DOCKER ENGINEERING REFERENCE

Author: ddonle.com

Core Model

Docker packages apps into isolated, portable units.

- image – read-only layered filesystem snapshot
- container – running instance of an image
- layer – immutable diff, shared across images
- registry – remote store for images (e.g. Docker Hub)

```
Dockerfile -> image - container
                    \ container
                    \ container
```

Dockerfile → `build` → Image → `run` → Container.
Images are content-addressed by digest (SHA256).

Image Naming & References

Format: `[registry/][namespace/]name[:tag][@digest]`

`ubuntu`	official image, implicit `:latest`
`ubuntu:22.04`	pinned tag
`myapp:v1.0`	local image
`ghcr.io/org/app:sha`	GitHub Container Registry
`localhost:5000/img`	local registry
`img@sha256:abc...`	digest pin (immutable)

Always pin by digest in production; tags are mutable.

Image Management

`docker pull image:tag` – pull image from registry.
`docker push image:tag` – push image to registry.
`docker images` – list local images.
`docker images --filter dangling=true` – list untagged images.
`docker rmi image` – remove image.
`docker image prune -a` – remove all unused images.
`docker tag src dst` – alias an image with a new name/tag.
`docker image inspect image` – show full image metadata.
`docker image history image` – show layer build history.
`docker save image | gzip > img.tar.gz` – export to tarball.
`docker load < img.tar.gz` – import from tarball.

Running Containers

`docker run image` – create and start a container.
`docker run -it image bash` – interactive shell.
`docker run -d image` – run detached (background).
`docker run --rm image` – auto-remove on exit.
`docker run --name myapp image` – assign a name.
`docker run -p 8080:80 image` – map host:container port.
`docker run -v /host:/app image` – bind mount a directory.
`docker run --env-file .env image` – load env vars from file.
`docker run --memory 512m --cpus 1.5 image` – set resource limits.
`docker run --restart=always image` – restart policy.
`docker run --network mynet image` – attach to named network.

Container Lifecycle

`docker ps` – list running containers.
`docker ps -a` – list all containers including stopped.
`docker stop container` – graceful stop (SIGTERM, then SIGKILL).
`docker kill container` – immediate SIGKILL.
`docker start container` – restart a stopped container.
`docker restart container` – stop then start.
`docker pause / unpause container` – freeze/resume processes.
`docker rm container` – remove stopped container.
`docker rm -f container` – force remove running container.
`docker container prune` – remove all stopped containers.

Exec & Inspection

`docker exec -it container bash` – shell into running container.
`docker exec container cmd` – run command without TTY.
`docker logs container` – print stdout/stderr logs.
`docker logs -f container` – follow log stream.
`docker logs --since 5m container` – logs from last 5 minutes.
`docker inspect container` – full JSON metadata.
`docker top container` – show running processes.
`docker stats` – live CPU/memory/network/IO usage.
`docker diff container` – show filesystem changes vs image.
`docker cp container:/path ./local` – copy file from container.
`docker cp ./local container:/path` – copy file into container.

Dockerfile Essentials

`FROM image:tag` – base image; every Dockerfile starts here.
`RUN cmd` – execute during build; creates a new layer.
`COPY src dst` – copy files from build context into image.
`ADD src dst` – like COPY but auto-extracts tars, supports URLs.
`ENV KEY=value` – set environment variable.
`ARG NAME=default` – build-time variable (`--build-arg`).
`WORKDIR /app` – set working directory for subsequent commands.
`EXPOSE 8080` – document intended port (does not publish).
`VOLUME ["/data"]` – declare a mount point.
`USER uid:gid` – switch user for subsequent commands.
`ENTRYPOINT ["executable"]` – primary process; not overridden easily.
`CMD ["arg1"]` – default args to ENTRYPOINT (easily overridden).
`HEALTHCHECK --interval=30s CMD curl -f http://localhost/` – health probe.
`LABEL key="value"` – attach metadata to image.

Dockerfile Best Practices

Layer caching: Order instructions least-to-most volatile. Copy dependency manifests and install *before* copying source code.

```
COPY package.json .
RUN npm install
COPY . .          # cache-busted only on src change
```

Minimize layers: Chain `RUN` with `&&` and clean up in the same step:

```
RUN apt-get update && apt-get install -y curl \
    && rm -rf /var/lib/apt/lists/*
```

- Never run as `root` in production — use `USER`.
- Use `.dockerignore` to exclude build context noise.
- Prefer `COPY` over `ADD` unless extraction is needed.
- Pin base image tags; pin digest for reproducibility.

Multi-Stage Builds

Produce lean final images by discarding build-time tooling:

```
FROM golang:1.22 AS builder
WORKDIR /src
COPY . .
RUN go build -o app .

FROM gcr.io/distroless/static
COPY --from=builder /src/app /app
ENTRYPOINT ["/app"]
```

`docker build --target builder .` – build up to a named stage only.
`COPY --from=stage src dst` – copy artifacts between stages.
`COPY --from=image src dst` – copy from an external image directly.
Combine with `--platform` to produce cross-architecture images.

Build System

`docker build -t name:tag .` – build from Dockerfile in context.
`docker build -f path/Dockerfile .` – specify alternate Dockerfile.
`docker build --no-cache .` – ignore layer cache entirely.
`docker build --build-arg KEY=val .` – pass build-time variable.
`docker build --target stage .` – build up to a named stage.
`docker build --platform linux/amd64,linux/arm64 .` – multi-arch.
`docker buildx build --push -t reg/img .` – build and push (BuildKit).
`DOCKER_BUILDKIT=1 docker build .` – enable BuildKit explicitly.
`docker buildx ls` – list available builders.
`docker buildx create --use` – create and activate a new builder.

