**Python-FastAPI docker container project**

260210 - 11

260210-11

Initial Docker container implementation

I want to provide me with docker-compose file for building a docker container for setting up a proper Python and FastAPI test environment.

Below is a solid “2-layer” setup for a FastAPI test environment using **docker-compose**:

* **Layer 1 (base image)**: installs OS deps + Python deps once (cached well).
* **Layer 2 (app/test image)**: copies your source and runs your tests (or starts the app).

This is the typical pattern: a Dockerfile.base that rarely changes + a Dockerfile that changes often.

**Folder layout (suggested)**

.

├─ docker-compose.yml

├─ Dockerfile.base

├─ Dockerfile

├─ requirements.txt

├─ requirements-dev.txt

├─ app/

│ ├─ main.py

│ └─ ...

└─ tests/

└─ test\_smoke.py

**docker-compose.yml**

services:

api:

build:

context: .

dockerfile: Dockerfile

target: runtime

args:

BASE\_IMAGE: fastapi-test-base:py312

image: fastapi-test-app:dev

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

APP\_ENV: "test"

ports:

- "8000:8000"

volumes:

- ./:/app:delegated

command: >

uvicorn app.main:app

--host 0.0.0.0

--port 8000

--reload

test:

build:

context: .

dockerfile: Dockerfile

target: test

args:

BASE\_IMAGE: fastapi-test-base:py312

image: fastapi-test-app:test

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

APP\_ENV: "test"

volumes:

- ./:/app:delegated

command: >

pytest -q

This gives you:

* docker compose up api for interactive dev
* docker compose run --rm test for tests

**Dockerfile.base (Layer 1)**

# syntax=docker/dockerfile:1.6

FROM python:3.12-slim AS base

ENV PYTHONDONTWRITEBYTECODE=1 \

PYTHONUNBUFFERED=1 \

PIP\_DISABLE\_PIP\_VERSION\_CHECK=1 \

PIP\_NO\_CACHE\_DIR=1

WORKDIR /app

# OS deps you often need for FastAPI testing stacks (adjust as needed)

RUN apt-get update && apt-get install -y --no-install-recommends \

build-essential \

curl \

&& rm -rf /var/lib/apt/lists/\*

# Install dependencies into a virtualenv to keep things clean & portable

RUN python -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Only copy requirement files for best caching

COPY requirements.txt requirements-dev.txt ./

RUN pip install --upgrade pip setuptools wheel \

&& pip install -r requirements.txt \

&& pip install -r requirements-dev.txt

# Optional: create a non-root user for runtime safety

RUN useradd -m appuser

Build it once:

docker build -f Dockerfile.base -t fastapi-test-base:py312 .

**Dockerfile (Layer 2)**

# syntax=docker/dockerfile:1.6

ARG BASE\_IMAGE=fastapi-test-base:py312

FROM ${BASE\_IMAGE} AS runtime

WORKDIR /app

ENV PATH="/opt/venv/bin:$PATH"

# Copy only application code (compose also mounts volume for live edits)

COPY app ./app

USER appuser

EXPOSE 8000

FROM ${BASE\_IMAGE} AS test

WORKDIR /app

ENV PATH="/opt/venv/bin:$PATH"

# Copy app + tests for CI-style test runs (even without volumes)

COPY app ./app

COPY tests ./tests

COPY pyproject.toml\* pytest.ini\* .ruff.toml\* ./

USER appuser

**Example requirements**

**requirements.txt**

fastapi

uvicorn[standard]

**requirements-dev.txt**

pytest

pytest-asyncio

httpx

**Run commands**

# 1) Build base layer (cached)

docker build -f Dockerfile.base -t fastapi-test-base:py312 .

# 2) Run API (dev)

docker compose up --build api

# 3) Run tests

docker compose run --rm test

**Notes / options you might want**

* If you use **Postgres/Redis** for tests, add them as services and set env vars in test.
* If you want tests to run against the *running* api service, you can make test depend on api and hit http://api:8000 (network alias).
* If you want maximum cache efficiency, switch to pip wheels or use BuildKit cache mounts.

If you tell me whether your tests need a database (and which one), I’ll add the extra service(s) plus a “wait-for” strategy so the test container doesn’t start before dependencies are ready.

I prefer the 1st layer to be based on Ubuntu and including both Python and FastAPI. Moreover, I also want more OS-based tools to be included (installed) as well, like:

wget

curl

net-tools

lsof

software-properties-common

unzip

p7zip-full

mc

etc.,

The 2nd layer should be based on MariaDB

Both layers should expose accessible ports for the host system, as well as to share a common shared folder with the host system

Got it — what you’re describing is best modeled as **two services in docker-compose** (an Ubuntu-based app/test container + a MariaDB container). Docker “layers” are inside a single image, but for Ubuntu + MariaDB you almost always want **separate containers** that share a network + volumes.

Below is a working setup that matches your requirements:

* **Service 1 (ubuntu-python)**: Ubuntu base image, installs **Python + FastAPI** plus your requested OS tools (wget curl net-tools lsof software-properties-common unzip p7zip-full mc etc.).
* **Service 2 (mariadb)**: official MariaDB image.
* Both expose **host-accessible ports**
* Both mount a **shared host folder** to the same in-container path (/shared)

## 1) docker-compose.yml

services:

ubuntu-python:

build:

context: .

dockerfile: Dockerfile.ubuntu

container\_name: ubuntu-python

working\_dir: /workspace

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

# DB connectivity for your app/tests

DB\_HOST: mariadb

DB\_PORT: "3306"

DB\_NAME: appdb

DB\_USER: appuser

DB\_PASSWORD: apppass

ports:

- "8000:8000" # FastAPI/uvicorn

volumes:

- ./workspace:/workspace

- ./shared:/shared

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"

mariadb:

image: mariadb:11

container\_name: mariadb

environment:

MARIADB\_ROOT\_PASSWORD: rootpass

MARIADB\_DATABASE: appdb

MARIADB\_USER: appuser

MARIADB\_PASSWORD: apppass

ports:

- "3306:3306" # MariaDB exposed to host

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared

# Optional: auto-run SQL scripts on first init

# - ./db/init:/docker-entrypoint-initdb.d:ro

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

volumes:

mariadb\_data:

### What this gives you

* Host can reach FastAPI at: http://localhost:8000
* Host can reach MariaDB at: localhost:3306
* Shared host folder is available in **both** containers at: /shared

## 2) Dockerfile.ubuntu (Ubuntu + Python + FastAPI + tools)

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

# OS tools + Python tooling

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

# (Optional) Convenience symlinks

RUN ln -sf /usr/bin/python3 /usr/local/bin/python \

&& ln -sf /usr/bin/pip3 /usr/local/bin/pip

# Upgrade pip and install FastAPI + typical dev/test tools

RUN python -m pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

# Default shell

CMD ["bash"]

This image already includes FastAPI and uvicorn. If you want it to install your project’s dependencies instead, I can switch it to copy requirements.txt and install those (better for real projects).

## 3) Minimal FastAPI app (example)

Create: workspace/app/main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/health")

def health():

return {"ok": True}

## 4) Run it

docker compose up --build

## Notes / common improvements (optional but useful)

* **If you want the Ubuntu container to run tests instead of uvicorn**, add another service test: that uses the same image and runs pytest.
* **If you want to connect to MariaDB from the Ubuntu container**, use host mariadb (service name), port 3306 (already wired via env vars).
* **If you want extra ports on the Ubuntu container** (debugger, admin tools, etc.), add more under ports: like "5678:5678".

If you want, I can also add:

* a **third service** like adminer / phpmyadmin for DB browsing (host port exposed),
* a test service that runs pytest against the same DB,
* a Makefile with common commands (up, down, test, logs).

I think adding a 3rd layer for phpMyAdmin would be fine.

Absolutely — adding phpMyAdmin as a 3rd service is straightforward. Here’s an updated docker-compose.yml that includes **Ubuntu+FastAPI**, **MariaDB**, and **phpMyAdmin**, with host-accessible ports and a shared folder mounted into all containers.

## Updated docker-compose.yml (with phpMyAdmin)

services:

ubuntu-python:

build:

context: .

dockerfile: Dockerfile.ubuntu

container\_name: ubuntu-python

working\_dir: /workspace

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

DB\_HOST: mariadb

DB\_PORT: "3306"

DB\_NAME: appdb

DB\_USER: appuser

DB\_PASSWORD: apppass

ports:

- "8000:8000" # FastAPI

volumes:

- ./workspace:/workspace

- ./shared:/shared

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"

mariadb:

image: mariadb:11

container\_name: mariadb

environment:

MARIADB\_ROOT\_PASSWORD: rootpass

MARIADB\_DATABASE: appdb

MARIADB\_USER: appuser

MARIADB\_PASSWORD: apppass

ports:

- "3306:3306" # MariaDB to host

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared

# Optional init scripts:

# - ./db/init:/docker-entrypoint-initdb.d:ro

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

phpmyadmin:

image: phpmyadmin:latest

container\_name: phpmyadmin

depends\_on:

mariadb:

condition: service\_healthy

environment:

PMA\_HOST: mariadb

PMA\_PORT: 3306

PMA\_ARBITRARY: 0

# Optional: if you want it to default to a user

# PMA\_USER: appuser

# PMA\_PASSWORD: apppass

UPLOAD\_LIMIT: 256M

ports:

- "8080:80" # phpMyAdmin to host

volumes:

- ./shared:/shared

volumes:

mariadb\_data:

## Dockerfile.ubuntu (same as before, included here for completeness)

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

RUN ln -sf /usr/bin/python3 /usr/local/bin/python \

&& ln -sf /usr/bin/pip3 /usr/local/bin/pip

RUN python -m pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

CMD ["bash"]

## How to use

docker compose up --build

Then:

* FastAPI: http://localhost:8000
* phpMyAdmin: http://localhost:8080
* MariaDB: localhost:3306

phpMyAdmin login:

* Server: mariadb (or leave blank if it auto-fills)
* Username: appuser (or root)
* Password: apppass (or rootpass)

### Optional hardening (recommended)

If you don’t actually need MariaDB exposed to the host (you only use phpMyAdmin + app), remove - "3306:3306" for better safety.

If you want, I can also add a **.env** file support so you don’t keep passwords in the compose file.

Yes, please add also a .env file support so I wouldn’t keep passwords in the compose file

Perfect 👍 — using a .env file is the cleanest way to keep passwords and ports out of the compose file.

Below is a **fully updated setup** with:

✅ Ubuntu + Python + FastAPI container  
✅ MariaDB container  
✅ phpMyAdmin container  
✅ Shared /shared folder  
✅ Host-exposed ports  
✅ Centralized .env configuration

# 1️⃣ .env file (NEW)

Create a file named:

.env

Example:

# =========================

# Database

# =========================

DB\_HOST=mariadb

DB\_PORT=3306

DB\_NAME=appdb

DB\_USER=appuser

DB\_PASSWORD=apppass

DB\_ROOT\_PASSWORD=rootpass

# =========================

# Ports (host side)

# =========================

FASTAPI\_PORT=8000

PHPMYADMIN\_PORT=8080

MARIADB\_PORT=3306

You can change values here without touching compose.

# 2️⃣ Updated docker-compose.yml

services:

ubuntu-python:

build:

context: .

dockerfile: Dockerfile.ubuntu

container\_name: ubuntu-python

working\_dir: /workspace

env\_file:

- .env

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

ports:

- "${FASTAPI\_PORT}:8000"

volumes:

- ./workspace:/workspace

- ./shared:/shared

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"

mariadb:

image: mariadb:11

container\_name: mariadb

env\_file:

- .env

environment:

MARIADB\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD}

MARIADB\_DATABASE: ${DB\_NAME}

MARIADB\_USER: ${DB\_USER}

MARIADB\_PASSWORD: ${DB\_PASSWORD}

ports:

- "${MARIADB\_PORT}:3306"

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

phpmyadmin:

image: phpmyadmin:latest

container\_name: phpmyadmin

env\_file:

- .env

depends\_on:

mariadb:

condition: service\_healthy

environment:

PMA\_HOST: ${DB\_HOST}

PMA\_PORT: ${DB\_PORT}

UPLOAD\_LIMIT: 256M

ports:

- "${PHPMYADMIN\_PORT}:80"

volumes:

- ./shared:/shared

volumes:

mariadb\_data:

# 3️⃣ Dockerfile.ubuntu (unchanged, included for completeness)

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

RUN ln -sf /usr/bin/python3 /usr/local/bin/python \

&& ln -sf /usr/bin/pip3 /usr/local/bin/pip

RUN python -m pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

CMD ["bash"]

# 4️⃣ How to run

docker compose up --build

Open:

* FastAPI → http://localhost:${FASTAPI\_PORT}
* phpMyAdmin → http://localhost:${PHPMYADMIN\_PORT}

Login in phpMyAdmin:

Server: mariadb

User: appuser

Password: apppass

# ⭐️ Pro tip (what most devs miss)

Docker Compose **automatically loads .env** — you don’t need to export variables manually.

Also:

.env → used by compose for substitution

env\_file: → injected into containers

These are **two different layers**, and you’re now using both correctly 👍

If you want, I can give you an **“advanced dev version”** next that adds:

✅ Adminer + phpMyAdmin toggle  
✅ FastAPI auto-restart + debugger port  
✅ Database auto-migration startup script  
✅ Test container (pytest) sharing same .env

That setup is extremely nice for FastAPI + DB testing.

Please add git to the 1st layer. Then, provide me with a folder structure of the project and a detailed description of each set-up file being used.

Sure — git is already included in the Ubuntu layer I gave (git is in the apt list), but I’ll keep it explicit and then give you a clean project folder structure + what each file does.

## Project folder structure

fastapi-mariadb-env/

├─ docker-compose.yml

├─ Docworkspacekerfile.ubuntu

├─ .env

├─ .gitignore

├─ workspace/

│ ├─ app/

│ │ ├─ \_\_init\_\_.py

│ │ └─ main.py

│ ├─ tests/

│ │ └─ test\_health.py

│ └─ README.md

├─ shared/

│ └─ .keep

└─ db/

└─ init/

└─ 001\_create\_schema.sql (optional)

* workspace/ = your FastAPI code + tests (mounted into the Ubuntu container)
* shared/ = a common folder mounted into **all** containers at /shared
* db/init/ = optional SQL scripts that MariaDB runs only on first initialization

## 1) Dockerfile.ubuntu (Ubuntu + tools + Python + FastAPI + git)

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

RUN apt-get update && apt-get install -y --no-install-recommends \

# requested OS tools

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

# commonly useful

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

# python

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

# Convenience symlinks (optional)

RUN ln -sf /usr/bin/python3 /usr/local/bin/python \

&& ln -sf /usr/bin/pip3 /usr/local/bin/pip

# Install FastAPI + common dev/test deps (baked into the image)

RUN python -m pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

CMD ["bash"]

### What it does

* Builds an Ubuntu-based image for your dev/test environment.
* Installs your OS tools (including **git**) so the container is useful for real dev work.
* Installs Python packages so the container can run FastAPI + tests immediately.

