# The Redis Architecture: Memory-First Persistence, Structures, and Distributed Caching

Date: 2026-01-08

## The 80/20 Summary

- In-memory first: design for sub-ms latency; add persistence only as needed.
- Pick persistence by role: pure cache (often none), warm restarts (RDB), semi-durable (AOF everysec or mixed).
- Use the right structure to avoid O(N) on hot paths.
- Scale with replicas (HA) or Cluster (sharding); default to cache-aside.
- Control RAM via `maxmemory` + `allkeys-lfu` or `allkeys-lru`.

## Memory-First Design

- Single-threaded event loop per instance; scale cores via more instances.
- RAM storage with async persistence; avoid blocking ops on hot keys.
- Fork + COW during RDB/AOF rewrite; budget extra memory while persisting.

## Persistence (Trade-offs)

- RDB snapshots: smallest files, fastest restart; may lose last N seconds. Use for caches that can rewarm.
- AOF: append writes; `everysec` ≈ 1s loss, `always` ≈ 0 loss; larger files. Use for semi-durable needs.
- Mixed (RDB preamble + AOF tail): fast boot + near-AOF recency.
- Defaults:
    - Pure cache: no persistence or infrequent RDB; set eviction.
    - Warm cache: RDB 5–15 min or AOF `everysec`.
    - Semi-durable: AOF `everysec` with monitored rewrites and SSD.

## Core Data Structures (Fit-for-Purpose)

- Strings: counters/blobs; prefer atomic ops (`INCRBY`, ranges, bit ops).
- Hashes: many small fields; partial updates are cheap.
- Lists: queues/recent items; cap size, avoid huge ranges.
- Sets: unique members/tags; fast membership and algebra.
- Sorted Sets: ranks/time windows; trim by score/time.
- Streams: log + consumer groups; durable-ish queues.
- Bitmaps/HyperLogLog: compact booleans/approx counts at scale.
- Geo: nearest lookups; good for proximity queries.

## Distributed Caching Roles

- Topologies: Replication + Sentinel (HA, read scale), Cluster (sharding, multi-master), or client-side sharding (simple, DIY failover).

- Patterns: cache-aside (most common), read-through (proxy/sidecar), write-through (stronger consistency), write-back (fast writes, risk).
- Invalidation: TTLs (bound staleness/memory) or event-driven updates.
- Eviction under `maxmemory`: `allkeys-lfu` (skewed traffic), `allkeys-lru` (simple default), `volatile-*` when only TTL keys should evict.

## Performance & Ops

- Cut RTTs with pipelining/batching; use Lua for tiny atomic multi-steps.
- Always set TTLs for cache keys; cap collection sizes (trim lists/zsets).
- Monitor: ops/sec, hit ratio, used memory, evictions, fork time, AOF fsync.
- Storage: fast disks for AOF/RDB; schedule rewrites off-peak; watch COW headroom.

## Pitfalls to Avoid

- Insufficient COW memory during snapshot/rewrite → OOM/latency spikes.
- Large blocking reads on hot keys → paginate/summary patterns.
- No TTL/eviction on caches → memory bloat, surprise evictions.
- Assuming replica strong consistency → stale reads; pin critical reads to primary or use `WAIT`.

## Final Take

Decide the role first (cache vs store), choose persistence to match loss tolerance, enforce `maxmemory` with LFU/LRU, and pick structures that make your workload O(1) or efficient ranges. Scale with replicas or Cluster and keep an eye on fork/COW and storage latency.