Redis: Remote Dictionary Server.

Key features:

- In-memory: Very fast (microseconds latency).
- Persistence: Can save data to disk (RDB, AOF).
- Data structures: Not just strings—supports lists, sets, hashes, sorted sets, streams, etc.
- Single-threaded core: Commands are atomic.
- Replication & clustering: Built-in high availability and scaling.

Why Use Redis?

- Speed: Data is kept in RAM, making it much faster than disk-based DBs.
- Versatile data types: Useful for caching, queues, leaderboards, pub/sub, etc.
- Scalability: Supports sharding via clustering.
- Durability options: Can configure persistence for different reliability vs. performance trade-offs.

Core Data Structs

Data Type	Example Usage	Key Commands	Example
String	Cache HTML, JSON, or counters	`SET`, `GET`, `INCR`, `DECR`	`SET page:home "<html>...</html>"`
List	Message queues, task lists	`LPUSH`, `RPUSH`, `LPOP`, `LRANGE`	`LPUSH queue:tasks "task1"`
Set	Unique user IDs, tags	`SADD`, `SREM`, `SMEMBERS`	`SADD users:active 1001 1002`
Sorted Set	Leaderboards, priority queues	`ZADD`, `ZRANGE`, `ZREVRANK`	`ZADD leaderboard 100 "player1"`
Hash	Store objects/records	`HSET`, `HGET`, `HGETALL`	`HSET user:1001 name "Alice" age 30`
Bitmap	Tracking active users	`SETBIT`, `GETBIT`, `BITCOUNT`	`SETBIT active_users 5 1`

Data Type	Example Usage	Key Commands	Example
HyperLogLog	Approximate unique counts	`PFADD`, `PFCOUNT`	`PFADD unique_visitors "user123"`
Streams	Event logs, real-time data feeds	`XADD`, `XREAD`	`XADD logs * event "user_login"`
Geospatial	Location-based search	`GEOADD`, `GEORADIUS`	`GEOADD places 13.361389 38.115556 "Palermo"`

Redis Core Data Types – Detailed Guide with Commands

1. Strings

Description: Binary-safe sequences of bytes (text, numbers, serialized objects).
Max size: **512 MB**
Use cases: Simple key-value storage, caching blobs, counters.

```
SET name "Alice"
GET name
DEL name
SET counter 10
INCR counter
DECR counter
APPEND name " Smith"
STRLEN name
MSET a 1 b 2 c 3
MGET a b c
```

2. Lists

Description: Ordered collection of strings, can push/pop from both ends.

Use cases: Message queues, task lists, logs.

```
LPUSH queue task1
RPUSH queue task2
LRANGE queue 0 -1
LPOP queue
```

```
RPOP queue
LLEN queue
LINDEX queue 1
LREM queue 2 "task1"
```

3. Sets

Description: Unordered collection of unique strings.
No duplicates, fast membership checks.

Use cases: Tags, unique visitors, friend lists.

```
SADD tags redis
SADD tags nosql
SADD tags redis
SMEMBERS tags
SISMEMBER tags redis
SREM tags nosql
SCARD tags
SUNION set1 set2
SINTER set1 set2
SDIFF set1 set2
```

4. Sorted Sets (Zsets)

Description: Like sets but each element has a score. Ordered by score.

Use cases: Leaderboards, rankings, priority queues.

```
ZADD leaderboard 150 player1
ZADD leaderboard 200 player2
ZRANGE leaderboard 0 -1 WITHSCORES
ZREVRANGE leaderboard 0 -1 WITHSCORES
ZRANK leaderboard player1
ZINCRBY leaderboard 50 player1
ZREM leaderboard player2
```

5. Hashes

Description: Maps between string fields and string values (like JSON objects).

Use cases: User profiles, product details, settings.

```
HSET user:1 name "John" age "30"
HGET user:1 name
HGETALL user:1
HDEL user:1 age
HLEN user:1
HKEYS user:1
HVALS user:1
HEXISTS user:1 age
```

6. Bitmaps

Description: Bit-level operations on strings. Efficient for large boolean sets.

Use cases: Login tracking, feature flags, activity tracking.

```
SETBIT active_users 100 1
GETBIT active_users 100
BITCOUNT active_users
BITOP AND dest key1 key2
```

7. HyperLogLogs

Description: Probabilistic structure for approximate unique counts (~0.81% error).
Uses ~12 KB memory fixed.

