Redis Troubleshooting Runbook Pack

A senior engineer's incident runbook for the failures that page you: OOM, persistence stalls, replication breaks, failover, and cluster splits. DevOps AI ToolKit · devopsaitoolkit.com

Redis fails loudly and in a handful of well-worn ways. A command that used to take microseconds suddenly errors with `OOM`, `MISCONF`, `LOADING`, `CLUSTERDOWN`, or `NOAUTH` — or it just hangs. Almost every one of these maps to a single subsystem: memory and eviction, persistence to disk, replication, Sentinel/Cluster topology, latency, connections, fork/fragmentation, or auth. This pack walks each one. Start with `INFO`, read the actual error string the client returned, and let it point you at the section.

> Examples use `redis-1:6379` and an app namespace `myapp`. Run the read-only checks first. `CONFIG SET`, `FLUSHALL`, `FLUSHDB`, `DEBUG`, and `FAILOVER` change live state — get a change record and a second engineer before you run them in production. If auth is on, add `-a "$REDISCLI_AUTH"` (or export `REDISCLI_AUTH`) so the password stays out of your shell history.

1. First-Response Triage

```
# Is it even up and answering?
redis-cli -h redis-1 -p 6379 PING                    # expect PONG
redis-cli -h redis-1 ping 2>&1 | grep -Ei "NOAUTH|LOADING|MASTERDOWN|CLUSTERDOWN"

# One-screen health snapshot — read these fields before anything else
redis-cli -h redis-1 INFO | grep -E \
  "redis_version|uptime_in_seconds|role:|connected_clients|blocked_clients|\
used_memory_human|maxmemory_human|mem_fragmentation_ratio|evicted_keys|\
rejected_connections|rdb_last_bgsave_status|aof_last_bgrewrite_status|\
master_link_status|loading:|instantaneous_ops_per_sec"

# Is latency real or perceived? (intrinsic = kernel/scheduler, not Redis)
redis-cli -h redis-1 --latency                       # Ctrl-C after ~10s; watch max
redis-cli -h redis-1 --intrinsic-latency 5
```

- `PONG` but the app is slow → not a liveness problem; go to §7 (latency).
- The error string is the fastest diagnosis: `OOM` → §2, `MISCONF` → §3, `master_link_status:down` → §4, `CLUSTERDOWN` → §6, `NOAUTH` / `WRONGPASS` → §10.
- `loading:1` → Redis is reading RDB/AOF into memory at startup and *will* reject most commands until done. Wait, don't restart.

2. Out-of-Memory & Eviction

```
redis-cli -h redis-1 INFO memory | grep -E \
  "used_memory:|used_memory_human|used_memory_rss_human|maxmemory:|maxmemory_human|maxmemory_policy"
redis-cli -h redis-1 INFO stats | grep -E "evicted_keys|keyspace_misses|expired_keys"
redis-cli -h redis-1 CONFIG GET maxmemory
redis-cli -h redis-1 CONFIG GET maxmemory-policy
```

- `OOM command not allowed when used memory > 'maxmemory'` on a write → `used_memory` hit the `maxmemory` ceiling **and** `maxmemory-policy` is `noeviction` (the default). Writes fail; reads still work.
- `evicted_keys` climbing → an eviction policy *is* active and Redis is shedding keys to stay under the cap. Fine for a cache, data-loss for a datastore — know which one this instance is.
- Policies that matter: `noeviction`, `allkeys-lru`, `allkeys-lfu`, `volatile-lru`, `volatile-ttl`. `volatile-*` only evict keys that have a TTL — if nothing has a TTL, a `volatile-*` policy behaves like `noeviction` and you still get `OOM`.

```
# Immediate relief (cache instances only — CONFIG SET is live, log it):
redis-cli -h redis-1 CONFIG SET maxmemory-policy allkeys-lru
# Find what's eating memory before raising the cap:
redis-cli -h redis-1 --bigkeys                       # sampled, safe in prod
redis-cli -h redis-1 MEMORY USAGE myapp:session:largest
```

> Never raise `maxmemory` past what the host can hold — pushing `used_memory_rss` toward physical RAM invites the kernel OOM-killer, which is a hard crash, not a graceful eviction.

