Linux Server Troubleshooting Runbook Pack

A senior engineer's incident runbook for triaging a sick or unreachable Linux box — from first login to root cause. DevOps AI ToolKit · devopsaitoolkit.com

Something is wrong with a server: a service is down, the box is slow, the disk is full, or you can't even SSH in. The instinct to start restarting things is what turns a five-minute fix into a two-hour outage. Work read-only first — read the state of the machine (load, memory, disk, logs, network) before you change anything, then act on the narrowest thing that explains the symptom. This pack walks the common failure classes top-to-bottom the way an on-call engineer actually does it: confirm what's broken, localize it, then fix the one component the evidence points at.

> Commands cover both Debian/Ubuntu (`apt`, `cron`) and RHEL/Rocky/Alma (`dnf`, `crond`); differences are called out inline. Most checks are read-only and safe to run anywhere. Anything that changes state (restart, remount, `fsck`, `kill`) should go in a change note and — for production — get a second engineer. Hostnames like `node-1` and paths like `/var/log` are illustrative; substitute your own.

1. First-Response Triage (the first 60 seconds)

You just logged in (or got paged). Before forming a theory, take the machine's vitals — how long has it been up, what's failed, and what has the kernel been complaining about.

```
uptime                    # load average (1/5/15 min) + how long since last boot
uptime -s                 # exact boot time — did it reboot unexpectedly?
systemctl --failed        # every unit in a failed state, at a glance
journalctl -p err -xb     # all errors since THIS boot, with -x hints
dmesg -T | tail -n 40     # kernel ring buffer, human-readable timestamps
```

- A load average well above your CPU count (`nproc`) means the box is saturated — jump to §5.
- A very recent `uptime -s` you didn't expect = an unplanned reboot; the *why* is in `journalctl -b -1 -p err` (the previous boot) and §4 (OOM) or §3 (disk).
- `systemctl --failed` is the fastest signal of *what* broke — feed each failed unit into §2.
- In `dmesg -T`, scan for `Out of memory`, `I/O error`, `EXT4-fs error`, `Hardware Error` — each points at a specific later section.

2. Failed systemd Services & Unit-Not-Found

A unit shows `failed`, or `systemctl start` says `Unit not found`. Read the status and the journal *for that unit* before touching config.

```
systemctl status nginx.service --no-pager -l   # state, last exit code, recent log lines
journalctl -u nginx.service -b --no-pager | tail -n 50
systemctl cat nginx.service                    # the effective unit file + drop-ins actually loaded

# After editing any unit file or drop-in you MUST reload the manager:
systemctl daemon-reload
systemctl reset-failed nginx.service           # clear the failed latch before retrying
systemctl restart nginx.service
```

- `Unit not found` → the `.service` file isn't where systemd looks. Confirm with `systemctl list-unit-files | grep nginx`; drop-in units live in `/etc/systemd/system/` or `/lib/systemd/system/`. A file added by hand needs `daemon-reload`.
- `status=203/EXEC` → the `ExecStart` binary path is wrong or not executable. `status=200/CHDIR` → bad `WorkingDirectory`.
- `Active: activating (auto-restart)` looping → the process crashes on start; the real error is in `journalctl -u <unit>`, not in systemd.
- `Result: exit-code` with a non-zero `Main PID exited` → the app itself failed; check its own log and config, not the unit.

3. Disk Full & Read-Only Filesystem

Writes are failing, or an app logs `No space left on device` / `Read-only file system`. Find *which* mount, then *what* filled it.

```
df -h                     # per-mount usage — find the 100% filesystem
df -i                     # inodes: 100% inodes = full even when df -h shows free space
du -x -h -d1 /var 2>/dev/null | sort -rh | head   # biggest dirs under a mount (-x = stay on it)

# Space held by a DELETED but still-open file (df full, du can't find it):
lsof +L1 2>/dev/null | awk '$5=="REG" && $NF==0 {print}'   # or: lsof | grep deleted
```

- `df -h` full but `du` doesn't add up → a process is holding a deleted file open (classic with rotated logs). Find it with `lsof +L1`; free the space by restarting or `truncate`-ing via the holder, *not* by re-deleting.
- `df -i` at 100% with space free → too many tiny files (mail spool, session dirs, PHP sessions). Delete files, not bytes.
- A filesystem gone **read-only** is the kernel protecting itself after an I/O or fs error — confirm in `dmesg -T | grep -iE "error|remount|read-only"`.

```
mount | grep ' / '                # confirm current mount options (ro vs rw)
# Only after checking dmesg — unclean fs needs fsck, not a blind remount:
mount -o remount,rw /             # re-mount read-write once the cause is understood
```

- Never blind-`remount,rw` a filesystem that went read-only from I/O errors — schedule an `fsck` (unmounted or via `fsck -f` at next boot); forcing rw can compound corruption.

