Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: REDI • **Updated:** 05 Nov 2026
https://mockattempt.com/courses/redis-training

REDI • IN-MEMORY CACHING, DISTRIBUTED LOCKS & REAL-TIME STREAMS

BEGINNER TO PRINCIPAL REDIS & DISTRIBUTED SYSTEMS ARCHITEC **100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER**

Redis Training

Master Redis 7 from Single-Threaded epoll Event Loops, Cache-Aside Stampede Defense & Rich Data Structures (Strings, Hashes + HEXPIRE, Lists, Sets, SkipList Sorted Sets, Streams) to WATCH/MULTI/EXEC, Atomic Lua Token-Bucket Rate Limiters, RTT Pipelining, RDB/AOF Persistence, ACL Security & 16,384-Slot Redis Cluster Partitioning — with 200 Mock Tests & Voice MockAttempt AI Interviews at ₹1,999 (Incl. 18% GST).

Duration: 14–16 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,999 (Net Tuition ₹1,694 + Included 18% GST ₹305) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹10/Guided Hr (~₹48/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (Python 3.12 & Atomic Lua): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: Redis 7.4 Server & redis-cli • Python 3.12 redis-py & ConnectionPool • Strings, Hashes (HEXPIRE), Lists, Sets & ZSets • Redis Streams, HyperLogLog & Bitmaps • WATCH / MULTI / EXEC & Atomic Lua (EVALSHA) • Pipelining, RDB + AOF Hybrid Persistence • Redis ACLs, TLS & 16,384-Slot Redis Cluster

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Select memory-optimal Redis data structures (`SDS`, `listpack` Hashes, `quicklist` Lists, `intset` Sets, `skiplist` Sorted Sets, Streams, HyperLogLog) with exact `O(1)` vs `O(log N)` complexity awareness
2. Eliminate Cache Stampedes, Cache Penetration, and Avalanche outages using jittered TTLs, singleflight locks, Bloom filters, and `allkeys-lfu` / `volatile-lru` eviction
3. Execute race-free distributed rate limiters and locks using `SET NX PX` + atomic `EVALSHA` Lua scripts, and cut network RTT overhead 20x via `redis-py` Pipelining
4. Clear 244 Timed Mock Exams (10,000 Redis & Distributed Caching MCQs) and 200 Spoken Voice + MockAttempt AI Systems Architecture Interviews
5. Choosing optimal Redis data structures: Strings, Hashes, Sets, Sorted Sets (`ZSET`), Bitmaps, HyperLogLog, and Streams.
6. Crack LeetCode and HackerRank problems in Python 3.12 & Atomic Lua with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Production caching patterns: Cache-Aside, Write-Through, jittered TTLs, and stampede protection (single-flight locks).
8. Distributed systems primitives: sliding-window rate limiters, atomic Lua scripts, Redlock patterns, and session stores.

Eligibility & Prerequisites: Basic familiarity with Python dictionaries/lists and client-server HTTP/database concepts — starts from in-memory fundamentals from scratch • Any laptop/PC with Python 3.10+ (`pip install redis fakeredis`) and Docker or local Redis 7 Server / WSL 2 • No prior Redis industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Single-Threaded I/O Multiplexing (`epoll`), RESP3 Protocol & Redis vs RDBMS Patterns

Understand how Redis processes >100,000 ops/sec on a single command-execution thread via kernel `epoll`/`kqueue` I/O multiplexing, and architect resilient Cache-Aside layers in front of SQL databases.

Objectives: ✓ Redis Architecture — Single-Threaded `epoll` Event Loop, I/O Threads, RESP3 & Sub-Millisecond RAM Speed — Why **REmote DIctionary Server** achieves `<0.5 ms` p99 latency and `100k+ ops/sec` on a single core: zero mutex lock contention on the main command-execution thread, kernel `epoll`/`kqueue` non-blocking socket multiplexing, Redis 6/7 threaded network socket read/write (`io-threads`), and binary-safe `RESP2`/`RESP3` wire framing. • ✓ Redis vs RDBMS — Polyglot Persistence, Cache-Aside vs Write-Through & Stampede / Avalanche Defense — Why Redis complements (rather than blindly replaces) PostgreSQL/MySQL: ACID system-of-record disk durability + ad-hoc SQL joins in RDBMS vs ephemeral/semi-persistent `O(1)` hot reads, counters, leaderboards, and locks in Redis; plus Cache-Aside (` Read: Cache -> DB -> Set Cache; Write: Update DB -> Delete Cache!`) and TTL jitter. • ✓ Build, verify, and ship Module Capstone: Stampede-Resistant Cache-Aside Engine with TTL Jitter & Null-Sentinel Protection.

1.1 Redis Architecture — Single-Threaded `epoll` Event Loop, I/O Threads, RESP3 & Sub-Millisecond RAM Speed

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Single-threaded command execution & Epoll / kqueue I/O multiplexing • Core Syntax, CLI & Production APIs: redis.Redis(decode_responses=True, protocol=3), PING, INFO server / stats, SLOWLOG GET • Deep-Dive Mechanics: RESP2 / RESP3 wire protocol & UNLINK async free vs blocking DEL • Advanced Patterns: Io-threads socket read/write offload & SLOWLOG (>10ms) tail-latency audit • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inspecting RESP Framing & Sub-Millisecond Cache Round-Trip in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Architecture

Purpose: To understand and apply Redis Architecture — Single-Threaded `epoll` Event Loop, I/O Threads, RESP3 & Sub-Millisecond RAM Speed in real Redis systems — specifically: Why...

What Topic Is Next → Next Topic 1.2: Redis vs RDBMS — Polyglot Persistence, Cache-Aside vs Write-Through & Stampede / Avalanche Defense

1.2 Redis vs RDBMS — Polyglot Persistence, Cache-Aside vs Write-Through & Stampede / Avalanche Defense

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: System-of-Record RDBMS + Speed-Layer Redis & Cache-Aside (Delete cache after DB write) • Core Syntax, CLI & Production APIs: Cache-Aside (GET -> SQL fallback -> SET k v EX ttl+jitter; SQL COMMIT -> DEL/UNLINK k) • Deep-Dive Mechanics: Cache Stampede (hot-key mutex lock) & Cache Avalanche (add random TTL jitter) • Advanced Patterns: Cache Penetration (null-value / Bloom) & Write-Behind async queue trade-offs • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cache-Aside Helper with Deterministic Jitter & Null-Penetration Shield • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis vs RDBMS

Purpose: To understand and apply Redis vs RDBMS — Polyglot Persistence, Cache-Aside vs Write-Through & Stampede / Avalanche Defense in real Redis systems — specifically: Why Redis...

What Topic Is Next → Module 1 Capstone (Stampede-Resistant Cache-Aside Engine with TTL Jitter & Null-Sentinel Protection) → Next Topic 2.1: Installing Redis 7 in Production — `vm.overcommit_memory=1`, Disabling `THP` & `systemd` Supervision

Module 1 Hands-On Capstone Project (100 Marks): Stampede-Resistant Cache-Aside Engine with TTL Jitter & Null-Sentinel Protection — Implement a Python `redis-py` Cache-Aside wrapper protecting PostgreSQL against both missing-ID Cache Penetration attacks and synchronized TTL-expiry Cache Avalanches.

MODULE 2 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Linux Kernel Tuning (`overcommit_memory`, Disable THP) & Production `redis.conf` Mastery

Deploy Redis 7 on Linux with mandatory OS kernel tuning to eliminate background `fork()` latency spikes, and configure `maxmemory`, `allkeys-lfu` eviction, and `CONFIG REWRITE`.

Objectives: ✓ Installing Redis 7 in Production — `vm.overcommit_memory=1`, Disabling `THP` & `systemd` Supervision — Installing official `packages.redis.io` / Alpine OCI images (`redis:7.4-alpine`), `supervised systemd`, and why Redis logs startup warnings if Linux `vm.overcommit_memory` is `0` (`fork()` fails with `Cannot allocate memory` even when 40% RAM is free!) or Transparent Huge Pages (`THP`) are enabled!. • ✓ Redis Configuration (`redis.conf`) — `maxmemory-policy` (`LFU` vs `LRU`), `lazyfree` & `CONFIG REWRITE` — Navigating `redis.conf`: network `bind 127.0.0.1 -::1` & `protected-mode yes`, `tcp-keepalive 300`, `maxmemory` & the 8 `maxmemory-policy` modes (`noeviction`, `allkeys-lru`, `allkeys-lfu`, `volatile-lru`, `volatile-lfu`, `volatile-ttl`), `lazyfree-lazy-eviction yes` non-blocking background thread eviction, and live `CONFIG SET` + `CONFIG REWRITE`. • ✓ Build, verify, and ship Module Capstone: Zero-OOM Production `redis.conf` & Kernel Copy-on-Write Readiness Validator.

