal): Microsoft Learn quickstart. (Microsoft Learn)
- Install Docker Engine on Ubuntu (official Docker docs). (Docker Documentation)
- Docker Compose docs / examples. (Docker Documentation)
- MySQL / Postgres official images (env var docs). (Docker Hub)
- BookStack (linuxserver) Docker docs. (LinuxServer)
- osTicket Docker image (tiredofit README). (GitHub)
- OpenProject Docker docs (database/config). (OpenProject.org)
- OpenEMR Docker image info. (Docker Hub)

If you want, I can now:

- generate the **exact** `docker-compose.yml` + `.env` and `initdb.sql` files tailored with secure random passwords (I will show them on-screen so you can paste), **or**
- produce a one-line script that will create the directories + files and run `docker compose up -d` on your Ubuntu VM, **or**
- add a Traefik reverse proxy + Let's Encrypt recipe so apps are served on standard ports 80/443 with proper hostnames.

Which of those would you like me to produce next?