TUTORIAL · SEPTEMBER 14, 2026

Docker for Delphi Developers

A five-part course on Docker for Delphi developers — from installing Docker Desktop and running a PostgreSQL container to images, tags, Compose, and a three-tier isolation architecture, closing with a complete FireDAC command-line application you can build and run.

By Dr. Holger Flick



You need a real backend to develop against — a Postgres database, say — on your Windows machine, right now. So you find the installer. It registers a Windows service, edits your `PATH`, drops config files in three places, and asks for a superuser password you'll forget by Thursday. It works, until one project needs version 15 and another needs 17 and two services start fighting over the same port. And uninstalling leaves orphaned services, stray directories, and a registry you don't dare touch.

Every seasoned desktop developer knows that particular dread. This tutorial is the way out of it, delivered as a five-part article series: one command gives you a real running database, one more command makes it vanish without a trace, and the same idea scales up to the full web ' backend ' database architecture behind essentially every production system you've ever used.

No Kubernetes, no DevOps liturgy, no YAML for its own sake — just the mental models that make containers click for someone who already thinks in classes and objects, with real Delphi and FireDAC code along the way. Read the parts in order: each one starts exactly where the previous one ended. Then stay for the sixth entry, a complete worked example that puts all five parts to work in a single buildable project.

Part 1 — What a container actually is

Install Docker Desktop and Run Your First Container (1/5)

Three words stand between you and everything else: **image**, **container**, **registry**. This part gets Docker Desktop onto Windows, runs `hello-world`, and then hands you the one analogy that makes the rest of the series easy — a Delphi-shaped analogy you already use every day, involving a class, an object, and a constructor. It also answers the licensing question honestly, with the open-source alternatives named rather than hidden.

Part 2 — A real PostgreSQL server, and FireDAC talking to it

PostgreSQL in a Container, Reached from Delphi FireDAC (2/5)

Now the promise gets cashed in: a production-grade PostgreSQL server started with a single command, no installer, no service wedged into your Windows startup — and a real Delphi console program using FireDAC to create a table, insert rows, and read them back out of that container. There's exactly one honest snag between you and that output, a dependency the container cannot hand you, and it stops most people on their first attempt. It gets a section of its own here.

Part 3 — Where that image actually came from

Docker Images, Tags, and Repositories Explained (3/5)

You typed `postgres` and Docker produced a database server. Where did it come from? This part decodes `docker.io/library/postgres:16` slice by slice until you read an image reference the way you read a file path — registry, repository, tag. It also explains why the second pull is nearly instant, how to tell a trustworthy image from a random one, and the single most widely misunderstood thing in all of Docker: what the `latest` tag does **not** mean.

Part 4 — Your whole stack in one file

Docker Compose: Your Whole Stack in One File (4/5)

A real application is never one container. Starting each service by hand means memorizing a paragraph of `docker run` flags per service and running them in the right order on every machine — miss one flag and the backend can't find the database. Compose erases that entire class of problem with one readable file committed to your repo. Along the way it delivers the two answers the series had been deliberately saving: how services find each other without a single IP address, and how your data survives a restart.

Part 5 — The architecture you can actually ship

Isolating the Database: Web ’ Backend ’ Postgres in Docker (5/5)

Part 4's stack ran, and it was tidy, and it had one quiet flaw left in on purpose: the database was reachable from the host. Convenient — and exactly the shape you must not ship. The finale builds the three-tier layering behind essentially every production web application: a web client, a backend that alone holds the credentials, and Postgres on a private network with no public door at all. The backend appears in both stacks worth reaching for — TMS XData in Delphi and Express in TypeScript — so you can pick by fit rather than by hype. And the isolation gets proven, not asserted.

The worked example — a complete FireDAC application, end to end

Delphi and PostgreSQL from Scratch: A FireDAC CLI in Four Units

Most database examples stop exactly where the interesting part starts: a form, a dropped `TFDConnection`, a hardcoded password, a screenshot. This one doesn't. It is the whole series put to work in one buildable project — a `compose.yaml` that starts the server, and a Delphi command-line application in four source files that creates its own database, builds two tables, inserts rows inside a transaction, reads them back with a join, and prints a formatted report. Every file appears in full, every command is one you can paste into a Windows terminal, and the build happens from the command line without touching a mouse. There's also an opinion in here worth arguing with: whether an application should create its own database at all.

What you'll be able to do afterwards

- Run a real PostgreSQL server on your machine in one command — and remove it just as cleanly, leaving nothing behind.
- Read any image reference on sight, pin versions deliberately instead of accidentally, and know why `latest` is a trap in anything you depend on.
- Describe a whole multi-service stack in one `compose.yaml` that behaves identically on every laptop and every server.
- Lay out web, backend, and database so the most valuable tier is genuinely unreachable except through code you wrote.
- Connect Delphi and FireDAC to a containerized database without getting stopped by the one dependency that stops everybody.

> A container is an object, an image is its class, and `docker run` is the constructor. Once that clicks, Docker stops being a mystery and starts being a tool you reach for without thinking.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —