Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: REDI • **Updated:** 01 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 Gemini AI Interviews at ₹2,999.

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

Fixed Non-Negotiable Tuition Fee: Base Fee **₹2,999** + 18% GST (₹540) = **Total Incl. GST: ₹3,539**

Live Batch Tracks: Weekday Live Online Batch (Mon – Fri) (Monday to Friday (5 Live Interactive Classes / Week) • 1.5 Hours / Day (60 Mins Live Architecture & Coding + 30 Mins Hands-On Lab) • 17–19 Weeks Fast-Track) | Weekend Intensive Live Online Batch (Sat – Sun) (Saturday & Sunday (Designed for Working Professionals & College Students) • 3.0 Hours / Day (2 Hrs Deep-Dive Live Coding + 1 Hr Guided Project Build) • 19–21 Weeks Weekend Track)

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 200 Timed Mock Exams (10,000 Redis & Distributed Caching MCQs) and 200 Spoken Voice + Google Gemini AI Systems Architecture Interviews
5. Choosing optimal Redis data structures: Strings, Hashes, Sets, Sorted Sets (`ZSET`), Bitmaps, HyperLogLog, and Streams.
6. Production caching patterns: Cache-Aside, Write-Through, jittered TTLs, and stampede protection (single-flight locks).
7. Distributed systems primitives: sliding-window rate limiters, atomic Lua scripts, Redlock patterns, and session stores.
8. Persistence & HA internals: RDB snapshots vs AOF fsync policies, memory eviction (`allkeys-lru`), Sentinel, and Cluster slots.

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. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • 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 MODULE & TOPIC-WISE SYLLABUS (16 MODULES • 42 LIVE TOPICS)

MODULE 1 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Explain how non-blocking I/O multiplexing (`epoll` / `kqueue`) + RAM access eliminates lock contention and context-switch overhead • ✓ Compare sub-millisecond in-memory Redis data structures against disk-backed ACID relational databases (PostgreSQL / MySQL) • ✓ Prevent the 3 classic

production caching outages: **Cache Stampede (Thundering Herd)**, **Cache Avalanche**, and **Cache Penetration**

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]

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...

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]

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...

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 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Tune Linux kernel settings (`vm.overcommit_memory = 1`, `net.core.somaxconn = 65535`, disable Transparent Huge Pages `THP = never`) • ✓ Configure `maxmemory` (`~70–75%` of host RAM leaving Copy-on-Write `fork()` headroom!) and select accurate `maxmemory-policy` (`allkeys-lfu` vs `noeviction`) • ✓ Apply live zero-restart changes safely via `CONFIG GET` / `CONFIG SET` + `CONFIG REWRITE`

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]

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

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]

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

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 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Select optimal structures across Strings, Hashes, Lists, Sets, Sorted Sets, Bitmaps, `HyperLogLog` (`12 KB` UV counter!), Geo, and Streams • ✓ Inspect internal compact C encodings via `OBJECT ENCODING` (`int`, `embstr` `<=44B`, `raw`, `listpack`, `intset`, `skiplist`) • ✓ Replace production-freezing `KEYS *` and `DEL` on large keys with cursor-based `SCAN` (`scan_iter`) and background-thread `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]

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...

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]

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

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 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Combine `SET key val NX PX ms` in a single atomic command (replacing broken two-step `SETNX` + `EXPIRE` scripts!) • ✓ Increment 64-bit signed integer & float counters atomically (`INCR`, `INCRBY`, `DECRBY`, `INCRBYFLOAT`) and batch reads/writes with `MGET`/`MSET` • ✓ Model domain objects as compact `listpack`-encoded Redis Hashes (`HSET`, `HMGET`, `HINCRBY`) instead of scattering 10 separate top-level String keys per user

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]

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...

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]

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

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 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Push/pop head & tail in `O(1)` (`LPUSH`, `RPOP`), cap activity feeds (`LPUSH` + `LTRIM 0 99`), and avoid `O(N)` `LINDEX`/`LRANGE` deep pagination traps • ✓ Implement crash-safe reliable work queues using atomic `LMOVE` / `BLMOVE src processing LEFT RIGHT` (or upgrade to Redis Streams `XREADGROUP`) • ✓ Perform Set algebra (`SADD`, `SISMEMBER`, `SMISMEMBER`, `SINTER`, `SUNION`, `SDIFF`, `SRANDMEMBER`) while protecting `intset` compact integer storage

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]

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...

