

# Docker Hands-On Lab 9 — Docker Images

**Trainer:** Sandeep Kumar Sharma

**Duration:** 60–90 minutes

**Goal:** In this lab, you'll learn what Docker Images are, why they are needed, how they work internally, and how to create your own images in different ways — from basic to advanced. By the end, you'll understand the full lifecycle of an image: creation, customization, tagging, and pushing to a registry.

---

## 1) What is a Docker Image?

A **Docker Image** is a lightweight, standalone, and immutable package that contains everything needed to run a piece of software — including the code, runtime, libraries, environment variables, and configurations.

**In simple terms:**

A Docker image is a blueprint for creating containers.

Every time you run `docker run`, Docker uses an image to create a **container instance**.

---

## 2) Why Do We Need Docker Images?

- **Consistency:** Same environment across development, testing, and production.
- **Portability:** Run the same image on any system with Docker installed.
- **Reusability:** One image can be reused to spawn multiple containers.
- **Version Control:** Images can be tagged (v1, v2, latest) to track versions.
- **Automation:** Base images (like Ubuntu or Nginx) can be extended with custom configurations.

**Example:** If you install Nginx manually on multiple servers, each may differ slightly. But with an image (`nginx:latest`), every container behaves identically.

---

## 3) Types of Docker Images

Docker images can be created in multiple ways:

1. **Base Images** — Created from scratch (no parent). Example: `scratch`, `alpine`.
2. **Official Images** — Maintained by Docker or the vendor. Example: `ubuntu`, `nginx`, `mysql`.
3. **Custom Images** — Built by users for specific applications (using Dockerfiles).
4. **Intermediate Images** — Temporary layers created during a Dockerfile build process.

---

## 4) Viewing Existing Images

List available images:

```
docker images
```

Pull an image from Docker Hub:

```
docker pull ubuntu:latest
```

Inspect an image:

```
docker inspect ubuntu:latest
```

Remove an image:

```
docker rmi <image_id>
```

---

## 5) Creating Docker Images

You can create Docker images in three ways:

### Method 1: Commit a Running Container

1. Start a container from a base image:

```
docker run -it --name sandeep-container ubuntu:latest /bin/bash
```

1. Inside the container, install Apache:

```
apt update && apt install -y apache2
```

1. Exit the container:

```
exit
```

1. Commit the container as a new image:

```
docker commit sandeep-container sandeep/apache:v1
```

1. Verify the new image:

```
docker images
```

Now you've created an image manually using container changes.

---

## Method 2: Create an Image Using a Dockerfile (Recommended)

Dockerfile is a text file containing instructions to build an image.

### Step 1: Create a Directory

```
mkdir ~/docker-image-lab  
cd ~/docker-image-lab
```

### Step 2: Create a File Named ``

```
# Base Image  
FROM ubuntu:latest  
  
# Maintainer Info  
LABEL maintainer="Sandeep Kumar Sharma <trainer@dockerlab.com>"  
  
# Install Apache  
RUN apt update && apt install -y apache2  
  
# Expose Port 80  
EXPOSE 80  
  
# Start Apache Service  
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

### Step 3: Build the Image

```
docker build -t sandeep/apache:v2 .
```

### Step 4: Run a Container from This Image

```
docker run -d --name sandeep-app -p 8080:80 sandeep/apache:v2
```

Visit <http://localhost:8080> — you should see the Apache default page.

---

### Method 3: Create an Image Using Docker Compose (Advanced)

Sometimes, images can be built automatically as part of a Compose setup.

**docker-compose.yml example:**

```
version: '3.8'
services:
  sandeep-web:
    build: .
    ports:
      - "8081:80"
```

Run it:

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

Docker Compose reads the Dockerfile and builds the image before running the container.

---

## 6) Tagging and Pushing Images to Docker Hub

### Step 1: Tag Your Image

```
docker tag sandeep/apache:v2 sandeepkumarsharma/apache:latest
```

### Step 2: Login to Docker Hub

```
docker login
```

### Step 3: Push the Image

```
docker push sandeepkumarsharma/apache:latest
```

## Step 4: Verify Online

Visit your Docker Hub repository to confirm the upload.

---

## 7) Layers and Caching in Docker Images

Docker images are made up of **layers** — each instruction in the Dockerfile creates a new layer.

Check image history:

```
docker history sandeep/apache:v2
```

Layers are cached, so rebuilding an unchanged Dockerfile is faster.

---

## 8) Cleanup

```
docker stop sandeep-app
docker rm sandeep-app

# Remove images
docker rmi sandeep/apache:v1 sandeep/apache:v2
```

---

## 9) Practice Tasks

1. Create a new Dockerfile using the `python:3.10` image and copy your Python script into it.
  2. Build and run the container to execute the Python app.
  3. Tag and push it to Docker Hub.
  4. Pull your image on another system and verify it runs correctly.
- 

## 10) Summary

- Docker images are blueprints for containers.
- They ensure consistency, portability, and easy deployment.
- You can create images manually, with Dockerfiles, or using Docker Compose.
- Images consist of multiple cached layers.
- You can share your images publicly via Docker Hub.

 **Checkpoint:** You have successfully learned to create, build, tag, and push Docker images using both manual and automated methods!