FREE · LEARN.AIWITHPRIYABRAT.COM

Docker for AI & Python Developers

Containerise Python and LLM apps the right way: production Dockerfiles, Compose for an AI stack, GPUs, Ollama and vLLM, model files, troubleshooting and security.

By **Priyabrat Dalbehera** · GenAI engineer

4 pages | 6 parts | 36 commands, files & checklists | 6 ready-to-copy files

What's inside

1 **Basics** — images, containers, run flags, everyday commands — p. 2

2 **Dockerfiles** — a production Dockerfile, caching, .dockerignore, multi-stage, size — p. 2

3 **Data, networks & Compose** — volumes, networking, compose.yaml for an AI app, secrets — p. 3

4 **Docker for AI** — GPUs, Ollama, vLLM, model files, vector DBs, Jupyter — p. 3

5 **Debug, secure & deploy** — 7 common errors, cleanup, security, CI, where to deploy — p. 4

6 **Interview answers** — containers vs VMs, 8 quick answers — p. 4

How to read it
Lines starting with `$` are commands. Grey file names label files to copy.

Tip & Watch out
Green boxes are good habits. Amber boxes are traps that leak secrets or lose data.

Works everywhere
Docker Desktop on Windows and macOS, Docker Engine on Linux.

learn.aiwithpriyabrat.com

© 2026 Priyabrat Dalbehera · Free to download and share unchanged, with credit · Not for resale

Docker for AI & Python Developers

Parts 1–2 · images and containers · run flags · everyday commands · Dockerfile · caching · multi-stage · size

1.1 The words you need

Image	a read-only package: OS files + Python + your code + dependencies
Container	a running instance of an image, isolated from other processes
Dockerfile	the recipe that builds an image, one layer per instruction
Registry	where images are stored: Docker Hub, GHCR, ECR, Artifact Registry
Volume	storage that survives when the container is deleted
Compose	runs several containers together from one YAML file

1.2 Install and check

```
$ docker version
$ docker run --rm hello-world
$ docker info | grep -i "server version"
```

Windows and macOS: Docker Desktop (on Windows it uses WSL 2). Linux: Docker Engine from your package manager; add yourself to the `docker` group to run without `sudo`.

1.3 A container's life

```
$ docker run -d --name web -p 8080:80 nginx
$ docker ps                # running containers
$ docker stop web
$ docker start web
$ docker rm -f web         # stop + delete
```

1.4 docker run flags

`-d`	run in the background
`-it`	interactive terminal
`--rm`	delete the container when it stops
`-p 8000:8000`	host port : container port
`-e KEY=value` · `--env-file .env`	environment variables
`-v data:/app/data`	mount a volume or a folder
`--name api`	a name instead of a random one
`--restart unless-stopped`	come back after crashes and reboots
`--memory 2g` · `--cpus 1.5`	resource limits

1.5 Everyday commands

`docker ps -a`	all containers, including stopped
`docker images`	local images and their sizes
`docker logs -f api`	follow a container's output
`docker exec -it api sh`	a shell inside a running container
`docker stats`	live CPU and memory per container
`docker cp api:/app/out.csv .`	copy a file out

2.1 A production Dockerfile for a FastAPI app

Dockerfile

```
FROM python:3.12-slim
ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
RUN useradd --create-home appuser
USER appuser
EXPOSE 8000
CMD ["uvicorn", "app.main:app", \
     "--host", "0.0.0.0", "--port", "8000"]
```

Tip: `--host 0.0.0.0` is required: an app listening on 127.0.0.1 can't be reached from outside the container.

2.2 Dockerfile instructions

`FROM`	the base image
`WORKDIR`	the working folder (created if missing)
`COPY`	copy files in (prefer it over `ADD`)
`RUN`	run a command at build time
`ENV` / `ARG`	variables at run time / build time only
`USER`	run as a non-root user
`EXPOSE`	documents the port (doesn't publish it)
`CMD`	default command; replaced by `docker run … cmd`
`ENTRYPOINT`	fixed command; CMD becomes its arguments
`HEALTHCHECK`	how Docker tests that the app is alive

2.3 Fast rebuilds: layer caching

- Each instruction is a cached layer; a change rebuilds it and **every layer after it**.
- Copy `requirements.txt` and install **before** `COPY . .`, so code edits don't reinstall packages.
- Order from least to most often changed.

2.4 .dockerignore

.dockerignore

```
.venv/
__pycache__/
.git/
.env
data/
models/
*.ipynb_checkpoints
```

Watch out: without it, `COPY . .` sends your secrets, datasets and virtualenv into the image.

