Docker CLI Fundamentals

docker pull, docker images, docker run, docker ps, docker logs, docker exec, docker stop /start, and docker rm -- walked through via the full lifecycle of a PostgreSQL container, with real examples.

Beginner 25 min

Last updated: August 31, 2026
Read online: https://www.learnforgex.com/en/topics/docker-cli-fundamentals

---

"What Is Docker?" built the vocabulary — image, container, Docker Engine, Docker Hub. This lesson puts that vocabulary to work at the command line: pulling an image, running it as a container, inspecting what's running, looking inside it, and cleaning up afterward. Every command here talks to the Docker Engine daemon described in "The Docker Engine" — the `docker` binary is just the client typing the requests.

Pulling an Image: `docker pull`

`docker pull` fetches an image from a registry (Docker Hub by default, see "Docker Hub and Image Registries") and stores it locally, without starting anything yet.

```
docker pull postgres:16
```

```
16: Pulling from library/postgres
a2318d6c47ec: Pull complete
c9c5e91e622b: Pull complete
...
Status: Downloaded newer image for postgres:16
docker.io/library/postgres:16
```

The `postgres` part is the image's **repository name**; the `16` after the colon is its **tag** — a label a repository can attach to multiple versions of an image. Omitting a tag entirely (`docker pull postgres`) implicitly means `:latest`, which is worth naming explicitly instead of relying on — "Best Practices" comes back to exactly why.

Listing Local Images: `docker images`

Once an image has been pulled (or built — covered in "Docker Images and Dockerfiles"), it sits in local storage until removed. `docker images` lists everything currently there:

```
docker images
```

```
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
postgres     16        a1b2c3d4e5f6   2 weeks ago   434MB
```

This is a purely local, offline listing — it does not reach out to Docker Hub. An image appearing here is exactly what "Images vs. Containers" described: a frozen template, not yet running anything.

Running a Container: `docker run`

`docker run` creates and starts a container from an image in one step — pulling the image first automatically if it isn't already local.

```
docker run --name learning-platform-db \
  -e POSTGRES_PASSWORD=secret \
  -p 5432:5432 \
  -d postgres:16
```

```
7f8e9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f
```

Four flags worth naming individually, since they show up constantly:

```
--name <name>     a human-readable name for the container, instead of a random one
-e KEY=VALUE       sets an environment variable inside the container (POSTGRES_PASSWORD here
                    is how the official postgres image expects its admin password to be set)
-p HOST:CONTAINER  publishes a port -- covered fully in "Docker Networking" later in this
course
-d                 "detached" -- runs in the background and returns the prompt immediately
```

The long string of characters printed is the new container's ID — `docker run` without `-d` would instead attach your terminal directly to the container's output, which is useful for a quick foreground check but blocks the terminal until the container stops.

Listing Running Containers: `docker ps`

`docker ps` lists containers, by default only the ones currently running:

```
docker ps
```

```
CONTAINER ID   IMAGE         COMMAND                  STATUS
PORTS                     NAMES
7f8e9a0b1c2d   postgres:16   "docker-entrypoint.s…"   Up 2 minutes   0.0.0.0:5432->5432
/tcp   learning-platform-db
```

> `docker ps` on its own only shows running containers — a container you stopped a moment ago (see "Stopping and Starting a Container: `docker stop` / `docker start`") won't appear. Add `-a` (`docker ps -a`) to see every container regardless of status, which is usually the first thing worth running when a container you expect to see is "missing."

Reading Container Output: `docker logs`

A container's standard output and standard error are captured by Docker even when running detached — `docker logs` is how you read them after the fact.

```
docker logs learning-platform-db
```

```
PostgreSQL init process complete; ready for start up.
...
database system is ready to accept connections
```

Add `-f` (`docker logs -f learning-platform-db`) to follow the log stream continuously, the same way `tail -f` follows a growing file — this is usually the very first thing to reach for when a container isn't behaving as expected, before anything more involved.

Getting a Shell Inside a Container: `docker exec`

`docker run` starts a *new* container; `docker exec` runs an *additional* command inside a container that's **already running** — most commonly, to get an interactive shell inside it.

```
docker exec -it learning-platform-db psql -U postgres
```

```
psql (16.x)
Type "help" for help.

postgres=#
```

`-i` keeps standard input open, `-t` allocates a terminal — together (`-it`) they're what make the session interactive instead of running one command and immediately exiting.

Here it runs `psql` directly, since it's already installed inside the official `postgres` image; running `docker exec -it learning-platform-db sh` instead would open a plain shell inside the container, useful for inspecting files or environment variables directly.

Stopping and Starting a Container: `docker stop` / `docker start`

A container that's running can be stopped, and a stopped container — as long as it hasn't been removed — can be started again, resuming with the same name, same configuration, and (see "Docker Volumes," later in this course) the same data.

```
docker stop learning-platform-db
```

```
learning-platform-db
```

```
docker start learning-platform-db
```

```
learning-platform-db
```

`docker stop` sends a graceful shutdown signal first and only forces termination after a timeout — the same reason a Spring Boot application gets a chance to run its own shutdown hooks rather than being killed outright.