2.1 Installing Redis 7 in Production — `vm.overcommit_memory=1`, Disabling `THP` & `systemd` Supervision

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Vm.overcommit_memory=1 fork() rule & Disable Transparent Huge Pages (4KB vs 2MB) • Core Syntax, CLI & Production APIs: sysctl vm.overcommit_memory=1, echo never > /sys/kernel/mm/transparent_hugepage/enabled, red... • Deep-Dive Mechanics: Net.core.somaxconn >= 4096 backlog & 25-30% RAM headroom above maxmemory • Advanced Patterns: Supervised systemd readiness notify & Jemalloc allocator fragmentation ratio • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating Linux Kernel & Memory Headroom Rules for Redis 7 • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Installing Redis 7 in Production

Purpose: To understand and apply Installing Redis 7 in Production — `vm.overcommit_memory=1`, Disabling `THP` & `systemd` Supervision in real Redis systems — specifically: Install...

What Topic Is Next → Next Topic 2.2: Redis Configuration (`redis.conf`) — `maxmemory-policy` (`LFU` vs `LRU`), `lazyfree` & `CONFIG REWRITE`

2.2 Redis Configuration (`redis.conf`) — `maxmemory-policy` (`LFU` vs `LRU`), `lazyfree` & `CONFIG REWRITE`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Maxmemory != 0 mandatory 64-bit cap & Allkeys-lfu vs allkeys-lru vs noeviction • Core Syntax, CLI & Production APIs: CONFIG GET maxmemory*, CONFIG SET maxmemory-policy allkeys-lfu, CONFIG REWRITE, lazyfree-laz... • Deep-Dive Mechanics: Volatile-* evicts ONLY keys with TTL! & Lazyfree-lazy-eviction yes async thread • Advanced Patterns: CONFIG SET + CONFIG REWRITE persistence & Client-output-buffer-limit pubsub cap • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Selecting Optimal `maxmemory-policy` & `lazyfree` Settings in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Configuration (`redis.conf`)

Purpose: To understand and apply Redis Configuration (`redis.conf`) — `maxmemory-policy` (`LFU` vs `LRU`), `lazyfree` & `CONFIG REWRITE` in real Redis systems — specifically: Navi...

What Topic Is Next → Module 2 Capstone (Zero-OOM Production `redis.conf` & Kernel Copy-on-Write Readiness Validator) → Next Topic 3.1: Redis Data Types Overview — Core 5 + `HyperLogLog` (`12 KB`), Bitmaps, Geospatial & Streams

Module 2 Hands-On Capstone Project (100 Marks): Zero-OOM Production `redis.conf` & Kernel Copy-on-Write Readiness Validator — Author a production `redis.conf` configuration validator ensuring `bind`, `protected-mode`, `maxmemory` CoW headroom, `allkeys-lfu` cache eviction, and disabled опасные commands (`FLUSHALL`...

MODULE 3 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Core & Probabilistic Redis Data Types (`OBJECT ENCODING`) & Non-Blocking Key Hygiene (`SCAN` / `UNLINK`)

Map domain problems to Redis's 9+ specialized data structures & compact memory encodings (`embstr`, `listpack`, `intset`, `skiplist`, `HyperLogLog`), and manage key lifecycles safely using cursor `SCAN` and async `UNLINK`.

Objectives: ✓ Redis Data Types Overview — Core 5 + `HyperLogLog` (`12 KB`), Bitmaps, Geospatial & Streams — Choosing the right data structure: Strings (`SDS` binary-safe <=`512 MB`), Hashes (field-value objects), Lists (`quicklist` queues), Sets (unique membership/intersections), Sorted Sets (`skiplist` leaderboards), **Bitmaps** (`1 bit` per user daily active flag!), **HyperLogLog** (`PFADD`/`PFCOUNT` counting billions of unique items in **`12 KB`**!), **Geo** (`GEOADD`/`GEOSEARCH`), and **Streams** (`XADD`/`XREADGROUP`). • ✓ Redis Key Management — Colon Namespaces, Cursor `SCAN` (Never `KEYS *`!), `UNLINK` vs `DEL` & `TTL` — Designing clean `app:domain:id` key schemas, inspecting existence & expiry (`EXISTS`, `TTL` returning `-2` missing / `-1` no expiry / `>=0` seconds, `PTTL`, `EXPIRE`, `PERSIST`), why `KEYS *` causes production outages (`O(N)` full-keyspace stop-the-world scan), `SCAN cursor MATCH pattern COUNT 500`, and `UNLINK` background thread reclamation. • ✓ Build, verify, and ship Module Capstone: Zero-Block Production Key Namespace Scanner (`scan_iter` + Batched `UNLINK`).

3.1 Redis Data Types Overview — Core 5 + `HyperLogLog` (`12 KB`), Bitmaps, Geospatial & Streams

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 5 core + 4 advanced Redis structures & HyperLogLog 12KB 0.81%-err cardinality • Core Syntax, CLI & Production APIs: TYPE, OBJECT ENCODING, PFADD / PFCOUNT (HyperLogLog), SETBIT / BITCOUNT, GEOADD / GEOSEARCH,... • Deep-Dive Mechanics: Bitmaps 1-bit boolean user flags & GEOSEARCH ZSet geohash indexing • Advanced Patterns: Redis Streams consumer-group log & OBJECT ENCODING compact listpack/embstr • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Comparing Set RAM vs `12 KB` HyperLogLog & `embstr` 44-Byte Threshold • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Data Types Overview

Purpose: To understand and apply Redis Data Types Overview — Core 5 + `HyperLogLog` (`12 KB`), Bitmaps, Geospatial & Streams in real Redis systems — specifically: Choosing the rig...

What Topic Is Next → Next Topic 3.2: Redis Key Management — Colon Namespaces, Cursor `SCAN` (Never `KEYS *`!), `UNLINK` vs `DEL` & `TTL`

3.2 Redis Key Management — Colon Namespaces, Cursor `SCAN` (Never `KEYS *`!), `UNLINK` vs `DEL` & `TTL`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Colon namespace convention (svc:obj:id) & Never run KEYS * in production! • Core Syntax, CLI & Production APIs: SCAN 0 MATCH user:* COUNT 500 (r.scan_iter), UNLINK vs DEL, TTL / PTTL (-2 vs -1), EXISTS, R... • Deep-Dive Mechanics: SCAN cursor != 0 stateless iteration & UNLINK O(1) detach + async bio free • Advanced Patterns: TTL -2 (missing) vs -1 (no TTL leak) & SCAN may return duplicates on rehash • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Batched `SCAN` + `UNLINK` Key-Purge Algorithm in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Key Management

Purpose: To understand and apply Redis Key Management — Colon Namespaces, Cursor `SCAN` (Never `KEYS *`!), `UNLINK` vs `DEL` & `TTL` in real Redis systems — specifically: Designin...

What Topic Is Next → Module 3 Capstone (Zero-Block Production Key Namespace Scanner (`scan_iter` + Batched `UNLINK`)) → Next Topic 4.1: Redis Strings (`SDS`) — `SET NX PX` Atomic Locks, `INCRBY` Counters, `MGET` & `GETDEL`

Module 3 Hands-On Capstone Project (100 Marks): Zero-Block Production Key Namespace Scanner (`scan_iter` + Batched `UNLINK`) — Build an administrative cache-invalidation tool in Python that purges `tenant:42:*` keys across a 20-million-key Redis instance without ever running `KEYS` or blocking the main event loop.

MODULE 4 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Binary-Safe Strings (`SDS`, Atomic `SET NX PX`, `INCR`) & Compact `listpack` Hashes

Use Simple Dynamic Strings (`SDS`) for atomic counters, idempotency keys, and `SET ... NX PX` distributed locks, and cut per-key metadata overhead up to 5x by modeling entities with Redis Hashes.

Objectives: ✓ Redis Strings (`SDS`) — `SET NX PX` Atomic Locks, `INCRBY` Counters, `MGET` & `GETDEL` — Why Redis uses C **Simple Dynamic Strings (`SDS`)** instead of null-terminated `char*` (`O(1)` `STRLEN`, binary safety for MessagePack/Protobuf bytes up to `512 MB`, buffer-overflow immunity), atomic `SET key token NX PX 5000` locks/idempotency keys, `GETEX`/`GETDEL` one-time OTP verification, and `INCR`/`INCRBY`. • ✓ Redis Hashes — Compact `listpack` Object Modeling, `HSET` / `HMGET` / `HINCRBY` & Avoiding `HGETALL` Stalls — Storing structured entities (`user:1001 -> {name, plan, credits}`) inside a single Redis Hash instead of 8 separate String keys (slashing `dictEntry` pointer overhead by `5x–8x`!), atomic field mutations (`HINCRBY user:1001 credits -1` without JSON deserialize-modify-serialize races!), and `HMGET` vs `HSCAN`. • ✓ Build, verify, and ship Module Capstone: Idempotent Payment Webhook Guard (`SET NX EX`) & Compact Session Hash Store.

