Docker Troubleshooting Runbook Pack

A senior engineer's incident runbook for the top Docker production failure modes — daemon, sockets, ports, restart loops, images, disk, networking, volumes, and health. DevOps AI ToolKit · devopsaitoolkit.com

Most Docker "outages" are one of a handful of failures wearing different masks: the daemon is down, the socket is unreachable or forbidden, a port is already taken, a container is crash-looping, an image won't build or pull, the disk is full, DNS/networking is broken, a volume mount fights permissions, or a HEALTHCHECK is flapping. Each section below is a self-contained runbook: read-only checks first, then the interpretation, then the narrowest safe fix. Work the symptom you actually have — don't restart the daemon hoping it clears.

> Examples use Linux with a systemd-managed daemon and the modern `docker` CLI (`docker compose`, not `docker-compose`). Adjust unit and path names for your distro. **Restarting `dockerd` restarts the daemon for every container on the host** — with `live-restore` off, that means all workloads bounce. Confirm the blast radius before you touch the daemon.

1. Daemon Down / Cannot Connect to the Socket

Symptom: `Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?`

```
# Is the daemon process/unit actually up?
systemctl is-active docker
systemctl status docker --no-pager -l

# Why did it fail to start? Read the daemon's own logs.
journalctl -u docker --no-pager -n 80

# Does the socket file exist and is anything listening?
ls -l /var/run/docker.sock
ss -lx | grep docker.sock
```

- `inactive` / `failed` unit → the daemon isn't running; the journal names the cause (bad `daemon.json`, port clash, storage driver error).
- Unit `active` but socket missing → a broken `-H` config or a masked `docker.socket`; check `systemctl status docker.socket`.
- A common root cause is invalid `/etc/docker/daemon.json` — validate it before restarting: `python3 -m json.tool /etc/docker/daemon.json`.

```
# Only after you know why it died:
sudo systemctl start docker      # or: sudo systemctl restart docker
```

2. Permission Denied on the Socket

Symptom: `permission denied while trying to connect to the Docker daemon socket ... /var/run/docker.sock`. The daemon is up; *your user* can't reach it.

```
# Who owns the socket, and which group grants access?
ls -l /var/run/docker.sock          # typically  srw-rw---- root docker
id                                   # are you in the 'docker' group?
getent group docker
```

- The socket is `root:docker` mode `660`. If you're not in the `docker` group, every call is denied.
- After adding a user to the group, the current shell keeps the old group set — you must re-login or start a fresh session.

```
sudo usermod -aG docker "$USER"     # add yourself to the docker group
newgrp docker                        # apply in THIS shell without a full re-login
docker version                       # confirm the client can now reach the daemon
```

- Prefer group membership over `chmod 666` on the socket — write access to `docker.sock` is effectively **root on the host**. Treat it as a privileged credential.

3. Port Already Allocated / Bind Conflicts

Symptom: `Error ... driver failed programming external connectivity ... bind: address already in use` or `port is already allocated`.

```
# What is already holding the port? (pick whichever tool you have)
sudo ss -ltnp 'sport = :8080'
sudo lsof -iTCP:8080 -sTCP:LISTEN -P -n

# Is it another CONTAINER publishing the same host port?
docker ps --format 'table {{.Names}}\t{{.Ports}}' | grep 8080
```

- If a container owns it, you have a duplicate publish — stop the old one or change the host-side port (`-p 8081:8080`).
- If a host process (another service, a stale `docker-proxy`) owns it, free that or bind to a different host port.

- Binding `0.0.0.0:8080` conflicts with anything on that port on *any* interface; publish to a specific address (`-p 127.0.0.1:8080:8080`) to scope it.

```
docker stop web-1 && docker rm web-1          # remove the stale publisher
docker run -d -p 8081:8080 --name web-1 myapp:1.4.2
```

4. Container Restart Loops & Exit Codes (137 / 143)

Symptom: a container flaps `Restarting` or sits in `Exited`. The exit code is the fastest clue.

```
docker ps -a --filter name=web-1 --format '{{.Names}} {{.Status}}'
docker inspect web-1 --format \
  'exit={{.State.ExitCode}} oom={{.State.OOMKilled}} restarts={{.RestartCount}} err={{.State.Error}}'
docker logs --tail=120 --timestamps web-1
```

- **Exit 137** = 128+9 (SIGKILL). With `OOMKilled=true` the kernel OOM-killer hit the container's memory limit; confirm with `dmesg -T | grep -i oom`. Raise `--memory` or fix the leak.
- **Exit 143** = 128+15 (SIGTERM) — a clean stop signal the app didn't handle in time before SIGKILL; usually a shutdown/timeout issue, not a crash.
- **Exit 1 / 2 / 127** = the app itself: unhandled error, bad flag, or `command not found` (127 → wrong `ENTRYPOINT`/`CMD` path).
- A tight loop with `restart: always` hammers the host — set `restart: on-failure:5` and read the last logs *before* the restart wipes them.

