Docker Complete Guide (Install + Images + Containers + Usage)

A practical PDF for beginners (Windows-focused) — with real commands and examples.

1) What is Docker?

Docker is a tool that packages your application + dependencies into a single unit called an **image**. That image can be run anywhere as a **container**. This solves the problem of: “It works on my machine”.

Key Concepts

- **Image**: Blueprint / snapshot of your app (read-only template).
- **Container**: Running instance of an image (actual running app).
- **Dockerfile**: Recipe to build an image.
- **Docker Hub**: Online registry where images are stored and shared.
- **Volume**: Persistent storage for container data.
- **Bind Mount**: Link your local folder into the container (best for dev).

2) Install Docker Desktop (Windows)

Docker Desktop is the easiest way to install Docker on Windows. It includes Docker Engine, Docker CLI, and GUI.

Requirements

- Windows 10/11 (64-bit)
- Virtualization enabled in BIOS
- WSL2 enabled (recommended)

Steps

1. Download Docker Desktop installer from the official Docker website.
2. Run installer → enable WSL2 backend when asked.
3. Restart your PC if required.
4. Open Docker Desktop and wait until it shows “Docker is running”.

Verify Installation (Commands)

```
docker --version
docker info
docker run hello-world
```

3) Docker Images vs Containers (Most Important)

Image is like a “class/template”. **Container** is like an “object/running process”. You build an image, then run containers from it.

Common Commands

```
# List images
docker images

# List running containers
docker ps

# List all containers (including stopped)
docker ps -a
```

4) Create a Simple Node/Express App Dockerfile

A Dockerfile tells Docker how to build your application image.

Example Dockerfile (Node + Express)

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 5000

CMD ["npm", "start"]
```

5) Build an Image

Build your Docker image from the Dockerfile:

```
docker build -t express-app:latest .
```

Here, **-t** means tag (name:version). **latest** is a common default tag.

6) Run a Container

Run your app inside a container and map ports:

```
docker run -p 5000:5000 --name express-container express-app:latest
```

Port mapping format is **HOST:CONTAINER**. Example: open http://localhost:5000 in your browser.

Run in Background (Detached Mode)

```
docker run -d -p 5000:5000 --name express-container express-app:latest
```

7) Manage Containers (Start/Stop/Logs/Shell)

```
# Stop a container
docker stop express-container
# Start a container
```

```
docker start express-container

# Restart
docker restart express-container

# Remove container
docker rm -f express-container

# View logs
docker logs express-container

# Follow logs (live)
docker logs -f express-container

# Open shell inside container
docker exec -it express-container sh
```

8) Tagging & Versioning (Professional Practice)

Tagging helps you keep versions of your images. Example tags: v1, v1.1, v2, prod, dev.

```
docker tag express-app:latest tanveer2246/express-app:v1
docker tag express-app:latest tanveer2246/express-app:latest
```

9) Push Image to Docker Hub

Steps:

1. Login to Docker Hub from terminal:
2. Tag your image with your Docker Hub username/repo.
3. Push it to Docker Hub.

```
docker login
docker push tanveer2246/express-app:v1
```

10) Pull & Run on Another Laptop

If you push your image to Docker Hub, you can run it on any machine with Docker installed.

```
docker pull tanveer2246/express-app:v1
docker run -d -p 5000:5000 --name express-app tanveer2246/express-app:v1
```

11) Volumes & Bind Mounts (Persistent Data + Dev Workflow)

Containers are temporary by default. When a container is removed, its internal data is lost. Volumes and bind mounts solve this.

A) Volumes (Best for Databases)

```
# Create volume
docker volume create appdata

# Use volume in container
docker run -d -p 5000:5000 \
  -v appdata:/app/data \
  --name express-container express-app:latest
```

B) Bind Mounts (Best for Development)

```
docker run -d -p 5000:5000 \
  -v %cd%:/app \
  --name express-dev express-app:latest
```

Bind mount links your local code into the container so changes reflect instantly (great for dev).

12) .dockerignore (Faster Builds)

A .dockerignore file prevents copying unnecessary files into the image (like node_modules).

```
node_modules
.git
.env
npm-debug.log
Dockerfile
.dockerignore
```

13) Docker Compose (Run Multiple Services)

Docker Compose helps run multiple containers together (example: backend + database).

Example docker-compose.yml (Express + Mongo)

```
version: '3.8'
services:
  api:
    build: .
    ports:
      - "5000:5000"
    environment:
      - PORT=5000
    depends_on:
      - mongo

  mongo:
    image: mongo:6
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
```

Run:

```
docker compose up -d
docker compose logs -f
docker compose down
```

14) Useful Cleanup Commands

```
# Remove unused containers, networks, images
docker system prune

# Remove all unused images
docker image prune -a

# Remove volumes not used
```

```
docker volume prune
```

15) Final Notes (Best Practices)

- Always use .dockerignore to keep builds clean and fast.
- Use version tags (v1, v2) instead of only latest.
- Use volumes for databases and persistent data.
- Use bind mounts for development workflow.
- Never push .env secrets inside images or GitHub.