4.1 Redis Strings (`SDS`) — `SET NX PX` Atomic Locks, `INCRBY` Counters, `MGET` & `GETDEL` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: SDS O(1) length & binary-safe 512MB & Atomic SET key token NX PX ms • Core Syntax, CLI & Production APIs: SET k v NX PX 5000, GETDEL (one-time OTP!), INCR / INCRBY / DECR, MGET / MSET, STRLEN • Deep-Dive Mechanics: Never split SETNX + EXPIRE (deadlock!) & GETDEL atomic 1-time OTP read+delete • Advanced Patterns: 64-bit signed atomic INCR / INCRBY & MGET single-RTT multi-key batching • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atomic Idempotency Gate (`SET NX EX`) & One-Time OTP (`GETDEL`) Simulation • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Strings (`SDS`)

Purpose: To understand and apply Redis Strings (`SDS`) — `SET NX PX` Atomic Locks, `INCRBY` Counters, `MGET` & `GETDEL` in real Redis systems — specifically: Why Redis uses C **Si...

What Topic Is Next → Next Topic 4.2: Redis Hashes — Compact `listpack` Object Modeling, `HSET` / `HMGET` / `HINCRBY` & Avoiding `HGETALL` Stalls

4.2 Redis Hashes — Compact `listpack` Object Modeling, `HSET` / `HMGET` / `HINCRBY` & Avoiding `HGETALL` Stalls [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 5x-8x less metadata RAM than N Strings & Listpack contiguous encoding (<=128) • Core Syntax, CLI & Production APIs: HSET key f1 v1 f2 v2 (mapping={}), HGET, HMGET, HINCRBY, HDEL, HLEN, HSCAN (prefer over HGET... • Deep-Dive Mechanics: Atomic HINCRBY avoids JSON lost updates & HMGET selective field projection • Advanced Patterns: HSCAN vs O(N) HGETALL big-key hazard & HSET multi-field mapping (replaces HMSET) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atomic Partial Hash Updates (`HSET` + `HINCRBY` + `HMGET` Projection) • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Hashes

Purpose: To understand and apply Redis Hashes — Compact `listpack` Object Modeling, `HSET` / `HMGET` / `HINCRBY` & Avoiding `HGETALL` Stalls in real Redis systems — specifically:...

What Topic Is Next → Module 4 Capstone (Idempotent Payment Webhook Guard (`SET NX EX`) & Compact Session Hash Store) → Next Topic 5.1: Redis Lists (`quicklist`) — `LPUSH`/`RPOP`, Capped `LTRIM` Feeds & Crash-Safe `BLMOVE` Queues

Module 4 Hands-On Capstone Project (100 Marks): Idempotent Payment Webhook Guard (`SET NX EX`) & Compact Session Hash Store — Build an atomic payment-webhook deduplicator using single-command `SET ... NX EX` plus a memory-dense user cart/profile store using `listpack`-optimized Redis Hashes.

MODULE 5 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: `quicklist` Lists (Blocking Queues & Capped Feeds) & `intset`/Hash Sets (Algebra & Tags)

Build bounded timeline feeds and reliable `BLMOVE` worker queues with `quicklist` Lists, and perform fast tag intersections, deduplication, and `intset` optimizations with Redis Sets.

Objectives: ✓ Redis Lists (`quicklist`) — `LPUSH`/`RPOP`, Capped `LTRIM` Feeds & Crash-Safe `BLMOVE` Queues — How Redis implements Lists as a **`quicklist`** (a doubly-linked list of compact `listpack` chunks — giving `O(1)` head/tail push-pop WITHOUT per-element `prev/next` pointer bloat!), `BLPOP` blocking waits (`0` CPU busy-polling!), why plain `BRPOP` loses the message if the worker dies mid-task, and atomic `BLMOVE` reliable queues. • ✓ Redis Sets — `O(1)` `SISMEMBER`, `SMISMEMBER` Batch Checks, `intset` Encoding & Set Algebra (`SINTER`) — Unordered collections of unique strings: `O(1)` membership (`SISMEMBER`) and Redis 6.2+ batch `SMISMEMBER` (checking 50 liked post IDs in 1 command!), `SPOP` (random removal, e.g. raffle/prize pool) vs `SRANDMEMBER` (non-destructive sample), `SINTER` / `SUNION` / `SDIFF`, and compact sorted **`intset`** encoding for integer Sets (`<=512` ints). • ✓ Build, verify, and ship Module Capstone: Crash-Safe `BLMOVE` Reliable Job Queue & `SINTER` Course Recommendation Engine.

5.1 Redis Lists (`quicklist`) — `LPUSH`/`RPOP`, Capped `LTRIM` Feeds & Crash-Safe `BLMOVE` Queues [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Quicklist (linked list of listpacks) & O(1) head/tail vs O(N) middle LRANGE • Core Syntax, CLI & Production APIs: LPUSH / RPUSH, LPOP / RPOP, BLPOP, LMOVE / BLMOVE src dst RIGHT LEFT, LPUSH + LTRIM 0 99, LL... • Deep-Dive Mechanics: LPUSH + LTRIM bounded ring-buffer & BLPOP zero-polling blocking consumer • Advanced Patterns: BLMOVE crash-safe in-flight queue & Lists vs Redis Streams (multi-group) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atomic `LMOVE` Reliable Queue + `LTRIM` Bounded Feed in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Lists (`quicklist`)

Purpose: To understand and apply Redis Lists (`quicklist`) — `LPUSH`/`RPOP`, Capped `LTRIM` Feeds & Crash-Safe `BLMOVE` Queues in real Redis systems — specifically: How Redis impl...

What Topic Is Next → Next Topic 5.2: Redis Sets — `O(1)` `SISMEMBER`, `SMISMEMBER` Batch Checks, `intset` Encoding & Set Algebra (`SINTER`)

5.2 Redis Sets — `O(1)` `SISMEMBER`, `SMISMEMBER` Batch Checks, `intset` Encoding & Set Algebra (`SINTER`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: O(1) SISMEMBER & batch SMISMEMBER & Intset compact integer-only encoding • Core Syntax, CLI & Production APIs: SADD, SREM, SISMEMBER, SMISMEMBER, SCARD, SPOP vs SRANDMEMBER, SINTER / SINTERCARD, SSCAN • Deep-Dive Mechanics: SPOP (mutates) vs SRANDMEMBER (reads) & SINTER starts from smallest SCARD set • Advanced Patterns: SINTERCARD ... LIMIT bounded CPU cap & SSCAN instead of O(N) SMEMBERS • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Batch `SMISMEMBER` & Bounded `SINTERCARD` Set Algebra in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter,

Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Sets

Purpose: To understand and apply Redis Sets — `O(1)` `SISMEMBER`, `SMISMEMBER` Batch Checks, `intset` Encoding & Set Algebra (`SINTER`) in real Redis systems — specifically: Unord...

What Topic Is Next → Module 5 Capstone (Crash-Safe `BLMOVE` Reliable Job Queue & `SINTER` Course Recommendation Engine) → Next Topic 6.1: Redis Sorted Sets (`ZSET`) — Dual `Dict + SkipList` Internals, Leaderboards & Sliding-Window Rate Limiting

Module 5 Hands-On Capstone Project (100 Marks): Crash-Safe `BLMOVE` Reliable Job Queue & `SINTER` Course Recommendation Engine — Implement a reliable queue pattern where crashed workers never drop in-flight jobs (`LMOVE` into a `processing` list until acked via `LREM`), plus a real-time Set intersection skill-matching...

MODULE 6 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: `skiplist` Sorted Sets (Real-Time Leaderboards & Sliding Windows) & `HSET` + `HEXPIRE` Field TTLs

Power million-player gaming/exam leaderboards and sliding-window rate limiters with `Hash + SkipList` Sorted Sets, and master Hash TTL lifecycle management including Redis 7.4 native per-field `HEXPIRE`!

Objectives: ✓ Redis Sorted Sets (`ZSET`) — Dual `Dict + SkipList` Internals, Leaderboards & Sliding-Window Rate Limiting — Redis's crown-jewel structure: every large `ZSET` maintains BOTH a **Hash Table** (`member -> score` in `O(1)`) AND a probabilistic multi-level **SkipList** (sorted by `IEEE-754 double score`, breaking score ties lexicographically by member bytes in `O(log N)`!), powering live leaderboards, priority delay queues (`ZMPOP`), and sliding-window rate limiters. • ✓ Redis `HSET` & Hash Expiration — Key-Level `EXPIRE` vs Redis 7.4 Native Per-Field `HEXPIRE` & `HTTL` — Solving a decade-long Redis interview classic! Prior to Redis 7.4, `EXPIRE` worked ONLY on top-level keys (not individual Hash fields) — AND note that `HSET` preserves existing key TTL (unlike `SET` without `KEEPTTL`, which wipes TTL!). In **Redis 7.4+**, native Hash Field Expiration (`HEXPIRE`, `HPEXPIRE`, `HEXPIREAT`, `HTTL`, `HPERSIST`) expires individual fields inside a Hash independently!. • ✓ Build, verify, and ship Module Capstone: Million-User Exam Leaderboard (`ZSET`) & Per-Device Session Token Hash (`HSET` + `HEXPIRE`).