## 2) .env (secrets + ports, not committed)

# =========================

# Database

# =========================

DB\_HOST=mariadb

DB\_PORT=3306

DB\_NAME=appdb

DB\_USER=appuser

DB\_PASSWORD=apppass

DB\_ROOT\_PASSWORD=rootpass

# =========================

# Ports (host side)

# =========================

FASTAPI\_PORT=8000

PHPMYADMIN\_PORT=8080

MARIADB\_PORT=3306

### What it does

* Keeps passwords and configuration out of docker-compose.yml.
* Provides variables for:
  + MariaDB init (root/user/password/db)
  + host port mappings (so you can change ports without editing compose)

## 3) docker-compose.yml (3 services + shared mounts + healthchecks)

services:

ubuntu-python:

build:

context: .

dockerfile: Dockerfile.ubuntu

container\_name: ubuntu-python

working\_dir: /workspace

env\_file:

- .env

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

ports:

- "${FASTAPI\_PORT}:8000"

volumes:

- ./workspace:/workspace

- ./shared:/shared

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"

mariadb:

image: mariadb:11

container\_name: mariadb

env\_file:

- .env

environment:

MARIADB\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD}

MARIADB\_DATABASE: ${DB\_NAME}

MARIADB\_USER: ${DB\_USER}

MARIADB\_PASSWORD: ${DB\_PASSWORD}

ports:

- "${MARIADB\_PORT}:3306"

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared

# Optional: auto-run SQL scripts only on first init

# - ./db/init:/docker-entrypoint-initdb.d:ro

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

phpmyadmin:

image: phpmyadmin:latest

container\_name: phpmyadmin

env\_file:

- .env

depends\_on:

mariadb:

condition: service\_healthy

environment:

PMA\_HOST: ${DB\_HOST}

PMA\_PORT: ${DB\_PORT}

UPLOAD\_LIMIT: 256M

ports:

- "${PHPMYADMIN\_PORT}:80"

volumes:

- ./shared:/shared

volumes:

mariadb\_data:

### What it does

* **ubuntu-python**: runs FastAPI via uvicorn; mounts your source code for live reload.
* **mariadb**: persists DB data in a named Docker volume (mariadb\_data) so DB survives restarts.
* **phpmyadmin**: connects to MariaDB on the internal compose network; exposed on host port 8080 (or whatever you set).

### Key behaviors

* depends\_on + service\_healthy prevents app/phpMyAdmin from starting before DB is ready.
* ./shared:/shared appears inside all containers — a simple way to exchange files across services and host.

## 4) workspace/app/main.py (minimal FastAPI app)

from fastapi import FastAPI

app = FastAPI()

@app.get("/health")

def health():

return {"ok": True}

### What it does

* Provides a simple endpoint you can test quickly:
  + GET http://localhost:8000/health

## 5) workspace/tests/test\_health.py (basic test)

from fastapi.testclient import TestClient

from app.main import app

client = TestClient(app)

def test\_health():

r = client.get("/health")

assert r.status\_code == 200

assert r.json() == {"ok": True}

### What it does

* Verifies the API is running and returning expected JSON.

(Note: *TestClient* relies on Starlette; FastAPI pulls it in. If you later want async tests with *httpx.AsyncClient*, you can.)

## 6) db/init/001\_create\_schema.sql (optional)

CREATE TABLE IF NOT EXISTS example (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

### What it does

* If you uncomment the ./db/init:/docker-entrypoint-initdb.d:ro mount in compose, MariaDB will run these scripts **only when the DB volume is created the first time**.
* Useful for initial schemas, seed data, etc.

## 7) .gitignore (recommended)

.env

\_\_pycache\_\_/

\*.pyc

.mypy\_cache/

.pytest\_cache/

.vscode/

.idea/

shared/\*

!shared/.keep

### What it does

* Ensures secrets (.env) aren’t committed.
* Keeps local caches and editor files out of git.
* Keeps the shared/ directory in the repo via .keep but ignores its contents.

## How you run it

docker compose up --build

* FastAPI: http://localhost:${FASTAPI\_PORT}/health
* phpMyAdmin: http://localhost:${PHPMYADMIN\_PORT}
  + server: mariadb
  + user/pass from .env

If you want, I can also add a **test compose service** that runs pytest inside the Ubuntu container (using the same .env), so you can do:

docker compose run --rm test

Initial files content and settings

**➜ fastapi-mariadb-env** tree -L 3

.

|-- Dockerfile.ubuntu

|-- README.md

|-- db

| `-- init

|-- docker-compose.yml

|-- myDocs

| |-- Python-FastAPI\ docker\ container\ projectdocx.docx

| `-- ~$thon-FastAPI\ docker\ container\ projectdocx.docx

|-- shared

`-- workspace

|-- app

| |-- \_\_init\_\_.py

| `-- main.py

`-- tests

`-- test\_health.py

8 directories, 8 files

**➜ fastapi-mariadb-env**

docker-compose.yml

# docker-compose.yml - # Docker Compose file for FastAPI + MariaDB + phpMyAdmin

#

# What it does

# ubuntu-python: runs FastAPI via uvicorn; mounts your source code for live reload.

# mariadb: persists DB data in a named Docker volume (mariadb\_data) so DB survives restarts.

# phpmyadmin: connects to MariaDB on the internal compose network; exposed on host port 8080 (or whatever you set).

# Key behaviors

# depends\_on + service\_healthy prevents app/phpMyAdmin from starting before DB is ready.

# ./shared:/shared appears inside all containers — a simple way to exchange files across services and host.

#

# Version 1 - 260210

services:

ubuntu-python:

build:

context: .

dockerfile: Dockerfile.ubuntu

container\_name: ubuntu-python

working\_dir: /workspace

env\_file:

- .env

environment:

PYTHONDONTWRITEBYTECODE: "1"

PYTHONUNBUFFERED: "1"

TZ: "Europe/Stockholm"

ports:

- "${FASTAPI\_PORT}:8000"

volumes:

# macOS bind mounts: delegated improves container-side performance

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload"

mariadb:

image: mariadb:11

container\_name: mariadb

env\_file:

- .env

environment:

TZ: "Europe/Stockholm"

MARIADB\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD}

MARIADB\_DATABASE: ${DB\_NAME}

MARIADB\_USER: ${DB\_USER}

MARIADB\_PASSWORD: ${DB\_PASSWORD}

ports:

# host:container (container stays 3306, host defaults to 3307)

- "${MARIADB\_PORT}:3306"

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared:delegated

# Optional init scripts (first-time DB init only):

# - ./db/init:/docker-entrypoint-initdb.d:ro

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

phpmyadmin:

image: phpmyadmin:latest

platform: linux/arm64/v8

container\_name: phpmyadmin

env\_file:

- .env

depends\_on:

mariadb:

condition: service\_healthy

environment:

TZ: "Europe/Stockholm"

PMA\_HOST: ${DB\_HOST}

PMA\_PORT: ${DB\_PORT}

PMA\_ARBITRARY: 0

UPLOAD\_LIMIT: 256M

ports:

- "${PHPMYADMIN\_PORT}:80"

volumes:

- ./shared:/shared:delegated

volumes:

mariadb\_data:

Dockerfile.ubuntu

# Dockerfile.ubuntu - Ubuntu + tools + Python + FastAPI + git

#

# What it does

# Builds an Ubuntu-based image for your dev/test environment.

# Installs your OS tools (including git) so the container is useful for real dev work.

# Installs Python packages so the container can run FastAPI + tests immediately.

#

# vesion 1 - 260210

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

# OS tools + Python

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

# Create a virtual environment for Python packages (avoids PEP 668 issues)

RUN python3 -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Upgrade pip inside venv and install dependencies into venv

RUN pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

CMD ["bash"]

.env

# .env file - This file contains environment variables for Docker Compose.

#

# What it does

# Keeps passwords and configuration out of docker-compose.yml.

# Provides variables for:

# MariaDB init (root/user/password/db)

# host port mappings (so you can change ports without editing compose)

#

# version 1 - 260210

# =========================

# Database

# =========================

DB\_HOST=mariadb

DB\_PORT=3306

DB\_NAME=appdb

DB\_USER=appuser

DB\_PASSWORD=apppass

DB\_ROOT\_PASSWORD=rootpass

# =========================

# Ports (host side)

# =========================

FASTAPI\_PORT=8000

PHPMYADMIN\_PORT=8080

# On macOS it's common to have MySQL/MariaDB already using 3306.

# Use 3307 on the host to avoid conflicts.

MARIADB\_PORT=3307

workspace/tests/**test\_health.py**

# workspace/tests/test\_health.py

#

# This is a very basic test, but it's a good starting point. You can expand it later with more complex tests as needed.

#

# What it does

# Verifies the API is running and returning expected JSON.

# (Note: TestClient relies on Starlette; FastAPI pulls it in. If you later want async tests with httpx.AsyncClient, you can.)

#

#version 1 - 260210

from fastapi.testclient import TestClient

from app.main import app

client = TestClient(app)

def test\_health():

r = client.get("/health")

assert r.status\_code == 200

assert r.json() == {"ok": True}

workspace/app/**main.py**

# workspace/app/main.py

#

# minimal FastAPI app

#

# What it does

# Provides a simple endpoint you can test quickly:

# GET http://localhost:8000/health

#

# version 1 - 260210

from fastapi import FastAPI

app = FastAPI()

@app.get("/health")

def health():

return {"ok": True}

Docker compose commands

docker compose up --build

docker compose -f docker-compose.yml up --build

docker compose build --no-cache ubuntu-python

docker compose up

docker compose pull phpmyadmin

docker compose up --force-recreate

docker compose down

docker compose up -d

docker image inspect mariadb:11 --format '{{.Os}}/{{.Architecture}}'

docker compose exec ubuntu-python bash

docker compose ps

After we have built the containers and run the **docker-compose.yml** file

docker compose ps

**➜ fastapi-mariadb-env** docker compose ps

NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS

mariadb mariadb:11 "docker-entrypoint.s…" mariadb 2 minutes ago Up 2 minutes (healthy) 0.0.0.0:3307->3306/tcp

phpmyadmin phpmyadmin:latest "/docker-entrypoint.…" phpmyadmin 2 minutes ago Up 2 minutes 0.0.0.0:8080->80/tcp

ubuntu-python fastapi-mariadb-env-ubuntu-python "bash -lc 'uvicorn a…" ubuntu-python 2 minutes ago Up 2 minutes 0.0.0.0:8000->8000/tcp

**➜ fastapi-mariadb-env**

or

docker ps --format 'table {{.Names}}\t{{.Image}}\t{{.Status}}'

**➜ fastapi-mariadb-env** docker ps --format 'table {{.Names}}\t{{.Image}}\t{{.Status}}'

NAMES IMAGE STATUS

ubuntu-python fastapi-mariadb-env-ubuntu-python Up 18 hours

phpmyadmin phpmyadmin:latest Up 18 hours

mariadb mariadb:11 Up 18 hours (healthy)

**➜ fastapi-mariadb-env**

First testing of the Implementation

Here’s a **clear, step-by-step way to test the whole setup** on your macOS host, from “containers are up” to “app ↔ DB ↔ phpMyAdmin all talk to each other”.

## 1️. Bring everything up cleanly

From the project root (where docker-compose.yml lives):

docker compose up -d

Check that all containers are running:

docker compose ps

You should see something like:

NAME STATUS

ubuntu-python Up

mariadb Up (healthy)

phpmyadmin Up

If mariadb is not healthy, wait a few seconds and re-check.

## 2️. Test FastAPI from your Mac (host → container)

Open a browser or use curl:

### Browser

http://localhost:8000/health

Expected response:

{"ok": true}

### Curl

curl http://localhost:8000/health

If this works, you’ve confirmed:

* Port mapping works
* FastAPI is running
* Code volume mount is correct

## 3️. Test phpMyAdmin (host → DB container)

Open:

http://localhost:8080

Login with values from .env:

| **Field** | **Value** |
| --- | --- |
| Server | mariadb |
| User | appuser |
| Pass | apppass |

If login succeeds:

* MariaDB container is reachable
* phpMyAdmin networking is correct
* DB credentials are correct

## 4️. Test container-to-container DB access (FastAPI → MariaDB)

### Exec into the Ubuntu container

docker compose exec ubuntu-python bash

Inside the container, connect to MariaDB:

mysql -h mariadb -u appuser -p

Enter password from .env.

Then run:

SHOW DATABASES;

USE appdb;

SHOW TABLES;

Exit:

exit

This confirms:

* Internal Docker DNS works (mariadb hostname)
* Credentials are valid
* DB is reachable from the app container

## 5️. Optional: test DB access via Python (real app scenario)

Inside ubuntu-python container:

python

Then:

import **pymysql**

conn = pymysql.connect(

host="mariadb",

user="appuser",

password="apppass",

database="appdb"

)

with conn.cursor() as cur:

cur.execute("SELECT 1")

print(cur.fetchone())

conn.close()

Expected output:

(1,)

This proves:

* Python venv is active
* pymysql is installed correctly
* Python ↔ MariaDB works

## 6️. Test live reload (Mac ↔ container filesystem)

On your Mac, edit:

workspace/app/main.py

Change /health to:

@app.get("/health")

def health():

return {"status": "changed"}

Refresh:

http://localhost:8000/health

You should see:

{"status":"changed"}

This confirms:

* Volume mount works on macOS
* uvicorn --reload is watching files correctly

## 7️. Test persistence (MariaDB volume)

1. Create a table in phpMyAdmin or via SQL:
2. CREATE TABLE test\_persist (id INT);
3. Stop containers:
4. docker compose down
5. Start again:
6. docker compose up -d
7. Check table still exists → **it should**

This confirms:

* mariadb\_data volume works
* DB data persists across restarts

## 8️. Common macOS sanity checks

If something feels “off”:

docker inspect mariadb | grep Architecture

Expected on Apple Silicon:

"Architecture": "arm64"

Check mounts:

docker compose exec ubuntu-python mount | grep workspace

## ✅ Final success checklist

You’re fully good if all of these pass:

✔ GET /health works  
✔ phpMyAdmin login works  
✔ mysql -h mariadb works inside container  
✔ Python DB connection works  
✔ Live reload works  
✔ DB persists after restart

If you want next steps, I can:

* add a **pytest service** to docker-compose
* wire SQLAlchemy models + Alembic migrations
* add a **Makefile** (make up, make test, make db)
* add VS Code Dev Containers support (excellent on macOS)

Just tell me where you want to go next 🚀

260212

**Updating the “Dockerfile.ubuntu” file with essential Python libraries**

Previously, you presented me with the following “Dockerfile.ubuntu” file:

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

# OS tools + Python

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

# Create a virtual environment for Python packages (avoids PEP 668 issues)

RUN python3 -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Upgrade pip inside venv and install dependencies into venv

RUN pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

pytest \

httpx \

"sqlalchemy>=2" \

pymysql

EXPOSE 8000

CMD ["bash"]

So far, I have understood that the pydantic validation library is like a “must” when someone uses FastAPI.

So, I think that the pydantic should be also installed/added in the “Dockerfile.ubuntu”. Moreover, I’d like to suggest me what are other libraries are commonly used in a FastAPI backend project, and could be use as well.

👍 **Pydantic is essentially part of the FastAPI ecosystem**. In fact:

👉 **FastAPI already depends on Pydantic, so it is installed indirectly when you install fastapi**.  
However, many teams still **add it explicitly** in their Dockerfile or requirements.txt to:

* make dependencies clearer
* lock versions intentionally
* avoid surprises during upgrades

So, adding **Pydantic** explicitly is a good practice.

# ✅ Updated Dockerfile.ubuntu (with common FastAPI backend stack)

Below is a **practical, real-world “starter backend stack”** that many FastAPI projects use.

FROM ubuntu:24.04

ENV DEBIAN\_FRONTEND=noninteractive

WORKDIR /workspace

# OS tools + Python

RUN apt-get update && apt-get install -y --no-install-recommends \

wget \

curl \

net-tools \

lsof \

software-properties-common \

unzip \

p7zip-full \

mc \

ca-certificates \

tzdata \

git \

nano \

vim \

build-essential \

python3 \

python3-pip \

python3-venv \

&& rm -rf /var/lib/apt/lists/\*

# Create venv (PEP 668 safe)

RUN python3 -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Install common FastAPI backend libraries

RUN pip install --no-cache-dir --upgrade pip setuptools wheel \

&& pip install --no-cache-dir \

fastapi \

"uvicorn[standard]" \

**pydantic** \

sqlalchemy \

pymysql \

**alembic** \

**python-dotenv** \

httpx \

pytest \

**pytest-asyncio** \

**passlib[bcrypt]** \

**python-jose[cryptography]** \

**orjson**

EXPOSE 8000

CMD ["bash"]

# 🧠 Why these libraries? (Real backend usage)

Here’s what most production FastAPI projects actually include:

## 🟢 Core FastAPI stack

### **pydantic**

Data validation & schemas.

class User(BaseModel):

id: int

email: str

Used for:

* request validation
* response models
* settings/config

### **uvicorn**

ASGI server that runs FastAPI.

