🐳 Docker Labs – Step by Step (Beginner to Real-Time)

Lab 1: Install Docker on Linux (Ubuntu Example)

Prerequisites: Ubuntu 20.04/22.04 VM, user with sudo privileges.

Steps:

```
# Update system
sudo apt update -y && sudo apt upgrade -y

# Install prerequisites
sudo apt install -y ca-certificates curl gnupg lsb-release

# Add Docker official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker.gpg

# Add Docker repository
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list

# Install Docker
sudo apt update -y
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

```
# Check version
docker --version

# Run test image
sudo docker run hello-world
```

Lab 2: Run First Container (Nginx Web Server)

Steps:

```
# Pull nginx image
docker pull nginx

# Run container (map port 8080 → 80 inside container)
docker run -d --name web1 -p 8080:80 nginx

# Verify container running
docker ps

# Open in browser
http://localhost:8080
```

Lab 3: Container Management (Ubuntu + Custom Image)

Steps:

```
# Run Ubuntu interactively
docker run -it --name myubuntu ubuntu bash

# Inside container → install curl
```

```
apt update && apt install -y curl

# Exit
exit

# Commit container to new image
docker commit myubuntu ubuntu-with-curl

# Run new image
docker run -it ubuntu-with-curl curl --version
```

Lab 4: Custom Dockerfile (Python Flask App)

Files: `app.py`

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Dockerfile:

```
FROM python:3.9
WORKDIR /app
COPY app.py /app
RUN pip install flask
```

```
CMD ["python", "app.py"]
```

Steps:

```
# Build image
docker build -t flask-app .

# Run container
docker run -d -p 5000:5000 flask-app

# Test
curl http://localhost:5000
```

Lab 5: Volumes (MySQL Persistence)

Steps:

```
# Create volume
docker volume create mysql_data

# Run MySQL with volume
docker run -d --name mydb -e MYSQL_ROOT_PASSWORD=admin -v
mysql_data:/var/lib/mysql mysql:5.7

# Stop container
docker stop mydb && docker rm mydb

# Run again → data should persist
docker run -d --name mydb2 -e MYSQL_ROOT_PASSWORD=admin -v
mysql_data:/var/lib/mysql mysql:5.7
```

Lab 6: Docker Networking (Bridge Network)

Steps:

```
# Create network
docker network create mynet

# Run nginx in network
docker run -d --name web --network mynet nginx

# Run busybox in network
docker run -it --name test --network mynet busybox sh

# Inside busybox container → test connectivity
ping web
```

Lab 7: Environment Variables (MySQL Example)

Steps:

```
docker run -d --name mydb -e MYSQL_ROOT_PASSWORD=root -e
MYSQL_DATABASE=school mysql:5.7
```

Lab 8: Port Mapping (Apache HTTPD)

Steps:

```
docker run -d --name apache -p 8081:80 httpd
```

Open → http://localhost:8081

Lab 9: Multi-Container App (WordPress + MySQL)

Steps:

```
# Run MySQL
docker run -d --name wpdb -e MYSQL_ROOT_PASSWORD=root -e
MYSQL_DATABASE=wordpress mysql:5.7

# Run WordPress
docker run -d --name mywp -p 8082:80 --link wpdb:mysql wordpress
```

Open → http://localhost:8082

Lab 10: Docker Compose (WordPress + MySQL)

File: docker-compose.yml

```
version: '3.1'
services:
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: wordpress
  wordpress:
    image: wordpress
```

```
    ports:
      - "8083:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: root
      WORDPRESS_DB_PASSWORD: root
```

Steps:

```
docker-compose up -d
```

Open → `http://localhost:8083`

Lab 11: Build & Push to Docker Hub

Steps:

```
# Login
docker login

# Build image
docker build -t myuser/nodeapp .

# Tag
docker tag myuser/nodeapp mydockerhubid/nodeapp:v1

# Push
docker push mydockerhubid/nodeapp:v1
```

Lab 12: Logs & Debugging

```
# Run container that exits quickly
docker run --name test alpine echo "Done"

# Logs
docker logs test

# Restart policy
docker run -d --name web --restart always -p 8084:80 nginx
```

Lab 13: Dockerize Existing Java/Python App

```
# Example Java Dockerfile
FROM openjdk:11
WORKDIR /app
COPY app.jar /app
CMD ["java", "-jar", "app.jar"]
```

Lab 14: Scaling with Docker Compose

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

Lab 15: Monitoring with Portainer

```
docker volume create portainer_data
docker run -d -p 9000:9000 -p 8000:8000 --name portainer \
    --restart always -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data portainer/portainer-ce
```

Open → `http://localhost:9000`
