What Docker Is and Why "It Runs in a Container" Matters

The thing everyone says and nobody explains

"It works on my machine" is one of the most expensive sentences in software. You build something, it runs perfectly, you move it to a server, it breaks. Different Python version. Missing library. Wrong system setting. Two days gone.

Docker exists to kill that sentence. That is the entire pitch.

You will hit it whether or not you ever write code, because self-hosted software now ships as a Docker image and nothing else. If you want a private transcription server, your own e-signature platform, or an analytics dashboard that does not phone home, the vendor's install instructions are going to be three lines of Docker. This tip is the concepts. The step-by-step install is in *Installing Docker: Windows, Mac, and Linux*.

What a container is

A container is your app plus everything it needs to run, sealed in a box: the runtime, the libraries, the config, the file layout. It runs the same on your laptop, on your server, and on someone else's machine, because it does not care what is installed on the host.

> Analogy: a shipping container holds cargo. The ship does not care what is inside. The port does not care what ship it came off. Docker works the same way; the app is packaged, isolated, and portable.

The practical consequence is what matters to you. When a project says "run it with Docker," you are not installing that project's dependencies onto your machine. Nothing is scattered across your system. You are handed a sealed box, you run it, and when you are done you delete the box and your machine is exactly as it was.

Image vs. container, because the words get used sloppily

An **image** is the blueprint. It is a read-only, downloadable template: "PostgreSQL 16, configured this way."

A **container** is a running copy of that image. One image, as many containers as you want.

Educational only. Not legal advice. No attorney-client relationship. ©2026 Chariton Media.

Think template versus document. A .dotx is the template; every letter you spawn from it is a separate document. `docker pull` gets the image; `docker run` starts a container from it. When someone says "just delete the container and start a new one," this is why that is a reasonable thing to say and not a panic move.

Container vs. virtual machine

	Virtual Machine	Docker Container
Boots	Full OS; minutes	Your app only; seconds
Size	Gigabytes	Megabytes to hundreds of MB
Isolation	Complete separate OS	Shares the host OS kernel
Use case	Run a different OS entirely	Ship an app consistently
Overhead	Heavy	Light

The kernel-sharing line is the one worth remembering. A VM carries a whole operating system on its back; a container borrows the host's. That is why you can run twelve containers on a machine that would choke on three VMs.

Where your data actually lives

This is the part that bites people, so learn it before you lose something.

A container's filesystem is disposable. Delete the container and everything written inside it is gone. That is the design, not a bug. Anything you need to keep gets stored in a **volume** or a **bind mount**, which is just a folder on the host machine mapped into the container.

In a compose file it looks like this:

```
volumes:
  - ./data:/var/lib/postgresql/data
```

Read it right to left: the container writes to `/var/lib/postgresql/data`, and that is really the `data` folder sitting next to your compose file on the host. Destroy the container, rebuild it, and your database is still there.

If you are running anything that touches client files, this is also your backup story. Back up the volume folders. The containers themselves are worth nothing; they rebuild from the image in seconds.

Educational only. Not legal advice. No attorney-client relationship. ©2026 Chariton Media.

The commands you will actually use

```
docker ps                          # list running containers
docker ps -a                       # list all containers, including stopped
docker logs container-name         # see output and errors from a container
docker logs -f container-name      # follow the log live
docker exec -it container-name bash  # open a shell inside a container
docker stop container-name         # stop a running container
docker restart container-name      # restart it
docker compose up -d               # start everything in docker-compose.yml
docker compose down                # stop and remove it
```

That is close to the whole working set. The `-d` means detached; it runs in the background instead of holding your terminal hostage.

The compose file

Real setups do not use long `docker run` commands with fifteen flags. They use Docker Compose: one file that defines every container, its ports, its volumes, and how the pieces talk to each other. Write it once, then `docker compose up -d`.

```
services:
  myapp:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html
    restart: unless-stopped
```

The ports line reads host:container. Port 8080 on your machine goes to port 80 inside the container, so you visit `http://127.0.0.1:8080`. Change the left number if something else already owns that port; leave the right one alone, because that is the app's business.

`restart: unless-stopped` brings the container back after a reboot. Add it to anything you actually depend on.

When something breaks

The first command is always `docker logs container-name`. Nine times out of ten the answer is sitting right there: a bad password, a missing environment variable, a port already in use.

If the logs are not enough, `docker exec -it container-name bash` drops you inside the running container and you can poke around exactly like a normal server.

Educational only. Not legal advice. No attorney-client relationship. ©2026 Chariton Media.

If it is still broken, `docker compose down && docker compose up -d` is safe as long as your data is in volumes. Rebuilding from scratch is a normal move here, not a last resort. That is the point of the whole system.

A word on trust

Pulling an image means running someone else's software on your machine with whatever access you gave it. Prefer official images and named publishers over a random account with eleven downloads, pin a version tag instead of `latest` on anything important, and read the compose file before you run it. If you would not run a stranger's installer on the firm's server, do not run their container either.

Using AI to help with this

> **You do not have to fully understand this to use it.** These prompts work:

"I need to run [app or service] in Docker. Write me a docker-compose.yml that sets it up with [specific requirements], and tell me which ports it opens."

"My Docker container is crashing. Here are the logs: [paste]. What is wrong and how do I fix it?"

"Explain what this docker-compose.yml is doing, and flag anything in it that would expose data or a port I did not intend: [paste]."

Educational only. Not legal advice. No attorney-client relationship. ©2026 Chariton Media.