---

4. Out of Memory / OOM Killer

The box got sluggish then a process vanished, or an app logs nothing and just dies. The kernel's OOM killer sacrifices a process to save the system — the victim is a symptom, not always the cause.

```
free -h                  # used / free / available + swap; "available" is the number that matters
dmesg -T | grep -iE "killed process|out of memory|oom-kill"   # who got killed and when
journalctl -k -b | grep -i "oom"                              # same, from the kernel journal

# Top memory consumers right now:
ps -eo pid,ppid,rss,comm --sort=-rss | head -n 10   # RSS in KB
```

- Trust `available`, not `free` — Linux uses free RAM for cache, which is reclaimable. Trouble is when `available` is near zero *and* swap is exhausted.
- The `dmesg` OOM block lists the killed PID and an `oom_score_adj` table — the killer picks the highest `oom_score`, usually the biggest RSS, not the actual leaker.
- Inspect a suspect process's OOM weighting: `cat /proc/<pid>/oom_score` and `cat /proc/<pid>/status | grep -i vmrss`.
- Fixes: cap the offender (systemd `MemoryMax=`), add swap as a buffer (`swapon --show` to check), or fix the leak. Protect a critical process with `oom_score_adj` (e.g. `-1000` for sshd via a drop-in) so it's killed last.

---

5. High CPU / Load Average

`uptime` shows a load average above your core count and everything feels slow. Separate *CPU-bound* from *I/O-wait* from *too many processes* — they look identical from the outside.

```
uptime                   # load avg vs nproc: load 8 on a 4-core box = 2x oversubscribed
nproc                    # how many cores you actually have
top -b -n1 | head -n 20  # snapshot: %Cpu line + top processes by CPU

# Per-process CPU over a real interval (need sysstat: apt/dnf install sysstat):
pidstat -u 2 3           # %CPU per process, sampled every 2s, 3 times
# Where is the time going — user, system, or I/O wait?
top -b -n1 | grep '%Cpu'  # high 'wa' = disk-bound (see §3), high 'sy' = kernel/syscalls
```

- High load + high `%wa` (I/O wait) means the CPU is *waiting on disk*, not computing — chase disk/storage (§3), not the CPU.
- High load with low CPU% often = hundreds of processes in `D` state (uninterruptible sleep, usually stuck I/O): `ps -eo state,pid,comm | grep '^D'`.
- A single process pinned at ~100% × one core is a hot loop or GC storm — get its stack with `top -H -p <pid>` (per-thread) before killing.
- Load is a *queue length*, not a percentage: load == `nproc` is fully busy; sustained above it means work is backing up.

---

6. SSH: Connection Refused & Permission Denied (publickey)

Two very different failures that get conflated. `Connection refused` means nothing is listening (or a firewall dropped you); `Permission denied (publickey)` means you reached sshd but auth failed.

```
# From your laptop — verbose handshake tells you HOW far you got:
ssh -v user@node-1       # watch for "Connection refused" vs "Permission denied (publickey)"

# On the box (via console/out-of-band if SSH is dead):
systemctl status sshd --no-pager       # RHEL: sshd; Debian: ssh.service (alias sshd)
ss -tlnp | grep ':22'                  # is sshd actually listening on 22?
journalctl -u ssh -b | tail -n 30      # Debian; use -u sshd on RHEL
sshd -t                                # validate sshd_config before any restart
```

- **Connection refused / timeout** → sshd isn't running (`systemctl status`), isn't listening on that port (`ss -tlnp`), or a firewall/security-group blocks it. Timeout = dropped by firewall; refused = reached host, no listener.
- **Permission denied (publickey)** → sshd is fine; the key/permissions are wrong. Server-side auth log: `journalctl -u ssh | grep sshd`. Common causes: home or `~/.ssh` group-writable (sshd refuses `700`/`600` violations), key not in `authorized_keys`, or `PubkeyAuthentication no`.
- Fix perms: `chmod 700 ~/.ssh && chmod 600 ~/.ssh/authorized_keys`; ownership must be the login user, not root.
- Always `sshd -t` before `systemctl restart ssh` — a bad config restart can lock you out of a remote box for good.

