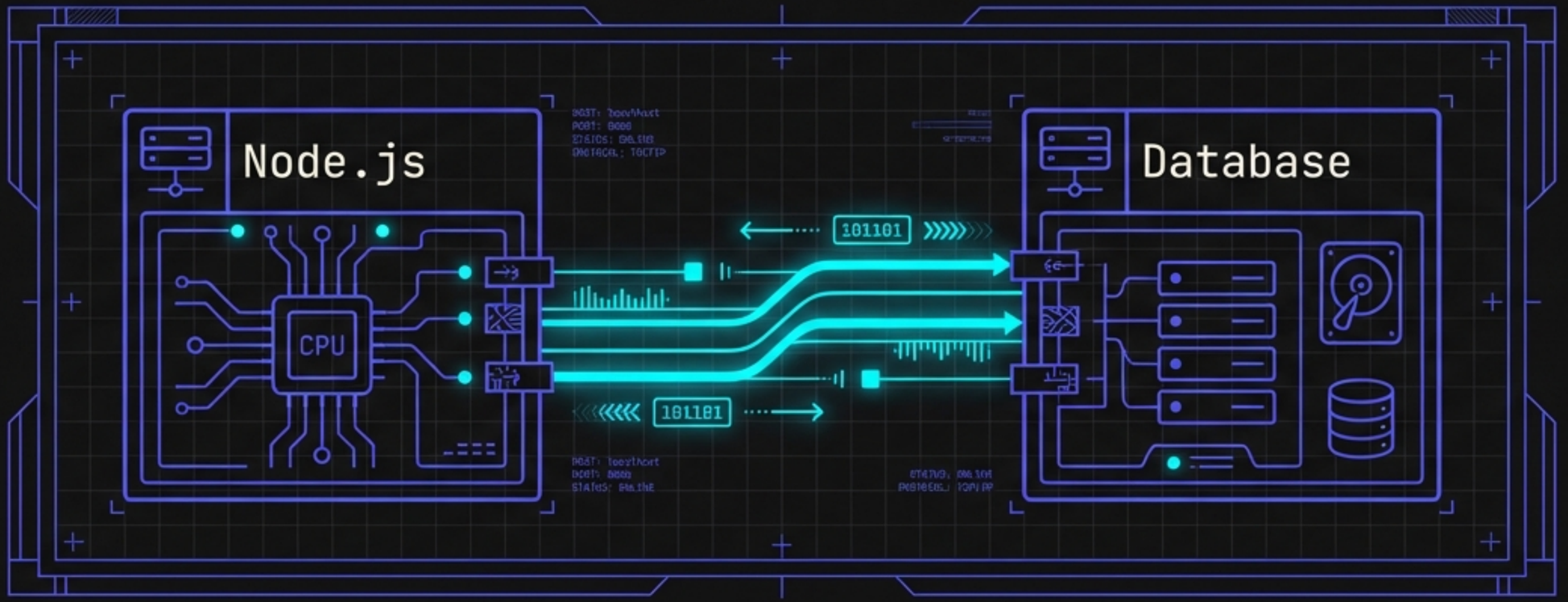
Orchestrating the Distributed Fleet

Professor Solo
Command Center

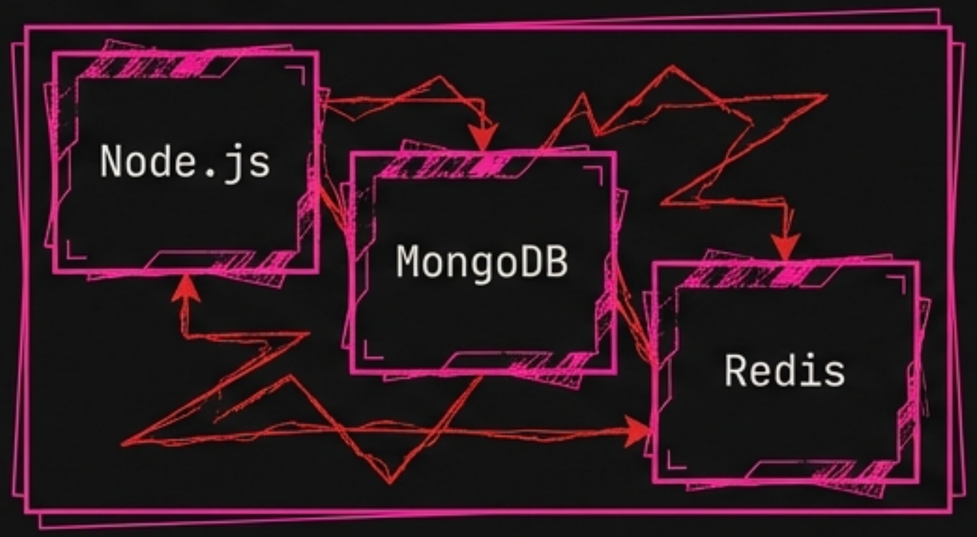
A ShipShape study guide for multi-container local development, Docker Compose, and surviving the localhost trap.




```
> docker compose up -d_
```

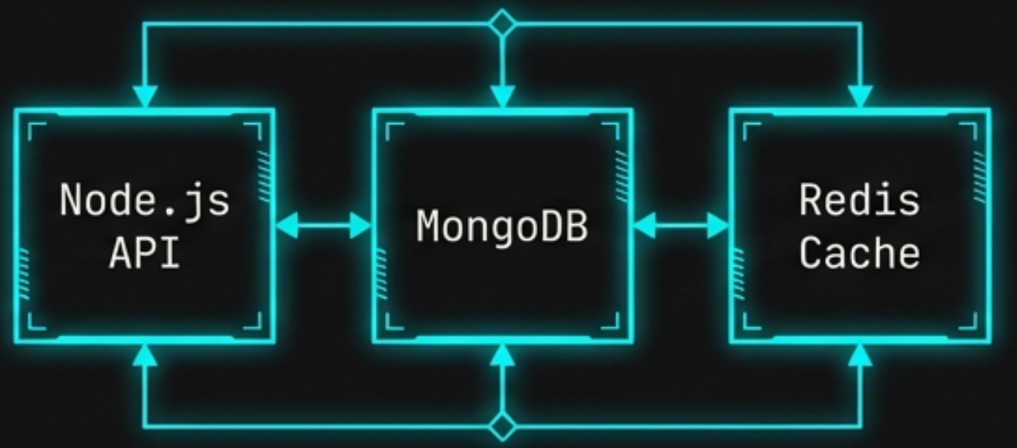
The Myth of the Do-It-All Container

The Monolith



- Updates break everything.
- Scaling duplicates the database.
- Crashes corrupt the whole filesystem.