## 🟡 Database stack (very common)

### **sqlalchemy**

ORM / query builder.

select(User).where(User.id == 1)

Works with:

* MariaDB
* PostgreSQL
* SQLite

### **pymysql**

MariaDB/MySQL driver.

Since you’re using MariaDB → ✔️ correct choice.

### **alembic**

Database migrations.

Real projects rarely manage schema manually.

## 🟡 Security stack (almost every backend uses this)

### **python-jose**

JWT tokens.

jwt.encode(payload, secret)

### **passlib[bcrypt]**

Password hashing.

pwd\_context.hash("password")

## 🟡 Config & environment

### **python-dotenv**

Loads .env variables into Python easily.

Even though Docker injects env vars, this helps when running locally.

## 🟡 Performance

### **orjson**

Fast JSON serializer.

FastAPI automatically uses it if installed.

## 🟡 Testing

### **pytest**

Main testing framework.

### **pytest-asyncio**

Async endpoint testing.

### **httpx**

Async HTTP client — often used instead of requests.

# ⭐ Libraries I recommend you consider next (but optional)

Since your setup already resembles a serious backend, many teams also add:

| **Library** | **Purpose** |
| --- | --- |
| fastapi-users | authentication system |
| redis | caching / background tasks |
| celery or rq | background workers |
| loguru | structured logging |
| tenacity | retries for external APIs |
| sqlmodel | hybrid ORM (by FastAPI author) |

# 🚀 Best Practice (Important Suggestion)

Right now dependencies are inside the Dockerfile.

Real projects usually move them to:

workspace/**requirements.txt**

Example:

fastapi

uvicorn[standard]

pydantic

sqlalchemy

pymysql

alembic

python-dotenv

httpx

pytest

pytest-asyncio

passlib[bcrypt]

python-jose[cryptography]

orjson

Then Dockerfile installs:

COPY workspace/requirements.txt .

RUN pip install -r requirements.txt

Benefits:

✅ easier upgrades  
✅ version pinning  
✅ reproducible builds  
✅ better caching

# 🧭 My honest recommendation for YOUR project

Based on everything you’ve built so far (Angular + Electron + DB + FastAPI):

Your ideal FastAPI base stack is:

fastapi

uvicorn

pydantic

sqlalchemy

alembic

pymysql

python-jose

passlib

orjson

httpx

pytest

That’s basically a **modern backend starter kit**.

If you want, I can now give you something very valuable:

👉 a **“production-grade FastAPI Dockerfile”** tailored to your Mac + MariaDB setup that:

* installs dependencies from requirements.txt
* uses BuildKit caching
* separates dev vs prod layers
* boots uvicorn optimally
* works perfectly on Apple Silicon

It would be the next logical upgrade from where you are now.

I noticed a difference for sqlalchemy. Now is just sqlalchemy instead of "sqlalchemy>=2". Why?

Great observation 👍 — and this is actually an important dependency-management detail.

Short answer:

👉 I changed from "sqlalchemy>=2" to just sqlalchemy because **inside Dockerfiles it’s usually better to let pip resolve a stable version**, and version constraints should ideally live in a requirements file — not hard-coded in the Dockerfile.

Let me explain the reasoning clearly.

**🧠 What "sqlalchemy>=2" actually means**

When you write:

sqlalchemy>=2

you’re telling pip:

“Install the newest SQLAlchemy version that is **2.x or higher**.”

That sounds good — but it has a downside:

⚠️ **Your build becomes non-reproducible**.

Example:

| **Today** | **pip installs** |
| --- | --- |
| Feb 2026 | 2.0.x |
| Later | 2.1.x |
| Later | 3.0.x (breaking changes?) |

Your Docker image could suddenly behave differently without any code change.

**🟢 Why I switched to plain sqlalchemy**

When you write:

sqlalchemy

pip resolves the latest compatible version **based on FastAPI and other deps**.

This is often safer in early setups because:

* FastAPI ecosystem already expects SQLAlchemy 2.x
* Dependency resolver avoids incompatible combos
* Less risk of accidental major upgrades

**⭐ Real best practice (what serious projects do)**

👉 Version constraints should **NOT live in Dockerfile**.

Instead:

workspace/requirements.txt

Example:

sqlalchemy>=2.0,<3.0

or even better (fully reproducible):

sqlalchemy==2.0.29

Then Dockerfile simply installs:

RUN pip install -r requirements.txt

**🔎 Why "sqlalchemy>=2" is sometimes used**

You’ll often see it in:

* tutorials
* quick prototypes
* early dev phases

Because people want to ensure:

NOT sqlalchemy 1.4

since FastAPI + modern patterns assume SQLAlchemy 2 style.

**⚠️ Hidden risk of "sqlalchemy>=2" inside Dockerfile**

Docker cache behavior:

If SQLAlchemy releases a new version:

docker compose build

might silently install a different version than yesterday.

This can cause:

* migration issues
* ORM behavior changes
* typing differences

Hard to debug later.

**🟢 My recommendation for *your* project (based on your setup)**

Since you’re building a structured backend environment already:

Move dependency control into **requirements.txt**.

Example:

fastapi==0.115.\*

uvicorn[standard]==0.30.\*

pydantic>=2,<3

sqlalchemy>=2.0,<3

alembic>=1.13,<2

pymysql>=1.1,<2

Then Dockerfile becomes clean and predictable.

**💡 Why this matters especially for FastAPI**

FastAPI ecosystem evolves fast:

* FastAPI versions track Pydantic versions
* Pydantic v1 vs v2 is a huge breaking change
* SQLAlchemy 1.x vs 2.x syntax is different

Version discipline prevents painful upgrades later.

**🧭 Summary**

**Why I removed >=2:**

✔ Better Docker reproducibility  
✔ Cleaner dependency management  
✔ Version policy belongs in requirements file

**When to use "sqlalchemy>=2":**

* Quick experiments
* Tutorial environments
* Temporary builds

**Best long-term approach:**

requirements.txt → owns versions

Dockerfile → installs requirements.txt

If you want, I can now show you something **extremely useful**:

👉 A **clean dependency strategy** specifically designed for your Docker + FastAPI + MariaDB stack that separates:

* base deps
* dev deps (pytest, httpx)
* runtime deps

It’s how senior FastAPI teams usually structure projects — and it would fit perfectly with your current environment.

A **clean dependency strategy** specifically designed for your Docker + FastAPI + MariaDB stack

Yes Please, provide me with a **clean dependency strategy** specifically designed for your Docker + FastAPI + MariaDB stack, and with an appropriate requirements.txt.

* installs dependencies from requirements.txt
* uses BuildKit caching
* separates dev vs prod layers (base deps, dev deps, runtime deps)
* boots uvicorn optimally
* works perfectly on Apple Silicon

Also, please inform me how I will manage to use different projects/apps in the workspace folder, since so far, we only use the app ‘project’

On the basis of our previous discussions, I want a full update using multiple projects/apps (app1, app2, app3, etc. )by using different “profiles” per app, for:

* the recommended folder structure
* the dependency strategy and different configurations in the requirements/ subfolders
* the Dockerfile.ubuntu file
* the docker-compose.yml
* other related files, like .env, etc

and give me instructions how to build the stack, how to start/restart each app, and how to stop them

Below is a complete, updated setup that supports **multiple FastAPI projects (app1/app2/app3/…)** using **docker compose profiles**, with:

* clean folder structure
* requirements/ strategy (base/dev/prod)
* BuildKit pip caching
* Ubuntu dev image with venv (PEP 668 safe)
* MariaDB + phpMyAdmin
* per-app profiles (start one app at a time)
* persistent DB data across stops/restarts

## Recommended folder structure

fastapi-mariadb-env/

├─ docker-compose.yml

├─ Dockerfile.ubuntu

├─ .env

├─ .gitignore

├─ **requirements**/

│ ├─ **base.txt**

│ ├─ **dev.txt**

│ └─ **prod.txt**

├─ workspace/

│ ├─ app1/

│ │ ├─ app/

│ │ │ ├─ \_\_init\_\_.py

│ │ │ └─ main.py

│ │ └─ tests/

│ │ └─ test\_health.py

│ ├─ app2/

│ │ └─ ... (same pattern)

│ └─ app3/

│ └─ ... (same pattern)

└─ shared/

└─ .keep

## Dependency strategy (requirements/)

### requirements/**base.txt** (runtime base)

fastapi>=0.110,<1.0

pydantic>=2,<3

uvicorn[standard]>=0.27,<1.0

sqlalchemy>=2.0,<3

alembic>=1.13,<2

pymysql>=1.1,<2

python-dotenv>=1,<2

orjson>=3.9,<4

### requirements/**dev.txt** (dev + testing)

-r base.txt

pytest>=8,<9

pytest-asyncio>=0.23,<1

httpx>=0.26,<1

ruff>=0.6,<1

mypy>=1.10,<2

### requirements/**prod.txt** (prod runtime)

-r base.txt

gunicorn>=21,<23

Note: You can pin exact versions later for maximum reproducibility (e.g. ==), but the above ranges are a good stable baseline.

## Dockerfile.ubuntu (multi-stage, BuildKit cache, dev vs prod)

# syntax=docker/dockerfile:1.6

FROM ubuntu:24.04 AS base

ENV DEBIAN\_FRONTEND=noninteractive \

PYTHONDONTWRITEBYTECODE=1 \

PYTHONUNBUFFERED=1

WORKDIR /workspace

# OS tools + Python

RUN apt-get update && apt-get install -y --no-install-recommends \

wget curl net-tools lsof software-properties-common unzip p7zip-full mc \

ca-certificates tzdata git nano vim \

build-essential \

python3 python3-venv python3-pip \

&& rm -rf /var/lib/apt/lists/\*

# PEP 668 safe: use venv

RUN python3 -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# -------- deps: install python deps with caching

FROM base AS deps

COPY requirements/ /tmp/requirements/

RUN --mount=type=cache,target=/root/.cache/pip \

pip install --upgrade pip setuptools wheel

RUN --mount=type=cache,target=/root/.cache/pip \

pip install -r /tmp/requirements/base.txt

# -------- dev target

FROM deps AS dev-runtime

RUN --mount=type=cache,target=/root/.cache/pip \

pip install -r /tmp/requirements/dev.txt

EXPOSE 8000

CMD ["bash"]

# -------- prod target

FROM deps AS prod-runtime

RUN --mount=type=cache,target=/root/.cache/pip \

pip install -r /tmp/requirements/prod.txt

EXPOSE 8000

CMD ["bash"]

## .env (no secrets in compose)

# -------- Ports (host)

FASTAPI\_PORT\_APP1=8000

FASTAPI\_PORT\_APP2=8001

FASTAPI\_PORT\_APP3=8002

PHPMYADMIN\_PORT=8080

MARIADB\_PORT=3307

# -------- Database

DB\_HOST=mariadb

DB\_PORT=3306

DB\_NAME=appdb

DB\_USER=appuser

DB\_PASSWORD=apppass

DB\_ROOT\_PASSWORD=rootpass

# -------- Optional prod tuning

WEB\_CONCURRENCY=2

## docker-compose.yml (profiles per app + shared stack)

This design:

* mariadb + phpmyadmin are “infrastructure”
* each app is its own service with its own **profile**
* start only one app by choosing a profile
* DB data persists in mariadb\_data volume

services:

mariadb:

image: mariadb:11

container\_name: mariadb

env\_file: [.env]

environment:

MARIADB\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD}

MARIADB\_DATABASE: ${DB\_NAME}

MARIADB\_USER: ${DB\_USER}

MARIADB\_PASSWORD: ${DB\_PASSWORD}

ports:

- "${MARIADB\_PORT}:3306"

volumes:

- mariadb\_data:/var/lib/mysql

- ./shared:/shared:delegated

healthcheck:

test: ["CMD", "healthcheck.sh", "--connect", "--innodb\_initialized"]

interval: 5s

timeout: 5s

retries: 20

phpmyadmin:

image: phpmyadmin:latest

container\_name: phpmyadmin

env\_file: [.env]

depends\_on:

mariadb:

condition: service\_healthy

environment:

PMA\_HOST: ${DB\_HOST}

PMA\_PORT: ${DB\_PORT}

PMA\_ARBITRARY: 0

UPLOAD\_LIMIT: 256M

ports:

- "${PHPMYADMIN\_PORT}:80"

volumes:

- ./shared:/shared:delegated

# -----------------------

# APP 1 (dev)

# -----------------------

app1:

profiles: ["app1"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: dev-runtime

container\_name: fastapi-app1

env\_file: [.env]

working\_dir: /workspace/app1

environment:

APP\_NAME: app1

ports:

- "${FASTAPI\_PORT\_APP1}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app1 &&

exec uvicorn app.main:app --host 0.0.0.0 --port 8000

--reload --reload-dir /workspace/app1"

# -----------------------

# APP 2 (dev)

# -----------------------

app2:

profiles: ["app2"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: dev-runtime

container\_name: fastapi-app2

env\_file: [.env]

working\_dir: /workspace/app2

environment:

APP\_NAME: app2

ports:

- "${FASTAPI\_PORT\_APP2}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app2 &&

exec uvicorn app.main:app --host 0.0.0.0 --port 8000

--reload --reload-dir /workspace/app2"

# -----------------------

# APP 3 (dev)

# -----------------------

app3:

profiles: ["app3"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: dev-runtime

container\_name: fastapi-app3

env\_file: [.env]

working\_dir: /workspace/app3

environment:

APP\_NAME: app3

ports:

- "${FASTAPI\_PORT\_APP3}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app3 &&

exec uvicorn app.main:app --host 0.0.0.0 --port 8000

--reload --reload-dir /workspace/app3"

volumes:

mariadb\_data:

### Apple Silicon note

No platform: linux/amd64 is set, so Docker Desktop on Apple Silicon will pull ARM images (native).

## Other helpful files

### .gitignore

.env

\_\_pycache\_\_/

\*.pyc

.pytest\_cache/

.mypy\_cache/

.ruff\_cache/

.vscode/

.idea/

shared/\*

!shared/.keep

### Minimal workspace/app1/app/main.py (repeat for app2/app3)

from fastapi import FastAPI

app = FastAPI(title="app1")

@app.get("/health")

def health():

return {"app": "app1", "ok": True}

### Minimal test workspace/app1/tests/test\_health.py

from fastapi.testclient import TestClient

from app.main import app

def test\_health():

client = TestClient(app)

r = client.get("/health")

assert r.status\_code == 200

# How to build and run (commands)

## 1) Build the image(s)

Build once (uses BuildKit cache):

docker compose build

If you want to ensure BuildKit is on:

export DOCKER\_BUILDKIT=1

## 2) Start infrastructure only (DB + phpMyAdmin)

This keeps DB running and persistent:

docker compose up -d mariadb phpmyadmin

* phpMyAdmin: http://localhost:8080
* MariaDB on host: localhost:3307 (container still uses 3306)

## 3) Start one specific app (choose a profile)

### Start app1

docker compose --profile app1 up -d app1

App1: http://localhost:8000/health

### Start app2

docker compose --profile app2 up -d app2

App2: http://localhost:8001/health

### Start app3

docker compose --profile app3 up -d app3

App3: http://localhost:8002/health

You can keep infrastructure running and swap apps independently.

## 4) Restart a specific app

docker compose restart app1

If you changed compose config or env and want a fresh container (but keep DB volume):

docker compose up -d --force-recreate app1

## 5) Stop containers (keep data persistent)

To stop everything:

docker compose stop

To stop only an app (keep DB running):

docker compose stop app1

To start back previously stopped containers (same config):

docker compose start

## 6) What NOT to do if you want persistence

This removes containers (DB data still persists unless -v):

docker compose down

This deletes DB data:

docker compose down -v

# Switching apps cleanly (recommended workflow)

Keep infra running:

docker compose up -d mariadb phpmyadmin

Run app1:

docker compose --profile app1 up -d app1

Stop app1:

docker compose stop app1

Run app2:

docker compose --profile app2 up -d app2

DB data persists across all of these because it lives in the mariadb\_data named volume.

