Compose System Architecture

Components Overview

NOTE: All services communicate via Docker’s internal DNS-based service names (e.g., *database-master*, *proxysql*, *application*) and are written in *italic*

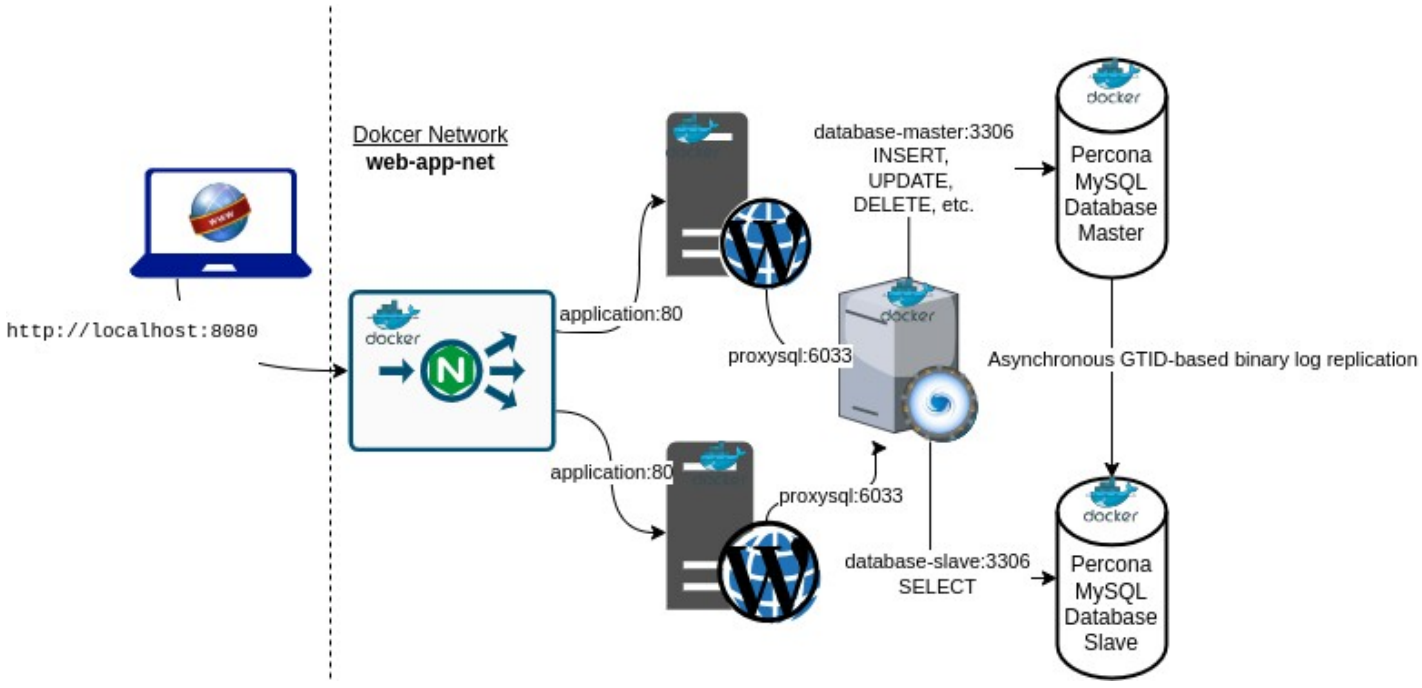
In **bold** are container names as seen with “docker ps” or “docker compose ps” commands

Service_name	**container_name**	**Role**
load-balancer	**nginx-ls**	HTTP Load Balancer that distributes requests to WP application replicas
application	**kp-wordpress-application**	Application running behind NGINX in 2 replicated containers
proxysql	**database-lb**	SQL load balancer between WordPress and MySQL master/slave
database-master/ database-slave	**master-db/slave-db**	Master-slave DB cluster

Internal Architecture and Network Flow

The system is built on Docker Compose using a custom bridge network (e.g., web-app-net). All services are connected to this internal network for service discovery, DNS-based addressing, and secure communication

Network Overview



Key Points

- The host browser accesses the system through the host's mapped port 8080
    - http://localhost:8080
- **nginx-lb** (*load-balancer*), running inside a container* listens on localhost:80, receives HTTP traffic and forwards it to application:80 service replicas using round-robin

    Note: In docker compose we mapped 8080(external):to:80(internal)
- **kp-wordpress-application** (*application*) containers are not exposed directly but are accessible via NGINX internally
    - *application* containers connect to *proxysql* on port 6033 to handle all MySQL queries
- **database-lb** (*proxysql*) routes queries to the respectful master/slave instance based on query type
    - reads(SELECT) → *database-slave*
    - writes (INSERT, UPDATE, etc.) → *database-master*
- **master-db/slave-db** (*database-master/database-slave*), databases behind *proxysql* utilizes master/slave setup to perform asynchronous binary log replication

If database volumes are empty, init.sql scripts are executed to:

- Set up the WordPress schema and content
- Create replication and monitoring users
- Configure master-slave replication

External Access Points

Component	External Access Point	Port	Notes	Credentials(user/pass)
kp-wordpress-application	http://localhost	8080	/ serves the webpage /wp-admin for admin GUI	admin/nimda
master-db/slave-db	localhost	3307/3308	Some database management tool MySQL Workbench or mysql cli	root/root user1/Password123!

Environment variables (.env)

```
MYSQL_ROOT_PASSWORD=root
MYSQL_ALLOW_EMPTY_PASSWORD=no
WORDPRESS_DB_HOST=proxysql:6033
WORDPRESS_DB_USER=user1
```

```
WORDPRESS_DB_PASSWORD=Password123!
WORDPRESS_DB_NAME=wordpress_data
```

Healthchecks (Docker-level)

Service	Check
WordPress	curl http://localhost
ProxySQL	mysqladmin ping on 6033
MySQL DBs	mysqladmin ping

Summary

- **nginx-lb** handles incoming HTTP traffic and load balances it to multiple WordPress app containers
- Each WordPress instance connects to ProxySQL(**database-lb**), not the DBs directly
- **database-lb** routes read/write SQL traffic to MySQL master or slave
- Percona MySQL database is set up in a master-slave replication model
- Docker Compose manages the entire stack: services, networks, volumes, and healthchecks
- The Docker network (web-app-net) facilitates internal service discovery and communication