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]

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

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 • 11 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Understand how Sorted Sets (`ZSET`) combine a **Hash Table** (`O(1)` `ZSCORE` lookup) + **SkipList** (`O(log N)` `ZADD` / `ZRANK` / `ZRANGE` ordered range queries) • ✓ Use unified Redis 6.2+/7 `ZRANGE ... REV BYSCORE LIMIT` (replacing deprecated `ZREVRANGEBYSCORE`) and tie-breaker composite scores • ✓ Manage Hash expiration accurately: top-level key `EXPIRE` traps on `HSET` AND Redis 7.4 per-field `HEXPIRE` / `HTTL` (`NX` / `XX` / `GT` / `LT`)

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]

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

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]

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...

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 • 10 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Distinguish serialized atomic isolation (`MULTI` ... `EXEC`) from RDBMS rollback transactions (runtime type errors inside `EXEC` do NOT roll back sibling commands!) • ✓ Guard read-modify-write workflows against concurrent races using `WATCH key` optimistic locking (`WatchError` retry loop) • ✓ Eliminate multi-RTT `WATCH` contention on hot keys by running single-RTT atomic server-side **Lua scripts** (`r.register_script` / `EVALSHA`)

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]

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...

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]

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...

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 • 10 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Eliminate `N * RTT` TCP stop-and-wait latency by batching `100–1,000` commands per network flush using Redis Pipelining • ✓ Understand `r.pipeline(transaction=False)` in `redis-py` (pure RTT batching WITHOUT `MULTI/EXEC` overhead!) vs `transaction=True` • ✓ Share a singleton `redis.ConnectionPool` (`socket_timeout`, `socket_connect_timeout`, `health_check_interval=30`) across worker threads/coroutines

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]

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

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]

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

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 • 10 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Contrast point-in-time binary `RDB` snapshots (`BGSAVE`, never blocking `SAVE`!) against write-ahead `AOF` (`appendfsync everysec`, max ~1s data loss) • ✓ Enable Redis 7 Multi-Part AOF + RDB hybrid preamble (`aof-use-rdb-preamble yes`) for both 1-second durability AND fast startup recovery • ✓ Replace shared root `requirepass` with least-privilege Redis 7 **ACLs** (`ACL SETUSER svc_checkout on >secret ~checkout:* +@read +@write -@dangerous`)

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]

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

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]

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...

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 • 10 LIVE + LAB HOURS • 2 TOPICS • 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: ✓ Run realistic benchmarks (`redis-benchmark -t set,get -n 500000 -c 50 -r 100000 -d 256 -P 16 --threads 4`) avoiding fake single-key `"key:__rand_int__"` illusions • ✓ Compare **Redis Sentinel** (HA failover when dataset fits in 1 node's RAM, e.g. `<= 64 GB`) vs **Redis Cluster** (sharding writes & RAM across `N` primaries) • ✓ Master **`CRC16(key) % 16384`** hash-slot routing, avoid `CROSSSLOT` errors via **`{user:42}` hash tags**, and prevent hot-shard skew!

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]

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

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]

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

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 • 16 LIVE + LAB HOURS • 4 TOPICS • 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]

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...

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]

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

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]

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...

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]

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...

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 • 16 LIVE + LAB HOURS • 4 TOPICS • 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]

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...

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

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...

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]

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...

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

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

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 • 16 LIVE + LAB HOURS • 4 TOPICS • 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]

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...

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

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

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

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

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]

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...

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 • 16 LIVE + LAB HOURS • 4 TOPICS • 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]

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...

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

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

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]

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...

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]

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...

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 • 14 LIVE + LAB HOURS • 3 TOPICS • 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 Gemini 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 + Google Gemini 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]

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...

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]

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...

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]

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

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 • 14 LIVE + LAB HOURS • 3 TOPICS • 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 Gemini 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 + Google Gemini 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]

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

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]

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...

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

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

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, GLOBAL CERTIFICATIONS & 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.

Aligned Global Vendor Certifications: **REDIS-DEV** (Redis Certified Developer) • **C100DEV** (MongoDB Certified Associate Developer)

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 98765 43210 • Web: https://mockattempt.com/courses/redis-training

Try 2 Free Topic-Wise 50-MCQ Mock Tests & Spoken Gemini AI Interviews daily on every topic above before enrolling.