Docker Compose

docker-compose.yml, services, depends_on, Compose's automatic service-name networking, and volumes -- reducing everything "Docker Networking" and "Docker Volumes" did by hand to one file and docker compose up.

Intermediate 35 min

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

---

Everything the last two lessons did by hand — `docker network create`, `docker volume create`, and two separate `docker run` commands with matching `--network` and `-v` flags typed on each ("Docker Networking" and "Docker Volumes") — is real, working Docker, but it's also a sequence of commands that has to be remembered, retyped, and kept consistent by hand every single time. Docker Compose is the tool that turns that sequence into a single, declarative file. This lesson rebuilds the exact same two-container setup — this project's own app and database — as one `docker-compose.yml`.

What Docker Compose Is

Docker Compose reads a YAML file describing a set of related containers (**services**, in Compose's own vocabulary) and brings all of them up — or tears all of them down — with one command each. Nothing about the underlying mechanism changes: Compose still creates a network, still creates volumes, still runs containers with the equivalent of `docker run` flags — it just derives all of that from one file instead of requiring each `docker network create` / `docker volume create` / `docker run` to be typed separately, in the right order, every time.

The Compose File: `docker-compose.yml`

A Compose file's top level has two sections that matter for this project: `services`, one entry per container to run, and `volumes`, any named volumes those services need (see "Docker Volumes" for what a named volume actually is).

```
services:
  db:
    # ...
  app:
    # ...

volumes:
  db-data:
```

`db-data` here, declared once at the top level, is the exact same kind of named volume "Named Volumes" already covered — Compose creates it automatically the first time `docker compose up` runs, the same way `docker volume create` did by hand.

Defining a Service: `db`

```
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: secret
    volumes:
      - db-data:/var/lib/postgresql/data
```

This is a direct translation of the `docker run` command "Docker Volumes" used by hand: `image` is the same `postgres:16` from "Pulling an Image: `docker pull`", `environment` is the same `-e POSTGRES_PASSWORD=secret`, and `volumes` mounts the same named volume at the same `/var/lib/postgresql/data` path "Where PostgreSQL Stores Its Data" already explained — just expressed as YAML instead of command-line flags.

Defining a Service: `app`

```
services:
  app:
    build: .
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/postgres
      SPRING_DATASOURCE_PASSWORD: secret
    ports:
      - "8080:8080"
```

`build: .` tells Compose to build this project's own multi-stage Dockerfile (from "Dockerizing a Spring Boot Application") instead of pulling a pre-built image — the same effect as running `docker build` by hand, just triggered automatically as part of `docker compose up`. `ports` is the same `-p 8080:8080` "Running a Container: `docker run`" already covered.

depends_on — Ordering Startup

`depends_on: [db]` tells Compose to start `db` before starting `app` — without it, both would start at effectively the same time, and `app` could try to connect to a `db` that hasn't finished initializing yet.

> `depends_on` on its own only waits for the `db` **container to start**, not for PostgreSQL *inside* it to actually be ready to accept connections — those are two different moments, and a slow-starting database can still cause `app` to fail its first connection attempt even with `depends_on` in place. A `healthcheck` closes this gap properly — "Production Docker for Java Applications," the next lesson in this category, covers it in full.

Volumes in Compose

The `db-data` volume referenced in the `db` service's `volumes:` list has to be declared once at the file's top level, exactly as shown in "The Compose File: `docker-compose.yml`" — a service can't invent a volume name Compose hasn't been told about anywhere else in the file. Compose actually prefixes the volume's real name with the project's own name at creation time (visible as `learning-platform_db-data` in `docker volume ls`, not just `db-data`) — the short name in the YAML file is what services reference; the prefixed name is how it avoids colliding with a same-named volume from a completely different Compose project on the same machine.

Networking in Compose (Automatic)

This is where Compose actually saves real, meaningful work: "Docker Networking" required an explicit `docker network create` and `--network` on every `docker run` before two containers could reach each other by name at all. Compose skips that step entirely — every service defined in one `docker-compose.yml` is automatically placed on the same, Compose-created network, and **each service's name becomes its hostname** for every other service in that same file, no configuration required. That's exactly why `app`'s `SPRING_DATASOURCE_URL` above reaches PostgreSQL at `db:5432` — `db` is simply the other service's name in this same file, resolving automatically the same way `learning-platform-db` did on the manually created network in "Container-to-Container Communication by Name."

Running It: `docker compose up` / `docker compose down`

```
docker compose up -d
```

`docker compose up` reads `docker-compose.yml` in the current directory, creates whatever network and volumes are missing, builds or pulls whatever images are needed, and starts every service — `-d` runs it detached, the same meaning as `docker run -d`. `docker compose`

`down` reverses it — stopping and removing every container and the network Compose created — while deliberately **not** touching named volumes by default, for the exact reason "Docker Volumes" covered: a volume is meant to outlive routine container lifecycle events, and `docker compose down` treats itself as exactly that, an ordinary teardown, not a data-destroying one. `docker compose down -v` is the explicit opt-in to remove volumes too, when that's genuinely intended.

Putting It All Together

The complete `docker-compose.yml` for this project — the exact same app-plus-database setup "Docker Networking" and "Docker Volumes" built by hand, expressed as one file:

```
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: secret
    volumes:
      - db-data:/var/lib/postgresql/data

  app:
    build: .
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/postgres
      SPRING_DATASOURCE_PASSWORD: secret
    ports:
      - "8080:8080"

volumes:
  db-data:
```

And running it, end to end:

```
#!/bin/sh
# Everything "Docker Networking" and "Docker Volumes" typed by hand --
# docker network create, docker volume create, and two docker run commands
# -- replaced by a single docker-compose.yml and one command.

docker compose up -d
# Output (trimmed):
# [+] Running 3/3
# => Network learning-platform_default   Created
# => Volume "learning-platform_db-data"  Created
# => Container learning-platform-db-1    Started
# => Container learning-platform-app-1   Started

docker compose ps
# Output:
# NAME                       IMAGE          STATUS         PORTS
# learning-platform-app-1    learning-platform-app   Up 5 seconds   0.0.0.0:8080->8080/tcp
# learning-platform-db-1     postgres:16             Up 5 seconds   5432/tcp

docker compose logs app
# Output (trimmed):
# app-1  | Started LearningPlatformApplication in 3.2 seconds

curl http://localhost:8080/en
# Output (trimmed):
# <!doctype html> ... the homepage renders, meaning app reached db by service name ...

# Stop and remove everything Compose created -- containers and the default
# network. The named volume, "learning-platform_db-data", is deliberately
# left behind (see "Docker Volumes") -- it needs an explicit -v to remove.
docker compose down
# Output (trimmed):
# => Container learning-platform-app-1   Removed
# => Container learning-platform-db-1    Removed
# => Network learning-platform_default   Removed

# The data survives -- confirm the volume is still there.
docker volume ls
# Output:
# DRIVER    VOLUME NAME
# local     learning-platform_db-data
```

Every piece here is something this course already covered individually — a service's `image` / `build`, its `environment`, its `volumes`, its `ports`, `depends_on` for startup order, and the automatic per-file network giving `app` a way to reach `db` by name. Compose doesn't introduce a new mechanism; it just replaces a sequence of commands that had to be retyped correctly every time with one file that's read the same way every time.

Common Mistakes

- Referencing a volume name in a service's `volumes:` list that was never declared at the file's top-level `volumes:` section — Compose has no other way to know that name exists.

- Assuming `depends_on` guarantees the dependency is fully *ready*, not just *started* — see the warning under "`depends_on` — Ordering Startup."
- Running `docker compose down -v` out of habit and being surprised a database's data is gone — the plain `docker compose down` (no `-v`) already handles ordinary teardown safely; `-v` is a deliberate, separate choice.
- Hardcoding a container's actual name (like `learning-platform-db` from earlier lessons) into a Compose service's configuration instead of using the *other service's name* from the same `docker-compose.yml` — Compose's own automatic networking uses service names, not whatever a container happened to be named by hand elsewhere.

Best Practices

- Reach for Compose, not a sequence of manual `docker network create` / `docker volume create` / `docker run` commands, for any real multi-container setup — including local development of a project like this one.
- Keep a service's environment variables in the Compose file itself for values like this lesson's (already non-sensitive, illustrative ones) — a real deployment typically moves secrets like `POSTGRES_PASSWORD` out to a separate, uncommitted `.env` file or a proper secrets mechanism instead.
- Use `depends_on` for startup ordering, but pair it with a real `healthcheck` (covered next, in "Production Docker for Java Applications") whenever a dependency's actual readiness — not just its process starting — genuinely matters.
- Default to plain `docker compose down` for routine teardown, and treat `-v` as a deliberate, separate decision every single time it's typed.

Summary, Cheat Sheet, and Glossary

Summary

- Docker Compose reads a `docker-compose.yml` describing a set of services and brings them all up, or all down, with one command — replacing a sequence of manual `docker network create` / `docker volume create` / `docker run` commands with one declarative file.
- A service's `image` / `build`, `environment`, `volumes`, and `ports` are direct YAML equivalents of the `docker run` flags already covered throughout this course.
- Named volumes are declared once at the file's top level and referenced by name from any service that needs them.
- Every service in one Compose file is automatically placed on the same network, and each service's name becomes its hostname for every other service — no manual `docker network create` or `--network` required.
- `docker compose down` removes containers and the network but deliberately leaves named volumes intact by default; `docker compose down -v` is the explicit choice to remove them too.

Cheat Sheet

```
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: secret
    volumes:
      - db-data:/var/lib/postgresql/data
  app:
    build: .
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/postgres
    ports:
      - "8080:8080"

volumes:
  db-data:
```

```
docker compose up -d      # create/start everything, detached
docker compose ps         # list this project's running services
docker compose logs <svc> # read one service's output
docker compose down       # stop and remove containers + network (volumes kept)
docker compose down -v    # also remove named volumes
```

Glossary

- **Service**: one container definition inside a `docker-compose.yml`, equivalent to a `docker run` command expressed as YAML.
- `depends_on`: a Compose setting that orders one service's container start after another's, without waiting for the dependency to be fully ready.
- **Compose network**: the network Compose automatically creates for every file, on which every service can reach every other by its service name.
- `docker compose down -v`: the explicit, opt-in form of teardown that also removes named volumes — the plain form does not.