6.1 Redis Sorted Sets (`ZSET`) — Dual `Dict + SkipList` Internals, Leaderboards & Sliding-Window Rate Limiting

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Dual Hash Table O(1) + SkipList O(log N) & IEEE-754 double score (2^53 exact int cap) • Core Syntax, CLI & Production APIs: ZADD (NX|XX|GT|LT), ZINCRBY, ZSCORE, ZRANK / ZREVRANK, ZRANGE key max min BYSCORE REV LIMIT... • Deep-Dive Mechanics: Lexicographical tie-breaking (BYLEX) & Unified ZRANGE REV BYSCORE syntax • Advanced Patterns: ZADD GT (update high-score only!) & Sliding-window ZREMRANGEBYSCORE + ZCARD • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: High-Score-Only `ZADD GT` & `ZREVRANK` Leaderboard Engine in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Sorted Sets (`ZSET`)

Purpose: To understand and apply Redis Sorted Sets (`ZSET`) — Dual `Dict + SkipList` Internals, Leaderboards & Sliding-Window Rate Limiting in real Redis systems — specifically: R...

What Topic Is Next → Next Topic 6.2: Redis `HSET` & Hash Expiration — Key-Level `EXPIRE` vs Redis 7.4 Native Per-Field `HEXPIRE` & `HTTL`

6.2 Redis `HSET` & Hash Expiration — Key-Level `EXPIRE` vs Redis 7.4 Native Per-Field `HEXPIRE` & `HTTL`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Redis 7.4+ native HEXPIRE & HTTL! & HEXPIRE NX / XX / GT / LT modifiers • Core Syntax, CLI & Production APIs: HSET key f v + EXPIRE key 3600 NX, Redis 7.4+ HEXPIRE key seconds [NX|XX|GT|LT] FIELDS numfi... • Deep-Dive Mechanics: SET wipes TTL (need KEEPTTL) vs HSET keeps TTL & Pipeline HSET + EXPIRE key ttl NX • Advanced Patterns: HTTL codes: -2 (gone), -1 (no field TTL) & Multi-device session Hash per-token TTL • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Redis 7.4 Per-Field `HEXPIRE ... NX FIELDS` & `HTTL` Semantics • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis `HSET`

Purpose: To understand and apply Redis `HSET` & Hash Expiration — Key-Level `EXPIRE` vs Redis 7.4 Native Per-Field `HEXPIRE` & `HTTL` in real Redis systems — specifically: Solving...

What Topic Is Next → Module 6 Capstone (Million-User Exam Leaderboard (`ZSET`) & Per-Device Session Token Hash (`HSET` + `HEXPIRE`)) → Next Topic 7.1: Redis Transactions — `MULTI` / `EXEC` / `DISCARD`, Why There Is NO Rollback & `WATCH` Optimistic CAS

Module 6 Hands-On Capstone Project (100 Marks): Million-User Exam Leaderboard (`ZSET`) & Per-Device Session Token Hash (`HSET` + `HEXPIRE`) — Build an `O(log N)` MockAttempt live exam leaderboard supporting instant rank + top-10 queries with timestamp tie-breaking, alongside a multi-device user session Hash expiring individual ref...

MODULE 7 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Optimistic `WATCH` / `MULTI` / `EXEC` Transactions & Atomic Server-Side `Lua` Scripting

Understand why Redis `MULTI/EXEC` does NOT support SQL-style rollbacks, implement CAS optimistic locking with `WATCH`, and write race-free conditional logic using server-side `EVALSHA` Lua scripts & Redis 7 Functions.

Objectives: ✓ Redis Transactions — `MULTI` / `EXEC` / `DISCARD`, Why There Is NO Rollback & `WATCH` Optimistic CAS — How `MULTI` queues commands (`QUEUED`) and `EXEC` runs the entire batch sequentially without interleaving any other client's commands; the #1 Redis interview gotcha (**Redis does NOT roll back already-executed commands if a later command inside `EXEC` fails at runtime, e.g., `INCR` on a List!**); and Check-And-Set (`CAS`) via `WATCH`. • ✓ Redis Lua Scripting & Functions — Atomic `EVALSHA`, Safe Lock Release & `KEYS[]` vs `ARGV[]` Rule — Why embedded Lua 5.1 (`EVAL`, `EVALSHA`, and Redis 7 persistent `FUNCTION LOAD` / `FCALL`) beats `WATCH/MULTI/EXEC`: zero network round-trips mid-transaction, zero `WatchError` retry storms, true conditional branching (`if/else` on `redis.call` results!), powering Redlock/lock-release and Token-Bucket rate limiters. • ✓ Build, verify, and ship Module Capstone: Atomic Token-Bucket API Rate Limiter & Lock Safe-Release in Redis `Lua`.

7.1 Redis Transactions — `MULTI` / `EXEC` / `DISCARD`, Why There Is NO Rollback & `WATCH` Optimistic CAS

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: MULTI queues -> EXEC runs atomically & NO rollback on runtime WRONGTYPE error! • Core Syntax, CLI & Production APIs: WATCH key, MULTI, EXEC, DISCARD, UNWATCH, pipe.watch() + WatchError in redis-py • Deep-Dive Mechanics: Syntax error during queue aborts EXEC & WATCH Check-And-Set optimistic lock • Advanced Patterns: WatchError retry loop in redis-py & Lua beats WATCH on high-contention keys • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Optimistic `WATCH` Check-And-Set Wallet Debit Simulation in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Transactions

Purpose: To understand and apply Redis Transactions — `MULTI` / `EXEC` / `DISCARD`, Why There Is NO Rollback & `WATCH` Optimistic CAS in real Redis systems — specifically: How `MU...

What Topic Is Next → Next Topic 7.2: Redis Lua Scripting & Functions — Atomic `EVALSHA`, Safe Lock Release & `KEYS[]` vs `ARGV[]` Rule

7.2 Redis Lua Scripting & Functions — Atomic `EVALSHA`, Safe Lock Release & `KEYS[]` vs `ARGV[]` Rule

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Atomic 1-RTT conditional read-modify-write & Always pass keys in KEYS[], args in ARGV[] • Core Syntax, CLI & Production APIs: EVAL / EVALSHA sha1 numkeys key [key ...] arg [arg ...], r.register_script(lua), FUNCTION LO... • Deep-Dive Mechanics: EVALSHA 40-byte SHA-1 bandwidth save & Owner-checked lock release Lua idiom • Advanced Patterns: Token-Bucket rate limiter in 10 lines & Redis 7 FUNCTION LOAD persisted libraries • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Canonical Owner-Checked Distributed Lock Release (`Lua` + Python Harness) • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Lua Scripting

Purpose: To understand and apply Redis Lua Scripting & Functions — Atomic `EVALSHA`, Safe Lock Release & `KEYS[]` vs `ARGV[]` Rule in real Redis systems — specifically: Why embedd...

What Topic Is Next → Module 7 Capstone (Atomic Token-Bucket API Rate Limiter & Lock Safe-Release in Redis `Lua`) → Next Topic 8.1: Redis Pipelining — Defeating Network `RTT` Latency, `transaction=False` & Bounded Batch Sizes

Module 7 Hands-On Capstone Project (100 Marks): Atomic Token-Bucket API Rate Limiter & Lock Safe-Release in Redis `Lua` — Build two mission-critical Lua scripts registered via `redis-py`: (1) an owner-verified distributed lock release (`if GET == token then DEL`) and (2) a race-free Token-Bucket API rate limite...

MODULE 8 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: High-Throughput Network `Pipelining` (`transaction=False`) & Thread-Safe `ConnectionPool`s

Boost batch throughput 10x–30x by eliminating per-command TCP Round-Trip Time (`RTT`) via `pipeline(transaction=False)`, and manage thread-safe/async `redis-py` `ConnectionPool`s cleanly.