5. Image Build Failures & Cache

Symptom: `docker build` fails partway, or "works on my machine" but rebuilds pull stale layers.

```
# Verbose, plain output so you can see which layer/step failed
docker build --progress=plain -t myapp:1.4.2 .

# Suspect a stale cached layer? Rebuild the affected steps clean
docker build --no-cache -t myapp:1.4.2 .

# BuildKit cache is separate from images — reclaim it deliberately
docker builder prune            # add -a to remove all build cache
```

- Read the failing `RUN` step's output, not just the final error — the real message is a few lines up.
- Cache invalidates from the first changed line down: put `COPY package.json` / dependency installs **before** `COPY . .` so code edits don't bust the dependency layer.
- `no space left` mid-build is a disk issue, not a build bug — see §7.
- For an amd64/arm64 mismatch (`exec format error` at runtime), build with `--platform linux/amd64`.

6. Image Pull: Auth, Rate Limits, Manifest Unknown

Symptom: `pull access denied`, `unauthorized`, `toomanyrequests`, or `manifest unknown`.

```
docker pull registry.example.com/myapp:1.4.2
docker login registry.example.com        # re-auth; token/creds may have expired
cat ~/.docker/config.json | python3 -m json.tool   # which registries have creds/auth
```

- `manifest unknown` / `not found` → the **tag doesn't exist** (typo, never pushed, or wrong arch). List what's really there: `docker buildx imagetools inspect registry.example.com/myapp:1.4.2`.
- `unauthorized` / `pull access denied` → creds missing/expired or the repo is private; re-run `docker login`.
- `toomanyrequests: ... rate limit` → Docker Hub anonymous/free pull cap. Authenticate, or pull through a pull-through cache / your own registry.
- On air-gapped or CI hosts, confirm the credential helper is present — a missing helper silently drops auth.

7. No Space Left on Device / overlay2

Symptom: `write ... no space left on device`, builds/pulls fail, containers won't start.

```
df -h /var/lib/docker           # is the Docker filesystem full?
df -i /var/lib/docker           # inodes can exhaust even with free bytes
docker system df                # space by images / containers / volumes / build cache
docker system df -v | head -40  # the biggest offenders
```

- `overlay2` lives under `/var/lib/docker` — a full root filesystem stalls every container.
- Dangling images and stopped containers accumulate silently; build cache is often the single largest consumer.

```
# Reclaim in increasing order of aggressiveness — READ each before running:
docker container prune          # remove stopped containers
docker image prune              # remove dangling images (add -a for all unused)
docker builder prune            # remove build cache
docker system prune             # all of the above at once
# NOTE: 'docker volume prune' and 'system prune --volumes' DELETE DATA. Never blind-run in prod.
```

8. Container Networking & DNS

Symptom: a container can't reach another container, an external host, or resolves names wrong.

```
docker network ls
docker inspect web-1 --format '{{range $n,$c := .NetworkSettings.Networks}}{{$n}}={{$c.IPAddress}} {{end}}'

# Test from INSIDE the container (its resolver differs from the host's)
docker exec web-1 getent hosts api-1        # DNS on the docker network
docker exec web-1 cat /etc/resolv.conf      # which resolver is it using?
docker exec web-1 sh -c 'wget -qO- http://api-1:8080/health || echo FAIL'
```

- Containers resolve each other by **service/container name only on a shared user-defined network** — the default `bridge` has no automatic name resolution. Put both on the same `docker network create` network.
- Can reach IP but not name → Docker's embedded DNS (127.0.0.11) or upstream DNS in `daemon.json` is the issue.
- Can't reach the outside at all → check `net.ipv4.ip_forward=1` and that a firewall (ufw/firewalld) isn't dropping the `docker0`/bridge forwards.

9. Volume Mount & Permission Issues

Symptom: `permission denied` writing to a mount, empty directory where data should be, or config not seen.

```
docker inspect web-1 --format '{{json .Mounts}}' | python3 -m json.tool
# What UID/GID does the process run as vs. who owns the mounted path?
docker exec web-1 id
ls -ln /srv/web-1/data          # numeric owner of the host bind-mount source
docker exec web-1 ls -ln /data  # how it looks inside the container
```

- Bind mounts preserve the **host's** numeric UID/GID. If the container runs as UID 1000 but the host path is owned by root, writes fail — align ownership (`chown`) or run the container as the owning UID (`--user 1000:1000`).
- A mount that shadows image content (empty host dir over a populated image path) hides the image's files — that's expected bind-mount behavior, not a bug.
- SELinux hosts (`Permission denied` despite correct UIDs) need the `:z`/`:Z` mount flag, e.g. `-v /srv/web-1/data:/data:Z`.
- Named volumes (not bind mounts) *do* seed from the image on first use — a subtle difference worth confirming with `docker inspect`.

