Docker Port Conflict Runbook

A senior engineer's incident runbook for when a container refuses to bind its published port. DevOps AI ToolKit · devopsaitoolkit.com

`Bind for 0.0.0.0:8080 failed: port is already allocated` (or `address already in use`) means Docker tried to publish a container port on the host and something already owns that host address. The "something" is one of: another running container publishing the same port, a stale `docker-proxy` from a container that didn't clean up, a plain host service (an `nginx`, a database, a dev server) listening on it, or a reverse proxy you forgot was there. Work outward: find *who* holds the port, decide whether it's Docker's own bookkeeping or a real host process, then free it or move yours.

> Assumes a Linux host running Docker Engine with the default bridge networking and `docker-proxy` (userland proxy) enabled. Read-only checks first — identifying the holder is free; killing processes and pruning networks is not. Never `kill -9` a listener before you know what it is.

---

1. Port Conflict Triage Checklist (top-to-bottom)

1. **Read the address.** The error names the exact host bind — `0.0.0.0:8080` (all interfaces) vs `127.0.0.1:8080` (loopback only). They can conflict independently.
2. **Who holds it?** `ss -ltnp` / `lsof -i` — is the listener a container, `docker-proxy`, or a host process?
3. **Is it another container?** `docker ps` with a `.Ports` format — most conflicts are two containers asking for the same host port.
4. **Is it a stale proxy?** A `docker-proxy` with no live container behind it means a container exited uncleanly.
5. **Host service?** A native `nginx` / `haproxy` /dev server on the port is a real conflict, not a Docker bug.
6. **Reverse proxy?** Your intended fronting proxy may already own `:80` / `:443` by design.
7. **Decide + act.** Free the port (stop the real owner) *or* change your host port. Record which.

---

2. Find What Holds the Port

```
PORT=8080
sudo ss -ltnp "sport = :$PORT"          # the definitive listener list, with PID/process
sudo lsof -nP -iTCP:$PORT -sTCP:LISTEN  # same answer, shows command + PID + user
sudo fuser $PORT/tcp 2>/dev/null         # just the PID(s), quick
```

- `ss` is the source of truth: `users:(("docker-proxy",pid=1234,...))` means Docker holds it; `users:(("nginx",...))` means a host service does.
- `0.0.0.0:8080` and `[::]:8080` are IPv4 and IPv6 of the same logical port — both can appear; that's normal for a single publish.
- If nothing shows in `ss` yet Docker still refuses the bind, the holder is likely bound to a *different* interface — re-check without the `sport` filter: `sudo ss -ltnp | grep :$PORT`.

---

3. Is It Another Container?

```
# Every container's published ports, one line each
docker ps --format 'table {{.Names}}\t{{.Image}}\t{{.Ports}}'

# Just the ones touching the port in question
docker ps --format '{{.Names}} {{.Ports}}' | grep ':8080->'

# Include stopped containers — a paused/exited one can still hold a proxy (see §5)
docker ps -a --format '{{.Names}}\t{{.Status}}\t{{.Ports}}' | grep 8080
```

- `0.0.0.0:8080->80/tcp` on `web-1` means `web-1` already publishes host `8080`. Two containers cannot share the same host port + interface — one must move.
- Distinguish **published** (`host->container`, in `.Ports` with an arrow) from merely **exposed** (`80/tcp`, no arrow) — only published ports bind the host and cause this error.
- If the conflicting container is the *previous version* of the same service (a redeploy that didn't stop the old one), stopping it is the fix: `docker stop <name>`.

---

4. The docker-proxy

```
sudo ss -ltnp "sport = :8080" | grep docker-proxy
ps -ef | grep '[d]ocker-proxy' | grep 8080          # shows -host-ip / -host-port / -container-ip
```

- For each published port, `dockerd` spawns a `docker-proxy` process that binds the host side. A live proxy with a matching running

container is normal — that's the port working as intended.

- A `docker-proxy` still listening with **no** corresponding container in `docker ps` is stale bookkeeping from a container that crashed or was killed uncleanly (§5).
- Don't `kill` a `docker-proxy` directly — that desyncs it from the daemon. Stop/remove the owning container, or restart the daemon, and let Docker reap the proxy.

---

5. Stale Containers and Networks

```
docker ps -a --filter status=exited --format '{{.Names}}\t{{.Status}}\t{{.Ports}}'
docker rm <stale-container>                # remove the dead container holding the mapping
docker network ls
docker network inspect bridge --format '{{range .Containers}}{{.Name}} {{end}}'
docker network prune                       # remove unused networks (prompts first)
```

- An exited container can leave a `docker-proxy` behind until it's removed. `docker rm <name>` releases the port; `docker start <name>` reuses it. Pick one deliberately.
- If port state looks wrong across the board (proxies for containers that are gone), a daemon reconcile clears it: `sudo systemctl restart docker` — but note that restarts every container on the host, so treat it as a blast-radius action.
- `docker network prune` only removes networks with **no** attached containers; it won't touch anything live, but confirm the prompt list before answering `y`.

