Docker Daemon Error Runbook

A senior engineer's incident runbook for when the Docker daemon stops answering the socket. DevOps AI ToolKit · devopsaitoolkit.com

`Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?` means your `docker` client tried to reach the daemon (`dockerd`) and got nothing back. The client is the messenger — the real problem is one of: `dockerd` is dead or crash-looping, the socket is missing or unreadable, your user lacks access, the disk is full, the container runtime (`containerd`) is wedged, or a bad `daemon.json` stopped the daemon from starting. Work top-to-bottom: confirm whether the daemon is even running, then walk outward to the socket, your permissions, and its dependencies.

> Assumes a Linux host with systemd-managed Docker Engine. Adjust unit and path names for rootless or non-systemd setups. Run read-only checks first; restarting `dockerd` kills every running container's foreground process and pauses all workloads on the host — treat it as a blast-radius action with a change record.

1. Daemon Error Triage Checklist (top-to-bottom)

1. **Is it the daemon or the client?** The error is client-side text; confirm whether `dockerd` is actually up before anything else.
2. **Service state.** `systemctl status docker` — active, failed, or `activating (auto-restart)` (crash loop)?
3. **Why did it stop?** `journalctl -u docker` for the last exit reason — panic, config parse error, or OOM.
4. **The socket.** Does `/var/run/docker.sock` exist, and is `DOCKER_HOST` pointing you somewhere else?
5. **Your access.** Are you in the `docker` group, or do you need `sudo` / rootless context?
6. **Disk.** `no space left on device` under `/var/lib/docker` will crash or wedge the daemon.
7. **The runtime.** `containerd` is a hard dependency; if it's down, `dockerd` can't run containers.
8. **Recent change.** A `daemon.json` edit or a package upgrade is the most common trigger — check last.
9. **Decide + act.** Restart the narrowest thing that explains the symptom. Record it.

2. Is dockerd Actually Running?

```
systemctl status docker --no-pager
systemctl is-active docker            # active / failed / activating
sudo journalctl -u docker --since "30 min ago" --no-pager | tail -n 60
pgrep -a dockerd                       # is the process actually there?
```

- `activating (auto-restart)` means systemd is restarting a daemon that keeps dying — a crash loop, almost always a config or runtime problem, not a client one.
- `inactive (dead)` and nobody started it → simply `systemctl start docker`, then find out why it was down.
- If `pgrep` shows a live `dockerd` but the client still can't connect, the problem is the socket or your context (§3, §4), not the daemon.

3. Socket and DOCKER_HOST Checks

```
ls -l /var/run/docker.sock            # expect: srw-rw---- root docker
echo "DOCKER_HOST=$DOCKER_HOST"       # empty = default local socket
docker context ls                     # is a non-default context active?
sudo curl -s --unix-socket /var/run/docker.sock http://localhost/_ping   # should print: OK
```

- No socket file **and** `dockerd` is down → this is a daemon problem; go back to §2.
- No socket file but `dockerd` is up → the daemon may be listening on a TCP host or a different path; check its `ExecStart` and `-H` flags with `systemctl cat docker`.
- A stray `DOCKER_HOST` (e.g. left over from a remote or `docker-machine` session) silently redirects the client. Unset it and retry: `unset DOCKER_HOST`.
- `_ping` returning `OK` proves the daemon is healthy and the issue is purely client-side context or permissions.

4. User Permission Checks

```
id -nG "$USER" | tr ' ' '\n' | grep -x docker   # are you in the docker group?
groups                                          # same, quick view
# Rootless installs answer on a per-user socket, not /var/run:
echo "$DOCKER_HOST"                             # e.g. unix:///run/user/1000/docker.sock
systemctl --user status docker 2>/dev/null      # rootless daemon lives under --user
```

- `permission denied` on `/var/run/docker.sock` while `dockerd` is up → you're not in the `docker` group. Add with `sudo usermod -aG docker`

`"$USER"`, then start a fresh login shell (`newgrp docker` or re-login) so the group takes effect.

- Membership in `docker` is root-equivalent — it grants full control of the host. Don't hand it out casually; use `sudo docker` for one-off admin instead.
- On a **rootless** setup the system socket won't exist at all; the daemon runs under `systemctl --user` and answers on `/run/user/<uid>/docker.sock`.

---

5. Disk Space Checks

```
df -h /var/lib/docker /var                    # look for 100% Use%
df -i /var/lib/docker                         # inode exhaustion looks like "no space" too
docker system df 2>/dev/null                  # only works if the daemon is up
sudo du -sh /var/lib/docker/* 2>/dev/null | sort -h | tail
```

- `no space left on device` in the journal (§2) means `dockerd` can't write image layers, container state, or its own logs — it will fail to start or wedge mid-operation.
- A full **inode** table (`df -i`) presents the identical error even when `df -h` shows free bytes — a swarm of tiny files, often from log-heavy containers.
- Reclaim safely once the daemon is up: `docker system prune` (add `-a` and `--volumes` only when you understand what they delete). If the daemon won't start at all, free space at the OS level first (old logs under `/var/log`, unused packages).

---

6. Containerd / Runtime State

```
systemctl status containerd --no-pager
sudo journalctl -u containerd --since "30 min ago" --no-pager | tail -n 40
sudo ctr version 2>/dev/null                  # low-level containerd client
ps -ef | grep -E 'containerd|runc' | grep -v grep
```

- Docker Engine delegates to `containerd`, which delegates to `runc`. If `containerd` is `failed`, `dockerd` will start but be unable to create or manage containers — and may exit.
- Check ordering in the journal: if `containerd` died first and `docker` followed, fix `containerd` first, then restart `docker`.
- A wedged `containerd-shim` for a stuck container can hang daemon operations; identify it in `ps` before considering a runtime restart.