---

7. Network Unreachable & DNS Resolution

"The server can't reach anything" spans several layers: no link, no route, no name resolution, or a firewall. Test them in order — link → route → DNS → application.

```
ip -br addr              # interfaces + IPs, one line each (is the NIC even UP?)
ip route                 # default gateway present? "default via ..." must exist
ping -c3 1.1.1.1         # raw IP reachability — isolates routing from DNS
ss -tlnp                 # what's listening locally (rule out "port closed" vs "host down")

# DNS specifically (name resolves but IP pings = pure DNS problem):
dig +short github.com    # or: dig @1.1.1.1 github.com to bypass local resolver
resolvectl status        # systemd-resolved: which nameservers are actually in use
cat /etc/resolv.conf     # the resolver the stub actually reads
```

- `ping 1.1.1.1` works but `dig github.com` fails → **DNS only**. Check `resolvectl status` / `/etc/resolv.conf` for the wrong or unreachable nameserver.
- No `default via` in `ip route` → no gateway; nothing off-subnet is reachable regardless of DNS.
- `Network is unreachable` from a command = routing/link (§ this section); `Connection refused` from a command = you reached the host but the port's closed (that's the *remote* service, not your network).
- On RHEL check firewall with `firewall-cmd --list-all`; on Ubuntu `ufw status`. A silently dropped packet = timeout; a rejected one = refused.

---

8. Permissions, sudoers & Mounts (fstab Boot Failure)

Permission-denied on files, a broken `sudo`, or — worst — a box that won't finish booting because of a bad `/etc/fstab` line.

```
ls -ld /var/www /var/www/html          # trace perms UP the tree; a parent 700 blocks all below
namei -l /var/www/html/index.html      # show ownership/mode at EVERY path component
id www-data                            # what groups does the service user actually have?

# sudoers — NEVER edit /etc/sudoers with a plain editor:
visudo -c                              # syntax-check sudoers + /etc/sudoers.d/*
```

- `namei -l` is the fastest way to find *which* directory in a path denies access — it's almost always a parent, not the file.
- A syntax error in `/etc/sudoers` can lock out all sudo. Always edit via `visudo` (validates on save) and put custom rules in `/etc/sudoers.d/` with `visudo -f`.

For a bad `fstab` mount that hangs or fails boot:

```
findmnt --verify                       # validate /etc/fstab entries WITHOUT rebooting
mount -a                               # dry-run the fstab mounts now (catches typos safely)
systemctl list-units --failed --type=mount
```

- A mount in `fstab` for a device that's missing will drop the boot into emergency mode (unless the entry has `nofail`). Add `nofail` (and a sane `x-systemd.device-timeout=`) to non-critical mounts so a missing disk doesn't block boot.
- Always run `mount -a` after editing `fstab` — if it errors *now*, it will fail the *next boot*; fix it before you reboot, not after.

---

9. Package Manager: Lock, Broken Deps & Half-Configured

`apt`/`dnf` refuses to run ("could not get lock"), or a previous install left the system half-configured.

```
# --- Debian / Ubuntu ---
ps aux | grep -E "apt|dpkg|unattended" | grep -v grep   # who holds the lock (often unattended-upgrades)
lsof /var/lib/dpkg/lock-frontend 2>/dev/null            # the exact holder PID
dpkg --configure -a                                     # finish any half-configured packages
apt-get -f install                                      # fix broken/unmet dependencies
apt-get update && apt-get install -f

# --- RHEL / Rocky / Alma ---
ps aux | grep -E "dnf|yum|packagekit" | grep -v grep
dnf check                                               # report broken deps / duplicates
rpm --rebuilddb                                         # rebuild a corrupt rpm database
dnf history                                             # find + undo the last transaction
```

- "Could not get lock `/var/lib/dpkg/lock-frontend`" is almost always `unattended-upgrades` running in the background — **wait for it** or find the PID with `lsof`; do *not* just `rm` the lock file while a process holds it (that corrupts the dpkg DB).
- If a previous run was killed mid-install, `dpkg --configure -a` (Debian) or `dnf history redo/undo <id>` (RHEL) is the correct recovery, not deleting packages.
- Unmet dependencies on Debian → `apt-get -f install`. Only remove a lock file (`rm /var/lib/apt/lists/lock`) after you've *confirmed* no apt/dpkg process is running.