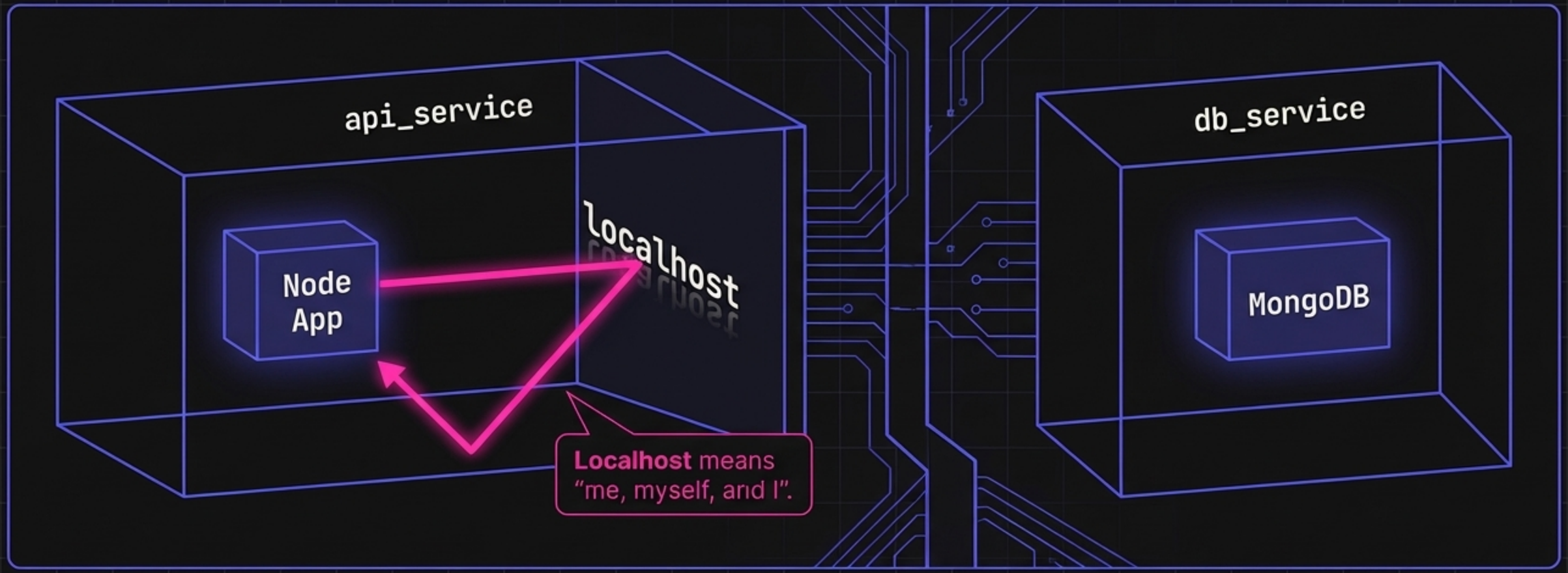
The Fleet



- One process per container.
- Independent scaling.
- Dedicated API (code) + Dedicated Database (data).

To build professional applications, we stop treating localhost as one giant machine and start managing a cooperating fleet.

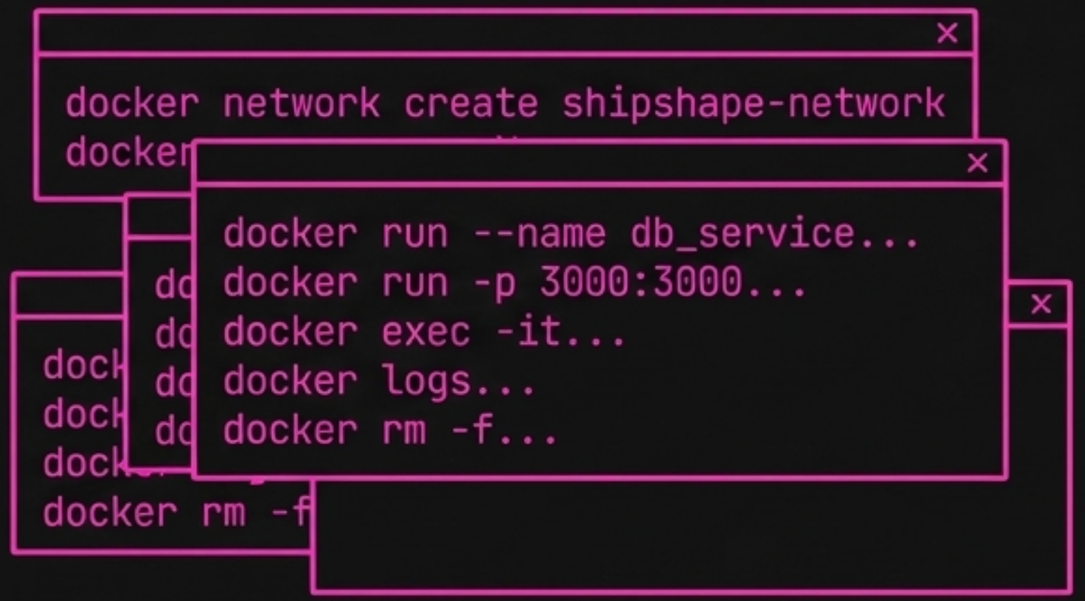
Escaping the Localhost Trap



Inside a container, **`localhost`** points back to that exact same container. It does **NOT** point to your laptop, and it does **NOT** point to the database container sitting next to it.

From Imperative Chores to Declarative Blueprints

Imperative Chores



Telling Docker exactly how to do every little step.

Declarative Blueprints

compose.yaml

```
services:
  db_service:
    image: postgres
    ports: 5432
  api_service:
    image: node_app
    ports: 3000
  networks:
    - shipshape-network
```

Describing the setup you want and letting Compose make reality match the file.

