**Container Lifecycle & Health Checks**

**Restart policies**

In Docker, the restart policy controls whether containers automatically restart under certain conditions.

| **Policy** | **Description** |
| --- | --- |
| no (default) | Container won’t restart automatically. |
| always | Container will always restart, no matter why it exited:   * if it crashed with some exit code, it restarts * if Docker demon restarts, it restarts * if host system reboots, it restarts   Use:   * For essential services that must always run, like databases or API gateways. |
| unless-stopped | * Behaves like always, **except** when we stop the container manually (docker stop). * After a manual stop, it won’t restart on daemon or system restarts **until we explicitly start it again** * If the system restarts, it’ll come back up. * But if we stopped it manually, Docker would **respect your decision** and not restart it.   **Use:**   * For apps we want to be resilient, but with some manual control. |
| on-failure[:max] | * Restarts the container **only** when it exits with a **non-zero exit code** (meaning: failed). * We can limit it: e.g., on-failure:3 means retry max **3 times**.   **Use:**   * For **lightweight, stateless apps** or jobs that can fail occasionally but don’t need infinite retries. |

A screenshot of a computer program

AI-generated content may be incorrect.

**Note: Somehow, restart: unless-stopped is not working in my case in Docker for Desktop.**

**We can also use in the docker run command:**

***docker run -d --restart=always --name db postgres:15***

**🔄 Restart policy applies to the container, not the process inside it**

**🧩 Example:**  
If your Flask app encounters a Python exception but doesn’t call sys.exit(1), the container will **remain running**, and Docker will **assume it's healthy**, even though it’s broken.

✅ **Solution**: Make sure your application **fails explicitly** (exit non-zero) on critical errors — otherwise, Docker won't restart it.

**🔄 restart=always can create "restart loops" if misused**

If your container **keeps crashing** (bad code, missing env vars, or DB not ready), always will keep restarting it **forever** — clogging logs and CPU.

✅ **Tip**: Use on-failure[:N] in development/testing environments to avoid this and only use always in stable production setups.

**🔄 restart doesn’t affect docker-compose down**

When we run: docker-compose down, all containers stop and **won’t restart**, regardless of their restart policy. This is an intentional, clean shutdown.

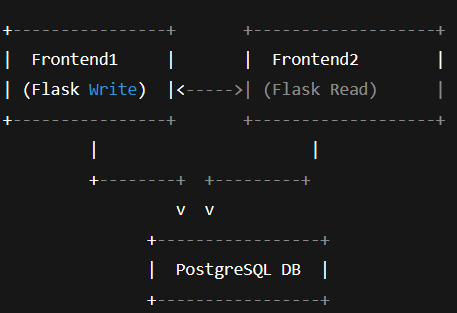
**🐳 Docker HEALTHCHECK**

The HEALTHCHECK instruction in a Dockerfile (or Docker compose) tells Docker how to test if a container is still working as expected. If the check fails continuously, Docker marks the container as unhealthy, which is useful in multi-container systems to detect and react to failure.

✅ Syntax of HEALTHCHECK

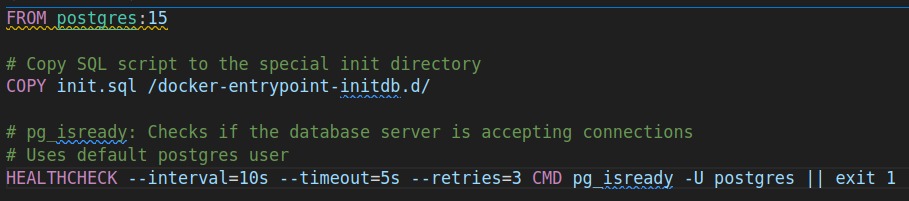
HEALTHCHECK [OPTIONS] CMD command

* CMD: The command to run to check container health.
* --interval: Time between checks (default: 30s).
* --timeout: Time before a check is considered failed (default: 30s).
* --start-period: Time to wait after container starts (default: 0s).
* --retries: Consecutive failures needed to mark container unhealthy (default: 3).