---

10. Cron Jobs Not Running & Time Sync

A scheduled job "didn't run." First prove the daemon is up and read its log, then rule out the two silent killers: environment differences and a wrong clock.

```
systemctl status cron       # Debian/Ubuntu: the daemon is 'cron'
systemctl status crond      # RHEL/Rocky/Alma: it's 'crond'
crontab -l                  # the CURRENT user's crontab (jobs are per-user!)
journalctl -u cron -b | tail -n 30        # (crond on RHEL) — did cron even fire the job?
grep -RiE "cron|CMD" /var/log/syslog 2>/dev/null | tail   # Debian cron activity

# Clock / time sync — cron fires on wall-clock time:
timedatectl                 # is NTP active + system clock synchronized?
chronyc tracking 2>/dev/null   # RHEL: chrony offset/stratum
```

- Cron runs with a **minimal environment** — no `PATH` from your shell, no profile. A job that works by hand but not in cron almost always uses a command not on cron's bare `PATH`; use absolute paths (`/usr/bin/python3`) and redirect output (`>> /var/log/myjob.log 2>&1`) so failures are visible.
- The journal shows cron *firing* the line but the job can still fail internally — that's why you log its stdout/stderr. No `CMD` line at all = cron never scheduled it (bad crontab syntax, missing trailing newline, or wrong user's crontab).
- `timedatectl` showing `System clock synchronized: no` or a large offset means jobs fire at the wrong time (or `@reboot` misbehaves). Fix time sync (`systemctl enable --now systemd-timesyncd` or `chronyd`) before chasing "missed" runs.

11. Triage Decision Tree

```
Server is sick / unreachable
 ├─ Can't SSH in? ──────────────── refused/timeout → §6 (sshd down / firewall / no listener)
 │                                 "Permission denied (publickey)" → §6 (keys / perms)
 ├─ Reachable but a service is down? → §2 (systemctl --failed, journalctl -u)
 ├─ Whole box slow / high load?
 │     ├─ top %wa high (I/O wait) ──────→ §3 (disk / storage)
 │     ├─ available RAM ~0 + swap full ──→ §4 (OOM)
 │     └─ CPU pinned, wa low ────────────→ §5 (hot process / oversubscribed)
 ├─ "No space left" / read-only fs? ──→ §3 (df -h, df -i, lsof +L1, dmesg before remount)
 ├─ Process got killed with no app error? → §4 (dmesg OOM)
 ├─ Can't reach the network / a name won't resolve? → §7 (ip route, ping IP, dig)
 ├─ Permission denied on files / sudo broken / boot hung on a mount? → §8
 ├─ apt/dnf won't run or left things half-installed? → §9
 └─ A cron job "didn't run"? ──────────→ §10 (daemon up? PATH? clock synced?)
```

12. Copy/Paste One-Shot Triage

Run this read-only block first on any sick box — it fingerprints all the common failure classes in one pass.

```
echo "== uptime/load ==";   uptime; echo "cores: $(nproc)"
echo "== failed units ==";  systemctl --failed --no-legend
echo "== recent errors ==";journalctl -p err -b --no-pager | tail -n 15
echo "== kernel/oom/io ==";  dmesg -T 2>/dev/null | grep -iE "oom-kill|killed process|I/O error|read-only|Hardware Error" | tail
echo "== disk ==";          df -h --output=source,pcent,target -x tmpfs -x devtmpfs | awk 'NR==1 || $2+0>=85'
echo "== inodes ==";        df -i --output=source,ipcent,target -x tmpfs -x devtmpfs | awk 'NR==1 || $2+0>=85'
echo "== memory ==";        free -h
echo "== top cpu ==";       ps -eo pid,pcpu,rss,comm --sort=-pcpu | head -n 6
echo "== top mem ==";       ps -eo pid,pcpu,rss,comm --sort=-rss  | head -n 6
echo "== net ==";           ip -br addr; ip route | grep '^default' || echo "NO DEFAULT ROUTE"
echo "== dns ==";           dig +short github.com || echo "DNS FAIL"
echo "== time ==";          timedatectl | grep -iE "synchronized|NTP"
```

13. Incident Notes Template

```
INCIDENT: Linux Server — <hostname / node-1>
Started (UTC):        ____    Detected by: ____    Sev: ____
Symptom:              [ ] service down  [ ] unreachable/SSH  [ ] slow/high load
                      [ ] disk full/ro  [ ] OOM  [ ] network/DNS  [ ] cron  [ ] pkg mgr
First vitals:         load __/__ cores   mem avail ____    swap used ____
                      failed units: ____
Localized to:         §__  component: ____
Evidence:             (df/inodes __%, dmesg OOM PID ____, top %wa ____, ss :22 ____,
                       ip route ____, dig ____, journalctl line ____)
Read-only checks done: [ ] yes  Second engineer for change: [ ] yes  [ ] n/a
Action taken:         (restart / remount / fsck / kill / config + who + when)
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (alerts, capacity, MemoryMax=, nofail in fstab, log rotation,
                       monitoring, oom_score_adj for sshd) ____
```

before running in production. When you're stuck, paste the errors into the free AI Incident Response assistant at devopsaitoolkit.com/dashboard/incident-response.