



Docker

Version 27.0 · verified 2026-10-18

Practical reference for Docker CLI commands,
Dockerfile optimization, Compose, debugging, and AI
agent sandboxes.

Contents

Mental model	2
Essential CLI commands	2
Container lifecycle	3
Building and Dockerfiles	3
Docker Compose and networking	4
System cleanup and debugging	5
Docker for AI agents and sandboxing	6
Further reading	7

Mental model

Docker packages applications and their dependencies into portable containers. Containers share the host operating system kernel while executing in isolated user space environments. An image is a read-only blueprint, while a container is a running instance of an image with a writable layer.

Essential CLI commands

Task	Command
Pull image	<code>docker pull <image></code>
List images	<code>docker image ls</code>
Remove image	<code>docker rmi <image></code>

The Docker CLI interacts with the Docker daemon to manage container images and runtime environments. Images are downloaded from registries like Docker Hub and stored locally.

```
docker pull node:22-alpine
docker image ls
docker rmi node:22-alpine
```

Tip: Always specify explicit image tags rather than `latest` to ensure reproducible environments.

Container lifecycle

Task	Command
Run container	<code>docker run <options> </code>
List active	<code>docker ps</code>
Stop container	<code>docker stop <id></code>

Containers move through distinct states: created, running, paused, stopped, and removed. Using interactive flags allows running ephemeral environments for testing or debugging.

```
docker run -d \  
  --name web \  
  -p 8080:80 \  
  nginx:alpine
```

Gotcha: Containers are ephemeral by default. Any data created inside a container is lost when the container is deleted unless mounted to a volume.

Building and Dockerfiles

Instruction	Purpose
FROM	Sets base image for build stage
COPY	Copies files from host to image
RUN	Executes command during build

Dockerfiles use layered caching. Each instruction creates an immutable image layer. Place infrequently changed layers (like dependency installation) before frequently changed source code.

```
FROM node:22-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
```

Tip: Multi-stage builds keep final images lean by discarding build toolchains and source files.

Docker Compose and networking

Command	Action
Start services	<code>docker compose up -d</code>
View logs	<code>docker compose logs -f</code>
Stop services	<code>docker stop <id></code>

Docker Compose defines and runs multi-container applications. Services within the same Compose file join a shared network by default and can resolve each other by service name.

```
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DB_HOST=db
  db:
    image: postgres:16-alpine
    environment:
      - POSTGRES_PASSWORD=secret
```

Gotcha: docker compose down stops containers but preserves named volumes; use -v flag to purge volume data.

System cleanup and debugging

Task	Command
Exec shell	docker exec -it <id> sh
View logs	docker logs -f <id>
Prune system	docker system prune -a

Debugging running containers requires checking stdout logs or executing interactive shells. Dangling images, stopped containers, and unused volumes consume disk space quickly.

```
docker logs --tail 50 -f my-app
docker exec -it my-app sh
docker system prune -af --volumes
```

Warning: `docker system prune -a --volumes` irreversibly deletes all stopped containers, unused networks, and unreferenced volumes.

Docker for AI agents and sandboxing

Goal	Flag / Setting
Limit CPU	<code>--cpus="2.0"</code>
Limit memory	<code>--memory="2g"</code>
Read-only root	<code>--read-only</code>

AI agents executing unverified code must be run in secure, resource-constrained sandbox containers. Restricting kernel privileges, memory, CPU, and filesystem write access prevents runaway agent processes and security breaches.

```
docker run --rm -it \  
  --memory="1g" \  
  --cpus="1.0" \  
  --read-only \  
  --network none \  
  --cap-drop=ALL \  
  /bin/sh
```

```
python:3.12-slim \  
python -c "print('agent test')"
```

Gotcha: Read-only root filesystems require explicit `--tmpfs /tmp` flags if the agent script or runtime needs temporary scratch space.

Further reading

- [Docker Documentation](#) — Official Docker manuals and reference guides.
- [Dockerfile Best Practices](#) — Guidelines for writing clean, efficient Dockerfiles.
- [Docker Compose Overview](#) — Multi-container application orchestration specification.