.dockerignore

Reduces build context size and prevents cache invalidation.
`.git` – exclude version control history.
`node_modules` – exclude installed dependencies.
`**/*.log` – exclude all log files recursively.
`!dist/` – negate: re-include a previously excluded path.
`Dockerfile*` – exclude Dockerfiles from context.
`.env` – keep secrets out of the image entirely.
Patterns follow the same syntax as `.gitignore`.
Smaller context = faster builds and fewer accidental secret leaks.

Volumes & Storage

Named volumes – managed by Docker; best for persistent data.
`docker volume create myvol` – create named volume.
`docker volume ls` – list volumes.
`docker volume inspect myvol` – show volume details.
`docker volume rm myvol` – remove volume.
`docker volume prune` – remove all unused volumes.
`docker run -v myvol:/data image` – mount named volume.
Bind mounts – host path mapped into container.
`docker run -v $(pwd):/app image` – mount current directory.
tmpfs mounts – in-memory, not persisted to disk.
`docker run --tmpfs /tmp image` – ephemeral in-memory mount.

Networking

`docker network ls` – list networks.
`docker network create mynet` – create a bridge network.
`docker network inspect mynet` – show network details.
`docker network connect mynet container` – attach container to network.
`docker network disconnect mynet container` – detach.
`docker network rm mynet` – remove network.
Driver types:

- `bridge` – default; isolated network on host
- `host` – shares host network stack (Linux only)
- `none` – no networking
- `overlay` – multi-host (Swarm/Kubernetes)

Containers on the same named network resolve each other by **container name** via Docker DNS.

Docker Compose

`docker compose up -d` – start all services detached.
`docker compose down` – stop and remove containers + networks.
`docker compose down -v` – also remove named volumes.
`docker compose build` – build/rebuild service images.
`docker compose pull` – pull latest images for all services.
`docker compose logs -f service` – follow service logs.
`docker compose exec service bash` – shell into a service.
`docker compose ps` – list service containers and status.
`docker compose run --rm service cmd` – one-off command.
`docker compose restart service` – restart a specific service.
`docker compose scale service=3` – set replica count.
`docker compose config` – validate and view merged config.

Compose File Structure

```
services:
  web:
    build: .
    image: myapp:latest
    ports: ["8080:80"]
    environment:
      - NODE_ENV=production
    env_file: .env
    volumes:
      - ./src:/app
      - data:/var/lib/db
    depends_on:
      db:
        condition: service_healthy
    networks: [backend]
    restart: unless-stopped

  db:
    image: postgres:16
    healthcheck:
      test: ["CMD", "pg_isready"]
      interval: 10s
      retries: 5

volumes:
  data:
networks:
  backend:
```

Registry & Distribution

`docker login registry.example.com` – authenticate to registry.
`docker logout` – remove stored credentials.
`docker search term` – search Docker Hub.
`docker pull image@sha256:digest` – pull by immutable digest.
`docker manifest inspect image` – view multi-arch manifest.
`docker buildx imagetools inspect image` – inspect remote image.
`docker buildx imagetools create --tag dst src1 src2` – assemble multi-arch manifest list.
Run a local registry:
`docker run -d -p 5000:5000 registry:2`

Security

[RISK] `--privileged` – grants full host access; avoid in production.
[RISK] `-v /var/run/docker.sock:/var/run/docker.sock` – exposes host Docker daemon.
`--read-only` – mount container root filesystem read-only.
`--cap-drop ALL --cap-add NET_BIND_SERVICE` – drop all capabilities, add only what is needed.
`--security-opt no-new-privileges` – prevent privilege escalation.
`docker scan image` – scan image for CVEs (Snyk-backed).
`docker sbom image` – generate Software Bill of Materials.
`docker trust sign image` – sign image with Notary.
`docker trust inspect image` – verify image trust data.
Best practice: non-root `USER`, minimal base image (distroless/alpine), no secrets in layers.

Resource Management & Cleanup

`docker system df` – show disk usage by object type.
`docker system prune` – remove all stopped containers, unused networks, dangling images.
`docker system prune -a --volumes` – full cleanup including volumes.
[DANGER] `--volumes` flag also removes named volumes with data.
`docker stats --no-stream` – snapshot resource usage.
`docker run --memory 256m --cpus 0.5 image` – hard resource cap.
`docker update --memory 512m container` – update limits live.

Debugging & Troubleshooting

`docker events` – stream real-time daemon events.
`docker inspect --format '.State.ExitCode' c` – extract field via Go template.
`docker run --entrypoint sh image` – override entrypoint for debugging.
`docker commit container debug-img` – snapshot container state as image.
`docker run --pid=host image` – share host PID namespace.
`docker run --network=container:other img` – share another container's network.
Check logs path:
`docker inspect --format '.LogPath' container`
OOM killed? Check `docker inspect` for `OOMKilled: true`.

Large-Scale Engineering Practices

- Pin image digests (`@sha256:...`) in production deployments.
- Use multi-stage builds to minimize final image attack surface.
- Never embed secrets in image layers; use secrets mounts: `RUN --mount=type=secret,id=key ...`
- Enforce non-root `USER` and `--read-only` in production.
- Use `HEALTHCHECK` so orchestrators detect unhealthy containers.
- Enable `BuildKit` for parallel, cache-efficient builds.
- Scan images in CI with `docker scout` or `trivy`.
- Use named volumes, not bind mounts, for stateful workloads.
- Set `restart: unless-stopped` for long-running services.

Core concepts
Build & Compose
Dangerous / security
Registry / networking
Config / Dockerfile
Workflow / runtime