---

6. Host Services vs Reverse Proxies

```
sudo systemctl list-units --type=service --state=running | grep -Ei 'nginx|haproxy|apache|httpd|traefik'
sudo ss -ltnp | grep -E ':(80|443|8080) '          # who owns the classic web ports?
sudo nginx -t 2>/dev/null                           # is a host nginx configured & running?
```

- A native `nginx` or `haproxy` already listening on `:80` / `:443` is doing exactly what it was installed to do. If it's meant to front your container, publish the container on a **different** host port (e.g. `8081`) and point the proxy upstream at it — don't fight it for `:80`.
- If the reverse proxy is *supposed* to be the only thing on the public port, the container shouldn't publish there at all; put both on a shared Docker network and let the proxy reach the container by name/exposed port.
- A dev server (`python -m http.server`, a `node` app, a `postgres` on `5432`) left running from an earlier session is the most common non-Docker holder. Confirm it's disposable before stopping it.

---

7. Changing the Host Port

```
# Move the HOST side; the container port stays the same (host:container)
docker run -d --name web-1 -p 8081:80 registry.example.com/myapp:1.4.2

# Bind to loopback only if it's meant to sit behind a proxy (avoids 0.0.0.0 clashes)
docker run -d --name web-1 -p 127.0.0.1:8081:80 registry.example.com/myapp:1.4.2
```

```
# docker-compose: only the left number (host) needs to change
services:
  web:
    image: registry.example.com/myapp:1.4.2
    ports:
      - "8081:80"          # was "8080:80"
```

- Changing the **host** port (left of the colon) is the safe, non-destructive fix when the current owner is legitimate — you avoid touching anything else on the box.
- Binding to `127.0.0.1:8081:80` publishes only on loopback, which both dodges `0.0.0.0` conflicts and is the correct shape for a service that should only be reachable via a front proxy.
- Let Docker pick a free ephemeral host port with `-p 80` (no host side) when you don't care which — read it back with `docker port web-1`.

---

8. Decision Tree

> **Free the port or move yours — but only after you know the owner.** Killing an unidentified listener can take down an unrelated production service. Identify in §2 first.

```
"port is already allocated" on 0.0.0.0:8080?
 ├─ ss/lsof shows a running container (docker ps has it)?
 │     ├─ it's an old/duplicate of my service? ── yes → docker stop/rm the old one, redeploy
 │     └─ it's a different, needed service?     ── yes → change MY host port (§7)
 ├─ ss shows docker-proxy but NO live container? ── yes → docker rm the exited container (§5); reconcile daemon if needed
 ├─ ss shows a host process (nginx/haproxy/dev)?
 │     ├─ it's a reverse proxy meant to be there? ── yes → publish container on another port, point proxy upstream (§6)
 │     └─ it's a stray dev/old service?           ── yes → stop that service (systemctl stop / kill its PID)
 └─ ss shows nothing on the port?                ── → holder is on another interface; re-check without filter (§2)
```

9. Copy/Paste One-Shot Triage

```
PORT=8080
echo "== listener ==";     sudo ss -ltnp "sport = :$PORT" 2>/dev/null || sudo lsof -nP -iTCP:$PORT -sTCP:LISTEN
echo "== containers ==";   docker ps --format '{{.Names}} {{.Ports}}' | grep ":$PORT->" || echo "no running container publishes :$PORT"
echo "== exited ==";       docker ps -a --filter status=exited --format '{{.Names}} {{.Status}} {{.Ports}}' | grep "$PORT" || echo "none"
echo "== proxy ==";        ps -ef | grep '[d]ocker-proxy' | grep "$PORT" || echo "no docker-proxy on :$PORT"
echo "== host svcs ==";    sudo ss -ltnp | grep -E ":$PORT " | grep -v docker-proxy || echo "no non-docker listener"
```

10. Incident Notes Template

```
INCIDENT: Docker Port Conflict / Bind Failed
Started (UTC):          ____   Detected by: ____   Sev: ____
Impact:                 (which service failed to start / deploy blocked)
Error:                  "Bind for ____:____ failed: port is already allocated"
Address:                [ ] 0.0.0.0:____    [ ] 127.0.0.1:____   [ ] [::]:____
Holder (from ss/lsof):  [ ] other container: ____  [ ] docker-proxy (stale?): ____
                        [ ] host service: ____  [ ] reverse proxy: ____  [ ] dev/stray: ____
Owner legitimate?       [ ] yes, moved my port  [ ] no, freed the port
Action taken:           (stopped/removed ____ / changed host port ____ + who + when)
Result:                 [ ] recovered @ ____   [ ] escalated to ____
Root cause:             ____
Follow-ups:             (port allocation policy, compose cleanup, proxy topology, alerts) ____
```