Use cases: Unique visitor count, large-scale counting.

```
PFADD visitors "user1" "user2"
PFCOUNT visitors
PFMERGE all_visitors visitors1 visitors2
```

8. Streams
   Description: Log-like structure with ID + field-value pairs, keeps order & persistence.

Use cases: Event sourcing, analytics, chat.

```
XADD mystream * user alice message "hello"
XREAD COUNT 2 STREAMS mystream 0
XRANGE mystream - +
XDEL mystream 1607513495873-0
XGROUP CREATE mystream group1 $ MKSTREAM
XREADGROUP GROUP group1 consumer1 STREAMS mystream >
```

9. Geospatial

Description: Store geo-coordinates and query by radius/distance.

Use cases: Store locators, nearby searches.

```
GEOADD cities 13.361389 38.115556 "Palermo"
GEOADD cities 15.087269 37.502669 "Catania"
GEODIST cities Palermo Catania km
GEORADIUS cities 15 37 200 km
GEOPOS cities Palermo
```

Persistence Modes

Redis is an in-memory database, offering high speed but volatile storage. It provides persistence options to prevent data loss on server restarts.

1. RDB (Redis Database Backup)

- **Description**: Takes snapshots of all data at configurable intervals (set in `redis.conf`).
- **Storage**: Saved in a `.rdb` file.
- **Pros**:
  - Compact files, easy to back up or transfer.
  - Fast to load into memory on restart.
- **Cons**:
  - Data changes after the last snapshot are lost if Redis crashes.
- **When to use**: Suitable for backups when losing a few minutes of data is acceptable.

2. AOF (Append-Only File)

- **Description**: Logs every write operation in a plain-text file. On restart, Redis replays commands to restore the dataset.
- **Settings**: Configurable `appendfsync` for durability vs. performance trade-offs.
- **Pros**:
  - Safer, with minimal data loss (down to milliseconds).
  - Human-readable file.
- **Cons**:
  - Larger file sizes, slower writes compared to RDB.
- **When to use**: Ideal when data safety is critical.

3. Mixed

- **Description**: Combines RDB for fast full recovery with AOF to capture recent changes.
- **Advantage**: Balances speed and durability.

Expiration & Eviction

Expiration

- Keys can be set to self-delete after a specified time.
- **Example**:

```
SET session:123 "user1" EX 60  # Expires in 60 seconds
EXPIRE session:123 120         # Change to 120 seconds
TTL session:123                # Check remaining time
```

Eviction Policies

- Configurable with `maxmemory-policy` when memory is full.
- **Common Policies**:
  - `noeviction` : Stops writes, returns error when memory is full.
  - `volatile-lru` : Removes least recently used keys with expiry set.
  - `allkeys-lru` : Removes least recently used keys, regardless of expiry.
  - `volatile-ttl` : Removes keys with the shortest time-to-live first.
  - `allkeys-random` : Randomly removes any key.

Transactions

- Redis supports atomic execution of multiple commands.
- **Workflow**:
  - `MULTI` : Starts a transaction.
  - Commands are queued but not executed immediately.
  - `EXEC` : Executes all queued commands atomically.
- **Example**:

```
MULTI
SET balance 100
```

```
DECR balance
EXEC
```

- **Note**: No rollback; if one command fails, others still execute.

Pub/Sub (Publish/Subscribe)

- Enables real-time messaging between processes.
- Publishers send messages to a channel, instantly received by all subscribers.
- **Example**:

```
# Subscriber 1
SUBSCRIBE news

# Subscriber 2
SUBSCRIBE news

# Publisher
PUBLISH news "Breaking update!"
```

- `PSUBSCRIBE` supports pattern matching for channel names.

Redis Streams

- Advanced alternative to Pub/Sub:
  - Messages persist until deleted.
  - Unique IDs (timestamp + sequence).
  - Supports replaying message history.
  - Consumer groups for distributed processing.
- **Example**:

```
XADD mystream * user alice message "Hello"
XRANGE mystream - +        # Read all messages
XREAD COUNT 2 STREAMS mystream 0
```

- **Use Cases**: Event sourcing, chat logs, IoT data feeds.