

## Home Lab Activity: Docker Containerization with Persistent Volumes

**Student Name:** Timothy Forte

**Course:** Elective 3

### Objective

This lab demonstrates understanding of containerization concepts by:

1. Accessing a running container's shell environment
2. Deploying a multi-container application (WordPress + MySQL)
3. Implementing persistent storage using Docker volumes

### Part 1: Shell Access to Running Container

```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Timothy>docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
20043066d3d5: Pull complete
Digest: sha256:c35e29c9458151419d9448b8fd75374fec4fff364a27f176fb458d472dfc9e54
Status: Downloaded newer image for ubuntu:latest
root@19615b061ceb:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@19615b061ceb:/# pwd
/
root@19615b061ceb:/# whoami
root
root@19615b061ceb:/# exit
exit

C:\Users\Timothy>
```

### Explanation:

I successfully created and accessed an interactive Ubuntu container using the command docker run -it ubuntu bash. This command pulled the Ubuntu image from Docker Hub and launched a container with an interactive bash shell.

Inside the container, I executed basic Linux commands to explore the environment:

- ls - listed directory contents showing the root filesystem structure
- pwd - confirmed I was in the root directory (/)
- whoami - verified I was logged in as the root user

This demonstrates that containers provide isolated environments with their own filesystem, separate from the host operating system.

## Part 2: WordPress and MySQL Deployment with Persistent Volumes

```
version: "3.8"
services:
  db:
    image: mysql:8.0
    container_name: mysql_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wpuser
      MYSQL_PASSWORD: wppassword
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - wp_network
  wordpress:
    image: wordpress:latest
    container_name: wordpress_site
    restart: always
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wpuser
      WORDPRESS_DB_PASSWORD: wppassword
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wp_data:/var/www/html
    depends_on:
      - db
    networks:
      - wp_network
volumes:
  db_data:
  wp_data:
networks:
```

### Explanation:

I created a docker-compose.yml file to define a multi-container application stack. This configuration file specifies two services:

#### MySQL Database Service:

- Uses mysql:8.0 image
- Environment variables configure database credentials
- Volume db\_data:/var/lib/mysql ensures database persistence

#### WordPress Service:

- Uses wordpress:latest image
- Maps port 8080 on host to port 80 in container
- Volume wp\_data:/var/www/html preserves WordPress files
- Depends on MySQL service for proper startup order

#### Key Configuration Elements:

- **Volumes:** Named volumes (db\_data and wp\_data) store data outside containers, ensuring persistence across restarts

- **Networks:** Custom network (wp\_network) enables container communication
- **Environment Variables:** Pass configuration securely without hardcoding in images

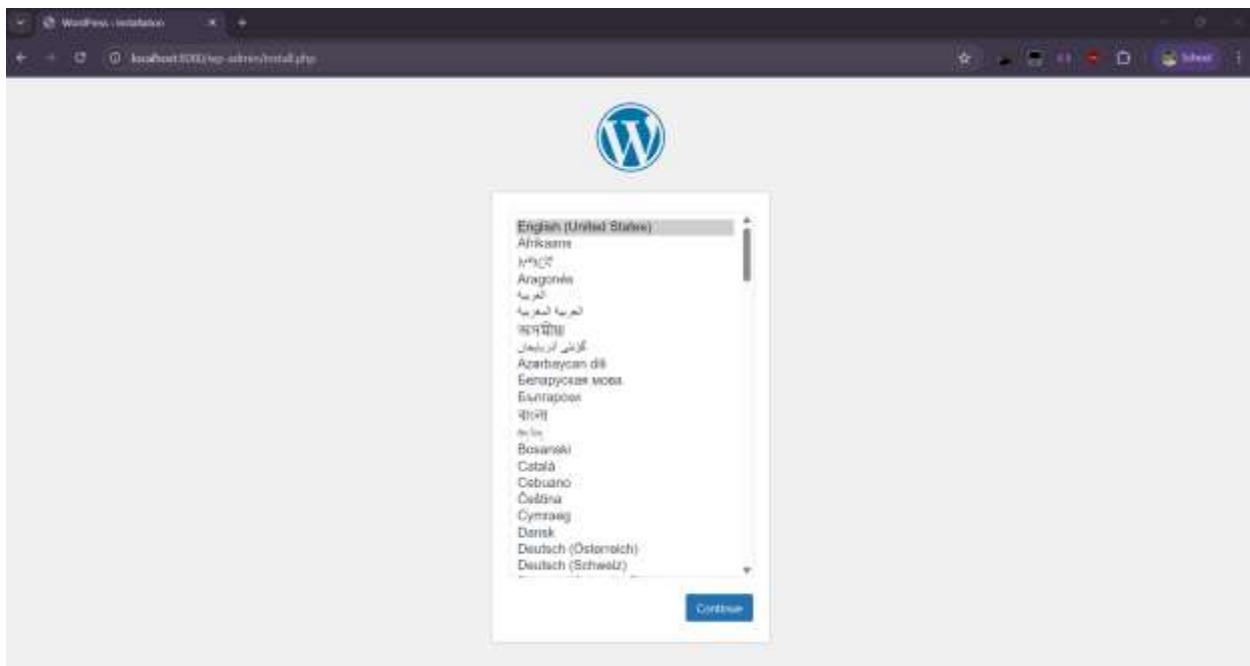
```
C:\Users\Timothy\wordpress-lab>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
063d5ac44f1f        wordpress:latest   "docker-entrypoint.s..."   19 seconds ago    Up 18 seconds      0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   wordpress_site
f1a119383488        mysql:8.0          "docker-entrypoint.s..."   28 seconds ago    Up 18 seconds      3306/tcp, 33060/tcp   mysql_db
```

