



8 AM: The Frozen Userspace

The Symptom

- Status: GCP Instance RUNNING. Port 22 OPEN.
- The Hang: SSH stuck at banner exchange.
- The Diagnosis: TCP is alive, but the userspace is entirely frozen. Resource exhaustion.

The Early Warning

- Logs stopped 20 hours ago. rsyslogd couldn't even write to its own files (action suspended).
- A hard reset was the only way back in.

Act 1: The Infinite Crash Loop



Snap Chrome + PM2 = Cgroup Mismatch

Snap packages require strict cgroup confinement. PM2 runs them in a different system slice. The result? Instant rejection.

The Avalanche Matrix

- **Action:** Chrome exits -> PM2 immediately restarts it.
- **Missing Safety:** No max_restarts. No restart_delay.
- **The Damage:** 3,055 rejections in 33 hours. Thousands of instant forks eating up PIDs, memory, and disk I/O.

The Fix

Replaced Snap with .deb package and enforced PM2 restart limits.

The Catalyst: Port 22 Under Siege

6,066 Brute Force Attempts

While Chrome was crash-looping, the public IP was being hammered. Over 33 hours, bots attempted to guess the SSH password 6,000+ times.

The Avalanche Effect

Each login attempt spawns a new sshd child process. Under normal conditions, this is harmless. Under resource pressure, it accelerates the crash.

The Missing Buffer

With 16GB of RAM and Zero Swap, the kernel had no buffer time to reclaim memory. It went straight to a deadlock.

Fixes Applied: Installed fail2ban, added 4GB Swap, and limited unauthenticated SSH connections via MaxStartups 5:50:10.



Act 2: Unbounded Memory

Treating the Symptom vs. The Constitution

Hours later, the server died again. The crash loop was gone, but the processes had no ceilings.



The Memory Footprint (16GB Total)

Chrome (6 processes): 861 MB

Docker & LLM Gateways: ~750 MB+

Claude Code Sessions: 1-2 GB each
(growing dynamically)

Defense in Depth Applied:

- **OOM Priorities:** Instructed the kernel (`oom_score_adj`) to never kill SSH, and sacrifice Chrome first.
- **Hard Limits:** Applied memory ceilings to Docker and PM2. Added an 85% memory watchdog cron script.

Right-Sizing: More Power, Less Cost

The Realization:

The server was running on a c3-standard-4 (optimized for HPC gaming, premium CPU). But the workload was entirely I/O bound—waiting for APIs and idle 99% of the time.



Ultra-rounde-styled comparison matrix

Instance Type	Specs	Cost (Approx.)	Status
c3-standard-4	4 vCPU / 16GB	~\$152/mo	Current (Overpriced)
e2-highmem-4	4 vCPU / 32GB	~\$131/mo	Save \$21, 2x RAM

The Takeaway: Don't assume you're safe after the first fix. Layer your defenses, limit your processes, and always check if you're paying for the right hardware.