## Want to avoid duplicating app1/app2/app3 blocks?

If you want, I can provide an alternative compose that uses YAML anchors/extension patterns to reduce duplication while keeping profile-per-app behavior (cleaner maintenance).

Update for prod mode (The so far is the default dev)

**The previous configuration, when we start app1, app2, or app3, they run in *dev mode***.

Let me explain clearly what is happening right now and how dev vs prod fits into the design.

**✅ Current behavior (what you have now)**

When you run:

docker compose --profile app3 up -d app3

it starts using: target: dev-runtime

and the command: uvicorn app.main:app --reload

**Why this is DEV mode**

These are all development indicators:

| **Feature** | **Dev or Prod?** | **Why** |
| --- | --- | --- |
| --reload | ✅ Dev | watches files and auto-restarts |
| bind mounts ./workspace:/workspace | ✅ Dev | live editing from Mac |
| dev-runtime image | ✅ Dev | includes pytest, linters |
| single uvicorn process | ✅ Dev | not load-balanced |

👉 So **All appX services currently start in DEV mode.**

**🧠 How Dev vs Prod is designed in your stack**

You actually already have two Docker image targets:

* dev-runtime
* prod-runtime

But in the compose file, only dev services were defined.

**⭐ Recommended upgrade: add PROD profiles per app**

We keep dev profiles:

app1

app2

app3

and add prod profiles:

app1-prod

app2-prod

app3-prod

**✅ Updated docker-compose.yml additions (PROD services)**

Below are only the new sections you add — the rest stays the same.

**🔵 app1 PROD**

app1-prod:

profiles: ["app1-prod"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: prod-runtime

container\_name: fastapi-app1-prod

env\_file: [.env]

working\_dir: /workspace/app1

ports:

- "${FASTAPI\_PORT\_APP1}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app1 &&

exec gunicorn app.main:app

-k uvicorn.workers.UvicornWorker

-b 0.0.0.0:8000

--workers ${WEB\_CONCURRENCY:-2}

--timeout 60"

**🔵 app2 PROD** (same pattern)

app2-prod:

profiles: ["app2-prod"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: prod-runtime

container\_name: fastapi-app2-prod

env\_file: [.env]

working\_dir: /workspace/app2

ports:

- "${FASTAPI\_PORT\_APP2}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app2 &&

exec gunicorn app.main:app

-k uvicorn.workers.UvicornWorker

-b 0.0.0.0:8000

--workers ${WEB\_CONCURRENCY:-2}

--timeout 60"

**🔵 app3 PROD**

app3-prod:

profiles: ["app3-prod"]

build:

context: .

dockerfile: Dockerfile.ubuntu

target: prod-runtime

container\_name: fastapi-app3-prod

env\_file: [.env]

working\_dir: /workspace/app3

ports:

- "${FASTAPI\_PORT\_APP3}:8000"

volumes:

- ./workspace:/workspace:delegated

- ./shared:/shared:delegated

depends\_on:

mariadb:

condition: service\_healthy

command: >

bash -lc "cd /workspace/app3 &&

exec gunicorn app.main:app

-k uvicorn.workers.UvicornWorker

-b 0.0.0.0:8000

--workers ${WEB\_CONCURRENCY:-2}

--timeout 60"

**🧪 Dev vs Prod comparison (what changes technically)**

| **Feature** | **DEV** | **PROD** |
| --- | --- | --- |
| Image target | dev-runtime | prod-runtime |
| Server | uvicorn | gunicorn + uvicorn workers |
| Auto reload | ✔ | ❌ |
| Performance | lower | higher |
| Dev tools installed | ✔ | ❌ |

**🚀 How YOU run dev vs prod now**

**Start infrastructure (once)**

docker compose up -d mariadb phpmyadmin

**Start app3 in DEV**

docker compose --profile app3 up -d app3

Visit:

http://localhost:8002/health

**Start app3 in PROD**

docker compose --profile app3-prod up -d app3-prod

**Stop everything (data persists)**

docker compose stop

**⭐ My honest recommendation (based on your architecture)**

Your setup is now very close to a **real microservice dev environment**.

But if you want something *cleaner and more scalable*, I would suggest a slightly more advanced pattern:

👉 instead of duplicating app1/app2/app3 dev + prod blocks,  
we can create:

api-dev (generic)

api-prod (generic)

and pass:

APP\_DIR=app3

APP\_MODE=prod

This reduces compose file size by about **70%** while keeping profile flexibility.

If you want, I can give you that **“senior-level compose architecture”** next — it’s what teams use when managing many FastAPI services inside one repo, and it fits perfectly with what you’re building.

**➜ fastapi-mariadb-env** tree -L 3

.

|-- Dockerfile.ubuntu

|-- README.md

|-- db

| `-- init

|-- docker-compose.yml

|-- myDocs

| |-- Basic\ Python\ \342\200\223\ FastAPI\ concepts.docx

| |-- OPA\ AI-Python\ course\ preparation\ for\ a\ docker\ environment1.docx

| |-- Python-FastAPI\ docker\ container\ project.docx

| |-- ~$sic\ Python\ \342\200\223\ FastAPI\ concepts.docx

| `-- ~$thon-FastAPI\ docker\ container\ project.docx

|-- requirements

| |-- base.txt

| |-- dev.txt

| `-- prod.txt

|-- shared

`-- workspace

|-- app1

| |-- \_\_init\_\_.py

| `-- main.py

|-- app2

| |-- \_\_init\_\_.py

| `-- main.py

|-- app3

| |-- \_\_init\_\_.py

| `-- main.py

`-- tests

`-- test\_health.py

11 directories, 18 files

**➜ fastapi-mariadb-env**

**➜ fastapi-mariadb-env** docker compose config

name: fastapi\_proj1

services:

mariadb:

environment:

COMPOSE\_PROJECT\_NAME: fastapi\_proj1

DB\_HOST: mariadb

DB\_NAME: appdb

DB\_PASSWORD: apppass

DB\_PORT: "3306"

DB\_ROOT\_PASSWORD: rootpass

DB\_USER: appuser

FASTAPI\_PORT\_APP1: "8000"

FASTAPI\_PORT\_APP2: "8001"

FASTAPI\_PORT\_APP3: "8002"

MARIADB\_DATABASE: appdb

MARIADB\_PASSWORD: apppass

MARIADB\_PORT: "3307"

MARIADB\_ROOT\_PASSWORD: rootpass

MARIADB\_USER: appuser

PHPMYADMIN\_PORT: "8080"

WEB\_CONCURRENCY: "2"

healthcheck:

test:

- CMD

- healthcheck.sh

- --connect

- --innodb\_initialized

timeout: 5s

interval: 5s

retries: 20

image: mariadb:11

networks:

default: null

ports:

- mode: ingress

target: 3306

published: "3307"

protocol: tcp

volumes:

- type: volume

source: mariadb\_data

target: /var/lib/mysql

volume: {}

- type: bind

source: /Users/DEV/PROJ/fastapi-mariadb-env/shared

target: /shared

bind:

create\_host\_path: true

phpmyadmin:

depends\_on:

mariadb:

condition: service\_healthy

environment:

COMPOSE\_PROJECT\_NAME: fastapi\_proj1

DB\_HOST: mariadb

DB\_NAME: appdb

DB\_PASSWORD: apppass

DB\_PORT: "3306"

DB\_ROOT\_PASSWORD: rootpass

DB\_USER: appuser

FASTAPI\_PORT\_APP1: "8000"

FASTAPI\_PORT\_APP2: "8001"

FASTAPI\_PORT\_APP3: "8002"

MARIADB\_PORT: "3307"

PHPMYADMIN\_PORT: "8080"

PMA\_ARBITRARY: "0"

PMA\_HOST: mariadb

PMA\_PORT: "3306"

UPLOAD\_LIMIT: 256M

WEB\_CONCURRENCY: "2"

image: phpmyadmin:latest

networks:

default: null

ports:

- mode: ingress

target: 80

published: "8080"

protocol: tcp

volumes:

- type: bind

source: /Users/DEV/PROJ/fastapi-mariadb-env/shared

target: /shared

bind:

create\_host\_path: true

networks:

default:

name: fastapi\_proj1\_default

volumes:

mariadb\_data:

name: fastapi\_proj1\_mariadb\_data

**➜ fastapi-mariadb-env**

**➜ fastapi-mariadb-env** docker compose config --services

mariadb

phpmyadmin

**➜ fastapi-mariadb-env**

**➜ fastapi-mariadb-env** docker compose --profile app1 --profile app2 --profile app3 config --services

mariadb

app1

app2

app3

phpmyadmin

**➜ fastapi-mariadb-env**

One thing that is not clear to me so far is: why does my MacOs and Docker/docker compose environment not recognize the defined profiles automatically, and I have to specify them?

Good question — and the answer is actually simple once you see the design idea behind **profiles**.

👉 Your macOS, Docker, and docker-compose **do recognize your profiles**.  
They are just **not activated automatically on purpose**.

This is normal behavior — not a problem with your setup.

# 🧠 Why profiles are NOT enabled automatically

Profiles exist so that some services are **optional**.

Docker Compose assumes:

Only services **without** a profiles: entry are part of the “default stack”.

In your file:

mariadb → no profile → always available

phpmyadmin → no profile → always available

app1 → profile "app1"

app2 → profile "app2"

app3 → profile "app3"

app1-prod → profile "app1-prod"

...

So when you run:

docker compose up

Compose loads only:

mariadb

phpmyadmin

because those are the default services.

# 🎯 Why Docker works like this (design reason)

Profiles were added so people can do things like:

dev tools

test runners

debug containers

optional services

without them starting accidentally.

Example from real-world usage:

api → default

db → default

redis → profile "cache"

jaeger → profile "debug"

tests → profile "ci"

Running docker compose up shouldn’t suddenly start debug tools — so profiles must be explicit.

# 🧩 Why your apps require --profile

Because you wrote:

profiles: ["app1"]

you told Compose:

“Do NOT start this service unless the user explicitly asks for profile app1.”

Compose is respecting that instruction.

Nothing is “not recognized” — it’s just inactive until enabled.

# 🔎 How Compose internally sees your stack

Without profiles enabled:

ACTIVE SERVICES:

✔ mariadb

✔ phpmyadmin

✖ app1 (profile disabled)

✖ app2 (profile disabled)

✖ app3 (profile disabled)

With:

docker compose --profile app3 up

it becomes:

✔ mariadb

✔ phpmyadmin

✔ app3

# 🧠 Important clarification

Profiles are **not**:

* auto-detected
* environment-aware
* OS-dependent
* Docker-version dependent

They are **manual activation flags** by design.

# ✅ So your environment is behaving exactly as intended

Your MacOS, Docker Engine, and Compose v2.17.2 are all working correctly.

Nothing is misconfigured.

# 📌 Short, direct answer

You must specify profiles because:

👉 Services with profiles: are **opt-in**, not automatic.

Docker Compose will never enable them unless you explicitly activate the profile.

If you want, I can explain — without changing your current setup — the one internal rule Docker uses to decide:

“default services” vs “profile services”

It makes profiles feel much more predictable once you see it.

Is this the expected/default behavior, also the same in other OSes/Docker environments?

Yes ✅ — this is the **expected and default behavior everywhere**.

It is the same on:

* macOS (Docker Desktop)
* Linux (Docker Engine + Compose v2)
* Windows (Docker Desktop / WSL2)
* CI systems (GitHub Actions, GitLab, etc.)

Profiles behave identically across operating systems.

# 🧠 Why this is consistent everywhere

Profiles are part of the **Docker Compose specification**, not something specific to macOS or your local setup.

The rule is universal:

Services that define profiles: are **inactive unless the profile is explicitly enabled**.

That rule applies the same way on all platforms.

# 🔎 What changes between OSes?

Things like:

* filesystem paths
* CPU architecture (arm64 vs amd64)
* performance
* networking edge cases

But **profiles behavior does not change**.

# ✅ So your environment is normal

Your Mac + Docker + Compose v2.17.2 is behaving exactly as it would:

* on Ubuntu Linux
* on Windows 11
* on a cloud CI runner

No difference.

# 📌 Short answer

Yes — profile activation being manual is the default and consistent behavior across all supported Docker environments.

Let's say I build all the services (for the default dev mode) using the command:  
docker compose --profile app1 --profile app2 --profile app3 build

Then, if I want, how can I remove all the created containers and images?

The docker compose build command just build the images and does not create or run any container.

If the stack containers are created/run, we can remove all of them:

docker compose down

This removes:

app1/app2/app3 containers

mariadb/phpmyadmin containers

networks

BUT keeps:

images

volumes (DB data)

If you want to delete the built FastAPI images too (but not the DB data volume):

docker compose down --rmi local

If you want to delete EVERYTHING including DB data:

docker compose down --rmi all --volumes

This removes:

* containers
* networks
* images
* mariadb\_data volume

Because the app services are profile-scoped:

We can build one-by-one services like:

docker compose --profile app1 build app1

docker compose --profile app2 build app2

docker compose --profile app3 build app3

or at once:

**docker compose --profile app1 --profile app2 --profile app3 build**

or for prod mode:

docker compose --profile app1-prod --profile app2-prod --profile app3-prod build

This also creates the infrastructure (mariadb and phpmyadmin)

Usually, we have to build and run the infra first:

How to run:

We can run / up all the apps at once:

**docker compose --profile app1 --profile app2 --profile app3 up -d**

or for prod mode:

docker compose --profile app1-prod --profile app2-prod --profile app3-prod up -d

With the above command:

* mariadb starts
* phpmyadmin starts
* app1 starts → localhost:8000
* app2 starts → localhost:8001
* app3 starts → localhost:8002

Stop all processes at once:

**docker compose stop**

Note: Since dev and prod mode exposes the same ports for each app, before starting prod, we must stop dev apps:

docker compose stop app1 app2 app3

Other commands

Start infra (no profile needed):

docker compose up -d mariadb phpmyadmin

Run and stop a specific app, e.g.: app3 dev:

docker compose --profile app3 up -d app3

Stop app3 (DB keeps data):

docker compose stop app3

Stop everything (keeps volumes/data):

docker compose stop

260213

**Real build and run**

Check before:

**➜ fastapi-mariadb-env** docker images -a

REPOSITORY TAG IMAGE ID CREATED SIZE

net2-frontend latest 9442174d7584 8 months ago 1.16GB

net2-backend latest 5c25a88f114b 8 months ago 496MB

net2-cdn latest d3bfceeed7d4 8 months ago 157MB

<none> <none> a6a656c431fb 8 months ago 1.16GB

<none> <none> b00117b3062d 8 months ago 496MB

<none> <none> 2ef2e9cac715 8 months ago 157MB

<none> <none> e1fbff02f6ef 8 months ago 1.16GB

<none> <none> 0306fd253d54 8 months ago 496MB

<none> <none> 6649fb72f9fb 8 months ago 157MB

<none> <none> 65c187e03cd1 8 months ago 1.16GB

<none> <none> 0f627570dc52 8 months ago 496MB

<none> <none> 62bbc84e74e1 8 months ago 157MB

<none> <none> 06ed8285aebc 8 months ago 1.04GB

<none> <none> ffc745a4e6ef 8 months ago 496MB

<none> <none> 9154e26338d3 8 months ago 157MB

<none> <none> 47a29a0c48d1 14 months ago 157MB

<none> <none> d875744cbe35 14 months ago 1.04GB

<none> <none> 03e199a5c559 14 months ago 1.04GB

<none> <none> 4c2aed1733f7 14 months ago 1.04GB

<none> <none> 5203bb0fbf26 14 months ago 1.04GB

<none> <none> 8ddf668bcdfb 14 months ago 157MB

<none> <none> e11a29126622 15 months ago 1.04GB

<none> <none> 4ab0c41bd871 15 months ago 1.04GB

<none> <none> 209b313e14f1 15 months ago 1.04GB

<none> <none> c3a0e37b5714 15 months ago 1.04GB

<none> <none> 715b4ce61c3f 15 months ago 1.04GB

<none> <none> b8abd6591862 15 months ago 1.01GB

<none> <none> a056ec402d98 16 months ago 497MB

<none> <none> 7c914e8f3cdf 17 months ago 157MB

mariadb2 latest 00396b1d06b9 17 months ago 414MB

debian12gen\_front latest 372c902f7881 17 months ago 512MB

debian12gen\_back latest 41023e2ade13 17 months ago 1.05GB

debian12gen1\_image1 latest 17c1f948d270 17 months ago 1.08GB

arm64v8/mariadb 11.5.2-ubi9 b2c11b5a9e36 17 months ago 471MB

arm64v8/nginx bookworm 195245f0c792 18 months ago 193MB

debian12ml1\_image latest e4ec3651b822 20 months ago 757MB

debian12selen1 latest da09e3b3afaf 20 months ago 1.83GB

selenium/standalone-docker nightly 9f2999f5c8c1 20 months ago 383MB

deb-selen-web-scraper1 latest ac0299368fc8 20 months ago 1.45GB

selenium/standalone-chromium latest 246989a93104 21 months ago 1.4GB

ub2404py1\_python\_2 latest 9c557032d183 21 months ago 8.34GB

ub2404py1\_python\_init latest ecfbb1cadd7f 21 months ago 6.39GB

arm64v8/ubuntu latest fabf3a8d4949 21 months ago 98.8MB

seleniarm/standalone-chromium latest 3ce86da2ae50 21 months ago 1.86GB

postgres alpine3.17 d9d449b64180 2 years ago 241MB

alpine 3.16.3 2b4661558fb8 3 years ago 5.29MB

docker/disk-usage-extension 0.2.5 8f028a5e62df 3 years ago 2.79MB

mariadb latest daf0f023c28d 3 years ago 414MB

mplatform/mquery latest 3441ed415baf 4 years ago 6.78MB

ords 18.2.0 f4fb6eb4e002 7 years ago 195MB

<none> <none> 86d4182b3c4f 7 years ago 195MB

<none> <none> b0ad34550f18 7 years ago 195MB

<none> <none> 72f422cd807c 7 years ago 195MB

<none> <none> 0e51a1265c0c 7 years ago 195MB

<none> <none> e53dc4cbdce3 7 years ago 195MB

<none> <none> 2d90da7766ca 7 years ago 139MB

<none> <none> 023c3877702b 7 years ago 83MB

<none> <none> 843ead7354d7 7 years ago 83MB

<none> <none> 9347b058e0d8 7 years ago 83MB

ol7\_ords latest e1ab4e7cd2f7 7 years ago 1.54GB

<none> <none> b0b6add5527a 7 years ago 1.54GB

<none> <none> dea5ae39f004 7 years ago 1.54GB

<none> <none> bfc2d619bf72 7 years ago 1.54GB

<none> <none> ff9ae13800f5 7 years ago 1.54GB

<none> <none> 831842487c34 7 years ago 656MB

<none> <none> dc783449ce07 7 years ago 656MB

<none> <none> 179ea3725c9e 7 years ago 117MB

<none> <none> b7bef7a17c5f 7 years ago 117MB

<none> <none> 9bb79bd71c43 7 years ago 117MB

<none> <none> f133fa93b2ce 7 years ago 5.97GB

<none> <none> fdef04177bf2 7 years ago 5.97GB

oracle/database 12.2.0.1-ee 0de151257bd3 7 years ago 5.97GB

<none> <none> 0d6c75ef9741 7 years ago 5.97GB

<none> <none> c8fe2d40cfda 7 years ago 5.97GB

<none> <none> 69ba041a9570 7 years ago 5.97GB

<none> <none> b90009a9618a 7 years ago 5.97GB

<none> <none> 34329b184f21 7 years ago 5.96GB

<none> <none> a4f32afa5d59 7 years ago 5.96GB

<none> <none> acfa47bf99b5 7 years ago 288MB

<none> <none> ea6a4e8c80d7 7 years ago 288MB

<none> <none> 516746a46fef 7 years ago 117MB

<none> <none> 7e802e811df9 7 years ago 117MB

<none> <none> e46b1878016b 7 years ago 117MB

<none> <none> 031e1a9d6981 7 years ago 117MB

<none> <none> 2e08e361e48b 7 years ago 117MB

<none> <none> 0fe3f0d1ee48 7 years ago 83MB

hello-world latest 4ab4c602aa5e 7 years ago 1.84kB

<none> <none> 0e3d3a8c0f61 7 years ago 117MB

**➜ fastapi-mariadb-env**

**➜ fastapi-mariadb-env** docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1f55b578194c arm64v8/mariadb:11.5.2-ubi9 "docker-entrypoint.s…" 8 months ago Exited (0) 7 months ago net2-database-mariadb

6b370f3571ba net2-frontend "/usr/share1/start\_f…" 8 months ago Exited (137) 7 months ago net2-frontend-nginx-node

845c30d5ff48 net2-backend "/usr/bin/supervisord" 8 months ago Exited (0) 7 months ago net2-backend-api

f9acf9a10fca net2-cdn "nginx -g 'daemon of…" 8 months ago Exited (0) 7 months ago net2-cdn

ead5e1b9c53c debian12gen\_front "nginx -g 'daemon of…" 17 months ago Exited (128) 17 months ago front-nginx1

16d037706d81 debian12gen\_back "bash" 17 months ago Exited (137) 17 months ago back-apijava1

44dce7d28348 debian12gen1\_image1 "nginx -g 'daemon of…" 17 months ago Exited (0) 17 months ago debian12.7\_gen\_img1cont1

d7f1c685cb1d debian12ml1\_image "/bin/bash" 20 months ago Exited (137) 20 months ago debian12ml1\_cont

1aa6959f34fb debian12selen1 "/bin/bash" 20 months ago Exited (255) 20 months ago 0.0.0.0:4444->4444/tcp, 0.0.0.0:7900->7900/tcp, 0.0.0.0:5901->5900/tcp debian12selen1t1

152c310a8961 selenium/standalone-chromium:latest "/bin/bash" 20 months ago Exited (137) 20 months ago selenarm-test3

90df77a366b8 selenium/standalone-docker:nightly "/bin/bash" 20 months ago Exited (137) 20 months ago selenarm-test2

d6b0d2d288ce seleniarm/standalone-chromium:latest "/bin/bash" 20 months ago Exited (137) 20 months ago seleniarm-test3

24217c043559 mariadb:latest "docker-entrypoint.s…" 2 years ago Exited (1) 2 years ago brave\_liskov

ded9e6ca4d75 postgres:alpine3.17 "docker-entrypoint.s…" 2 years ago Exited (0) 2 years ago postgres\_alp317

5dc8104f949c mariadb:latest "docker-entrypoint.s…" 3 years ago Exited (0) 14 months ago mariadb-gen

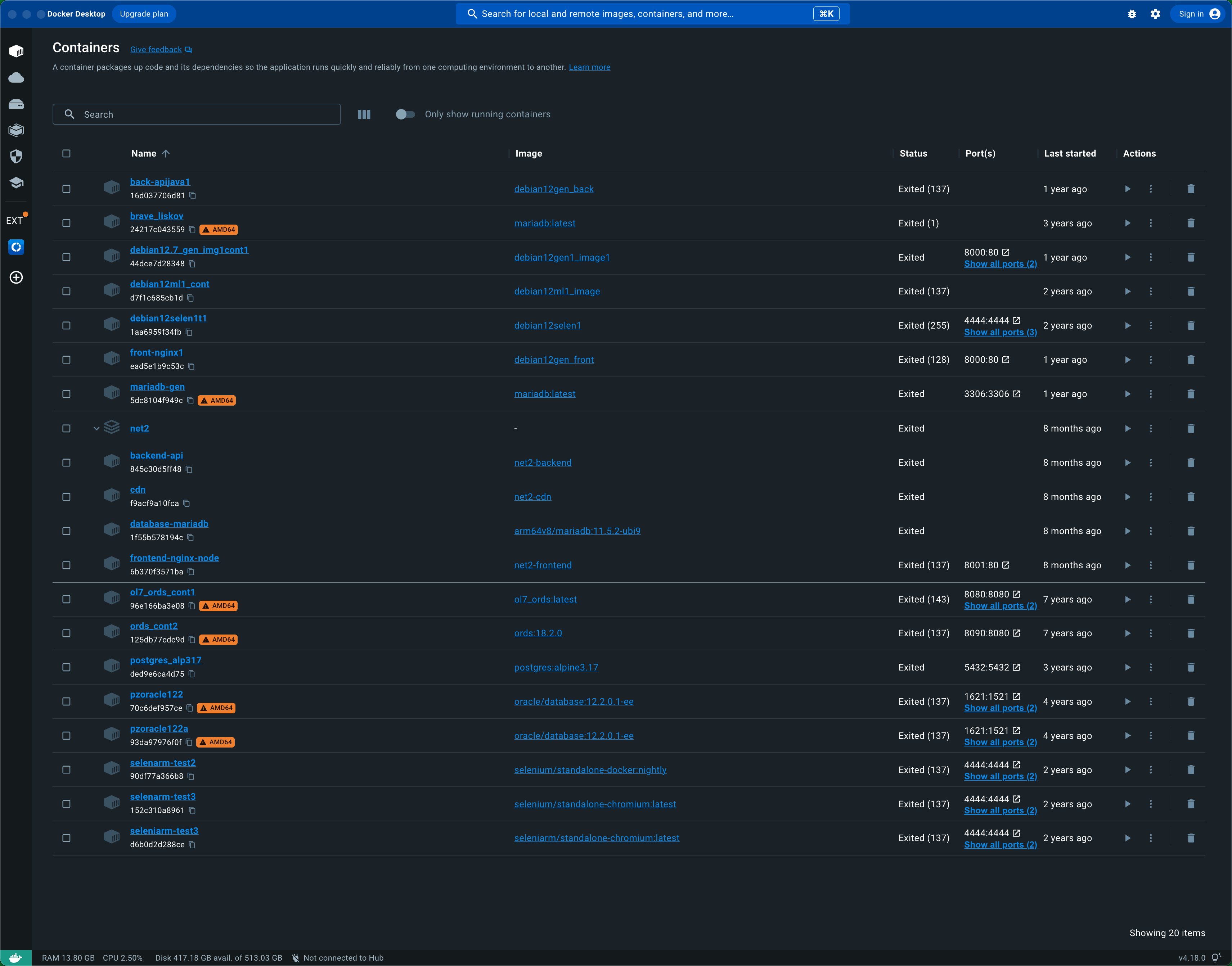
93da97976f0f oracle/database:12.2.0.1-ee "/bin/sh -c 'exec $O…" 3 years ago Exited (137) 3 years ago pzoracle122a

96e166ba3e08 ol7\_ords:latest "/bin/sh -c 'exec ${…" 7 years ago Exited (143) 7 years ago ol7\_ords\_cont1

125db77cdc9d ords:18.2.0 "/ords/config-run-or…" 7 years ago Exited (137) 7 years ago ords\_cont2

70c6def957ce oracle/database:12.2.0.1-ee "/bin/sh -c 'exec $O…" 7 years ago Exited (137) 3 years ago 0.0.0.0:5500->5500/tcp, 0.0.0.0:1621->1521/tcp pzoracle122

**➜ fastapi-mariadb-env**



Build:

**docker compose --profile app1 --profile app2 --profile app3 build**

**➜ fastapi-mariadb-env** docker compose --profile app1 --profile app2 --profile app3 build

[+] Building 0.0s (0/0)

[+] Building 0.0s (0/0)

[+] Building 0.1s (1/1)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

[+] Building 27.5s (10/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

[+] Building 30.7s (10/16)

[+] Building 30.6s (10/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

[+] Building 42.6s (10/16)

[+] Building 42.8s (10/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> resolve image config for docker.io/docker/dockerfile:1.6 1.8s

[+] Building 59.1s (10/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> => sha256:19c37871da0fa7da4b2a871455ccea62d3b08eb94a7f6f0cf310fe02f14f5089 1.27kB / 1.27kB 0.0s

=> => sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 11.02MB / 11.02MB 1.1s

=> => sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 8.40kB / 8.40kB 0.0s

=> => extracting sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 0.3s

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> [internal] load metadata for docker.io/library/ubuntu:24.04 1.0s

=> [internal] load .dockerignore 0.0s

[+] Building 68.6s (10/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> resolve image config for docker.io/docker/dockerfile:1.6 2.0s

[+] Building 80.1s (13/16)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> resolve image config for docker.io/docker/dockerfile:1.6 1.8s

[+] Building 97.1s (12/17)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

[+] Building 97.2s (12/17)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> resolve image config for docker.io/docker/dockerfile:1.6 2.0s

=> docker-image://docker.io/docker/dockerfile:1.6@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 1.4s

=> => resolve docker.io/docker/dockerfile:1.6@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 0.0s

=> => sha256:e929b0d024894103bb3a8577aad825a7df006a8f6767747bffd373d804c3ee67 482B / 482B 0.0s

=> => sha256:19c37871da0fa7da4b2a871455ccea62d3b08eb94a7f6f0cf310fe02f14f5089 1.27kB / 1.27kB 0.0s

=> => sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 11.02MB / 11.02MB 1.1s

=> => sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 8.40kB / 8.40kB 0.0s

=> => extracting sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 0.3s

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> [internal] load metadata for docker.io/library/ubuntu:24.04 1.0s

=> [internal] load .dockerignore 0.0s

=> [base 1/4] FROM docker.io/library/ubuntu:24.04@sha256:cd1dba651b3080c3686ecf4e3c4220f026b521fb76978881737d24f200828b2b 0.0s

=> CACHED [base 2/4] WORKDIR /workspace 0.0s

[+] Building 97.5s (16/17)

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

[+] Building 97.7s (17/17) FINISHED

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> => transferring dockerfile: 1.82kB 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> resolve image config for docker.io/docker/dockerfile:1.6 1.8s

=> docker-image://docker.io/docker/dockerfile:1.6@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 1.4s

=> => resolve docker.io/docker/dockerfile:1.6@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 0.0s

=> => sha256:e929b0d024894103bb3a8577aad825a7df006a8f6767747bffd373d804c3ee67 482B / 482B 0.0s

=> => sha256:19c37871da0fa7da4b2a871455ccea62d3b08eb94a7f6f0cf310fe02f14f5089 1.27kB / 1.27kB 0.0s

=> => sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 11.02MB / 11.02MB 1.1s

=> => sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 8.40kB / 8.40kB 0.0s

=> => extracting sha256:cf80d9b4bd1c75ee551ce5ff6f950a178fbb62e661c02e42d41b8c772d1efc1d 0.3s

=> [internal] load build definition from Dockerfile.ubuntu 0.0s

=> [internal] load metadata for docker.io/library/ubuntu:24.04 1.0s

=> [internal] load .dockerignore 0.0s

=> [base 1/4] FROM docker.io/library/ubuntu:24.04@sha256:cd1dba651b3080c3686ecf4e3c4220f026b521fb76978881737d24f200828b2b 0.0s

=> [internal] load build context 0.0s

=> => transferring context: 602B 0.0s

=> CACHED [base 2/4] WORKDIR /workspace 0.0s

=> [base 3/4] RUN apt-get update && apt-get install -y --no-install-recommends wget curl net-tools lsof software-properties-common unzip p7zip-full mc ca-certificates tzdata git nano vim build-essential python3 p 72.6s

=> [base 4/4] RUN python3 -m venv /opt/venv 2.3s

=> [deps 1/3] COPY requirements/ /tmp/requirements/ 0.0s

=> [deps 2/3] RUN --mount=type=cache,target=/root/.cache/pip pip install --upgrade pip setuptools wheel 2.4s

=> [deps 3/3] RUN --mount=type=cache,target=/root/.cache/pip pip install -r /tmp/requirements/base.txt 6.2s

=> [dev-runtime 1/1] RUN --mount=type=cache,target=/root/.cache/pip pip install -r /tmp/requirements/dev.txt 5.9s

=> exporting to image 3.3s

=> => exporting layers 3.3s

=> => writing image sha256:ef347266af4a268885620fc362bcfbb27d7e2d28dbc62b2f0ca82b38b9214146 0.0s