Objectives: ✓ Redis Pipelining — Defeating Network `RTT` Latency, `transaction=False` & Bounded Batch Sizes — Why Redis throughput in microservices is almost always bottlenecked by **network Round-Trip Time (`RTT` ~`0.5 ms`)** + `read()/write()` syscall context switches rather than Redis CPU (`~1 us`!), how Pipelining streams `K` commands in `1` TCP write and reads `K` replies in `1` read (`20x` speedup!), and why Pipelining != Atomic Transaction. • ✓ Redis Connections & Pooling — `ConnectionPool`, Timeouts, `CLIENT LIST` & Async `redis.asyncio` — Why opening a brand-new TCP + TLS + `AUTH` handshake per HTTP request kills latency and exhausts ephemeral ports (`maxclients` default `10,000`), sharing a thread-safe `redis.ConnectionPool` (or `redis.asyncio` in FastAPI/uvloop), `CLIENT SETNAME` observability in `CLIENT LIST`, RESP3 client-side caching, and socket timeout discipline. • ✓ Build, verify, and ship Module Capstone: Chunked `1,000`-Op Non-Transactional Pipeline Loader & Resilient `ConnectionPool` Client.

8.1 Redis Pipelining — Defeating Network `RTT` Latency, `transaction=False` & Bounded Batch Sizes

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: K * RTT sequential vs 1 * RTT pipelined & Fewer kernel read()/write() syscalls • Core Syntax, CLI & Production APIs: with r.pipeline(transaction=False) as pipe: pipe.get(...); pipe.set(...); results = pipe.exe... • Deep-Dive Mechanics: Redis-py pipeline(transaction=False) & Pipelining is NOT atomic (interleaves) • Advanced Patterns: Chunk size 500-1000 memory cap & Redis Cluster slot-grouped pipelines • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Benchmarking Sequential `K * RTT` vs Chunked `pipeline(transaction=False)` Math • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Pipelining

Purpose: To understand and apply Redis Pipelining — Defeating Network `RTT` Latency, `transaction=False` & Bounded Batch Sizes in real Redis systems — specifically: Why Redis thro...

What Topic Is Next → Next Topic 8.2: Redis Connections & Pooling — `ConnectionPool`, Timeouts, `CLIENT LIST` & Async `redis.asyncio`

8.2 Redis Connections & Pooling — `ConnectionPool`, Timeouts, `CLIENT LIST` & Async `redis.asyncio`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Singleton thread-safe ConnectionPool & Mandatory socket_timeout != None! • Core Syntax, CLI & Production APIs: redis.ConnectionPool(max_connections=50, socket_connect_timeout=1, socket_timeout=2, health_... • Deep-Dive Mechanics: Health_check_interval=30 stale-TCP guard & Post-fork socket FD safety in Gunicorn • Advanced Patterns: Client_name tag for CLIENT LIST & Redis.asyncio for FastAPI event loops • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Production-Hardened `redis-py` `ConnectionPool` Configuration Spec • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Connections

Purpose: To understand and apply Redis Connections & Pooling — `ConnectionPool`, Timeouts, `CLIENT LIST` & Async `redis.asyncio` in real Redis systems — specifically: Why opening...

What Topic Is Next → Module 8 Capstone (Chunked `1,000`-Op Non-Transactional Pipeline Loader & Resilient `ConnectionPool` Client) → Next Topic 9.1: Redis Backup, Restore & Persistence — `BGSAVE` (`RDB`) vs `AOF` (`everysec`) & Hybrid RDB-Preamble

Module 8 Hands-On Capstone Project (100 Marks): Chunked `1,000`-Op Non-Transactional Pipeline Loader & Resilient `ConnectionPool` Client — Build a Python `redis-py` bulk feature-flag hydrator that chunks 25,000 writes into bounded 1,000-command `pipeline(transaction=False)` flushes over a timeout-hardened `ConnectionPool`.

MODULE 9 OF 16 • STAGE 3 • 10 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Durability (`RDB` + `AOF` Hybrid Persistence) & Zero-Trust Redis 7 `ACL` / TLS Security

Balance speed and crash recovery using `BGSAVE` RDB snapshots + `appendfsync everysec` Multi-Part AOF (`aof-use-rdb-preamble yes`), and lock down Redis 7 with granular `ACL` users, TLS 1.3, and network isolation.

Objectives: ✓ Redis Backup, Restore & Persistence — `BGSAVE` (`RDB`) vs `AOF` (`everysec`) & Hybrid RDB-Preamble — Comparing **RDB** (compact point-in-time binary `dump.rdb` created via child `fork()` `BGSAVE` — fast restores & S3 off-site backups, but loses minutes since last snapshot) vs **AOF** (`appendfsync everysec` losing at most `1 second` of writes + `BGREWRITEAOF` compaction), and Redis 7 Multi-Part AOF (`manifest` + `base.rdb` + `incr.aof`). • ✓ Redis 7 Security Hardening — Granular `ACL`s (Key & Command Prefixes), `default off`, TLS 1.3 & VPC Isolation — Why exposed unauthenticated `0.0.0.0:6379` Redis instances get cryptomined within minutes (`CONFIG SET dir /root/.ssh` exploit!): network `bind` + `protected-mode yes` + private VPC Security Groups, disabling `user default off`, creating per-service **Redis 6/7 ACLs** (`~prefix:*`, `&channel:*`, `+@read +@write -@dangerous -@admin`), and mTLS encryption in transit. • ✓ Build, verify, and ship Module Capstone: Hybrid `RDB + AOF` Disaster-Recovery Procedure & Least-Privilege `ACL` Policy Generator.

9.1 Redis Backup, Restore & Persistence — `BGSAVE` (`RDB`) vs `AOF` (`everysec`) & Hybrid RDB-Preamble [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: BGSAVE fork() vs blocking SAVE ban & RDB (fast boot/backup) vs AOF (1s RPO) • Core Syntax, CLI & Production APIs: BGSAVE, LASTSAVE, appendonly yes, appendfsync everysec, aof-use-rdb-preamble yes, BGREWRITEA... • Deep-Dive Mechanics: Appendfsync everysec sweet spot & Aof-use-rdb-preamble yes hybrid speed • Advanced Patterns: Redis loads AOF over dump.rdb on boot! & Redis 7 appendonlydir multi-part AOF • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Persistence Boot-Source & RPO Durability Analyzer in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Backup

Purpose: To understand and apply Redis Backup, Restore & Persistence — `BGSAVE` (`RDB`) vs `AOF` (`everysec`) & Hybrid RDB-Preamble in real Redis systems — specifically: Comparing...

What Topic Is Next → Next Topic 9.2: Redis 7 Security Hardening — Granular `ACL`s (Key & Command Prefixes), `default off`, TLS 1.3 & VPC Isolation

9.2 Redis 7 Security Hardening — Granular `ACL`s (Key & Command Prefixes), `default off`, TLS 1.3 & VPC Isolation [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Never expose 6379 to public internet & ACL SETUSER default off hardening • Core Syntax, CLI & Production APIs: ACL SETUSER <name> on >pass ~app:* +@read +@write -@dangerous -@admin, ACL LIST, ACL WHOAMI,... • Deep-Dive Mechanics: ~prefix:* key-pattern ACL isolation & -@dangerous -@admin blocks FLUSHALL/KEYS • Advanced Patterns: ACL LOG unauthorized audit trail & Tls-port 6379 + port 0 encryption • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Generating Least-Privilege Redis 7 `ACL SETUSER` Rules in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis 7 Security Hardening

Purpose: To understand and apply Redis 7 Security Hardening — Granular `ACL`s (Key & Command Prefixes), `default off`, TLS 1.3 & VPC Isolation in real Redis systems — specifically...

What Topic Is Next → Module 9 Capstone (Hybrid `RDB + AOF` Disaster-Recovery Procedure & Least-Privilege `ACL` Policy Generator) → Next Topic 10.1: Redis Benchmarking & Latency Forensics — `redis-benchmark` (`-r`, `-d`, `-P`), `--intrinsic-latency` & Tail p99

Module 9 Hands-On Capstone Project (100 Marks): Hybrid `RDB + AOF` Disaster-Recovery Procedure & Least-Privilege `ACL` Policy Generator — Design a safe Redis persistence & restore runbook avoiding the empty-`appendonly.aof` data-wipe trap, paired with a microservice-scoped Redis 7 `ACL` rule generator.

MODULE 10 OF 16 • STAGE 3 • 10 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Realistic Load Benchmarking (`redis-benchmark` `-r` / `-P`) & `16,384`-Slot Redis Cluster Partitioning

Benchmark throughput, p99 tail latency, and keyspace randomization realistically with `redis-benchmark` / `memtier_benchmark`, and scale horizontally past single-node RAM using Redis Sentinel HA vs `16,384`-Slot Redis Cluster & `{hash_tag}` co-location.