2.5 Build, tag and push

```
$ docker build -t chatbot:1.0 .
$ docker tag chatbot:1.0 ghcr.io/asha/chatbot:1.0
$ docker push ghcr.io/asha/chatbot:1.0
$ docker buildx build --platform \
    linux/amd64,linux/arm64 -t <tag> --push .
```

Building on an Apple Silicon Mac for a Linux x86 server? Build for `linux/amd64`.

2.6 Multi-stage build with uv

Dockerfile

```
FROM python:3.12-slim AS build
RUN pip install --no-cache-dir uv
WORKDIR /app
COPY pyproject.toml uv.lock ./
RUN uv sync --frozen --no-dev --no-install-project

FROM python:3.12-slim
WORKDIR /app
COPY --from=build /app/.venv /app/.venv
COPY . .
ENV PATH="/app/.venv/bin:$PATH"
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0"]
```

Build tools stay in the first stage; the final image gets only the installed packages.

2.7 Smaller images

`python:3.12-slim`	good default; avoid `alpine` for Python (wheels often fail)
CPU-only PyTorch	`pip install torch --index-url https://download.pytorch.org/whl/cpu` saves GBs
`--no-cache-dir`	don't keep pip's download cache in the image
Models outside the image	mount or download at start (4.4)

learn.aiwithpriyabrat.com

3.1 Keep data: volumes and bind mounts

Type	Example	Use for
Named volume	`-v pgdata:/var/lib/postgresql/data`	databases, model caches
Bind mount	`-v ./src:/app/src`	live code reloads in development
tmpfs	`--tmpfs /tmp`	fast scratch space in memory

```
$ docker volume ls
$ docker volume inspect pgdata
```

3.2 Containers talking to each other

- Containers on the same network reach each other by **name**: `redis://redis:6379`.
- Compose puts all services on one network automatically.
- `localhost` inside a container is the container itself, not your computer.
- Reach your computer from a container: `host.docker.internal` (Docker Desktop).

3.3 compose.yaml for an AI app

compose.yaml

```
services:
  api:
    build: .
    ports: ["8000:8000"]
    env_file: .env
    depends_on:
      redis: {condition: service_healthy}
    restart: unless-stopped
  redis:
    image: redis:7-alpine
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
  qdrant:
    image: qdrant/qdrant
    volumes: ["qdrant_data:/qdrant/storage"]
volumes:
  qdrant_data:
```

3.4 Compose commands

```
$ docker compose up -d --build
$ docker compose ps
$ docker compose logs -f api
$ docker compose exec api sh
$ docker compose down          # keeps volumes
$ docker compose down -v       # deletes volumes!
```

3.5 Secrets and environment

- Pass secrets at **run time**: `--env-file`, `env_file:` or your platform's secret store.
- Never `COPY .env` or set API keys with `ENV`: anyone with the image can read them.
- Build-time secrets (private packages): `RUN --mount=type=secret`.
- Commit a `.env.example` with empty values.

3.6 Hot reload while you develop

compose.override.yaml

```
services:
  api:
    volumes: ["./app:/app/app"]     # your code, live
    command: uvicorn app.main:app --host 0.0.0.0 --reload
```

Compose merges `compose.override.yaml` automatically: production settings stay in `compose.yaml`, development tweaks live here.

4.1 Use the GPU

```
$ docker run --rm --gpus all \
    nvidia/cuda:12.4.1-base-ubuntu22.04 nvidia-smi
```

Needs an NVIDIA driver on the host and the NVIDIA Container Toolkit. In Compose: `deploy.resources.reservations.devices` with `capabilities: [gpu]`.

4.2 Run a local LLM with Ollama

```
$ docker run -d --name ollama -p 11434:11434 \
    -v ollama:/root/.ollama ollama/ollama
$ docker exec -it ollama ollama run llama3.2
```

The API is then at `http://localhost:11434`; add `--gpus all` on a GPU machine.

4.3 Serve an open model with vLLM

```
$ docker run --gpus all --ipc=host -p 8000:8000 \
    -v ~/.cache/huggingface:/root/.cache/huggingface \
    -e HF_TOKEN \
    vllm/vllm-openai --model Qwen/Qwen2.5-1.5B-Instruct
```

It serves an **OpenAI-compatible** API on port 8000, so the OpenAI client works with a changed `base_url`.

4.4 Model files: in the image or not?

Mount a cache volume	small image, fast deploys; first start downloads the model
Bake into the image	fast, offline cold starts; huge image, slow pushes

Tip: set `HF_HOME` to a mounted folder so every container shares one download.

4.5 Health checks for slow-starting models

Dockerfile

```
HEALTHCHECK --interval=30s --start-period=120s \
  CMD curl -f http://localhost:8000/health || exit 1
```

`--start-period` gives the model time to load before failures count. The base image must have `curl`, or use a small Python check.