10. HEALTHCHECK Unhealthy

Symptom: `docker ps` shows `(unhealthy)`; orchestrators refuse traffic or restart the container.

```
docker ps --filter health=unhealthy --format '{{.Names}} {{.Status}}'
docker inspect web-1 --format '{{json .State.Health}}' | python3 -m json.tool
# Run the exact healthcheck command yourself, inside the container:
docker inspect web-1 --format '{{json .Config.Healthcheck.Test}}'
docker exec web-1 sh -c 'wget -qO- http://localhost:8080/health; echo " rc=$?"'
```

- `.State.Health.Log` keeps the last few probe outputs and exit codes — read the actual failure, not just the `unhealthy` label.
- Non-zero probe exit = unhealthy. A `curl`/`wget` that isn't installed in the image makes the check fail even when the app is fine — match the probe to the image's tools.
- `start_period` too short marks a slow-booting app unhealthy during normal startup; `interval`/`timeout`/`retries` too tight causes flapping under load. Tune before you assume the app is broken.

11. Restart Decision Tree

> **Restart the narrowest thing that explains the symptom.** A single container almost never justifies bouncing `dockerd`. Restarting the daemon affects every workload on the host.

```
Docker problem?
 ├─ CLI can't reach daemon (§1/§2)?
 │    ├─ 'permission denied'     → fix group membership (§2); DO NOT restart the daemon
 │    └─ daemon down/failed      → read journalctl -u docker, fix cause, then systemctl restart docker
 ├─ One container misbehaving?
 │    ├─ exit 137 / OOMKilled    → raise --memory or fix leak (§4); restart the CONTAINER only
 │    ├─ exit 143 / clean stop   → app shutdown handling (§4); restart container
 │    ├─ port conflict (§3)      → stop the other publisher / change host port; no daemon restart
 │    ├─ unhealthy (§10)         → read Health.Log; fix probe/app; restart container
 │    └─ can't reach peers (§8)  → put both on a user-defined network; no daemon restart
 ├─ Build/pull failing (§5/§6)?  → fix Dockerfile/auth/tag; no restart needed
 ├─ 'no space left' (§7)?        → prune (READ each prune first); then retry
 └─ Only after config change to daemon.json → systemctl restart docker (blast radius = ALL containers)
```

12. Copy/Paste One-Shot Triage

```
echo "== daemon ==";       systemctl is-active docker; docker version --format 'client={{.Client.Version}} server={{.Server.Version}}' 2>&1
echo "== socket ==";       ls -l /var/run/docker.sock 2>&1; id -nG | tr ' ' '\n' | grep -qx docker && echo "in docker group" || echo "NOT in
docker group"
echo "== unhealthy ==";    docker ps --filter health=unhealthy --format '{{.Names}} {{.Status}}'
echo "== exited/looping ==";docker ps -a --filter status=exited --format '{{.Names}} {{.Status}}' | head
echo "== oom recent ==";   docker ps -aq | xargs -r docker inspect --format '{{.Name}} oom={{.State.OOMKilled}} exit={{.State.ExitCode}}'
2>/dev/null | grep -E 'oom=true|exit=(137|143)'
echo "== disk ==";         df -h /var/lib/docker | tail -1; docker system df
echo "== port holders ==";  sudo ss -ltnp | grep -E ':(80|443|8080|5432) ' 2>/dev/null
```

13. Incident Notes Template

```
INCIDENT: Docker production failure
Started (UTC):        ____   Detected by: ____   Sev: ____
Host / node:          ____   Docker server version: ____
Impact:               (which containers / services / % of traffic)
Failure mode:         [ ] daemon down (§1)  [ ] socket perm (§2)  [ ] port conflict (§3)
                      [ ] restart loop / exit ____ (§4)  [ ] build (§5)  [ ] pull/auth (§6)
                      [ ] disk full (§7)  [ ] network/DNS (§8)  [ ] volume perm (§9)  [ ] unhealthy (§10)
Key evidence:         exit code ____  OOMKilled ____  restarts ____
                      journalctl/logs line: ____
                      disk /var/lib/docker: __% used   inodes: __%
Blast radius:         [ ] single container  [ ] daemon restart (ALL containers)
Action taken:         (what was restarted / pruned / reconfigured + who + when)
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (memory limits, restart policy, disk alerts, registry auth, healthcheck tuning) ____
```