Objectives: ✓ Redis Benchmarking & Latency Forensics — `redis-benchmark` (`-r`, `-d`, `-P`), `--intrinsic-latency` & Tail p99 — Running honest load tests with `redis-benchmark`: why omitting `-r <keyspacelen>` hammers the **same 1 single key** in L1 CPU cache, testing realistic payload bytes (`-d 512`), measuring pipeline scaling (`-P 16`), multi-threaded client generation (`--threads 4` from a **separate** VM so benchmark client CPU doesn't starve `redis-server`!), and `redis-cli --latency-history`. • ✓ Redis Partitioning & Cluster HA — Sentinel vs `16,384` Hash Slots (`CRC16`), `{HashTags}` & `CROSSSLOT` — Scaling Redis horizontally: why naive `hash(key) % N` loses `(N-1)/N` of keys when adding a node vs Consistent Hashing vs **Redis Cluster's `16,384` deterministic Hash Slots** (`slot = CRC16(key) & 16383`), minimum **6-node production topology** (`3 Primaries + 3 Replicas`), smart client `MOVED`/`ASK` slot maps (`RedisCluster`), and `{user:42}` **Hash Tags** for multi-key operations. • ✓ Build, verify, and ship Module Capstone: 6-Node (`3 Primary + 3 Replica`) `16,384`-Slot Redis Cluster Router & `{HashTag}` Validator.

10.1 Redis Benchmarking & Latency Forensics — `redis-benchmark` (`-r`, `-d`, `-P`), `--intrinsic-latency` & Tail p99 [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: -r keyspacelen realistic random keys & -d bytes payload size realism • Core Syntax, CLI & Production APIs: redis-benchmark -h <host> -c 50 -n 200000 -r 100000 -d 512 -P 16 --threads 4, redis-cli --la... • Deep-Dive Mechanics: -P 16 pipeline throughput multiplier & Run benchmark from separate client VM • Advanced Patterns: P99 / p99.9 tail-latency percentiles & LATENCY DOCTOR & SLOWLOG GET diagnosis • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating Realistic Production `redis-benchmark` CLI Invocations • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Benchmarking

Purpose: To understand and apply Redis Benchmarking & Latency Forensics — `redis-benchmark` (`-r`, `-d`, `-P`), `--intrinsic-latency` & Tail p99 in real Redis systems — specifical...

What Topic Is Next → Next Topic 10.2: Redis Partitioning & Cluster HA — Sentinel vs `16,384` Hash Slots (`CRC16`), `{HashTags}` & `CROSSSLOT`

10.2 Redis Partitioning & Cluster HA — Sentinel vs `16,384` Hash Slots (`CRC16`), `{HashTags}` & `CROSSSLOT` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Sentinel (HA <=64GB) vs Cluster (sharded) & CRC16(key) % 16384 hash-slot routing • Core Syntax, CLI & Production APIs: redis.cluster.RedisCluster, CLUSTER INFO / NODES / KEYSLOT, CRC16(key) % 16384, {hash_tag} c... • Deep-Dive Mechanics: 3 Primaries + 3 Replicas min topology & CROSSSLOT multi-key error & {hash_tag} • Advanced Patterns: Empty {} hashes full key (spec rule!) & Never use global {prod} hot-shard tag! • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Exact Redis Cluster `CRC16 % 16384` & `{HashTag}` Slot Calculator in Python • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Partitioning

Purpose: To understand and apply Redis Partitioning & Cluster HA — Sentinel vs `16,384` Hash Slots (`CRC16`), `{HashTags}` & `CROSSSLOT` in real Redis systems — specifically: Scal...

What Topic Is Next → Module 10 Capstone (6-Node (`3 Primary + 3 Replica`) `16,384`-Slot Redis Cluster Router & `{HashTag}` Validator) → Next Topic 11.1: Cache-Aside, Read/Write-Through, Negative Caching & TTL Jitter

Module 10 Hands-On Capstone Project (100 Marks): 6-Node (`3 Primary + 3 Replica`) `16,384`-Slot Redis Cluster Router & `{HashTag}` Validator — Implement the exact XMODEM `CRC16(key) % 16384` Redis Cluster hash-slot router in Python supporting `{...}` hash tags, verifying multi-key Lua/`MGET` slot co-location without creating single...

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

High-Scale Caching Architecture: Stampede Locks, Pipelining & Memory Tuning

Production-grade Redis Training engineering module covering High-Scale Caching Architecture: Stampede Locks, Pipelining & Memory Tuning with hands-on labs and capstone build.

Objectives: ✓ Cache-Aside, Read/Write-Through, Negative Caching & TTL Jitter — Purpose: Protect SQL/NoSQL databases from both hot-key floods and repeated missing-ID lookups. • ✓ Preventing Cache Stampedes (Thundering Herd) with Single-Flight Locks & XFetch — Purpose: Recompute expiring popular keys on 1 worker while 5,000 readers still get instant responses. • ✓ Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS — Purpose: Slash network round-trip latency 10x and use cursor SCAN so Redis's single thread never stalls. • ✓ Compact Encodings (ziplist/listpack), maxmemory-policy Eviction & MEMORY USAGE — Purpose: Cut Redis RAM bills by 50% and choose allkeys-lfu vs volatile-lru intentionally.

11.1 Cache-Aside, Read/Write-Through, Negative Caching & TTL Jitter [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Read/Write • Integration with Preventing Cache Stampedes (Thundering Herd) with Single • Live Debugging, Telemetry & Technical Interview Trade • What Is Cache • In Redis Training, Cache • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Cache-Aside, Read/Write-Through, Negative Caching & TTL Jitter in real Redis systems — specifically: Protect SQL/NoSQL databases from both hot-key...

What Topic Is Next → Next Topic 11.2: Preventing Cache Stampedes (Thundering Herd) with Single-Flight Locks & XFetch

11.2 Preventing Cache Stampedes (Thundering Herd) with Single-Flight Locks & XFetch [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of XFetch • Integration with Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS • Live Debugging, Telemetry & Technical Interview Trade • What Is Preventing Cache Stampedes (Thundering Herd) with Single • In Redis Training, Preventing Cache Stampedes (Thundering Herd) with Single • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Preventing Cache Stampedes (Thundering Herd) with Single-Flight Locks & XFetch in real Redis systems — specifically: Recompute expiring popular ke...

What Topic Is Next → Next Topic 11.3: Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS

11.3 Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Connection Pooling • Integration with Compact Encodings (ziplist/listpack), maxmemory • Live Debugging, Telemetry & Technical Interview Trade • What Is Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS & Why Industry Teams Use It • In Redis Training, Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS exists for one clear real • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Command Pipelining, Connection Pooling & Avoiding O(N) KEYS/SMEMBERS in real Redis systems — specifically: Slash network round-trip latency 10x an...

What Topic Is Next → Next Topic 11.4: Compact Encodings (ziplist/listpack), maxmemory-policy Eviction & MEMORY USAGE

11.4 Compact Encodings (ziplist/listpack), maxmemory-policy Eviction & MEMORY USAGE [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of maxmemory • Integration with Cache • Live Debugging, Telemetry & Technical Interview Trade • What Is Compact Encodings (ziplist/listpack), maxmemory • In Redis Training, Compact Encodings (ziplist/listpack), maxmemory • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Compact Encodings (ziplist/listpack), maxmemory-policy Eviction & MEMORY USAGE in real Redis systems — specifically: Cut Redis RAM bills by 50% an...

What Topic Is Next → Module 11 Capstone (Sub-2ms Multi-Tier Cache Layer Sustaining 50k RPS) → Next Topic 12.1: Server-Side Atomic EVALSHA Lua Scripts vs Optimistic WATCH/MULTI

Module 11 Hands-On Capstone Project (100 Marks): Sub-2ms Multi-Tier Cache Layer Sustaining 50k RPS — Design, implement, test, and document 'Sub-2ms Multi-Tier Cache Layer Sustaining 50k RPS' applying every production concept from Module 11 (High-Scale Caching Architecture: Stampede Locks, P...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Atomic Lua Scripting, Sliding-Window Rate Limiters & Distributed Locks

Production-grade Redis Training engineering module covering Atomic Lua Scripting, Sliding-Window Rate Limiters & Distributed Locks with hands-on labs and capstone build.

Objectives: ✓ Server-Side Atomic EVALSHA Lua Scripts vs Optimistic WATCH/MULTI — Purpose: Execute multi-step check-and-decrement logic atomically in microseconds with zero race windows. • ✓ Token-Bucket & ZSET Sliding-Window Rate Limiters — Purpose: Enforce exact per-user, per-IP, and per-API-key request quotas across 20 API servers. • ✓ Safe Distributed Mutexes (SET key token NX PX + Lua Safe-Unlock) — Purpose: Guarantee an expired worker never accidentally deletes a lock acquired by a newer worker!. • ✓ Probabilistic Structures: HyperLogLog Unique Counts & Geo/Leaderboard ZSETs — Purpose: Count 100M unique daily visitors in just 12 KB of RAM and rank live leaderboards in O(log N).

