

Docker Hands-On Lab 8 — Docker Compose from Basic to Advanced

Trainer: Sandeep Kumar Sharma

Duration: 90–120 minutes

Goal: In this lab, you'll learn **Docker Compose** from the ground up — what it is, why we use it, how to create and configure `docker-compose.yml` files, and how to manage multi-container applications using Compose. We'll build from a simple single-service app to an advanced multi-container setup with custom networks and volumes.

1) What is Docker Compose?

Docker Compose is a tool that allows you to **define and run multi-container Docker applications** using a single YAML configuration file — `docker-compose.yml`.

Instead of running multiple `docker run` commands manually, Compose lets you define your containers, networks, and volumes in one place and bring them up with a single command.

Simple analogy:

Think of Docker Compose as an orchestra conductor — it coordinates how different containers (services) run and interact.

2) Why Use Docker Compose?

- Manage multi-container applications easily.
 - Simplify networking between containers.
 - Use declarative configuration (`docker-compose.yml`).
 - Bring up or down the entire stack with one command.
 - Useful for **development**, **testing**, and **CI/CD** environments.
-

3) Install Docker Compose

If you have Docker Desktop (Windows/Mac), Compose is already included.

For Linux:

```
sudo apt update
sudo apt install docker-compose-plugin -y
```

Verify the installation:

```
docker compose version
```

4) Basic Example — Single Container Using Docker Compose

Step 1: Create a Project Directory

```
mkdir ~/compose-lab-basic
cd ~/compose-lab-basic
```

Step 2: Create `docker-compose.yml`

```
version: '3.8'
services:
  sandeep-app:
    image: nginx:latest
    container_name: sandeep-app
    ports:
      - "8080:80"
```

Step 3: Run the Application

```
docker compose up -d
```

Check the running container:

```
docker ps
```

Visit `http://localhost:8080` — you should see the Nginx default page.

Step 4: Stop the Application

```
docker compose down
```

5) Intermediate Example — Multi-Container Web + Database Stack

Now we'll create a **WordPress + MySQL** stack using Compose.

Step 1: Create a New Directory

```
mkdir ~/compose-lab-wordpress
cd ~/compose-lab-wordpress
```

Step 2: Create `docker-compose.yml`

```
version: '3.8'
services:
  db:
    image: mysql:5.7
    container_name: sandeep-db
    restart: always
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wp_user
      MYSQL_PASSWORD: wp_pass
      MYSQL_ROOT_PASSWORD: rootpass
    volumes:
      - db_data:/var/lib/mysql

  wordpress:
    image: wordpress:latest
    container_name: sandeep-wordpress
    depends_on:
      - db
    ports:
      - "8081:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wp_user
      WORDPRESS_DB_PASSWORD: wp_pass
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wp_data:/var/www/html

volumes:
  db_data:
  wp_data:
```

Step 3: Start the Application

```
docker compose up -d
```

Step 4: Verify

```
docker ps
```

Open your browser and go to `http://localhost:8081` to see the WordPress setup page.

Step 5: Stop and Remove

```
docker compose down
```

6) Advanced Example — Custom Network and Multiple Services

Let's create a more advanced setup with custom networking and named volumes.

Step 1: Create a Directory

```
mkdir ~/compose-lab-advanced  
cd ~/compose-lab-advanced
```

Step 2: Create `docker-compose.yml`

```
version: '3.8'  
  
networks:  
  sandeep-network:  
    driver: bridge  
    ipam:  
      config:  
        - subnet: 172.25.0.0/24  
  
volumes:  
  app_data:  
  db_data:  
  
services:  
  app1:
```

```

image: nginx:latest
container_name: sandeep-app1
ports:
  - "8082:80"
networks:
  - sandeep-network
volumes:
  - app_data:/usr/share/nginx/html

app2:
image: httpd:latest
container_name: sandeep-app2
ports:
  - "8083:80"
networks:
  - sandeep-network
volumes:
  - app_data:/usr/local/apache2/htdocs

database:
image: mysql:8.0
container_name: sandeep-db
restart: always
environment:
  MYSQL_ROOT_PASSWORD: rootpass
  MYSQL_DATABASE: demo_db
  MYSQL_USER: demo_user
  MYSQL_PASSWORD: demo_pass
volumes:
  - db_data:/var/lib/mysql
networks:
  - sandeep-network

```

Step 3: Deploy the Stack

```
docker compose up -d
```

Step 4: Verify Network and Containers

```

docker network ls
docker ps
docker network inspect compose-lab-advanced_sandeep-network

```

You'll see all containers (`sandeep-app1` , `sandeep-app2` , and `sandeep-db`) connected to the same custom network.

7) Scaling Services

Compose allows scaling stateless services easily. Example:

```
docker compose up -d --scale app1=3
```

This will spin up 3 instances of `app1` sharing the same network and configuration.

Check them:

```
docker ps
```

8) Viewing Logs and Managing Containers

View real-time logs for all services:

```
docker compose logs -f
```

Restart specific service:

```
docker compose restart app2
```

Stop and remove everything:

```
docker compose down -v
```

9) Practice Tasks

1. Modify the advanced Compose file to add a Redis service and link it to `app1`.
 2. Create a health check for `app1` using the `healthcheck` directive.
 3. Try scaling `app2` and observe how ports are handled.
 4. Connect multiple Compose projects to the same custom Docker network.
-

10) Summary

- **Docker Compose** simplifies running multi-container applications.
- Everything is defined declaratively in `docker-compose.yml`.
- You can easily manage services, volumes, and networks.
- Scaling and orchestration become simple with `docker compose` commands.
- In this lab, we used containers like **sandeep-app1**, **sandeep-app2**, and **sandeep-db** connected via a custom **sandeep-network**.

✅ **Checkpoint:** You have mastered Docker Compose from basics to advanced — including networks, scaling, and multi-service orchestration!