Cleaning Up: `docker rm`

`docker rm` removes a **stopped** container permanently — its filesystem, its configuration, everything about that specific container instance (not the image it was created from, which stays in `docker images` untouched).

```
docker stop learning-platform-db
docker rm learning-platform-db
```

```
learning-platform-db
learning-platform-db
```

> `docker rm` refuses to remove a container that's still running — you'll get an error naming the container as running instead. `docker rm -f <name>` skips the stop step and force-removes it directly, but that's a hard kill, not a graceful shutdown; prefer `docker stop` first when the container is doing anything that benefits from shutting down cleanly, like flushing writes to disk.

Putting It All Together: Running PostgreSQL in a Container

The full lifecycle — pull, run, verify, inspect, and clean up — end to end, using the same PostgreSQL container this section has been building toward, and the exact container this project's own database will eventually run in ("Docker Compose," in the next category, automates this same sequence for a full Spring Boot + PostgreSQL setup).

```
#!/bin/sh
# A developer needs a local PostgreSQL instance to run this project against,
# without installing PostgreSQL itself on their own machine.

docker pull postgres:16
# Output (trimmed):
# Status: Downloaded newer image for postgres:16

docker run --name learning-platform-db \
  -e POSTGRES_PASSWORD=secret \
  -p 5432:5432 \
  -d postgres:16
# Output:
# 7f8e9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f

docker ps
# Output:
# CONTAINER ID   IMAGE         STATUS         PORTS                     NAMES
# 7f8e9a0b1c2d   postgres:16   Up 2 seconds   0.0.0.0:5432->5432/tcp   learning-platform-db

docker logs learning-platform-db
# Output (trimmed):
# database system is ready to accept connections

# The database is now reachable at localhost:5432, exactly as if PostgreSQL
# had been installed directly -- but running as an isolated container instead.

docker exec -it learning-platform-db psql -U postgres -c "SELECT version();"
# Output:
# PostgreSQL 16.x on x86_64-pc-linux-gnu, compiled by gcc ...

# Work is done for the day -- stop the container without deleting it.
docker stop learning-platform-db
# Output:
# learning-platform-db

# The next morning, resume with the exact same data still intact.
docker start learning-platform-db
# Output:
# learning-platform-db

# Done with this database entirely -- remove it for good.
docker stop learning-platform-db
docker rm learning-platform-db
# Output:
# learning-platform-db
# learning-platform-db

# The postgres:16 image itself is untouched -- "docker images" still lists it,
# ready to start a brand-new container from at any time.
```

Nothing here is new mechanism — every single command was already introduced above; this only shows them chained together the way they'd actually be typed in one working session.

Common Mistakes

- Running `docker rm` on a still-running container and being surprised by the error — stop it first, or use `-f` deliberately, not by accident.
- Forgetting `-d` and being confused when the terminal appears to "hang" — without it, `docker run` attaches to the container's output in the foreground.
- Assuming `docker ps` shows every container that exists — it only shows running ones by default; use `docker ps -a` for the full picture (see "Listing Running Containers: `docker ps` ").
- Reaching for `docker exec` to start a brand-new container — `exec` only works against a container that's already running; a fresh container comes from `docker run`.

Best Practices

- Always pull and run a specific tag (`postgres:16`), not the implicit `:latest` — an image that updates silently underneath a running system is much harder to reason about than one pinned to a known version.
- Give containers meaningful `--name` values instead of accepting Docker's randomly generated ones — `learning-platform-db` is far easier to reference correctly in every later command than a random name.
- Reach for `docker logs -f` before anything more involved when a container seems to be misbehaving — most early debugging questions are answered by what the container itself already printed.
- Use `docker ps -a` (see "Tip" above) as a first troubleshooting step whenever a container you expect to see seems to be missing.

Summary, Cheat Sheet, and Glossary

Summary

- `docker pull` fetches an image from a registry without starting anything; `docker images` lists what's stored locally.
- `docker run` creates and starts a new container from an image; `docker ps` lists currently running containers (`docker ps -a` for all of them).
- `docker logs` reads a container's captured output; `docker exec` runs an additional command — typically an interactive shell — inside a container that's already running.
- `docker stop` / `docker start` pause and resume an existing container without deleting it; `docker rm` deletes a stopped container permanently (the image it came from is untouched).
- All of these are client commands talking to the Docker Engine daemon described in "What Is Docker?" — the CLI itself does none of the actual work.

Cheat Sheet

```
docker pull <image>:<tag>          # fetch an image from a registry
docker images                      # list locally stored images
docker run -d --name <name> <img>  # create + start a container, detached
docker ps                          # list running containers
docker ps -a                       # list all containers, any status
docker logs <name>                 # read a container's output
docker logs -f <name>              # follow a container's output live
docker exec -it <name> <cmd>       # run a command inside a running container
docker stop <name>                 # gracefully stop a running container
docker start <name>                # resume a stopped container
docker rm <name>                   # permanently remove a stopped container
```

Glossary

- **Tag**: a label attached to a specific version of an image within a repository (`postgres:16`); omitting one implies `:latest`.
- **Detached mode (`-d`)**: running a container in the background, returning the terminal prompt immediately instead of attaching to its output.
- **`docker exec`**: running an additional command inside an already-running container, most often an interactive shell.
- **Graceful stop**: `docker stop`'s default behavior — sending a shutdown signal and giving the process a chance to exit cleanly before forcing termination.