12.1 Server-Side Atomic EVALSHA Lua Scripts vs Optimistic WATCH/MULTI [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Token • Live Debugging, Telemetry & Technical Interview Trade • What Is Server • In Redis Training, Server • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Server-Side Atomic EVALSHA Lua Scripts vs Optimistic WATCH/MULTI in real Redis systems — specifically: Execute multi-step check-and-decrement logi...

What Topic Is Next → Next Topic 12.2: Token-Bucket & ZSET Sliding-Window Rate Limiters

12.2 Token-Bucket & ZSET Sliding-Window Rate Limiters [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of ZSET Sliding • Integration with Safe Distributed Mutexes (SET key token NX PX + Lua Safe • Live Debugging, Telemetry & Technical Interview Trade • What Is Token • In Redis Training, Token • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Token-Bucket & ZSET Sliding-Window Rate Limiters in real Redis systems — specifically: Enforce exact per-user, per-IP, and per-API-key request quo...

What Topic Is Next → Next Topic 12.3: Safe Distributed Mutexes (SET key token NX PX + Lua Safe-Unlock)

12.3 Safe Distributed Mutexes (SET key token NX PX + Lua Safe-Unlock) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Probabilistic Structures • Live Debugging, Telemetry & Technical Interview Trade • What Is Safe Distributed Mutexes (SET key token NX PX + Lua Safe • In Redis Training, Safe Distributed Mutexes (SET key token NX PX + Lua Safe • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Safe Distributed Mutexes (SET key token NX PX + Lua Safe-Unlock) in real Redis systems — specifically: Guarantee an expired worker never accidenta...

What Topic Is Next → Next Topic 12.4: Probabilistic Structures: HyperLogLog Unique Counts & Geo/Leaderboard ZSETs

12.4 Probabilistic Structures: HyperLogLog Unique Counts & Geo/Leaderboard ZSETs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Geo/Leaderboard ZSETs • Integration with Server • Live Debugging, Telemetry & Technical Interview Trade • What Is Probabilistic Structures • In Redis Training, Probabilistic Structures • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Probabilistic Structures: HyperLogLog Unique Counts & Geo/Leaderboard ZSETs in real Redis systems — specifically: Count 100M unique daily visitors...

What Topic Is Next → Module 12 Capstone (Race-Free Flash-Sale Inventory & API Throttling Engine in Redis) → Next Topic 13.1: Pub/Sub Fire-and-Forget Fanout vs Persistent Redis Streams (XADD/XREADGROUP)

Module 12 Hands-On Capstone Project (100 Marks): Race-Free Flash-Sale Inventory & API Throttling Engine in Redis — Design, implement, test, and document 'Race-Free Flash-Sale Inventory & API Throttling Engine in Redis' applying every production concept from Module 12 (Atomic Lua Scripting, Sliding-Window...

MODULE 13 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Redis Streams Consumer Groups, Reliable Queues & Pub/Sub Fanout

Production-grade Redis Training engineering module covering Redis Streams Consumer Groups, Reliable Queues & Pub/Sub Fanout with hands-on labs and capstone build.

Objectives: ✓ Pub/Sub Fire-and-Forget Fanout vs Persistent Redis Streams (XADD/XREADGROUP) — Purpose: Know when ephemeral WebSocket broadcast fits vs durable consumer-group event logs. • ✓ Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM — Purpose: Reassign unacknowledged messages automatically when a worker container crashes mid-job. • ✓ Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead-Letter Streams — Purpose: Bound stream memory predictably and quarantine poison messages cleanly. • ✓ Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion — Purpose: Build rock-solid scheduled reminders and retry queues on pure Redis primitives.

13.1 Pub/Sub Fire-and-Forget Fanout vs Persistent Redis Streams (XADD/XREADGROUP) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of • Integration with Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM • Live Debugging, Telemetry & Technical Interview Trade • What Is Pub/Sub Fire • In Redis Training, Pub/Sub Fire • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Pub/Sub Fire-and-Forget Fanout vs Persistent Redis Streams (XADD/XREADGROUP) in real Redis systems — specifically: Know when ephemeral WebSocket b...

What Topic Is Next → Next Topic 13.2: Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM

13.2 Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Pending Entries (XPENDING) • Integration with Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead • Live Debugging, Telemetry & Technical Interview Trade • What Is Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM & Why Industry Teams Use It • In Redis Training, Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM exists for one cl... • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Consumer Group Acknowledgments (XACK), Pending Entries (XPENDING) & XAUTOCLAIM in real Redis systems — specifically: Reassign unacknowledged messa...

What Topic Is Next → Next Topic 13.3: Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead-Letter Streams

13.3 Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead-Letter Streams [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Idempotent Deduplication • Integration with Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion • Live Debugging, Telemetry & Technical Interview Trade • What Is Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead • In Redis Training, Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Stream Trimming (MAXLEN ~), Idempotent Deduplication & Dead-Letter Streams in real Redis systems — specifically: Bound stream memory predictably a...

What Topic Is Next → Next Topic 13.4: Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion

13.4 Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Pub/Sub Fire • Live Debugging, Telemetry & Technical Interview Trade • What Is Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion & Why Industry Teams Use It • In Redis Training, Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion exists for one clear real • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Delayed Job Schedulers Using ZSET Timestamps + Atomic Lua Promotion in real Redis systems — specifically: Build rock-solid scheduled reminders and...

What Topic Is Next → Module 13 Capstone (Durable Real-Time Event Stream & Notification Backplane) → Next Topic 14.1: RDB Snapshots vs Append-Only File (appendfsync everysec) Hybrid Persistence

Module 13 Hands-On Capstone Project (100 Marks): Durable Real-Time Event Stream & Notification Backplane — Design, implement, test, and document 'Durable Real-Time Event Stream & Notification Backplane' applying every production concept from Module 13 (Redis Streams Consumer Groups, Reliable Queu...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Persistence (RDB + AOF), Sentinel Failover, Redis Cluster Sharding & Security

Production-grade Redis Training engineering module covering Persistence (RDB + AOF), Sentinel Failover, Redis Cluster Sharding & Security with hands-on labs and capstone build.

Objectives: ✓ RDB Snapshots vs Append-Only File (appendfsync everysec) Hybrid Persistence — Purpose: Balance sub-millisecond write speed with at-most-1-second crash durability. • ✓ Redis Sentinel Quorum Monitoring & Client-Aware Automatic Failover — Purpose: Promote replica nodes automatically when a primary Redis host goes unreachable. • ✓ Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding — Purpose: Scale write throughput horizontally and co-locate multi-key Lua operations via {tenantId} tags. • ✓ Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor — Purpose: Lock down dangerous FLUSHALL/CONFIG commands and diagnose micro-stalls in production.

14.1 RDB Snapshots vs Append-Only File (appendfsync everysec) Hybrid Persistence [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Redis Sentinel Quorum Monitoring & Client • Live Debugging, Telemetry & Technical Interview Trade • What Is RDB Snapshots vs Append • In Redis Training, RDB Snapshots vs Append • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply RDB Snapshots vs Append-Only File (appendfsync everysec) Hybrid Persistence in real Redis systems — specifically: Balance sub-millisecond write sp...

What Topic Is Next → Next Topic 14.2: Redis Sentinel Quorum Monitoring & Client-Aware Automatic Failover

14.2 Redis Sentinel Quorum Monitoring & Client-Aware Automatic Failover [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Client • Integration with Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding • Live Debugging, Telemetry & Technical Interview Trade • What Is Redis Sentinel Quorum Monitoring & Client • In Redis Training, Redis Sentinel Quorum Monitoring & Client • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Redis Sentinel Quorum Monitoring & Client-Aware Automatic Failover in real Redis systems — specifically: Promote replica nodes automatically when...

What Topic Is Next → Next Topic 14.3: Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding

14.3 Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of 384 Hash Slots • Integration with Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor • Live Debugging, Telemetry & Technical Interview Trade • What Is Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding & Why Industry Teams Use It • In Redis Training, Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding exists for one clear real • Even if you come from a non • Runnable Code

Implementation (LUA)

Purpose: To understand and apply Redis Cluster 16,384 Hash Slots, {HashTags} & Live Slot Resharding in real Redis systems — specifically: Scale write throughput horizontally and c...

What Topic Is Next → Next Topic 14.4: Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor

14.4 Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Protected Mode • Integration with RDB Snapshots vs Append • Live Debugging, Telemetry & Technical Interview Trade • What Is Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor & Why Industry Teams Use It • In Redis Training, Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor exists for one clear.... • Even if you come from a non • Runnable Code Implementation (LUA)

Purpose: To understand and apply Redis 7 ACL Users, Protected Mode, TLS Encryption & SLOWLOG/Latency Doctor in real Redis systems — specifically: Lock down dangerous FLUSHALL/CONF...