Compose is not magic. It just takes the networks, service names, builds, and ports you used to type by hand, and locks them into version control.

Anatomy of the Master Blueprint

```
services:
  api_service:
    build: .
    ports:
      - "3000:3000"
  db_service:
    image: mongo:8.0
```

Built from Source

Tells Compose to look for a local `Dockerfile` and build the image from our project folder.

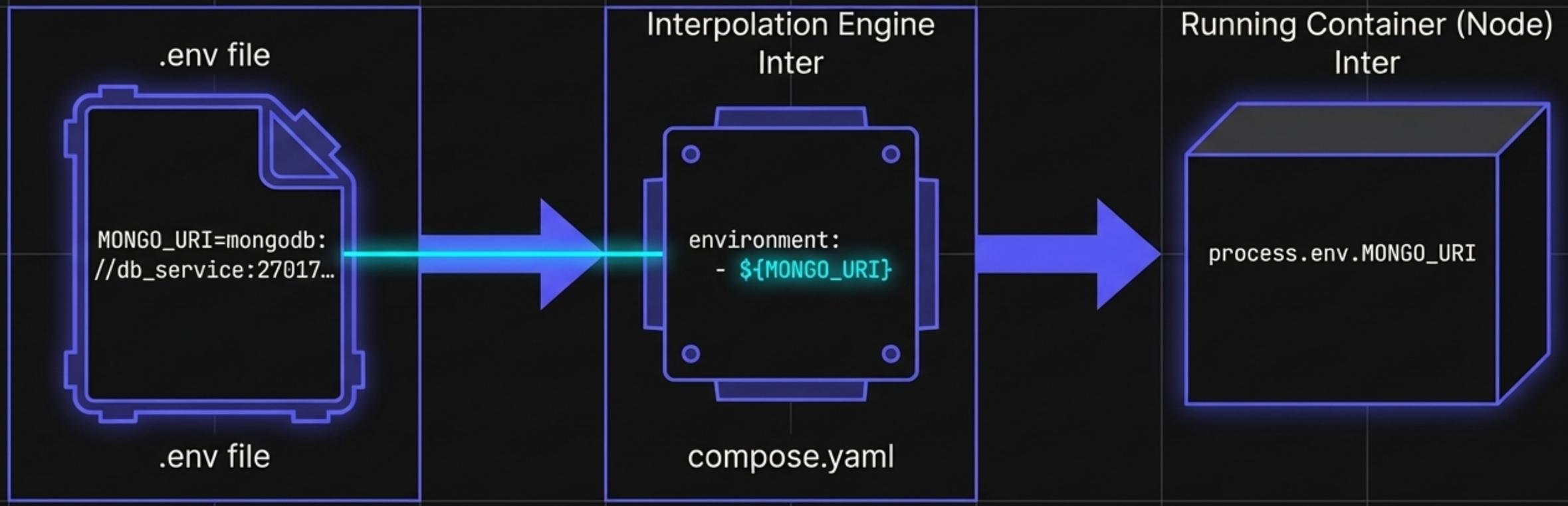
Pulled from Registry.

Bypasses local building. Grabs the official, pre-made database tool directly from Docker Hub.

The Bridge.

