# database # guide # tooling

# PostgreSQL & Docker Setup

**TCSS 460 — Client/Server Programming**

Your Express API has been returning data from in-memory arrays and third-party APIs. Real applications store data in a **database** — and for this course, that database is PostgreSQL. But installing a database engine directly on your laptop is messy: version conflicts, background services, platform-specific quirks. Instead, you will run PostgreSQL inside a **Docker container** — an isolated, reproducible environment that works the same on every machine. This guide walks you through both tools, from installation to your first connection.

---

## 1 What is PostgreSQL?

**PostgreSQL** (often just "Postgres") is an open-source relational database management system. It stores data in tables with rows and columns, enforces relationships between tables, and lets you query data using SQL (Structured Query Language).

### 1.1 Why PostgreSQL?

PostgreSQL is not an academic toy. It is the database behind Instagram, Spotify, the International Space Station's ground systems, and thousands of production applications. It is free, open-source, and runs on every major operating system.

Why PostgreSQL for this course:

- **Production-grade** — The skills you learn here transfer directly to real jobs. Companies use PostgreSQL because it is reliable, standards-compliant, and handles everything from a personal project to millions of rows.
- **SQL standard** — PostgreSQL follows the SQL standard more closely than most databases. What you learn in PostgreSQL's SQL dialect works in MySQL, SQL Server, and other relational databases with minor adjustments.
- **Ecosystem** — PostgreSQL has excellent tooling, documentation, and community support. The ORM you will use later in the course (Prisma) has first-class PostgreSQL support.

- **Free** — No license fees, no student discounts to manage, no expiring trials.

> **Relational vs. Non-Relational**
>
> You may have heard of non-relational (NoSQL) databases like MongoDB or Firebase. Those are valid choices for some applications, but relational databases remain the industry default for structured data with relationships — which describes most business applications. This course teaches the relational model because it is foundational knowledge you will use regardless of which specific database you encounter in your career.

---

# 2 What is Docker?

Docker is a platform that lets you run software in **containers** — lightweight, isolated environments that package an application with everything it needs to run: the runtime, libraries, configuration, and dependencies.

## 2.1 The Problem Docker Solves

Imagine you need PostgreSQL for this course. Without Docker, you would:

1. Download the PostgreSQL installer for your specific OS
2. Run the installer, choose a port, set a password, configure a data directory
3. Start the PostgreSQL service (which now runs in the background permanently)
4. Hope that the version you installed does not conflict with anything else on your machine
5. Repeat all of this when your teammate has a different OS or a different version

This is the classic "works on my machine" problem. You configured PostgreSQL one way; your teammate configured it differently; the production server has a third configuration. Debugging these differences wastes hours.

## 2.2 Containers: Runs the Same Everywhere

A Docker container is a self-contained package that includes the application and its entire environment. When you run PostgreSQL in a container:

- The PostgreSQL version is locked to a specific image (e.g., `postgres:17`)
- The configuration is defined in a file you can share with your team

- Starting the database is one command: `docker compose up`
- Stopping it is one command: `docker compose down`
- Your machine stays clean — nothing is installed globally

Every student in the class runs the exact same PostgreSQL version with the exact same configuration. "Works on my machine" stops being a problem.

### 2.3 Containers Are Not Virtual Machines

You may have used virtual machines (VMs) in other courses. Containers are similar in concept — they isolate software — but different in implementation:

| | Virtual Machine | Container |
|---|---|---|
| **Includes** | Full guest OS + application | Application + dependencies only |
| **Size** | Gigabytes | Megabytes |
| **Startup time** | Minutes | Seconds |
| **Resource usage** | Heavy (runs a full OS) | Lightweight (shares host OS kernel) |
| **Use case** | Running a different OS entirely | Running an isolated application |

A VM boots an entire operating system inside your operating system. A container shares your computer's OS kernel and only isolates the application layer. This is why containers start in seconds and use a fraction of the resources.

**You Don't Need to Become a Docker Expert**

Docker is a deep tool used by DevOps engineers to orchestrate hundreds of services. For this course, you need exactly one skill: running a PostgreSQL container with Docker Compose. The starter code provides the configuration file — you just need to understand what it does and how to start and stop it.