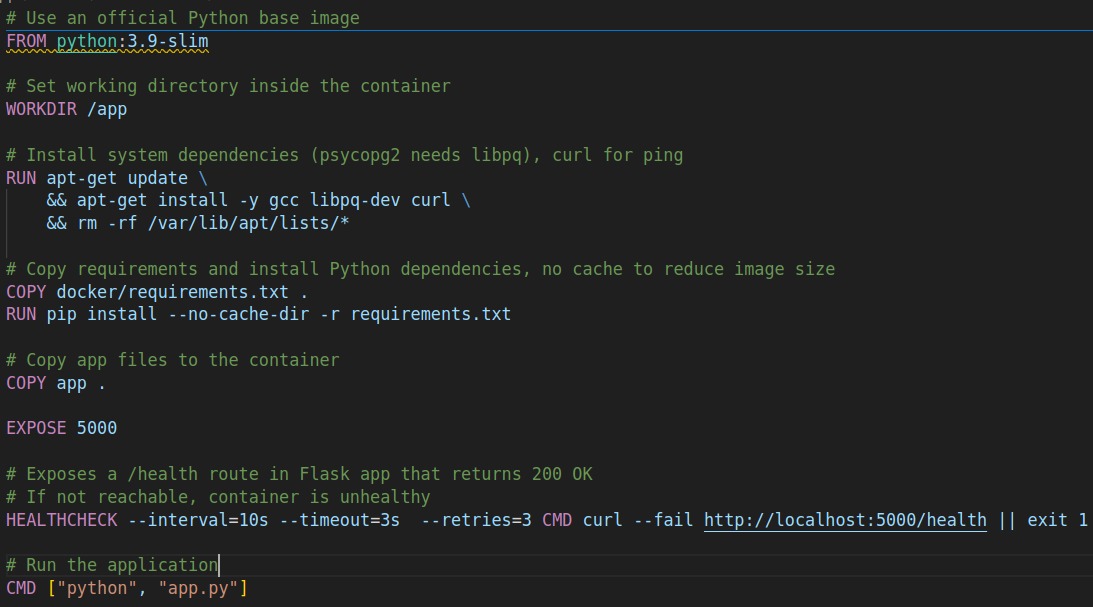
**🔧 Dockerfiles with HEALTHCHECK**

**🧩 DB (PostgreSQL)**



* pg\_isready: Checks if the database server is accepting connections.
* Uses default postgres user.

🧩 Flask App – Writer’s Dockerfile



* Exposes a /health route in Flask that returns 200 OK.
* If not reachable, container is unhealthy.

🧩 Flask App – Writer’s app.py

A computer code on a black background

AI-generated content may be incorrect.

🧩 Docker Compose

A screen shot of a computer

AI-generated content may be incorrect.

**🔍 HEALTHCHECK Status**

* We can check container health with
  + docker ps   
    We’ll see healthy, unhealthy, or starting under the STATUS column.
  + docker logs <container name or id>
  + Gives us latest health check status:  
    docker inspect <container name or id>| grep -A 10 '"Health":'
  + Gives us few latest health check status:  
    docker inspect --format='{{json .State.Health}}' <container name or id>| jq

🔍 Why HEALTHCHECK is defined in the **Dockerfile**, not in docker-compose.yml?

**✅ 1. Dockerfile is image-centric**

* The HEALTHCHECK instruction in a Dockerfile **becomes part of the image**.
* This means any container started from that image will have the healthcheck logic **pre-baked**.
* This is useful when distributing or reusing images (e.g., in production environments or CI/CD).

**🔄 2. docker-compose focuses on runtime**

* docker-compose.yml is primarily about **orchestrating containers**—what to run, environment variables, volumes, networks, etc.

**✅ We *can* use healthcheck in docker-compose.yml (Compose v3+)**

Although it's **less common**, we can override or define healthchecks directly in docker-compose.yml:



This is useful when:

* We don’t want to modify the Dockerfile.
* We want environment-specific overrides (e.g., dev vs prod).

For production-ready setups, it’s common to define base healthchecks in the Dockerfile, and override or fine-tune them in docker-compose.yml when needed.

**Practise:** Run the Writer-app and Reader-app containers after the DB-app container is ready and ready for accepting the connection.