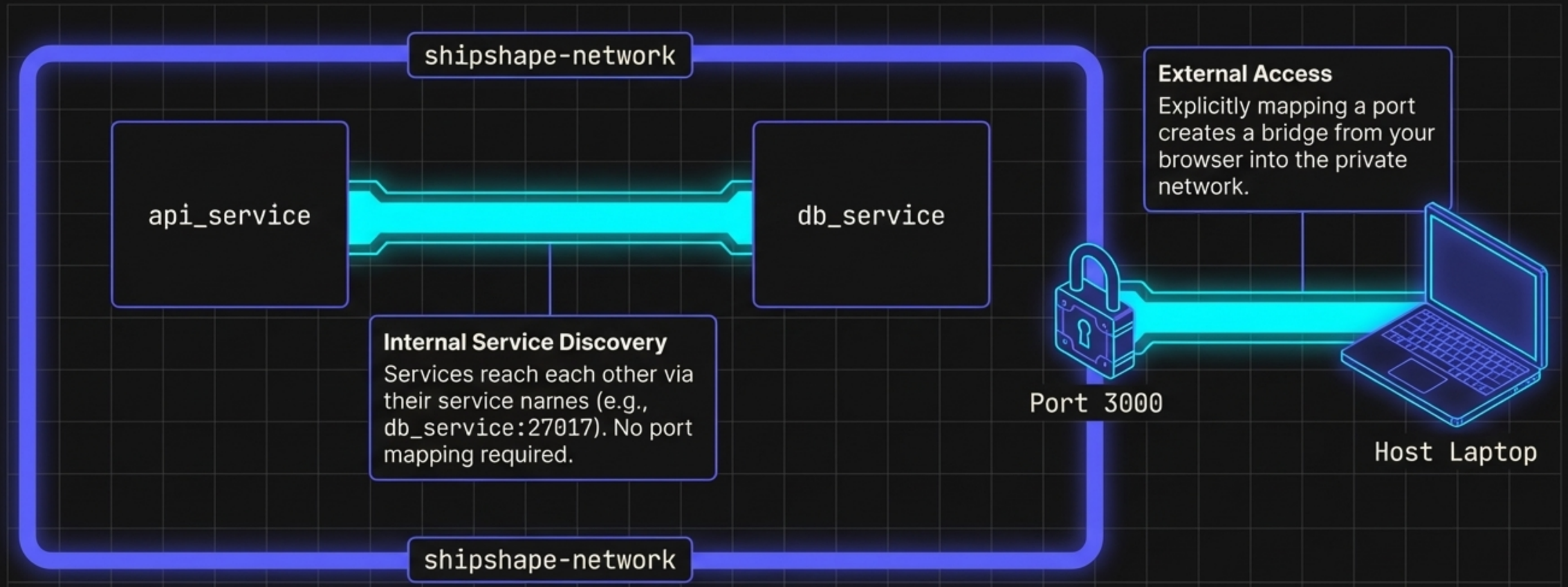
Maps port 3000 on your host machine to port 3000 inside the container.

The Configuration Bridge