---

## 3 Installing Docker

Docker runs through **Docker Desktop**, a GUI application that manages the Docker engine on your machine.

## 3.1 Download and Install

1. Go to docker.com/products/docker-desktop
2. Download Docker Desktop for your operating system (Mac or Windows)
3. Run the installer and follow the prompts

**Windows-specific notes:**

- Docker Desktop requires **WSL 2** (Windows Subsystem for Linux). The installer will prompt you to enable it if it is not already installed.
- If prompted to choose between WSL 2 and Hyper-V backends, choose **WSL 2** — it is faster and uses fewer resources.
- You may need to restart your computer after installation.

**Mac-specific notes:**

- Choose the correct installer for your chip: **Apple Silicon (M1/M2/M3/M4)** or **Intel**. If you are unsure, click the Apple menu → "About This Mac" and look for "Chip."
- You may need to grant Docker permission in System Settings → Privacy & Security.

## 3.2 Verify Installation

After installation, open a terminal and run:

```
docker --version
```

You should see something like:

```
Docker version 28.1.1, build ...
```

Also verify Docker Compose (included with Docker Desktop):

```
docker compose version
```

```
Docker Compose version v2.35.1
```

**docker compose vs docker-compose**

The old command was `docker-compose` (with a hyphen) — that was Compose v1, written in Python, and it reached end of life in July 2023. The current command is `docker compose` (with a space) — Compose v2, built into the Docker CLI. Always use `docker compose` (no hyphen). If you see tutorials using `docker-compose`, the commands still work the same way, but use the modern syntax.

**Gen AI & Learning: Watch for Old Docker Syntax**

AI coding assistants frequently generate the older `docker-compose` (hyphen) syntax instead of the modern `docker compose` (space) command. The old command dominates training data because it was the standard for years. If an AI tool generates Docker commands for you, check for the hyphen — it is one of the most common outdated suggestions you will encounter.

## 3.3 Docker Desktop Basics

Docker Desktop runs as a background application. You will see a whale icon in your menu bar (Mac) or system tray (Windows). The Docker engine only runs when Docker Desktop is running — if you close it, your containers stop.

You generally do not need to adjust Docker Desktop's settings. The defaults allocate sufficient memory and CPU for running a PostgreSQL container. If you are running multiple heavy containers and experience slowness, you can increase resources in Docker Desktop → Settings → Resources, but this is unlikely for this course.

**Try It Yourself**

1. Open Docker Desktop
2. Run `docker --version` and `docker compose version` in your terminal
3. If both commands produce version numbers, you are ready to proceed

---

# 4 Running PostgreSQL Directly (Reference)

> **For Reference — You Will Use Docker Compose**
>
> In this course, you will use Docker Compose (Section 5) to run PostgreSQL — not the `docker run` command shown here. This section explains the underlying flags and concepts so you understand what Docker Compose is doing for you. Read it for understanding, not as steps to follow.

Docker images are pre-built packages that define what software a container runs. The official PostgreSQL image is maintained on Docker Hub and is the standard way to run PostgreSQL in Docker.

## 4.1 The `docker run` Command

You can start a PostgreSQL container with a single command:

```
docker run --name my-postgres \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -e POSTGRES_DB=mydb \
  -p 5432:5432 \
  -v pgdata:/var/lib/postgresql/data \
  -d postgres:17
```

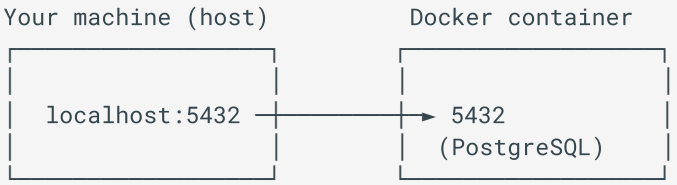
That is a lot of flags. Here is what each one does:

| Flag | Purpose |
| --- | --- |
| `--name my-postgres` | Give the container a human-readable name (instead of a random string) |
| `-e POSTGRES_USER=postgres` | Set the database superuser username |
| `-e POSTGRES_PASSWORD=mysecretpassword` | Set the superuser password |
| `-e POSTGRES_DB=mydb` | Create a database named `mydb` on first run |
| `-p 5432:5432` | Map port 5432 on your machine to port 5432 inside the container |

| Flag | Purpose |
| --- | --- |
| `-v pgdata:/var/lib/postgresql/data` | Persist database files in a named volume (survives container restarts) |
| `-d` | Run in detached mode (background) |
| `postgres:17` | Use the official PostgreSQL 17 image |

## 4.2 Understanding Port Mapping

The `-p 5432:5432` flag maps a port on your machine (the host) to a port inside the container. PostgreSQL listens on port 5432 by default.




When your Express application connects to `localhost:5432`, the connection is forwarded into the container where PostgreSQL is running. From your application's perspective, it looks like PostgreSQL is installed locally — but it is actually running inside Docker.

## 4.3 Understanding Volumes

Without the `-v` flag, all data inside the container disappears when the container is removed. The volume flag ( `-v pgdata:/var/lib/postgresql/data` ) tells Docker to store PostgreSQL's data files in a **named volume** called `pgdata` that persists independently of the container.

Think of it this way: the container is disposable, but the volume is permanent. You can delete and recreate the container, and your data survives — as long as you reattach the same volume.

## 4.4 Managing the Container

Once the container is running, you can manage it with these commands:

```
# Check which containers are running
docker ps
```

```
# Stop the container (PostgreSQL shuts down, data is preserved)
docker stop my-postgres

# Start it again
docker start my-postgres

# Remove the container entirely (data in the volume is preserved)
docker rm my-postgres

# View container logs (useful for debugging)
docker logs my-postgres
```

**docker run vs docker start**

`docker run` **creates and starts** a new container from an image. `docker start` **restarts** an existing stopped container. If you run `docker run` twice with the same name, you get an error because the container already exists. Use `docker start` to restart a stopped container.

---

# 5 Docker Compose

The `docker run` command in the previous section works, but it is long, easy to mistype, and hard to share with teammates. **Docker Compose** solves this by defining your container configuration in a YAML file.

## 5.1 Why Docker Compose?

Instead of memorizing a long `docker run` command, you write a `docker-compose.yml` file once and then:

- Start everything: `docker compose up -d`
- Stop everything: `docker compose down`
- Share the file with your team (it goes in your git repo)

One file, one command, reproducible setup. Every team member runs the exact same database configuration.

## 5.2 The Compose File

Here is a `docker-compose.yml` file for a PostgreSQL service:

```yaml
services:
  db:
    image: postgres:17
    container_name: tcss460-db
    restart: unless-stopped
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: mysecretpassword
      POSTGRES_DB: tcss460
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

This file describes the same configuration as the `docker run` command from Section 4, but in a readable, version-controlled format. Let's walk through each part:

| Key | Purpose |
|---|---|
| `services:` | Lists the containers to run (you can have multiple services) |
| `db:` | The name of this service (you choose the name) |
| `image: postgres:17` | Which Docker image to use |
| `container_name: tcss460-db` | A fixed name for the container |
| `restart: unless-stopped` | Automatically restart the container if it crashes or Docker restarts |
| `environment:` | Environment variables passed to the container (same as `-e` flags) |
| `ports:` | Port mapping (same as `-p` flag) |
| `volumes:` | Data persistence (same as `-v` flag) |
| `volumes: pgdata:` (bottom) | Declares the named volume so Docker manages it |

**The Compose File Will Be Provided**

Your project starter code will include a `docker-compose.yml` file. You do not need to write one from scratch — but you do need to understand what it does, because you may need to change the password, database name, or port if there is a conflict on your machine.

## 5.3 Starting and Stopping

Navigate to the directory containing your `docker-compose.yml` file and run:

```
# Start the database (detached mode — runs in background)
docker compose up -d
```

The first time you run this, Docker downloads the PostgreSQL image (about 150 MB). Subsequent starts are instant because the image is cached locally.

```
# Stop the database (data is preserved in the volume)
docker compose down
```

```
# Stop AND delete the volume (destroys all data — use with caution)
docker compose down -v
```

**docker compose down -v Deletes Your Data**

The `-v` flag removes the named volume, which means all your database tables, rows, and schema are permanently deleted. This is useful when you want a completely fresh start, but never do it accidentally. Without the `-v` flag, `docker compose down` preserves your data.

## 5.4 Checking Status

To see if your database container is running:

```
docker compose ps
```

This shows the status of all services defined in your Compose file. You should see something like:

```
NAME           IMAGE         COMMAND                  SERVICE   STATUS
PORTS
tcss460-db     postgres:17   "docker-entrypoint.s…"   db        Up 3 minutes
0.0.0.0:5432->5432/tcp
```

If the STATUS column says "Up," your database is running and ready for connections.

**Try It Yourself**

1. Create a new directory and add a `docker-compose.yml` file with the content from Section 5.2
2. Run `docker compose up -d`
3. Run `docker compose ps` — verify the container is running
4. Run `docker compose down` — verify the container stops
5. Run `docker compose up -d` again — notice how fast it starts the second time (no image download)

---

# 6 Connecting to Your Database

Your PostgreSQL container is running. Now you need to connect to it — both from the command line (for manual exploration) and from your Express application (in the next guide).

## 6.1 Connection Strings

A **connection string** (also called a **database URL**) is a single string that contains everything needed to connect to a database: the protocol, username, password, host, port, and database name.

```
postgresql://postgres:mysecretpassword@localhost:5432/tcss460
└──────┘   └──────┘ └──────────────┘ └────────┘ └──┘ └──────┘
protocol   user     password         host      port  database
```

This format is standardized. Every PostgreSQL client — psql, Prisma, pgAdmin, your application code — understands this connection string format. You will use this exact pattern when connecting Prisma to your database in the next guide.

## 6.2 The `.env` File Pattern

You already know this pattern from the Proxy Pattern guide — secrets like API keys belong in a `.env` file that is never committed to git. Database credentials follow the same rule.

Add your connection string to your project's `.env` file:

```
PORT=3000
DATABASE_URL=postgresql://postgres:mysecretpassword@localhost:5432/tcss460
```

Your application reads `DATABASE_URL` from the environment at runtime. Your `.gitignore` ensures the `.env` file is never committed. Your teammates create their own `.env` files with their own credentials.

> ⚠ **Never Commit Database Credentials**
>
> The same rule applies here as with API keys: database passwords in your git history are a security vulnerability. If your database is cloud-hosted (Section 8), a leaked connection string gives anyone full access to your data.

## 6.3 Connecting with psql

**psql** is PostgreSQL's built-in command-line client. Since PostgreSQL is running inside a Docker container, you use `docker exec` to run psql inside that container:

```
docker exec -it tcss460-db psql -U postgres -d tcss460
```

Breaking this down:

| Part | Purpose |
|---|---|
| `docker exec` | Run a command inside a running container |
| `-it` | Interactive mode with a terminal (so you can type commands) |
| `tcss460-db` | The container name (from your `docker-compose.yml`) |
| `psql` | The command to run inside the container |
| `-U postgres` | Connect as the `postgres` user |
| `-d tcss460` | Connect to the `tcss460` database |

You should see a prompt like:

```
tcss460=#
```

This is the psql interactive shell. You are now connected to your PostgreSQL database and can run SQL commands directly.

## 6.4 Essential psql Commands

psql has its own set of "backslash commands" for navigating the database. These are not SQL — they are psql-specific shortcuts:

| Command | Purpose |
|---|---|
| `\l` | List all databases |
| `\dt` | List all tables in the current database |
| `\d tablename` | Describe a table (show columns, types, constraints) |
| `\du` | List all users/roles |
| `\q` | Quit psql |

You can also run SQL directly at the psql prompt:

```
SELECT version();
```

This returns the PostgreSQL version running inside your container, confirming your connection works.

**Try It Yourself**

1. Make sure your Docker container is running (`docker compose up -d`)
2. Connect with psql: `docker exec -it tcss460-db psql -U postgres -d tcss460`
3. Run `\l` to list databases — you should see `tcss460` in the list
4. Run `\dt` — it will show no tables (you have not created any yet)
5. Run `SELECT 1 + 1;` — you should see `2` (confirming SQL works)
6. Run `\q` to exit

# 7 Cloud-Hosted PostgreSQL

Docker is perfect for local development — you have full control, it is fast, and it costs nothing. But when you deploy your API to production, you need a database that is accessible from the internet, not just from `localhost`.

## 7.1 When to Use Cloud vs. Local Docker

| Scenario | Use |
|---|---|
| **Local development** | Docker (fast, free, full control) |
| **Production deployment** | Cloud-hosted PostgreSQL (accessible from your deployed API) |
| **Team collaboration during development** | Docker locally (each person runs their own instance) |

During this course, you will use Docker for local development and a cloud-hosted database when you deploy your API in later sprints.

## 7.2 Cloud PostgreSQL Providers

Several services offer managed PostgreSQL databases with free or student-friendly tiers suitable for course projects. These align with the hosting platforms covered in the Deploying a Simple Web API guide:

- **Render** — Offers managed PostgreSQL alongside your deployed API. The free tier is available but expires after 30 days — plan for this limitation.
- **Heroku** — Offers Heroku Postgres as an add-on. The Essential-0 plan ($5/month) is covered by GitHub Student Developer Pack credits.
- **Supabase** — A backend-as-a-service platform built on PostgreSQL with a generous free tier (500 MB). Works with any hosting platform — useful if your hosting provider's database offering is limited or expired.

> **No Endorsement**
>
> The course does not require a specific cloud provider. When you reach the deployment sprint, you will choose a provider based on your team's needs. All options above provide a standard PostgreSQL connection string — your application code does not change regardless of which provider you choose.

### 7.3 The Connection String is the Same Pattern

This is the key insight: whether your database runs in Docker on your laptop or in the cloud, your application connects using the same connection string format.

**Local Docker:**

```
DATABASE_URL=postgresql://postgres:mysecretpassword@localhost:5432/tcss460
```

**Cloud-hosted (example from Neon):**

```
DATABASE_URL=postgresql://username:password@ep-cool-name-123456.us-east-2.aws.neon.tech/tcss460?sslmode=require
```

The only differences are the host (a cloud URL instead of `localhost`), possibly a different port, and typically an SSL requirement (`?sslmode=require`). Your application code reads `DATABASE_URL` from the environment and does not care where the database actually lives.

This is why the `.env` pattern matters. When you switch from local development to production deployment, you change one line in your `.env` file (or set the environment variable on your hosting platform). No code changes.

---

## 8 GUI Tools (Optional)

The command-line psql client is sufficient for everything in this course, but some students prefer a graphical interface for exploring their database. Here are a few options:

- **pgAdmin** — The official PostgreSQL GUI. Free, open-source, full-featured. Can be overwhelming for beginners due to its many options.
- **DBeaver** — A universal database client that supports PostgreSQL and many other databases. Free community edition available.

- **TablePlus** — A clean, modern database client for Mac and Windows. Free tier with some limitations.
- **Prisma Studio** — A web-based GUI that comes built into Prisma, the ORM you will use in this course. You will learn about it in the Prisma ORM guide. This is likely the most convenient option since it is already part of your project.

> **Not Required**
>
> GUI tools are entirely optional. You can complete every assignment in this course using psql and Prisma Studio. If you enjoy visual tools, try one — but do not spend time debugging a GUI client when the command line works.

---

# 9 Common Issues

## 9.1 Port 5432 Already in Use

If you see an error like `port is already allocated` or `address already in use`, something else on your machine is using port 5432. This usually means:

- You have another PostgreSQL instance running (maybe from a previous course or a native installation)
- You have another Docker container using port 5432

**Fix:** Change the host port in your `docker-compose.yml`:

```
ports:
  - "5433:5432"  # Use 5433 on your machine, still 5432 inside the container
```

Then update your connection string to use port 5433:

```
DATABASE_URL=postgresql://postgres:mysecretpassword@localhost:5433/tcss460
```

**Finding What's Using Port 5432**

On Mac/Linux: `lsof -i :5432`

On Windows (PowerShell): `netstat -ano | findstr :5432`

These commands show which process is using the port so you can decide whether to stop it or change your Docker port.

## 9.2 Container Not Running

If your application cannot connect to the database, the first thing to check is whether the container is actually running:

```
docker compose ps
```

If the container is not listed or shows a status other than "Up," start it:

```
docker compose up -d
```

If the container keeps crashing, check the logs:

```
docker compose logs db
```

## 9.3 Authentication Errors

If you see `password authentication failed for user "postgres"`, the password in your connection string does not match the password the container was created with.

The `POSTGRES_PASSWORD` environment variable is only used when the container **first creates** the data volume. If you change the password in `docker-compose.yml` after the volume already exists, the old password is still in effect.

**Fix:** Remove the volume and recreate the container:

```
docker compose down -v
docker compose up -d
```

**This Deletes All Data**

The `-v` flag removes the volume, which destroys all your database data. This is fine during initial setup, but be careful once you have data you care about.

### 9.4 "Connection Refused"

If you see `connection refused` when trying to connect, check these things in order:

1. **Is Docker Desktop running?** Look for the whale icon in your menu bar / system tray. If Docker Desktop is not running, no containers can run.
2. **Is the container running?** Run `docker compose ps` to check.
3. **Is the port correct?** Make sure your connection string uses the same port as your `docker-compose.yml` port mapping.
4. **Did the container finish starting?** PostgreSQL takes a few seconds to initialize on first run. Check `docker compose logs db` — look for the message `database system is ready to accept connections`.

**Gen AI & Learning: Debugging Container Issues**

Docker error messages can be cryptic, especially when they involve networking or volume permissions. If you hit an error you do not understand, paste the full error message into your AI coding assistant and ask it to explain what went wrong. Container issues are well-documented, and AI tools are good at diagnosing them from error output. The key is giving the tool the complete error message, not a summary — the details matter.

---

## 10 Summary

| Concept | Key Point |
|---|---|
| PostgreSQL | Open-source relational database used in production by major companies |
| Docker | Platform for running software in isolated, reproducible containers |
| Container vs. VM | Containers share the host OS kernel — lighter, faster, and disposable |
| Docker Desktop | GUI application that runs the Docker engine on Mac and Windows |
| `docker compose` | Modern CLI command for Docker Compose (replaces `docker-compose`) |

| Concept | Key Point |
| --- | --- |
| `docker-compose.yml` | YAML file defining your container configuration — one file, one command |
| `docker compose up -d` | Start all services in detached (background) mode |
| `docker compose down` | Stop all services (data preserved in volumes) |
| `docker compose down -v` | Stop all services AND delete volumes (destroys data) |
| Port mapping | `-p 5432:5432` connects your machine's port to the container's port |
| Named volumes | Persist data independently of the container lifecycle |
| Connection string | `postgresql://user:password@host:port/database` — same format everywhere |
| `.env` file | Store database credentials outside of source code, never commit to git |
| psql | PostgreSQL's command-line client, run via `docker exec` |
| Cloud PostgreSQL | Same connection string format, different host — no code changes needed |

---

## 11 References

**Official Documentation:**

- PostgreSQL Documentation — Official PostgreSQL reference
- Docker Documentation — Docker platform documentation
- Docker Compose Overview — Compose file reference and CLI commands

- Official PostgreSQL Docker Image — Image tags, environment variables, and configuration
- Docker Desktop — Installation guides for Mac, Windows, and Linux

---

## 12 Further Reading

**External Resources**

- SQL Fundamentals Guide — Introduction to SQL queries, the language you use to interact with PostgreSQL
- Prisma ORM Guide — Connecting your Express application to PostgreSQL using Prisma
- The Proxy Pattern Guide — Where you first learned the `.env` pattern for secrets
- Neon Documentation — Serverless PostgreSQL setup and connection guides
- Supabase Documentation — PostgreSQL platform with additional backend features
- Docker Getting Started Guide — Docker fundamentals beyond what this course requires

---

*This guide is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.*