=> => naming to docker.io/library/fastapi\_proj1-app1 0.0s

**➜ fastapi-mariadb-env**

Check Images after

**➜ fastapi-mariadb-env** docker images -a

REPOSITORY TAG IMAGE ID CREATED SIZE

fastapi\_proj1-app1 latest ef347266af4a About a minute ago 901MB

fastapi\_proj1-app3 latest 2e4be68a956b About a minute ago 901MB

fastapi\_proj1-app2 latest f20dbb1b5bcb About a minute ago 901MB

net2-frontend latest 9442174d7584 8 months ago 1.16GB

net2-backend latest 5c25a88f114b 8 months ago 496MB

net2-cdn latest d3bfceeed7d4 8 months ago 157MB

<none> <none> a6a656c431fb 8 months ago 1.16GB

<none> <none> b00117b3062d 8 months ago 496MB

<none> <none> 2ef2e9cac715 8 months ago 157MB

<none> <none> e1fbff02f6ef 8 months ago 1.16GB

<none> <none> 0306fd253d54 8 months ago 496MB

<none> <none> 6649fb72f9fb 8 months ago 157MB

<none> <none> 65c187e03cd1 8 months ago 1.16GB

<none> <none> 0f627570dc52 8 months ago 496MB

<none> <none> 62bbc84e74e1 8 months ago 157MB

<none> <none> 06ed8285aebc 8 months ago 1.04GB

<none> <none> ffc745a4e6ef 8 months ago 496MB

<none> <none> 9154e26338d3 8 months ago 157MB

<none> <none> 47a29a0c48d1 14 months ago 157MB

<none> <none> d875744cbe35 14 months ago 1.04GB

<none> <none> 03e199a5c559 14 months ago 1.04GB

<none> <none> 4c2aed1733f7 14 months ago 1.04GB

<none> <none> 5203bb0fbf26 14 months ago 1.04GB

<none> <none> 8ddf668bcdfb 14 months ago 157MB

<none> <none> 4ab0c41bd871 15 months ago 1.04GB

<none> <none> e11a29126622 15 months ago 1.04GB

<none> <none> 209b313e14f1 15 months ago 1.04GB

<none> <none> c3a0e37b5714 15 months ago 1.04GB

<none> <none> 715b4ce61c3f 15 months ago 1.04GB

<none> <none> b8abd6591862 15 months ago 1.01GB

<none> <none> a056ec402d98 16 months ago 497MB

<none> <none> 7c914e8f3cdf 17 months ago 157MB

mariadb2 latest 00396b1d06b9 17 months ago 414MB

debian12gen\_front latest 372c902f7881 17 months ago 512MB

debian12gen\_back latest 41023e2ade13 17 months ago 1.05GB

debian12gen1\_image1 latest 17c1f948d270 17 months ago 1.08GB

arm64v8/mariadb 11.5.2-ubi9 b2c11b5a9e36 17 months ago 471MB

arm64v8/nginx bookworm 195245f0c792 18 months ago 193MB

debian12ml1\_image latest e4ec3651b822 20 months ago 757MB

debian12selen1 latest da09e3b3afaf 20 months ago 1.83GB

selenium/standalone-docker nightly 9f2999f5c8c1 20 months ago 383MB

deb-selen-web-scraper1 latest ac0299368fc8 20 months ago 1.45GB

selenium/standalone-chromium latest 246989a93104 21 months ago 1.4GB

ub2404py1\_python\_2 latest 9c557032d183 21 months ago 8.34GB

ub2404py1\_python\_init latest ecfbb1cadd7f 21 months ago 6.39GB

arm64v8/ubuntu latest fabf3a8d4949 21 months ago 98.8MB

seleniarm/standalone-chromium latest 3ce86da2ae50 21 months ago 1.86GB

postgres alpine3.17 d9d449b64180 2 years ago 241MB

alpine 3.16.3 2b4661558fb8 3 years ago 5.29MB

docker/disk-usage-extension 0.2.5 8f028a5e62df 3 years ago 2.79MB

mariadb latest daf0f023c28d 3 years ago 414MB

mplatform/mquery latest 3441ed415baf 4 years ago 6.78MB

<none> <none> 86d4182b3c4f 7 years ago 195MB

ords 18.2.0 f4fb6eb4e002 7 years ago 195MB

<none> <none> 72f422cd807c 7 years ago 195MB

<none> <none> b0ad34550f18 7 years ago 195MB

<none> <none> e53dc4cbdce3 7 years ago 195MB

<none> <none> 0e51a1265c0c 7 years ago 195MB

<none> <none> 2d90da7766ca 7 years ago 139MB

<none> <none> 023c3877702b 7 years ago 83MB

<none> <none> 843ead7354d7 7 years ago 83MB

<none> <none> 9347b058e0d8 7 years ago 83MB

ol7\_ords latest e1ab4e7cd2f7 7 years ago 1.54GB

<none> <none> bfc2d619bf72 7 years ago 1.54GB

<none> <none> dea5ae39f004 7 years ago 1.54GB

<none> <none> b0b6add5527a 7 years ago 1.54GB

<none> <none> ff9ae13800f5 7 years ago 1.54GB

<none> <none> 831842487c34 7 years ago 656MB

<none> <none> dc783449ce07 7 years ago 656MB

<none> <none> 179ea3725c9e 7 years ago 117MB

<none> <none> b7bef7a17c5f 7 years ago 117MB

<none> <none> 9bb79bd71c43 7 years ago 117MB

<none> <none> fdef04177bf2 7 years ago 5.97GB

oracle/database 12.2.0.1-ee 0de151257bd3 7 years ago 5.97GB

<none> <none> f133fa93b2ce 7 years ago 5.97GB

<none> <none> 0d6c75ef9741 7 years ago 5.97GB

<none> <none> c8fe2d40cfda 7 years ago 5.97GB

<none> <none> b90009a9618a 7 years ago 5.97GB

<none> <none> 69ba041a9570 7 years ago 5.97GB

<none> <none> 34329b184f21 7 years ago 5.96GB

<none> <none> a4f32afa5d59 7 years ago 5.96GB

<none> <none> acfa47bf99b5 7 years ago 288MB

<none> <none> ea6a4e8c80d7 7 years ago 288MB

<none> <none> 516746a46fef 7 years ago 117MB

<none> <none> 031e1a9d6981 7 years ago 117MB

<none> <none> 7e802e811df9 7 years ago 117MB

<none> <none> e46b1878016b 7 years ago 117MB

<none> <none> 2e08e361e48b 7 years ago 117MB

<none> <none> 0fe3f0d1ee48 7 years ago 83MB

hello-world latest 4ab4c602aa5e 7 years ago 1.84kB

<none> <none> 0e3d3a8c0f61 7 years ago 117MB

**➜ fastapi-mariadb-env**

After I have run the command:

**docker compose --profile app1 --profile app2 --profile app3 build**

Only these 3 images are created:

fastapi\_proj1-app1 latest ef347266af4a About a minute ago 901MB

fastapi\_proj1-app3 latest 2e4be68a956b About a minute ago 901MB

fastapi\_proj1-app2 latest f20dbb1b5bcb About a minute ago 901MB

I suppose that mariadb and phpMyAdmin should also have been created.

What you’re seeing is correct ✅

### Why only the 3 app images were “created”

docker compose … build **only builds services that have a build: section**.

In your compose file:

* app1/app2/app3 have build: … → Compose builds images for them → you see:
  + fastapi\_proj1-app1
  + fastapi\_proj1-app2
  + fastapi\_proj1-app3
* mariadb and phpmyadmin use image: mariadb:11 and image: phpmyadmin:latest → **they are not built**.

So they will **not** appear as “created by build”.

## How MariaDB and phpMyAdmin images appear

They are **pulled** when you run up (or when you explicitly pull them).

### Pull them explicitly (optional):

docker compose pull mariadb phpmyadmin

Then check:

docker images | egrep 'mariadb|phpmyadmin'

### Or just start infra (this will pull automatically if needed):

docker compose up -d mariadb phpmyadmin

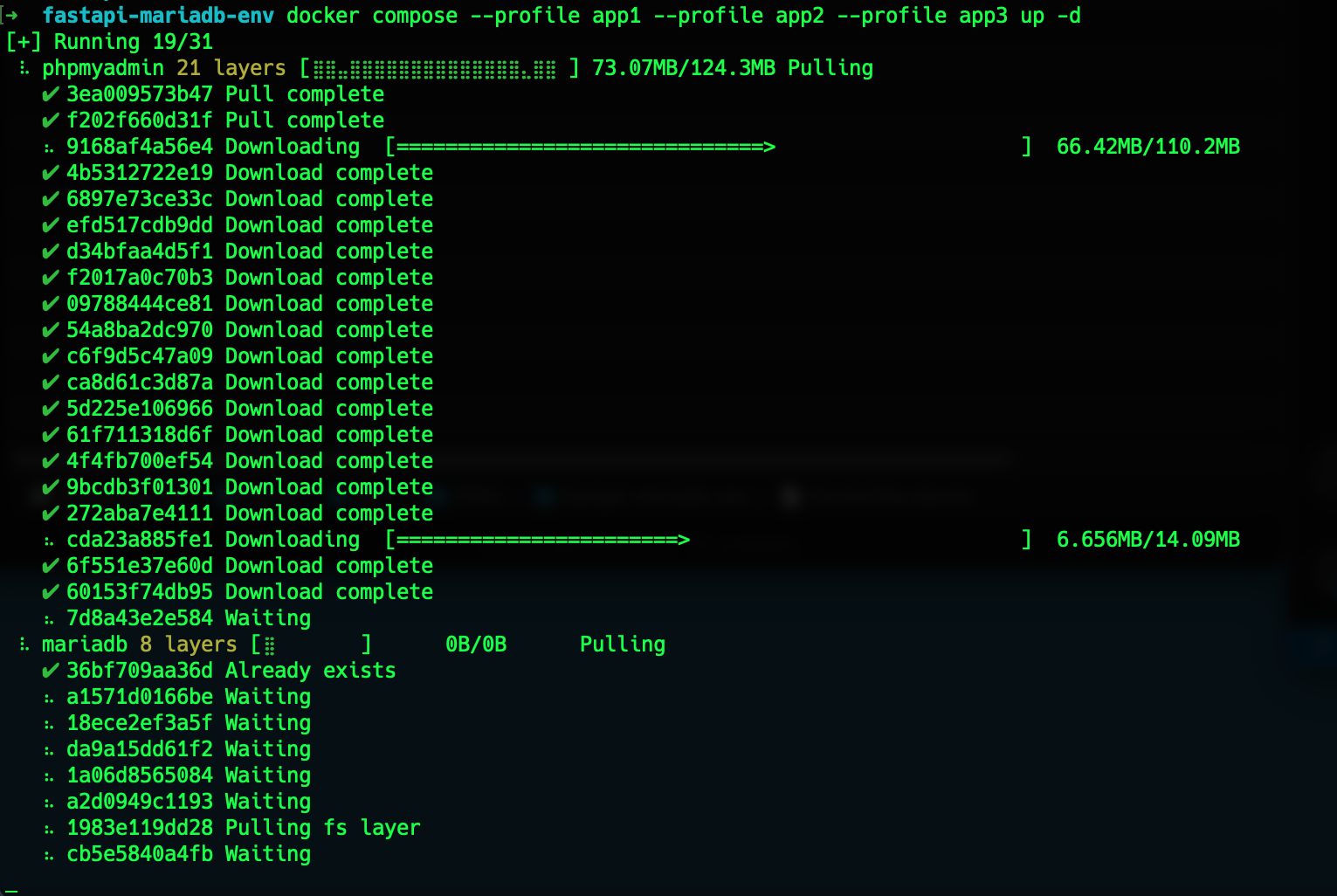
## Quick summary

* **Build** → only builds your FastAPI images (build: services)
* **Pull/Up** → fetches images for infra (image: services)

So yes: it’s expected that only the 3 FastAPI images show up after build.

Run

**docker compose --profile app1 --profile app2 --profile app3 up -d**



✔ phpmyadmin 21 layers [**⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿**] 0B/0B Pulled 28.4s

✔ 3ea009573b47 Pull complete 10.4s

✔ f202f660d31f Pull complete 10.5s

✔ 9168af4a56e4 Pull complete 21.5s

✔ 4b5312722e19 Pull complete 21.6s

✔ 6897e73ce33c Pull complete 21.9s

✔ efd517cdb9dd Pull complete 22.0s

✔ d34bfaa4d5f1 Pull complete 22.1s

✔ f2017a0c70b3 Pull complete 22.3s

✔ 09788444ce81 Pull complete 22.4s

✔ 54a8ba2dc970 Pull complete 23.0s

✔ c6f9d5c47a09 Pull complete 23.2s

✔ ca8d61c3d87a Pull complete 23.3s

✔ 5d225e106966 Pull complete 23.5s

✔ 61f711318d6f Pull complete 23.7s

✔ 4f4fb700ef54 Pull complete 23.8s

✔ 9bcdb3f01301 Pull complete 24.2s

✔ 272aba7e4111 Pull complete 24.2s

✔ cda23a885fe1 Pull complete 25.8s

✔ 6f551e37e60d Pull complete 26.0s

✔ 60153f74db95 Pull complete 26.1s

✔ 7d8a43e2e584 Pull complete 26.2s

✔ mariadb 8 layers [**⣿⣿⣿⣿⣿⣿⣿⣿**] 0B/0B Pulled 29.8s

✔ 36bf709aa36d Already exists 0.0s

✔ a1571d0166be Pull complete 13.6s

✔ 18ece2ef3a5f Pull complete 20.5s

✔ da9a15dd61f2 Pull complete 20.7s

✔ 1a06d8565084 Pull complete 20.8s

✔ a2d0949c1193 Pull complete 27.6s

✔ 1983e119dd28 Pull complete 27.6s

✔ cb5e5840a4fb Pull complete 27.7s

[+] Running 7/7

✔ Network fastapi\_proj1\_default Created 0.1s

✔ Volume "fastapi\_proj1\_mariadb\_data" Created 0.0s

✔ Container fastapi\_proj1-mariadb-1 Healthy 6.6s

✔ Container fastapi\_proj1-app2-1 Started 6.8s

✔ Container fastapi\_proj1-phpmyadmin-1 Started 6.3s

✔ Container fastapi\_proj1-app1-1 Started 6.7s

✔ Container fastapi\_proj1-app3-1 Started 6.5s

**➜ fastapi-mariadb-env**

Check Images again:

**➜ fastapi-mariadb-env** docker images -a

REPOSITORY TAG IMAGE ID CREATED SIZE

fastapi\_proj1-app2 latest f20dbb1b5bcb 10 minutes ago 901MB

fastapi\_proj1-app3 latest 2e4be68a956b 10 minutes ago 901MB

fastapi\_proj1-app1 latest ef347266af4a 10 minutes ago 901MB

mariadb 11 7ba0489760b3 3 days ago 366MB

phpmyadmin latest 8568c32212e9 10 days ago 589MB

net2-frontend latest 9442174d7584 8 months ago 1.16GB

net2-backend latest 5c25a88f114b 8 months ago 496MB

net2-cdn latest d3bfceeed7d4 8 months ago 157MB

<none> <none> a6a656c431fb 8 months ago 1.16GB

<none> <none> b00117b3062d 8 months ago 496MB

<none> <none> 2ef2e9cac715 8 months ago 157MB

<none> <none> e1fbff02f6ef 8 months ago 1.16GB

<none> <none> 0306fd253d54 8 months ago 496MB

<none> <none> 6649fb72f9fb 8 months ago 157MB

<none> <none> 65c187e03cd1 8 months ago 1.16GB

<none> <none> 0f627570dc52 8 months ago 496MB

<none> <none> 62bbc84e74e1 8 months ago 157MB

<none> <none> 06ed8285aebc 8 months ago 1.04GB

<none> <none> ffc745a4e6ef 8 months ago 496MB

<none> <none> 9154e26338d3 8 months ago 157MB

<none> <none> 47a29a0c48d1 14 months ago 157MB

<none> <none> d875744cbe35 14 months ago 1.04GB

<none> <none> 03e199a5c559 14 months ago 1.04GB

<none> <none> 4c2aed1733f7 14 months ago 1.04GB

<none> <none> 5203bb0fbf26 14 months ago 1.04GB

<none> <none> 8ddf668bcdfb 14 months ago 157MB

<none> <none> 4ab0c41bd871 15 months ago 1.04GB

<none> <none> e11a29126622 15 months ago 1.04GB

<none> <none> 209b313e14f1 15 months ago 1.04GB

<none> <none> c3a0e37b5714 15 months ago 1.04GB

<none> <none> 715b4ce61c3f 15 months ago 1.04GB

<none> <none> b8abd6591862 15 months ago 1.01GB

<none> <none> a056ec402d98 16 months ago 497MB

<none> <none> 7c914e8f3cdf 17 months ago 157MB

mariadb2 latest 00396b1d06b9 17 months ago 414MB

debian12gen\_front latest 372c902f7881 17 months ago 512MB

debian12gen\_back latest 41023e2ade13 17 months ago 1.05GB

debian12gen1\_image1 latest 17c1f948d270 17 months ago 1.08GB

arm64v8/mariadb 11.5.2-ubi9 b2c11b5a9e36 17 months ago 471MB

arm64v8/nginx bookworm 195245f0c792 18 months ago 193MB

debian12ml1\_image latest e4ec3651b822 20 months ago 757MB

debian12selen1 latest da09e3b3afaf 20 months ago 1.83GB

selenium/standalone-docker nightly 9f2999f5c8c1 20 months ago 383MB

deb-selen-web-scraper1 latest ac0299368fc8 20 months ago 1.45GB

selenium/standalone-chromium latest 246989a93104 21 months ago 1.4GB

ub2404py1\_python\_2 latest 9c557032d183 21 months ago 8.34GB

ub2404py1\_python\_init latest ecfbb1cadd7f 21 months ago 6.39GB

arm64v8/ubuntu latest fabf3a8d4949 21 months ago 98.8MB

seleniarm/standalone-chromium latest 3ce86da2ae50 21 months ago 1.86GB

postgres alpine3.17 d9d449b64180 2 years ago 241MB

alpine 3.16.3 2b4661558fb8 3 years ago 5.29MB

docker/disk-usage-extension 0.2.5 8f028a5e62df 3 years ago 2.79MB

mariadb latest daf0f023c28d 3 years ago 414MB

mplatform/mquery latest 3441ed415baf 4 years ago 6.78MB

ords 18.2.0 f4fb6eb4e002 7 years ago 195MB

<none> <none> 86d4182b3c4f 7 years ago 195MB

<none> <none> 72f422cd807c 7 years ago 195MB

<none> <none> b0ad34550f18 7 years ago 195MB

<none> <none> 0e51a1265c0c 7 years ago 195MB

<none> <none> e53dc4cbdce3 7 years ago 195MB

<none> <none> 2d90da7766ca 7 years ago 139MB

<none> <none> 9347b058e0d8 7 years ago 83MB

<none> <none> 843ead7354d7 7 years ago 83MB

<none> <none> 023c3877702b 7 years ago 83MB

ol7\_ords latest e1ab4e7cd2f7 7 years ago 1.54GB

<none> <none> b0b6add5527a 7 years ago 1.54GB

<none> <none> bfc2d619bf72 7 years ago 1.54GB

<none> <none> dea5ae39f004 7 years ago 1.54GB

<none> <none> ff9ae13800f5 7 years ago 1.54GB

<none> <none> 831842487c34 7 years ago 656MB

<none> <none> dc783449ce07 7 years ago 656MB

<none> <none> 179ea3725c9e 7 years ago 117MB

<none> <none> 9bb79bd71c43 7 years ago 117MB

<none> <none> b7bef7a17c5f 7 years ago 117MB

<none> <none> f133fa93b2ce 7 years ago 5.97GB

oracle/database 12.2.0.1-ee 0de151257bd3 7 years ago 5.97GB

<none> <none> fdef04177bf2 7 years ago 5.97GB

<none> <none> 0d6c75ef9741 7 years ago 5.97GB

<none> <none> c8fe2d40cfda 7 years ago 5.97GB

<none> <none> b90009a9618a 7 years ago 5.97GB

<none> <none> 69ba041a9570 7 years ago 5.97GB

<none> <none> 34329b184f21 7 years ago 5.96GB

<none> <none> a4f32afa5d59 7 years ago 5.96GB

<none> <none> acfa47bf99b5 7 years ago 288MB

<none> <none> ea6a4e8c80d7 7 years ago 288MB

<none> <none> 516746a46fef 7 years ago 117MB

<none> <none> 031e1a9d6981 7 years ago 117MB

<none> <none> e46b1878016b 7 years ago 117MB

<none> <none> 7e802e811df9 7 years ago 117MB

<none> <none> 2e08e361e48b 7 years ago 117MB

<none> <none> 0fe3f0d1ee48 7 years ago 83MB

hello-world latest 4ab4c602aa5e 7 years ago 1.84kB

<none> <none> 0e3d3a8c0f61 7 years ago 117MB

**➜ fastapi-mariadb-env**

Check (running) containers:

**➜ fastapi-mariadb-env** docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1a915b840e2f fastapi\_proj1-app3 "bash -lc 'cd /works…" 2 minutes ago Up 2 minutes 0.0.0.0:8002->8000/tcp fastapi\_proj1-app3-1

843b63de5163 fastapi\_proj1-app1 "bash -lc 'cd /works…" 2 minutes ago Up 2 minutes 0.0.0.0:8000->8000/tcp fastapi\_proj1-app1-1

a901fd271eb0 fastapi\_proj1-app2 "bash -lc 'cd /works…" 2 minutes ago Up 2 minutes 0.0.0.0:8001->8000/tcp fastapi\_proj1-app2-1

a662bcad58e0 phpmyadmin:latest "/docker-entrypoint.…" 2 minutes ago Up 2 minutes 0.0.0.0:8080->80/tcp fastapi\_proj1-phpmyadmin-1

9197707dfeb7 mariadb:11 "docker-entrypoint.s…" 2 minutes ago Up 2 minutes (healthy) 0.0.0.0:3307->3306/tcp fastapi\_proj1-mariadb-1

**➜ fastapi-mariadb-env**

**➜ fastapi-mariadb-env** docker compose ps

NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS

fastapi\_proj1-app1-1 fastapi\_proj1-app1 "bash -lc 'cd /works…" app1 9 minutes ago Up 9 minutes 0.0.0.0:8000->8000/tcp

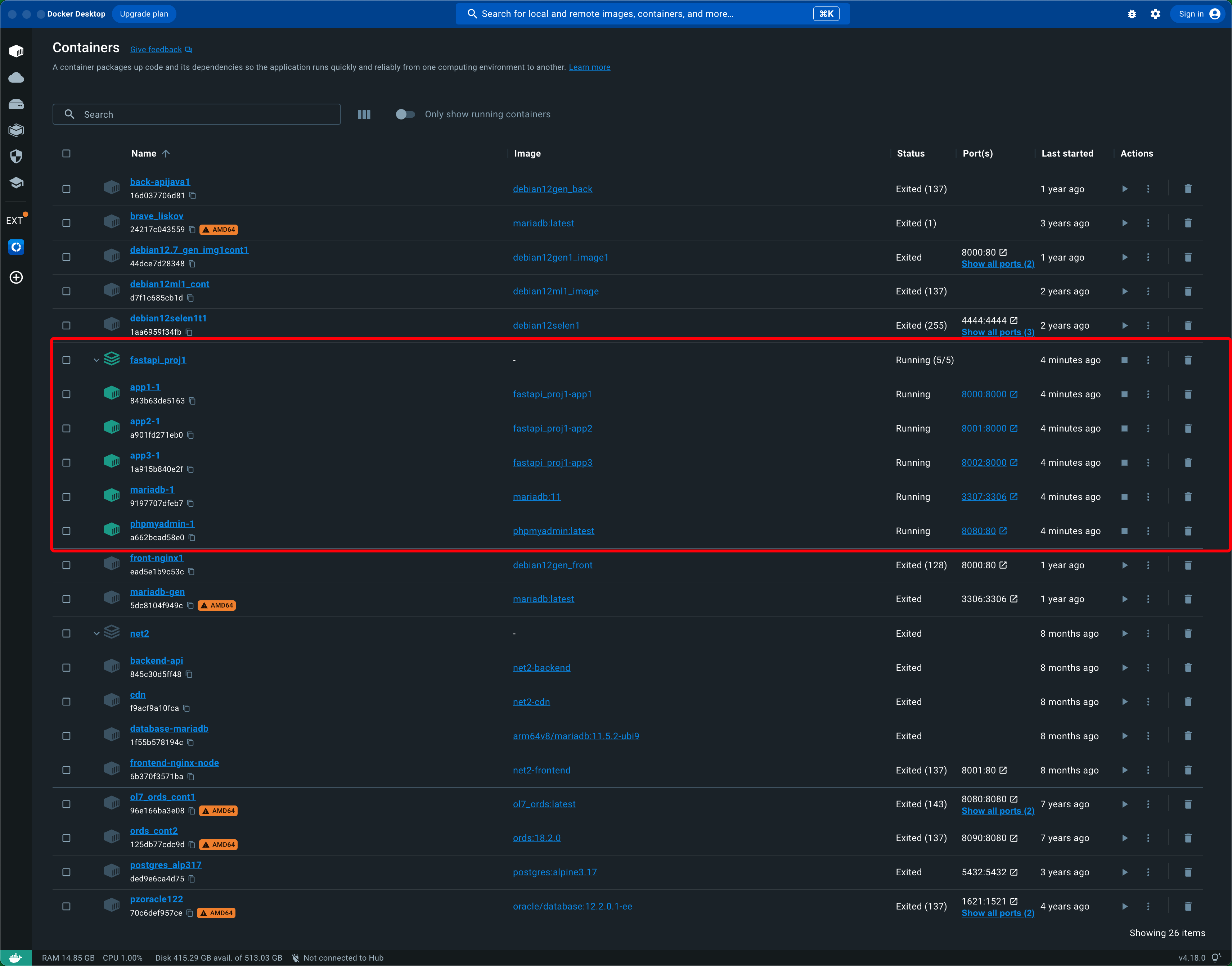
fastapi\_proj1-app2-1 fastapi\_proj1-app2 "bash -lc 'cd /works…" app2 9 minutes ago Up 9 minutes 0.0.0.0:8001->8000/tcp

fastapi\_proj1-app3-1 fastapi\_proj1-app3 "bash -lc 'cd /works…" app3 9 minutes ago Up 9 minutes 0.0.0.0:8002->8000/tcp

fastapi\_proj1-mariadb-1 mariadb:11 "docker-entrypoint.s…" mariadb 9 minutes ago Up 9 minutes (healthy) 0.0.0.0:3307->3306/tcp

fastapi\_proj1-phpmyadmin-1 phpmyadmin:latest "/docker-entrypoint.…" phpmyadmin 9 minutes ago Up 9 minutes 0.0.0.0:8080->80/tcp

**➜ fastapi-mariadb-env**



Creating an Initial Git repo

gh status

git init

git branch -M master

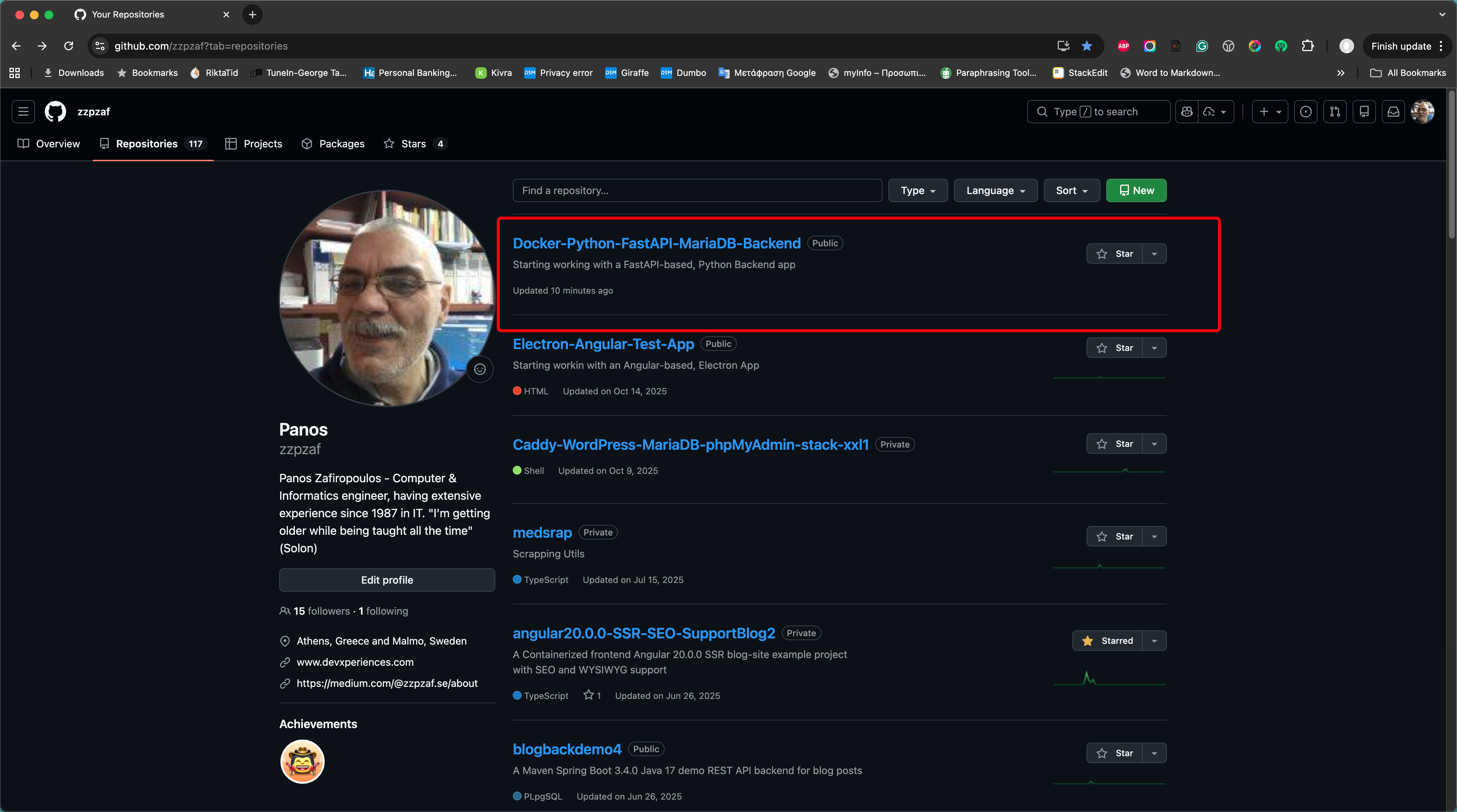
gh repo create 'Docker-Python-FastAPI-MariaDB-Backend' --description 'Starting working with a FastAPI-based, Python Backend app' --public