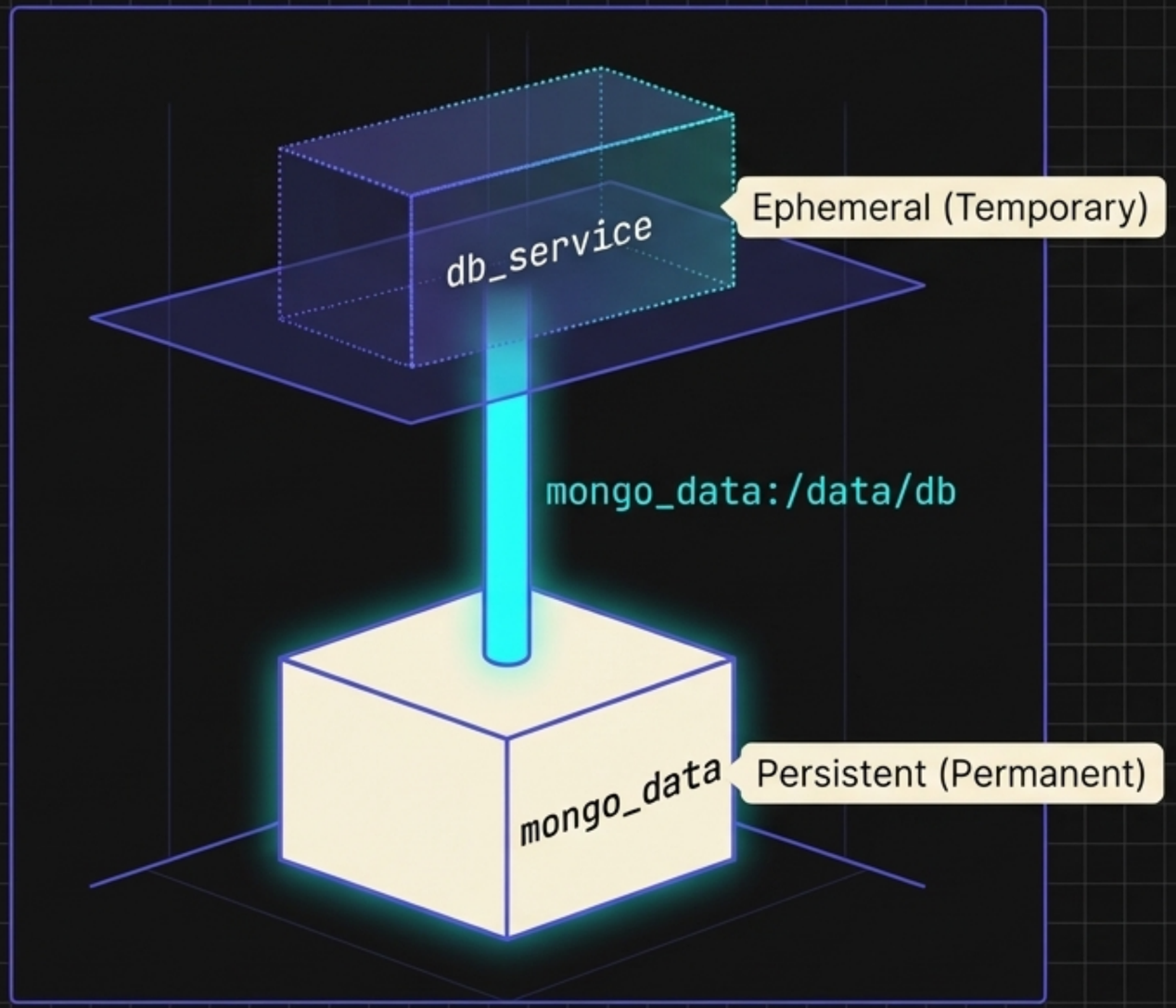
Compose automatically looks for a .env file next to your compose.yaml.
You don’t hardcode secrets; you let Compose inject them at runtime.