3. Persistence Failures

```
redis-cli -h redis-1 INFO persistence | grep -E \
  "rdb_last_bgsave_status|rdb_last_save_time|rdb_changes_since_last_save|\
rdb_bgsave_in_progress|aof_enabled|aof_last_bgrewrite_status|aof_last_write_status|\
aof_rewrite_in_progress"
redis-cli -h redis-1 CONFIG GET stop-writes-on-bgsave-error
redis-cli -h redis-1 CONFIG GET dir
df -h "$(redis-cli -h redis-1 CONFIG GET dir | tail -1)"   # disk under the RDB/AOF dir
```

- `MISCONF Redis is configured to save RDB snapshots, but it's currently unable to persist to disk` → a background save failed and, because `stop-writes-on-bgsave-error yes` (the default), Redis now **refuses all writes** to protect you from silent data loss.
- `rdb_last_bgsave_status:err` or `aof_last_write_status:err` → the last save/append failed. The usual cause is a full disk, wrong permissions on `dir`, or `fork()` failing under memory pressure (see §9).
- `rdb_changes_since_last_save` large and growing while `rdb_last_save_time` is stale → snapshots aren't completing; unsaved writes are piling up in RAM.

```
# Fix the real cause first — free disk / fix permissions on `dir` — then:
redis-cli -h redis-1 BGSAVE                            # trigger a fresh snapshot; watch it succeed
redis-cli -h redis-1 INFO persistence | grep rdb_last_bgsave_status   # want :ok
# Only as a deliberate stopgap to unblock writes while you fix disk:
redis-cli -h redis-1 CONFIG SET stop-writes-on-bgsave-error no
```

4. Replication Break

```
# On the replica:
redis-cli -h redis-2 INFO replication | grep -E \
  "role:|master_host|master_link_status|master_last_io_seconds_ago|\
master_sync_in_progress|slave_read_only|slave_repl_offset"
# On the master:
redis-cli -h redis-1 INFO replication | grep -E "role:|connected_slaves|master_repl_offset|slave[0-9]"
```

- `master_link_status:down` on the replica → the link to the master is broken. Check `master_last_io_seconds_ago` (how long since data flowed) and the replica log for `MASTER <-> REPLICA sync` errors.
- `master_sync_in_progress:1` → a full resync (RDB transfer) is running; the replica is `LOADING` and won't serve until done. Big datasets take minutes.
- `READONLY You can't write against a read only replica` → your app wrote to a replica. `slave_read_only yes` is correct and protective; the bug is the client's connection target (or a failover that moved the master).
- Repeated **full** resyncs instead of partial → the replication backlog is too small. Check `repl_backlog_size` / `repl_backlog_active`; a bigger backlog lets a briefly-disconnected replica catch up via PSYNC instead of a full RDB dump.

```
redis-cli -h redis-1 INFO replication | grep -E "repl_backlog_size|repl_backlog_active"
# Point a stray replica back at the right master (topology change — coordinate it):
redis-cli -h redis-2 REPLICAOF redis-1 6379
```

5. Sentinel Failover

```
# Ask Sentinel what it thinks the topology is (Sentinel listens on 26379)
redis-cli -h sentinel-1 -p 26379 SENTINEL masters | grep -E \
  "name|ip|port|flags|num-slaves|num-other-sentinels|quorum|failover-timeout"
redis-cli -h sentinel-1 -p 26379 SENTINEL master myapp
redis-cli -h sentinel-1 -p 26379 SENTINEL get-master-addr-by-name myapp
redis-cli -h sentinel-1 -p 26379 SENTINEL sentinels myapp | grep -c name   # peers seen
```

- `flags` containing `s_down` (subjectively down) or `o_down` (objectively down) → this Sentinel, or a quorum of them, believes the master is dead. `o_down` triggers failover.
- Failover won't start unless `num-other-sentinels + 1` meets `quorum` **and** a majority of Sentinels agree to elect a leader. A 2-Sentinel deployment can't reach majority if one is down — you need an odd number, 3+.
- Stuck mid-failover → check `failover-timeout`; a new attempt won't begin until it elapses. `get-master-addr-by-name` shows who Sentinel currently believes is master — reconcile that against what `INFO replication role:` says on each node.

```
# Force a failover (manual, promotes a healthy replica — log it, it moves live traffic):
redis-cli -h sentinel-1 -p 26379 SENTINEL failover myapp
```

6. Cluster Errors

```
redis-cli -h redis-1 -p 6379 CLUSTER INFO | grep -E "cluster_state|cluster_slots_assigned|cluster_slots_ok|cluster_known_nodes|cluster_size"
redis-cli -h redis-1 CLUSTER NODES | grep -E "master|slave|fail|handshake"
redis-cli --cluster check redis-1:6379                 # slot coverage + config sanity
```

- `CLUSTERDOWN Hash slot not served` → `cluster_state:fail`. Some of the 16384 slots have no serving master. `cluster_slots_assigned` should be 16384 and `cluster_slots_ok` should match; any gap means an unreplaced master died.

