Docker Hands-On Lab 5 — Bind Mounts

Trainer: Sandeep Kumar Sharma

Duration: 45–60 minutes

Goal: In this lab, you will understand the concept of **Bind Mounts** in Docker, why they are used, how they differ from volumes, and how to create and test a Bind Mount using a container named **sandeep-container**.

---

1) Understanding Bind Mounts

Bind mounts allow you to **map a specific directory from your host machine** into a directory inside a Docker container. Unlike Docker-managed volumes, bind mounts give you **direct control** over the data location on the host.

In simple terms:

> Bind mounts connect your host folder directly to a folder inside the container — any change on one side immediately reflects on the other.

Analogy: If a volume is like an external USB drive managed by Docker, a bind mount is like directly plugging your system's hard drive folder into the container.

---

2) Why Do We Use Bind Mounts?

- To share configuration files or source code between the host and container.
- To view or modify container-generated files directly from the host.
- To enable **real-time file sync** during development.
- To store logs or outputs in a host directory for easy access.

Example use cases:

- Mounting your application source code for development (`/home/user/app:/var/www/html`)
- Mounting logs for monitoring or debugging

---

3) Difference Between Volumes and Bind Mounts

Feature	Docker Volume	Bind Mount
Managed By	Docker	User/Host
Storage Path	`/var/lib/docker/volumes`	Any path on the host
Use Case	Data persistence & sharing	Development & direct access to host data
Backup	Easy (via Docker commands)	Manual
Flexibility	Limited to Docker path	Any host directory

---

4) Lab Exercise — Creating and Using a Bind Mount

Step 1: Create a Host Directory

On your host machine, create a directory that will be mounted inside the container.

```
mkdir -p ~/docker_data
cd ~/docker_data
echo "Welcome to the Bind Mount Lab!" > host_file.txt
```

This directory (`~/docker_data`) will be mounted into the container.

Step 2: Run a Container with a Bind Mount

Now we'll run an **Ubuntu** container named **sandeep-container** and mount the host directory.

```
docker run -it --name sandeep-container
  -v ~/docker_data:/data
  ubuntu:latest /bin/bash
```

Explanation:

- `-v ~/docker_data:/data` → Mounts the host directory into `/data` inside the container.
- Any file created in `/data` will appear instantly on your host, and vice versa.

Step 3: Verify the Mount Inside the Container

Inside the container:

```
cd /data
ls
cat host_file.txt
```

You should see `host_file.txt` created on the host.

Step 4: Create Files Inside the Container

Now create a new file from within the container:

```
echo "This file was created inside the container." > container_file.txt
ls -l
```

Exit the container:

```
exit
```

Step 5: Verify from Host

Back on your host system, check if the new file exists:

```
ls ~/docker_data
cat ~/docker_data/container_file.txt
```

You'll see the file created inside the container — this demonstrates the **two-way sync** of bind mounts.

Step 6: Start the Container Again

If the container is stopped, start it again:

```
docker start -ai sandeep-container
```

Inside the container:

```
cd /data
ls
```

You'll still see both `host_file.txt` and `container_file.txt`.

Step 7: Remove the Container but Keep Data

```
docker stop sandeep-container
docker rm sandeep-container
```

Your data remains safe on the host inside `~/docker_data`.

---

5) Practice Tasks

1. Create another bind mount at `/myapp` and link it to `~/app_data` on your host.
2. Write a file in the host folder and verify its visibility inside the container.
3. Delete the container and confirm that the data on the host remains.

---

6) Cleanup (Optional)

```
rm -rf ~/docker_data
```

---

7) Summary

- Bind mounts directly link host folders to container directories.
- Any change inside the container reflects instantly on the host.
- Ideal for development, configuration, and log sharing.
- Data persists outside the container lifecycle.
- In this lab, we used the container **sandeep-container** consistently.

✅ **Checkpoint:** You've successfully created and verified a bind mount between your host system and container **sandeep-container**!

Next, we can explore **Named Volumes** to manage Docker data with better portability and isolation.