Exposure vs. Publication



By default, Compose puts all your services on a shared private network. The database remains hidden from the outside world, but perfectly accessible to your API.

The Volume Anchor



A container is **not a long-term** home for stateful data. A named volume decouples your **database files from the container's lifecycle**, allowing the data to **survive teardowns and restarts**.

Crossing the Boundary: Seeding Data

Step 1: docker cp


```
docker cp projects.json $(docker compose ps
-q db_service):/projects.json
```

Step 2: mongoimport

db_service


```
docker compose exec db_service mongoimport...
```

Working with containerized databases requires container-aware tools.
We copy the file in, then step inside to execute the import.

Choosing Your Reset Level

`docker compose stop`

Halts the processes. Leaves containers and networks perfectly intact. Like turning off the ship's engine at the dock.

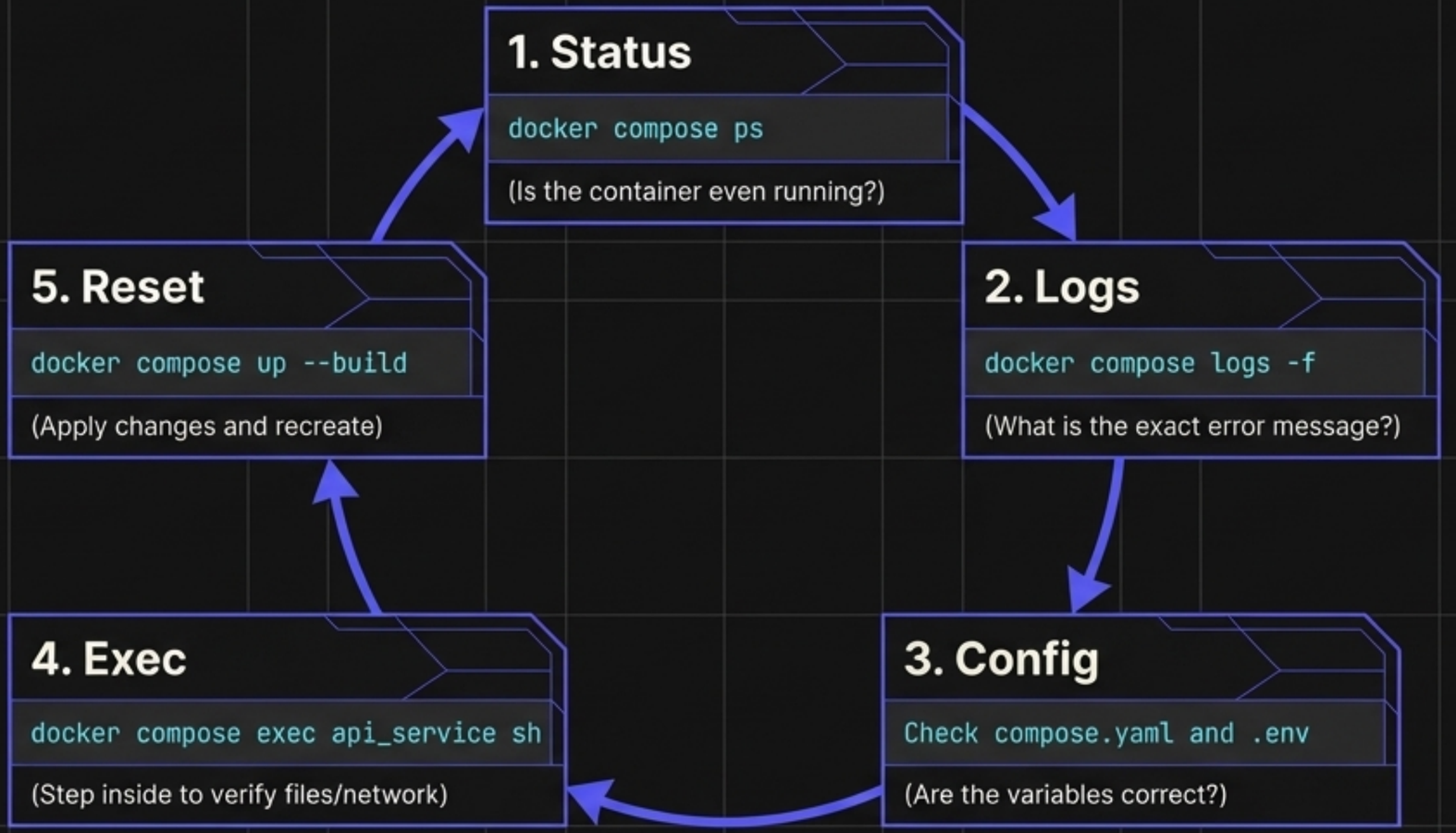
`docker compose down`

The definitive reset. Stops and removes containers and the project network. Clean slate for code/config changes.

`docker compose down -v`

The nuclear option. Removes containers, networks, AND named volumes. Deletes your database data. Use with caution.

The Dev Workflow Loop



Resist the urge to mash commands and hope for a miracle. Debug the system that is actually running, not the one you think is running.

Dockyard Hazard Map

Symptom	Cause	Fix
`ECONNREFUSED` using `localhost`.	API is looking inside its own container.	Change the hostname to `"db_service"`.
Code changes aren't showing up.	Running container is using an old, cached image.	Run `"docker compose up --build"`
'Port is already allocated'.	Another host app is using the port (e.g., 3000).	Stop conflicting process or or remap in YAML.
DB data disappears after teardown.	No named volume attached, or -v was used.	Add `"mongo_data:/data/db"` to YAML.

Tactical Command Syntax

Lifecycle

`> docker compose up -d`	(Start stack in background)
`> docker compose up --build`	(Force image rebuild before starting)
`> docker compose down`	(Tear down containers & networks)
`> docker compose down`	(Tear down containers & networks)

Inspection & Interaction

`> docker compose ps`	(Show status of stack services)
`> docker compose logs -f`	(Tail live log output)
`> docker compose exec db_service mongosh`	(Run interactive shell inside a service)

Solo's Pro-Notes

Localhost Means Self

Inside a container, `localhost` points back to that same container, not the host machine or a sibling service.

Volumes Are About State

Containers are disposable; database data is not. Stateful services need named volumes.

Compose is Declarative

Change the `compose.yaml` file when the setup needs to change. Stop relying on one-off manual commands.

The Bleeding Edge Can Bleed

Use pinned image tags (like `mongo:8.0`) instead of `latest` for stability.

p.s., keep learning!