---

7. daemon.json Config Errors

```
sudo cat /etc/docker/daemon.json                                # the usual culprit
python3 -m json.tool /etc/docker/daemon.json >/dev/null && echo "JSON OK"
sudo dockerd --validate 2>&1 | tail                             # dry-run config check (newer engines)
sudo journalctl -u docker --no-pager | grep -Ei "daemon.json|unable to configure|invalid|failed to load"
```

- A single trailing comma or bad key in `daemon.json` makes `dockerd` refuse to start — the journal will name the parse error explicitly.
- `--validate` parses the config without launching the daemon; use it to test an edit before you restart the service.
- Common breakers: an unreachable `registry-mirrors` entry, an invalid `data-root`, a `storage-driver` the kernel doesn't support, or a malformed `log-opts` block. Fix the key the journal names; don't rewrite the whole file blind.

---

8. Recent Config / Upgrade Changes

```
sudo journalctl -u docker --no-pager | grep -Ei "starting up|version" | tail    # did the version just change?
# Debian/Ubuntu upgrade history:
grep -i docker /var/log/dpkg.log* 2>/dev/null | tail
# RHEL/CentOS/Fedora:
sudo grep -i docker /var/log/dnf.log* /var/log/yum.log* 2>/dev/null | tail
sudo ls -lt /etc/docker/ /lib/systemd/system/docker.service     # recently touched files
```

- A package upgrade that swapped the `docker.service` unit, changed the default `storage-driver`, or bumped `containerd` is a frequent, easily-missed trigger.
- If the daemon broke right after an upgrade, compare the running config against the packaged default and check for a `docker.service.dpkgnew` / `.rpmnew` you were meant to merge.
- Correlate the daemon's "starting up" version line with your change window before assuming an infrastructure fault.

---

9. Restart Decision Tree

> **Restart the narrowest component your evidence points at.** Restarting `dockerd` stops the foreground process of every running container on the host — plan for it. Confirm the cause in §2–§8 first.

```
docker client can't connect?
 ├─ dockerd process not running?
 │    ├─ journal shows daemon.json / config error? — yes → fix config (§7), then: systemctl start docker
 │    ├─ journal shows "no space left"?            — yes → free disk (§5), then: systemctl start docker
 │    ├─ containerd failed first?                  — yes → systemctl restart containerd, then start docker
 │    └─ clean "inactive (dead)", no cause?        — yes → systemctl start docker; watch the journal
 └─ dockerd IS running
      ├─ _ping returns OK?           — yes → client-side: unset DOCKER_HOST / fix context (§3)
      ├─ permission denied on sock?  — yes → add user to docker group or use sudo (§4)
      └─ socket missing but up?      — yes → check ExecStart -H flags (systemctl cat docker)
```

```
# Least-blast-radius: reload config without dropping containers (works for some options, e.g. registry-mirrors, log-level)
sudo systemctl reload docker

# Full restart — pauses/stops container foregrounds; use with a change note
sudo systemctl restart docker
sudo journalctl -u docker -f          # watch it come back cleanly
```

10. Copy/Paste One-Shot Triage

```
echo "== service ==";    systemctl is-active docker; systemctl is-active containerd
echo "== last exit ==";  sudo journalctl -u docker --since "30 min ago" --no-pager | grep -Ei "error|fatal|no space|daemon.json|invalid" | tail -
n 5
echo "== socket ==";     ls -l /var/run/docker.sock 2>&1; echo "DOCKER_HOST=${DOCKER_HOST:-<unset>}"
echo "== ping ==";       sudo curl -s --unix-socket /var/run/docker.sock http://localhost/_ping 2>&1 || echo "no answer"
echo "== group ==";      id -nG "$USER" | tr ' ' '\n' | grep -x docker || echo "NOT in docker group"
echo "== disk ==";       df -h /var/lib/docker | tail -1; df -i /var/lib/docker | tail -1
echo "== config ==";     sudo test -f /etc/docker/daemon.json && python3 -m json.tool /etc/docker/daemon.json >/dev/null 2>&1 && echo
"daemon.json OK" || echo "daemon.json missing/invalid"
```

11. Incident Notes Template

```
INCIDENT: Docker Daemon Unreachable
Started (UTC):          ____    Detected by: ____    Sev: ____
Impact:                 (host ____ / which containers / % of workloads down)
Client error:           "Cannot connect to the Docker daemon at ____"
Daemon state:           [ ] active  [ ] failed  [ ] activating/crash-loop  [ ] inactive
Exit reason (journal):  ____
Socket:                 [ ] present  [ ] missing   DOCKER_HOST: ____
Access:                 [ ] user in docker group  [ ] rootless  [ ] used sudo
Disk:                   /var/lib/docker __% used   inodes __% used
Runtime:                containerd: [ ] active  [ ] failed
Config/upgrade:         [ ] daemon.json edited ____  [ ] recent package upgrade ____
Action taken:           (component restarted / config fixed + who + when)
Result:                 [ ] recovered @ ____   [ ] escalated to ____
Root cause:             ____
Follow-ups:             (disk alerts, config validation in CI, group policy) ____
```