- `MOVED 3999 redis-2:6379` → the key you asked for lives on another shard; a cluster-aware client follows this automatically. Seeing it constantly from a plain `redis-cli` is normal — use `redis-cli -c` to follow redirects.
- `ASK` → a slot is mid-migration; follow it to the target node for that one key. Transient during resharding.
- `CROSSSLOT Keys in request don't hash to the same slot` → a multi-key op (MSET, transaction, Lua) spanned shards. Fix by hash-tagging keys so they co-locate, e.g. `myapp:{user123}:profile` and `myapp:{user123}:cart` share the `{user123}` slot.

```
# Recover slot coverage after a node loss (topology change — coordinate carefully):
redis-cli --cluster fix redis-1:6379
```

7. Latency & Slow Commands

```
redis-cli -h redis-1 SLOWLOG GET 10                    # last 10 slow commands (µs + args)
redis-cli -h redis-1 CONFIG GET slowlog-log-slower-than  # threshold in microseconds
redis-cli -h redis-1 LATENCY DOCTOR                     # plain-English latency analysis
redis-cli -h redis-1 LATENCY LATEST                     # per-event spikes
redis-cli -h redis-1 INFO commandstats | sort -t= -k2 -rn | head
```

- Redis is single-threaded for command execution: one slow `O(n)` command blocks *every* other client. The usual culprits are `KEYS *`, `SMEMBERS` / `HGETALL` / `LRANGE 0 -1` on huge collections, and un-paginated scans.
- `SLOWLOG GET` shows the exact offending command and its argument list — this is usually a smoking gun straight to the responsible service.
- Use `SCAN` / `HSCAN` / `SSCAN` (cursor-based, non-blocking) instead of `KEYS`. Find the fat collections with `--bigkeys`.
- `blocked_clients` high in `INFO clients` → clients parked on `BLPOP` / `BRPOP` / `XREAD BLOCK` / `WAIT`. Often intended, but a leak of blocking calls looks like a hang.

```
redis-cli -h redis-1 --bigkeys                         # largest key per type (sampled)
redis-cli -h redis-1 SLOWLOG RESET                      # clear after you've captured it
```

8. Connections & Clients

```
redis-cli -h redis-1 INFO clients | grep -E "connected_clients|blocked_clients|maxclients|client_recent_max_input_buffer"
redis-cli -h redis-1 CONFIG GET maxclients
redis-cli -h redis-1 INFO stats | grep -E "rejected_connections|total_connections_received"
redis-cli -h redis-1 CLIENT LIST | awk '{for(i=1;i<=NF;i++) if($i ~ /^addr=/) print $i}' \
  | cut -d= -f2 | cut -d: -f1 | sort | uniq -c | sort -rn | head   # connections per client host
```

- `ERR max number of clients reached` and `rejected_connections` climbing → you hit `maxclients` (default 10000). Either a connection leak (no pooling / clients not closing) or genuine growth.
- `total_connections_received` very high relative to `connected_clients` → churn: clients open a connection per request instead of pooling. Fix the client, not Redis.
- Note: Redis reserves ~32 file descriptors for its own use, so effective `maxclients` can be a touch below the configured value if the OS `ulimit -n` is low. Check `INFO clients` for the effective number.

```
redis-cli -h redis-1 CLIENT LIST | grep -E "idle=[0-9]{4,}"        # very idle connections
# Raising the cap is a stopgap; verify the host FD limit can back it first:
redis-cli -h redis-1 CONFIG SET maxclients 20000
```

9. Memory Fragmentation & Fork

```
redis-cli -h redis-1 INFO memory | grep -E \
  "used_memory_human|used_memory_rss_human|mem_fragmentation_ratio|mem_allocator|allocator_frag_ratio|activedefrag"
redis-cli -h redis-1 INFO stats | grep -E "latest_fork_usec|total_forks"
redis-cli -h redis-1 INFO persistence | grep -E "current_fork_perc"
```

- `mem_fragmentation_ratio` = `used_memory_rss` / `used_memory`. Above ~1.5 → real fragmentation (RSS holds memory the allocator won't return). **Below 1.0** → Redis has been swapped to disk by the OS, which is far worse for latency; check host swap immediately.
- `latest_fork_usec` large → `BGSAVE` / `BGREWRITEAOF` forks a child, and the copy-on-write (COW) fork itself stalls the parent for that many microseconds. On a big, write-heavy dataset the fork pause alone can trip client timeouts.
- COW means a background save can transiently double memory if the parent rewrites most keys during the save — this is why fork can fail (`Cannot allocate memory`) even when steady-state `used_memory` fits. Set `vm.overcommit_memory = 1` on the host so fork can reserve.

```
# Reclaim fragmentation live without a restart (jemalloc only — CPU cost while it runs):
redis-cli -h redis-1 CONFIG SET activedefrag yes
redis-cli -h redis-1 MEMORY DOCTOR                      # Redis's own read on memory health
```