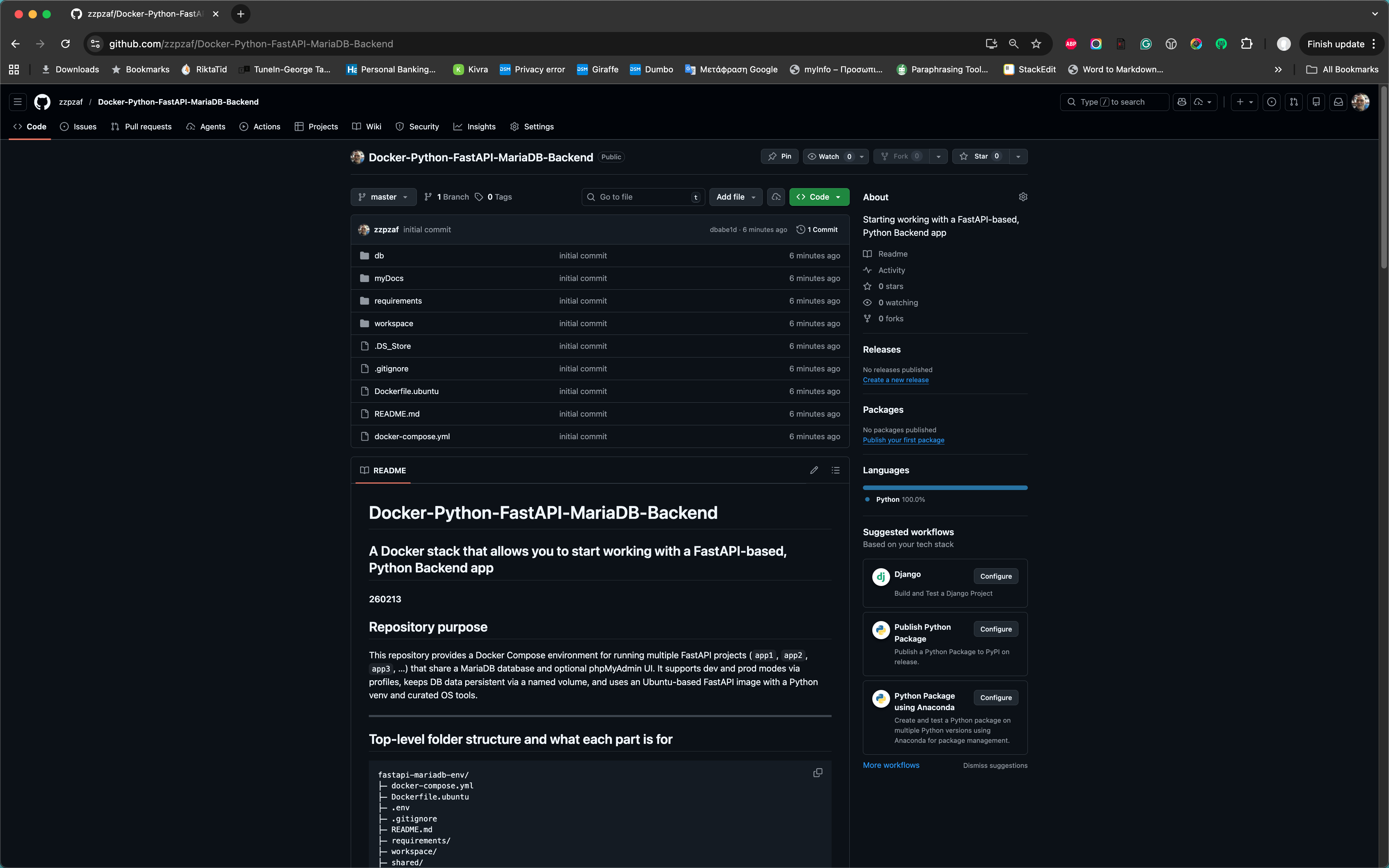
git remote add origin https://github.com/zzpzaf/Docker-Python-FastAPI-MariaDB-Backend.git

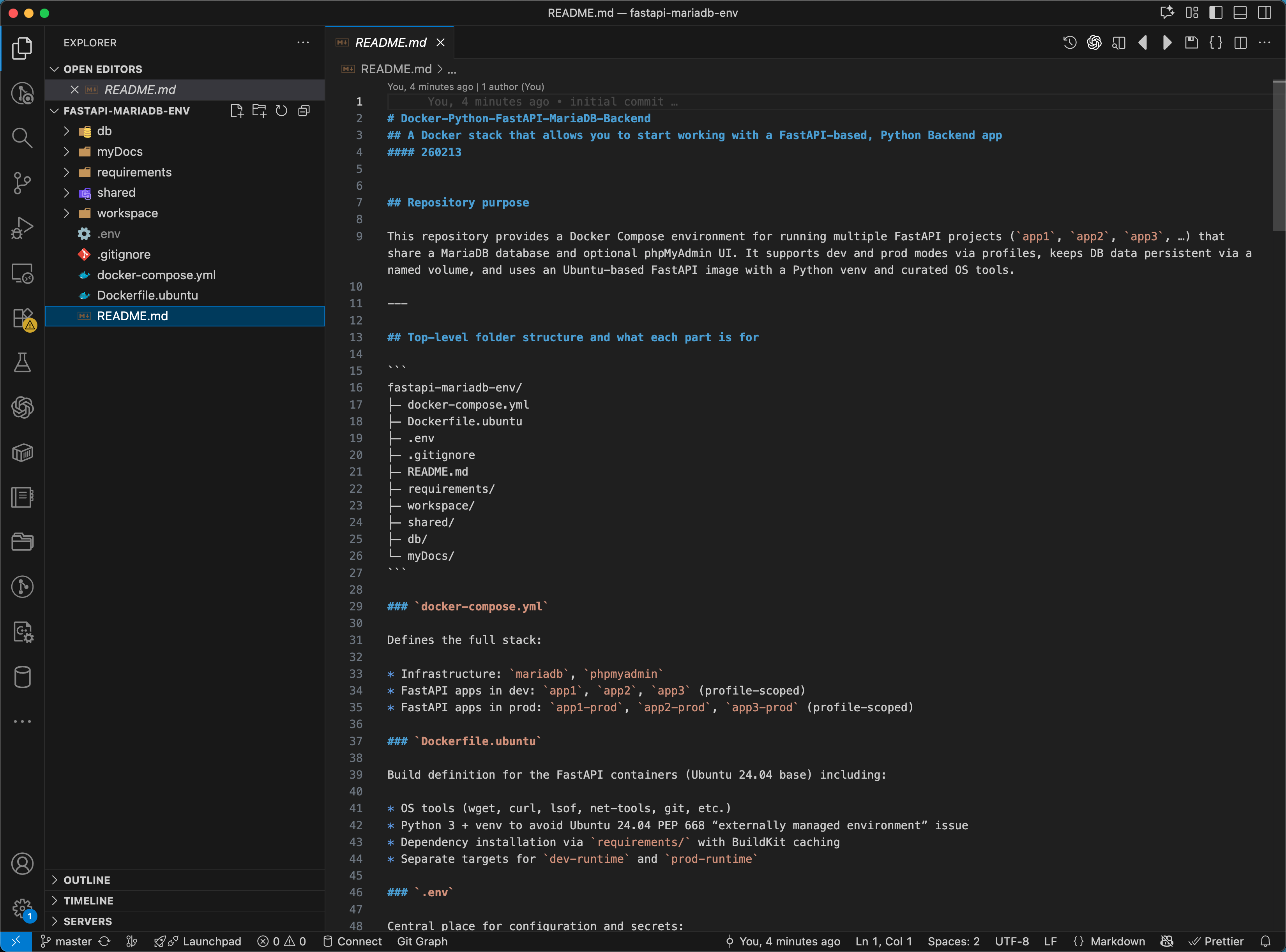
git add .

git commit -m "initial commit"

git push --set-upstream origin master







Initial schema and initial data insertion

**How the db/init/ is commonly used with MariaDB in Docker**

MariaDB’s official Docker image supports automatic execution of .sql scripts placed in db/init/, such as:

001\_schema.sql – It is used to define database schemas, tables, etc.,

002\_seed.sql – It inserts initial (demo) data into tables

So, the db/init/ → is used for placing first-time initialization scripts. MariaDB will run those scripts only the first time the database volume is created (i.e., on initial initialization).

In order to fire the initial execution process of the scripts, the following line should be included in the mariadb volumes section, in docker-compose.yml under mariadb: volumes:

- ./db/init:/docker-entrypoint-initdb.d:ro



001\_schema.sql

/\*

----------------------------------------------------------------------------

File name: db/init/001\_schema.sql

Bookstore Demo DB Tables and Data Objects for MariaDB/MySQL

Requires:

- MariaDB 10.2.1+ OR MySQL 8.0.13+

(uses DEFAULT (UUID()) and CURRENT\_TIMESTAMP(6))

(C) Copyright Panos Zafeiropoulos - 2022-26

-----------------------------------------------------------------------------

Updates:

260213: Tables: 3

Triggers: 0

Procedures: 0

Triggers: 0

Views: 0

----------------------------------------------------------------------------

Last update: 260213

----------------------------------------------------------------------------

Important reminder:

--------------------

When MariaDB initializes for the first time, it executes:

001\_schema.sql

002\_seed.sql

Seed scripts in /docker-entrypoint-initdb.d run only when mariadb\_data volume is empty

If you already started MariaDB once:

First, stop and remove the containers, network(s) and the volume(s) created by docker compose up:

docker compose --profile app1 --profile app2 --profile app3 down --volumes

Then, start the MariaDB container again:

docker compose --profile app1 --profile app2 --profile app3 up -d

Thi is required to re-run init scripts.

\*/

-- --------------------------------------------------------

-- CREATE DATABASE bookstore1

-- --------------------------------------------------------

CREATE DATABASE IF NOT EXISTS bookstore1

CHARACTER SET utf8mb4

COLLATE utf8mb4\_unicode\_ci;

USE bookstore1;

-- --------------------------------------------------------

-- Table: categories

-- --------------------------------------------------------

CREATE TABLE IF NOT EXISTS categories (

categoryId INT UNSIGNED NOT NULL AUTO\_INCREMENT,

categoryName VARCHAR(100) NOT NULL,

categoryStatusId SMALLINT UNSIGNED NOT NULL DEFAULT 1,

categoryCrUUID CHAR(36) NOT NULL DEFAULT (UUID()),

categoryCrTimestamp TIMESTAMP(6) NOT NULL DEFAULT CURRENT\_TIMESTAMP(6),

categoryClientUUID VARCHAR(40) NULL,

PRIMARY KEY (categoryId),

UNIQUE KEY uq\_categories\_name (categoryName)

) ENGINE=InnoDB;

-- --------------------------------------------------------

-- Table: items

-- (many-to-many: no category id here)

-- --------------------------------------------------------

CREATE TABLE IF NOT EXISTS items (

itemId INT UNSIGNED NOT NULL AUTO\_INCREMENT,

itemName VARCHAR(100) NOT NULL,

itemListPrice DECIMAL(10,2) NOT NULL,

itemModelYear SMALLINT UNSIGNED NULL,

itemStatusId SMALLINT UNSIGNED NOT NULL DEFAULT 1,

itemCrUUID CHAR(36) NOT NULL DEFAULT (UUID()),

itemCrTimestamp TIMESTAMP(6) NOT NULL DEFAULT CURRENT\_TIMESTAMP(6),

itemClientUUID VARCHAR(40) NULL,

PRIMARY KEY (itemId),

KEY ix\_items\_name (itemName)

) ENGINE=InnoDB;

-- --------------------------------------------------------

-- Table: categoryitems (junction table)

-- --------------------------------------------------------

CREATE TABLE IF NOT EXISTS categoryitems (

categoryitemId INT UNSIGNED NOT NULL AUTO\_INCREMENT,

categoryitemCategoryId INT UNSIGNED NOT NULL,

categoryitemItemId INT UNSIGNED NOT NULL,

categoryitemCrUUID CHAR(36) NOT NULL DEFAULT (UUID()),

categoryitemCrTimestamp TIMESTAMP(6) NOT NULL DEFAULT CURRENT\_TIMESTAMP(6),

categoryitemClientUUID VARCHAR(40) NULL,

PRIMARY KEY (categoryitemId),

-- Prevent duplicates like (category=5,item=10) appearing twice

UNIQUE KEY uq\_categoryitems\_pair (categoryitemCategoryId, categoryitemItemId),

-- Helpful indexes for joins

KEY ix\_categoryitems\_categoryId (categoryitemCategoryId),

KEY ix\_categoryitems\_itemId (categoryitemItemId),

CONSTRAINT fk\_categoryitems\_category

FOREIGN KEY (categoryitemCategoryId)

REFERENCES categories(categoryId)

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT fk\_categoryitems\_item

FOREIGN KEY (categoryitemItemId)

REFERENCES items(itemId)

ON UPDATE CASCADE

ON DELETE CASCADE

) ENGINE=InnoDB;

002\_seed.sql

/\*

----------------------------------------------------------------------------

File name: db/init/002\_seed.sql

Bookstore Demo DB Tables and Data Objects for MariaDB/MySQL

-----------------------------------------------------------------------------

Updates:

260213: Initial entries

Categories: 7

Items: 20

Many-to-many relationships via categoryitems

----------------------------------------------------------------------------

Last update: 260213

----------------------------------------------------------------------------

Important reminder:

--------------------

When MariaDB initializes for the first time, it executes:

001\_schema.sql

002\_seed.sql

Seed scripts in /docker-entrypoint-initdb.d run only when mariadb\_data volume is empty

If you already started MariaDB once:

First, stop and remove the containers, network(s) and the volume(s) created by docker compose up:

docker compose --profile app1 --profile app2 --profile app3 down --volumes

Then, start the MariaDB container again:

docker compose --profile app1 --profile app2 --profile app3 up -d

Thi is required to re-run init scripts.

\*/

-- --------------------------------------------------------

-- ADD RECORDS TO TABLES

-- --------------------------------------------------------

USE bookstore1;

-- --------------------------------------------------------

-- Insert Categories

-- --------------------------------------------------------

INSERT INTO categories (categoryName) VALUES

('Fiction'),

('Science'),

('Technology'),

('History'),

('Business'),

('Philosophy'),

('Education');

-- --------------------------------------------------------

-- Insert Items (20 entries)

-- --------------------------------------------------------

INSERT INTO items (itemName, itemListPrice, itemModelYear) VALUES

('The Quantum World', 29.90, 2022),

('Python Backend Essentials', 45.00, 2024),

('World War II Atlas', 39.50, 2020),

('Modern Sci-Fi Stories', 22.00, 2023),

('AI Fundamentals', 49.00, 2023),

('Deep Learning Explained', 55.00, 2021),

('Startup Strategy', 27.50, 2022),

('Lean Business Models', 31.20, 2019),

('Ancient Philosophy Reader', 24.00, 2018),

('Ethics in Technology', 28.90, 2024),

('Database Design Mastery', 42.00, 2023),

('Docker in Practice', 36.00, 2022),

('Distributed Systems Guide', 47.80, 2024),

('Physics for Everyone', 18.50, 2017),

('Educational Psychology', 26.00, 2021),

('Teaching Programming', 33.00, 2020),

('Cybersecurity Basics', 40.00, 2023),

('Historical Maps Collection', 30.00, 2016),

('Science Fiction Classics', 21.50, 2015),

('The Logic Handbook', 19.90, 2014);

-- --------------------------------------------------------

-- Many-to-many relations

-- --------------------------------------------------------

-- Science

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Science' AND i.itemName IN (

'The Quantum World','Physics for Everyone','Deep Learning Explained'

);

-- Technology

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Technology' AND i.itemName IN (

'Python Backend Essentials','AI Fundamentals','Database Design Mastery',

'Docker in Practice','Distributed Systems Guide','Cybersecurity Basics',

'Ethics in Technology'

);

-- History

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='History' AND i.itemName IN (

'World War II Atlas','Historical Maps Collection'

);

-- Fiction

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Fiction' AND i.itemName IN (

'Modern Sci-Fi Stories','Science Fiction Classics'

);

-- Business

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Business' AND i.itemName IN (

'Startup Strategy','Lean Business Models'

);

-- Philosophy

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Philosophy' AND i.itemName IN (

'Ancient Philosophy Reader','The Logic Handbook','Ethics in Technology'

);

-- Education

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId FROM categories c JOIN items i

WHERE c.categoryName='Education' AND i.itemName IN (

'Educational Psychology','Teaching Programming'

);

-- --------------------------------------------------------

-- Example cross-category assignments (true many-to-many)

-- --------------------------------------------------------

-- Modern Sci-Fi Stories belongs also to Technology

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId

FROM categories c JOIN items i

WHERE c.categoryName='Technology' AND i.itemName='Modern Sci-Fi Stories';

-- AI Fundamentals belongs also to Science

INSERT INTO categoryitems (categoryitemCategoryId, categoryitemItemId)

SELECT c.categoryId, i.itemId

FROM categories c JOIN items i

WHERE c.categoryName='Science' AND i.itemName='AI Fundamentals';

260214