### Explanation:

After running docker-compose up -d, I verified both containers were running using docker ps. The output shows:

- **mysql\_db:** MySQL container running on port 3306
- **wordpress\_site:** WordPress container with port mapping 8080→80

Both containers show "Up" status, confirming successful deployment. The -d flag runs containers in detached mode (background), allowing the terminal to remain available for other commands.

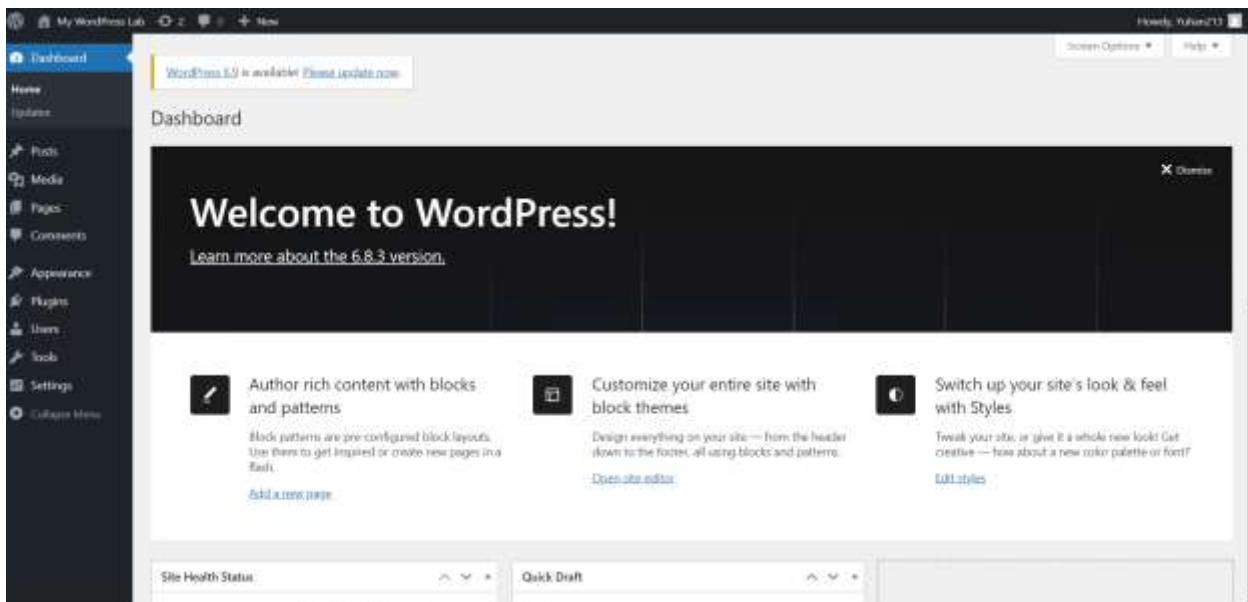


### Explanation:

Accessing `http://localhost:8080` in my browser displayed the WordPress installation page. This confirms:

1. The WordPress container is running correctly
2. Port mapping is working (host port 8080 → container port 80)
3. WordPress successfully connected to the MySQL database

The installation page allows configuration of site title, admin username, password, and email. This is the initial setup required before WordPress becomes fully functional.



### Explanation:

After completing the installation, I successfully logged into the WordPress admin dashboard. This interface provides access to:

- Posts and Pages management
- Theme and plugin installation
- Site settings and configuration
- User management

### Testing Persistent Storage:

To verify volume persistence, I performed the following test:

1. Stopped all containers: docker-compose down
2. Restarted containers: docker-compose up -d
3. Accessed <http://localhost:8080> again

The WordPress site remained intact with all configurations preserved, confirming that the persistent volumes successfully stored data outside the containers.

## **Key Concepts Demonstrated**

### **1. Containerization**

Containers package applications with their dependencies, ensuring consistent behavior across different environments. Each container runs in isolation from the host system and other containers.

### **2. Persistent Volumes**

By default, container data is ephemeral (lost when container stops). Named volumes solve this by storing data on the host filesystem, surviving container restarts and removals.

### **3. Multi-Container Orchestration**

Docker Compose simplifies managing applications requiring multiple interconnected services. It handles startup order, networking, and configuration through a single YAML file.

### **4. Container Networking**

The custom network (wp\_network) allows WordPress to communicate with MySQL using the service name (db:3306) instead of IP addresses, providing service discovery.

## **Conclusion**

This lab successfully demonstrated Docker containerization fundamentals. I deployed a production-ready WordPress application with persistent data storage, proving understanding of:

- Container lifecycle management
- Volume persistence mechanisms
- Multi-container application architecture
- Docker Compose orchestration

The persistent volumes ensured that even after stopping and restarting containers, all WordPress content and MySQL data remained intact, which is critical for real-world applications.