10. Auth & Security

```
redis-cli -h redis-1 PING                              # NOAUTH here means auth is required
redis-cli -h redis-1 -a "$REDISCLI_AUTH" PING          # supply the password (from env, not literal)
redis-cli -h redis-1 -a "$REDISCLI_AUTH" ACL WHOAMI    # which ACL user am I?
redis-cli -h redis-1 -a "$REDISCLI_AUTH" ACL LIST      # configured users + rules
redis-cli -h redis-1 -a "$REDISCLI_AUTH" CONFIG GET requirepass
redis-cli -h redis-1 -a "$REDISCLI_AUTH" CONFIG GET protected-mode
```

- `NOAUTH Authentication required` → `requirepass` (or an ACL) is set and the client sent no `AUTH`. Fix the client's credentials; don't unset `requirepass` to "make it work."
- `WRONGPASS invalid username-password pair or user is disabled` → wrong password, or an ACL user is disabled/misconfigured. `ACL LIST` shows each user's rules (`on`/`off`, key patterns, allowed commands).
- `NOPERM this user has no permissions to run the '...' command` → ACLs are working as designed; the app's user lacks that command or key pattern. Grant narrowly via `ACL SETUSER`, don't hand it `allcommands`.
- `protected-mode yes` + no `requirepass` + a non-loopback bind → Redis refuses external connections entirely. Set a password and an explicit `bind`; never expose an unauthenticated Redis to a public interface.

---

11. Diagnostic Decision Tree

```
Redis incident — what did the client actually get back?
 ├─ Connection refused / timeout ──────── PING fails → process down or LOADING (§1); check `bind`/firewall/`protected-mode` (§10)
 ├─ "OOM command not allowed" ─────────── §2  used_memory vs maxmemory + maxmemory-policy
 ├─ "MISCONF ... unable to persist" ────── §3  rdb/aof status + disk under `dir` + stop-writes-on-bgsave-error
 ├─ "READONLY" / master_link_status:down  §4  replication link, then §5 Sentinel if the master truly moved
 ├─ "CLUSTERDOWN" / "MOVED" / "CROSSSLOT" §6  cluster_state, slot coverage, hash tags
 ├─ "NOAUTH" / "WRONGPASS" / "NOPERM" ──── §10 requirepass / ACL user + rules
 └─ PONG but slow / hanging ────────────── §7 SLOWLOG + LATENCY DOCTOR; then §8 clients, §9 fork/frag/swap
```

---

12. Copy/Paste One-Shot Triage

```
H=redis-1; P=6379    # add -a "$REDISCLI_AUTH" to each line if auth is on
echo "== liveness ==";     redis-cli -h $H -p $P PING
echo "== role/link ==";    redis-cli -h $H -p $P INFO replication | grep -E "role:|master_link_status|connected_slaves"
echo "== memory ==";       redis-cli -h $H -p $P INFO memory | grep -E
"used_memory_human|maxmemory_human|maxmemory_policy|mem_fragmentation_ratio"
echo "== eviction ==";     redis-cli -h $H -p $P INFO stats | grep -E "evicted_keys|rejected_connections"
echo "== persistence =="; redis-cli -h $H -p $P INFO persistence | grep -E
"rdb_last_bgsave_status|aof_last_write_status|rdb_changes_since_last_save"
echo "== clients ==";      redis-cli -h $H -p $P INFO clients | grep -E "connected_clients|blocked_clients|maxclients"
echo "== cluster ==";      redis-cli -h $H -p $P CLUSTER INFO 2>/dev/null | grep -E "cluster_state|cluster_slots_ok" || echo "(standalone)"
echo "== slowlog ==";      redis-cli -h $H -p $P SLOWLOG GET 3
echo "== latency ==";      redis-cli -h $H -p $P LATENCY DOCTOR
```

---

13. Incident Notes Template

```
INCIDENT: Redis <symptom / error string>
Started (UTC):          ____    Detected by: ____    Sev: ____
Instance:               host:port ____    role: [ ] master [ ] replica   topology: [ ] standalone [ ] Sentinel [ ] Cluster
Client error string:    ____ (e.g. OOM / MISCONF / READONLY / CLUSTERDOWN / NOAUTH)
Impact:                 (which app / % of ops failing / reads vs writes)
Memory:                 used ____  maxmemory ____  policy ____  frag_ratio ____  evicted_keys ____
Persistence:            rdb_last_bgsave_status ____  aof_last_write_status ____  disk free under dir ____
Replication:            master_link_status ____  last_io_seconds_ago ____  full resyncs? ____
Sentinel/Cluster:       quorum ____  cluster_state ____  slots_ok ____
Latency:                slowlog top command ____  blocked_clients ____
Action taken:           (CONFIG SET / restart / failover / disk fix + who + when)
Result:                 [ ] recovered @ ____   [ ] escalated to ____
Root cause:             ____
Follow-ups:             (maxmemory sizing, TTLs, backlog size, ACLs, overcommit, alerts) ____
```