What Topic Is Next → Module 14 Capstone (High-Availability Sharded Redis Cluster with TLS & ACLs) → Next Topic 15.1: Solving Cache Stampede, Cache Penetration (Bloom/Null TTL) & Cache Avalanche (TTL Jitter)

Module 14 Hands-On Capstone Project (100 Marks): High-Availability Sharded Redis Cluster with TLS & ACLs — Design, implement, test, and document 'High-Availability Sharded Redis Cluster with TLS & ACLs' applying every production concept from Module 14 (Persistence (RDB + AOF), Sentinel Failover,...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Cache Stampede Protection, Atomic Lua Scripts & Sliding-Window Rate Limiters

Self-explanatory list-and-bullet production mastery module covering Cache Stampede Protection, Atomic Lua Scripts & Sliding-Window Rate Limiters in Redis Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Solving Cache Stampede, Cache Penetration (Bloom/Null TTL) & Cache Avalanche (TTL Jitter) — Purpose: Keep databases protected when hot cached keys expire under 50,000+ concurrent requests. • ✓ Atomic Server-Side Lua Scripting & MULTI/EXEC/WATCH Optimistic Pipelines — Purpose: Execute multi-step check-and-decrement logic atomically in sub-millisecond round trips. • ✓ Redis Streams Consumer Groups, Pending Entries (XPENDING/XAUTOCLAIM) & Dead-Letter Handling — Purpose: Build durable, replayable event pipelines with guaranteed consumer crash recovery. • ✓ Build, verify, and document 'Atomic Lua Token-Bucket Rate Limiter & Stampede-Safe Cache-Aside Library' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Solving Cache Stampede, Cache Penetration (Bloom/Null TTL) & Cache Avalanche (TTL Jitter) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Solving Cache Stampede, Cache Penetration (Bloom/Null TTL) & Cache Avalanc... • Performance Optimization, Complexity & Latency Impact of Solving Cache Stampede, Cache Penetration (Bloom/Null TTL)... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Solving Cache Stampede • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply Solving Cache Stampede, Cache Penetration (Bloom/Null TTL) & Cache Avalanche (TTL Jitter) in real Redis systems — specifically: Keep databases pro...

What Topic Is Next → Next Topic 15.2: Atomic Server-Side Lua Scripting & MULTI/EXEC/WATCH Optimistic Pipelines

15.2 Atomic Server-Side Lua Scripting & MULTI/EXEC/WATCH Optimistic Pipelines [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Atomic Server • Performance Optimization, Complexity & Latency Impact of Atomic Server • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Atomic Server-Side Lua Scripting • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply Atomic Server-Side Lua Scripting & MULTI/EXEC/WATCH Optimistic Pipelines in real Redis systems — specifically: Execute multi-step check-and-decrem...

What Topic Is Next → Next Topic 15.3: Redis Streams Consumer Groups, Pending Entries (XPENDING/XAUTOCLAIM) & Dead-Letter Handling

15.3 Redis Streams Consumer Groups, Pending Entries (XPENDING/XAUTOCLAIM) & Dead-Letter Handling [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Redis Streams Consumer Groups, Pending Entries (XPENDING/XAUTOCLAIM) & Dea... • Performance Optimization, Complexity & Latency Impact of Redis Streams Consumer Groups, Pending Entries (XPENDING/X... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Redis Streams Consumer Groups • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply Redis Streams Consumer Groups, Pending Entries (XPENDING/XAUTOCLAIM) & Dead-Letter Handling in real Redis systems — specifically: Build durable, r...

What Topic Is Next → Module 15 Capstone (Atomic Lua Token-Bucket Rate Limiter & Stampede-Safe Cache-Aside Library) → Next Topic 16.1: maxmemory Policies (allkeys-lru vs volatile-lfu), BigKey/HotKey Detection & Pipelining

Module 15 Hands-On Capstone Project (100 Marks): Atomic Lua Token-Bucket Rate Limiter & Stampede-Safe Cache-Aside Library — Architect, implement, test, and document 'Atomic Lua Token-Bucket Rate Limiter & Stampede-Safe Cache-Aside Library' applying every bullet-wise engineering concept from Module 15 (Cache Stamp...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Redis Memory Eviction Tuning, Sentinel/Cluster High Availability & System Design

Self-explanatory list-and-bullet production mastery module covering Redis Memory Eviction Tuning, Sentinel/Cluster High Availability & System Design in Redis Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ maxmemory Policies (allkeys-lru vs volatile-lfu), BigKey/HotKey Detection & Pipelining — Purpose: Prevent unexpected evictions and single-shard bottlenecks caused by oversized keys. • ✓ RDB vs AOF Persistence Trade-Offs, Redis Sentinel Failover & Cluster Hash Slots ({tag}) — Purpose: Balance durability vs latency and co-locate multi-key operations safely on Redis Cluster. • ✓ Distributed Locking Pitfalls & Redis System Design Interview Scenarios — Purpose: Design global leaderboards, session stores, and real-time counters in senior interviews. • ✓ Build, verify, and document 'Redis Cluster Hash-Tag Sharding Lab, BigKey Scanner & Failover Verification' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 maxmemory Policies (allkeys-lru vs volatile-lfu), BigKey/HotKey Detection & Pipelining [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for maxmemory Policies (allkeys • Performance Optimization, Complexity & Latency Impact of maxmemory Policies (allkeys • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for maxmemory Policies (allkeys-lru vs volatile-lfu) • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply maxmemory Policies (allkeys-lru vs volatile-lfu), BigKey/HotKey Detection & Pipelining in real Redis systems — specifically: Prevent unexpected ev...

What Topic Is Next → Next Topic 16.2: RDB vs AOF Persistence Trade-Offs, Redis Sentinel Failover & Cluster Hash Slots ({tag})

16.2 RDB vs AOF Persistence Trade-Offs, Redis Sentinel Failover & Cluster Hash Slots ({tag}) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for RDB vs AOF Persistence Trade • Performance Optimization, Complexity & Latency Impact of RDB vs AOF Persistence Trade • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for RDB vs AOF Persistence Trade-Offs • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply RDB vs AOF Persistence Trade-Offs, Redis Sentinel Failover & Cluster Hash Slots ({tag}) in real Redis systems — specifically: Balance durability v...

What Topic Is Next → Next Topic 16.3: Distributed Locking Pitfalls & Redis System Design Interview Scenarios

16.3 Distributed Locking Pitfalls & Redis System Design Interview Scenarios [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Distributed Locking Pitfalls & Redis System Design Interview Scenarios • Performance Optimization, Complexity & Latency Impact of Distributed Locking Pitfalls & Redis System Design Intervi... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic REDI Implementation: Cache-Aside with TTL jitter, Pipelining, Redis Streams & EVALSHA Lua • Redis 7 Single-Threaded epoll Event Loop, SDS/listpack/skiplist & RESP3 Execution & Memory Lifecycle for Distributed Locking Pitfalls • Production Hardening & Security: Redis 7 ACLs, TLS, protected-mode & UNLINK non-blocking eviction

Purpose: To understand and apply Distributed Locking Pitfalls & Redis System Design Interview Scenarios in real Redis systems — specifically: Design global leaderboards, session s...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: FlashCache Sub-Millisecond API Cache, Distributed Lock & Redis Streams Leaderboard

Module 16 Hands-On Capstone Project (100 Marks): Redis Cluster Hash-Tag Sharding Lab, BigKey Scanner & Failover Verification — Architect, implement, test, and document 'Redis Cluster Hash-Tag Sharding Lab, BigKey Scanner & Failover Verification' applying every bullet-wise engineering concept from Module 16 (Redis Me...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: FlashCache Sub-Millisecond API Cache, Distributed Lock & Redis Streams Leaderboard • **Confirmed Role:** Redis In-Memory Data Systems & Performance Intern

Domain & Production Infra: High-Concurrency Gaming, Flash-Sale & FinTech API Client (Redis 7 + AOF/RDB Persistence + Lua Atomic Scripts + Redis Streams + Sentinel HA)

- Implemented cache-aside and write-through caching layers with jittered TTLs to prevent cache stampedes.
- Authored atomic Redis Lua scripts for sliding-window API rate limiters and distributed checkout locks.
- Built real-time sorted-set (ZSET) leaderboards, Pub/Sub notifications, and durable Redis Streams consumer groups.
- Configured Redis memory eviction policies, AOF persistence, and failover monitoring in live client production.

Target Job Roles: High-Performance Backend & Caching Engineer, Distributed Data Systems Developer, Platform Performance Engineer, Backend SDE (Scale & Low-Latency)

Realistic Annual CTC Scope: **Backend / Data Engineer with Redis (0–1 Yr):** ₹5.5 LPA – ₹10.0 LPA **High-Scale Backend Systems Engineer (1–3 Yrs):** ₹11.5 LPA – ₹23.0 LPA **Staff Systems / Performance Architect (3–6+ Yrs):** ₹24.0 LPA – ₹45.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/redis-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.