4.6 Vector databases in one command

```
$ docker run -d -p 6333:6333 qdrant/qdrant
$ docker run -d -p 8001:8000 chromadb/chroma
$ docker run -d -p 5432:5432 -e POSTGRES_PASSWORD=dev \
    pgvector/pgvector:pg16
```

Tip: add `-v` with a named volume, or the vectors are gone when the container is removed.

4.7 Jupyter in a container

```
$ docker run -p 8888:8888 \
    -v "$PWD":/home/jovyan/work \
    quay.io/jupyter/scipy-notebook
```

Open the link with the token from the logs. Notebooks saved in `work/` land in your folder.

learn.aiwithpriyabrat.com

Docker for AI & Python Developers

Parts 5–6 · troubleshooting · disk space · security · CI · containers vs VMs · interview answers

Parts 5–6 · Debug, secure & interview

5.1 Troubleshooting

Problem	Fix
port is already allocated	another process uses it: change the host port (`-p 8001:8000`)
Container exits at once	`docker logs` ; the main process must stay in the foreground
Exit code **137**	killed for using too much memory: raise `--memory` or use a smaller model
exec format error	wrong CPU architecture: build with `--platform linux/amd64`
Works locally, not in Docker	listen on `0.0.0.0` ; check the `-p` mapping
Can't reach the database	use the service name, not `localhost`
Permission denied on a volume	the container's user ID doesn't own the folder

5.2 Look inside a container

```
$ docker logs --tail 100 -f api
$ docker exec -it api sh
$ docker inspect api --format "{{.State.ExitCode}}"
$ docker top api
```

5.3 Free disk space

```
$ docker system df            # what uses space
$ docker container prune      # stopped containers
$ docker image prune -a       # unused images
$ docker builder prune        # build cache
```

Watch out: `docker volume prune` and `docker system prune --volumes` delete data: databases, model caches.

5.4 Security checklist

- ☐ Run as a **non-root** user (`USER`)
- ☐ Pin versions: `python:3.12.7-slim` , not `latest`
- ☐ No secrets in the image or its layers
- ☐ Small base image, few packages
- ☐ Scan images: `docker scout cves` or Trivy
- ☐ Rebuild regularly to pick up security fixes
- ☐ Set memory and CPU limits

5.5 Build and push from GitHub Actions

.github/workflows/docker.yml

```
- uses: docker/login-action@v3
  with:
    registry: ghcr.io
    username: ${{ github.actor }}
    password: ${{ secrets.GITHUB_TOKEN }}
- uses: docker/build-push-action@v6
  with:
    push: true
    tags: ghcr.io/${{ github.repository }}:${{ github.sha }}
```

5.6 Tags you can trust

- Tag every build with the **git commit hash** and releases with a version: `1.4.2` .
- Never deploy `latest` : you can't tell what's running or roll back.
- Roll back = deploy the previous tag.

5.7 Where to run a container

Railway, Render, Fly.io	push a repo with a Dockerfile; easiest
Google Cloud Run	serverless containers, scale to zero
AWS App Runner / ECS	managed containers on AWS
Azure Container Apps	managed containers on Azure
Kubernetes	many services, autoscaling, GPUs; most ops work

6.1 Containers vs virtual machines

	Container	VM
Shares the host kernel	yes	no, has its own OS
Starts in	seconds	minutes
Size	MBs	GBs
Isolation	process level	hardware level, stronger

6.2 Quick interview answers

Image vs container	the template vs a running instance of it
CMD vs ENTRYPOINT	default command (replaceable) vs fixed command (CMD = its args)
COPY vs ADD	ADD also unpacks archives and fetches URLs; use COPY unless you need that
Layer caching	unchanged layers are reused; put rarely changed steps first
Volume vs bind mount	managed by Docker vs a folder from the host
Why multi-stage?	smaller, safer final image without build tools
Why non-root?	an attacker who breaks the app doesn't get root
Compose vs Kubernetes	one machine, simple vs a cluster, scaling and self-healing

6.3 Commands at a glance

`build` · `push` · `pull`	make, upload, download images
`run` · `ps` · `stop` · `rm`	manage containers
`logs` · `exec` · `inspect` · `stats`	debug a container
`volume` · `network`	storage and networking
`compose up` · `down`	a whole multi-container app
`system df` · `prune`	disk space

6.4 Keep learning, free

- Notes: Docker and deployment, serving open models (LLMOps)
- Git & GitHub Cheat Sheet
- Developer Templates Pack: Dockerfile, compose and CI files ready to copy

learn.aiwithpriyabrat.com

learn.aiwithpriyabrat.com