# 🐳 Simple & Clear Docker Guide

## What is Docker?

Docker is a tool that helps developers package applications into containers. A container bundles everything the app needs (code, libraries, settings) into a portable box. This makes sure your app works the same everywhere—on your machine, someone else's computer, or a server.

## What is a Container?

A container is like a lightweight virtual machine. It's an isolated environment that runs your application. It includes the application code, runtime, system tools, libraries—everything needed to run your app.

## What is an Image?

An image is a snapshot or template of a container. It's the instructions or recipe for creating containers. Once built, an image can be used to create one or many containers.

## What is a Dockerfile?

A Dockerfile is a text file with step-by-step instructions on how to build a Docker image. It tells Docker what base image to use, what files to copy, what commands to run, and what command to use to start the app.

Example Dockerfile:

FROM python:3.10-slim  
WORKDIR /app # Create a folder inside the container called /app and switch into it  
COPY . . # Copy everything from your current folder (on your computer) into the /app folder in the container  
RUN pip install uv # install the Python packages your app needs

# Sync dependencies from pyproject.toml and uv.lock

RUN uv sync

CMD ["python", "main.py"] # Tell Docker what command to run when the container starts (runs your app)

**The programming language (or framework) you use determines the base image you should choose.**

Line by line explanation:

- FROM: Use a base image with Python 3.10.  
- WORKDIR: Set working directory inside the container.  
- COPY: Copy all files from your current folder into the container.  
- RUN: Install Python dependencies.  
- CMD: Define the command to run when the container starts.

## .dockerignore File

A screenshot of a computer program

AI-generated content may be incorrect.

This file tells Docker which files to ignore when building the image (like .gitignore).

\_\_pycache\_\_/

\*.pyc

\*.log

.env

.git/

.vscode/

## Common Docker Commands

* Check Docker version: docker --version
* Build an image: docker build -t myapp .
* Run a container: docker run myapp
* Run with ports: docker run -p 5000:5000 myapp
* List running containers: docker ps
* List all containers: docker ps -a
* Stop a container: docker stop <container\_id>
* Remove a container: docker rm <container\_id>
* Remove an image: docker rmi myapp
* Enter a container: docker exec -it <container\_id> /bin/bash

## Example: Python App in Docker

Folder structure:

myapp/  
├── app.py  
├── requirements.txt  
└── Dockerfile

app.py

print("Hello from Docker!")

requirements.txt (can be empty or have packages)

Build and run:

cd myapp  
docker build -t my-python-app .  
docker run my-python-app

Output: Hello from Docker!

## Optional but Useful Commands

| **Command** | **What it does** | **Example** |
| --- | --- | --- |
| EXPOSE | Documents the port your app uses (optional, not mandatory) | EXPOSE 8000 |
| ENV | Sets environment variables | ENV PYTHONUNBUFFERED=1 |
| ENTRYPOINT | Like CMD, but more strict/overridable | ENTRYPOINT ["python"] |
| .dockerignore | Skips files/folders when building (like .gitignore) | .dockerignore file |

## Docker Hub

Docker Hub is a cloud registry where you can find and share Docker images. You can pull official images like Python, or push your own.

docker pull python  
docker login  
docker tag myapp username/myapp  
docker push username/myapp

## Summary

- Docker: tool to containerize apps.  
- Image: recipe to build a container.  
- Container: running instance of an image.  
- Dockerfile: instructions to build an image.  
- Docker Hub: image sharing service.  
- Commands: build, run, stop, exec, ps, etc.

## How to Build and Run a Docker Image & Container

**Step 1: Build the Docker Image**

You create an image by running docker build in the directory where your Dockerfile lives.

docker build -t myapp:latest .

* docker build — builds an image.
* -t myapp:latest — tags the image with name myapp and tag latest.
* . — current directory (where Dockerfile and code are).

**Step 2: List Your Docker Images (optional)**

**docker images**

**Step 3: Run a Container from Your Image**

**docker run --name myapp-container -d -p 8000:8000 myapp:latest**

**Explanation:**

* **docker run — run a container.**
* **--name myapp-container — give the container a name.**
* **-d — run container in detached mode (in the background).**
* **-p 8000:8000 — map port 8000 of the container to port 8000 on your computer.**
* **myapp:latest — the image to use.**

**If your app doesn’t use ports, you can omit -p.**

**Step 4: See Running Containers (optional)**

**docker ps**

**Step 5: See Logs (optional)**

**docker logs myapp-container**

**Step 6: Stop a Running Container**

**docker stop myapp-container**

**Step 7: Remove a Container (after stopping it)**

**docker rm myapp-container**

**Step 8: Remove an Image (optional)**

**docker rmi myapp:latest**

### Summary of Key Commands

| **Purpose** | **Command Example** |
| --- | --- |
| **Build image** | **docker build -t myapp:latest .** |
| **List images** | **docker images** |
| **Run container** | **docker run --name myapp-container -d -p 8000:8000 myapp:latest** |
| **List running containers** | **docker ps** |
| **View container logs** | **docker logs myapp-container** |
| **Stop container** | **docker stop myapp-container** |
| **Remove container** | **docker rm myapp-container** |
| **Remove image** | **docker rmi myapp:latest** |