Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

**Redis Pipelining**

A Request/Response server can be implemented so that it is able to process new requests even if the client didn't already read the old responses. This way it is possible to send multiple commands to the server without waiting for the replies at all, and finally read the replies in a single step.

This is called pipelining, and is a technique widely in use since many decades. For instance many POP3 protocol implementations already supported this feature, dramatically speeding up the process of downloading new emails from the server.

**Redis Commands**

**SET**: Set a string variable

**GET** : get a value of variable

**DEL** : delete a variable

**EXISTS** : Check if value exist

**MSET** KEY "VALUE"// To set multiple key to respective values and also replace existing with new ones

**MSETNX** KEY "Value" // To set multiple key to respective values. Will not overrite existing value . Will not perform if even single key already exists

**APPEND** mykey "Stringtoappend"

**GETRANGE** mykey 0 -1 : return the substring -1 means

**RENAME** mykey myrenamekey

**RENAMENX** mykey myrenamekey // rename key to newkey if newkey doesn't exists . Return error if key doesn't exists

**GETSET** //return old value work like set

**SETEX** mykey 10 "hello" ///equivalant to set mykey "hello" expire mykey 10 Set key to hold a string value and timeout after given amount of seconds

**PSETEX** mykey 1000 "hello" ///equivalant to set mykey "hello" expire mykey 10 Set key to hold a string value and timeout after given amount of miliseconds

**TTL** Remaining time Seconds

**PTTL** Remaining time in miliseconds

**PERSIST** Remove existing timeout on a key

**SETNX** mykey "hello" // Works like set if key doesn't exists. if the key already exists it will not change

**SCAN** //itertate the set of key on database //return small of data per call// can be used in production// takes cursor / positions as parameter

Cursor based Iterator

1) The server returns an updated cursor with every call of the command

2) this can be used as an argument of next call

3) Iteration starts when cursor is set to 0

4) Terminates when cursor returned from server the server is 0

Scan Gurantees Full iteration will retrieve all element that were present in the collection from start to the end.Never return any element that was not present in collection from start to finish

Count Option

Count can be defined in a scan to override the default returned per iteration

default 10

the user can specify the amount of work done at every call

COUNT can be changed from one iteration to the next

**SSCAN**:Used with sets, Returns list of set members

**HSCAN**:Used with hashes, Return array of element with field and value

**ZSCAN**:Used with sorted set, Return array of element with associated score

**MATCH** used with SCAN with match expression

Data Types:

**Redis Data Types**

1) **List** // List or group of String// Sorted by insertion order// Element can be pushed on the head or tail of the list// Often used as Producer/Consumer queries

1)Inserting element into List

// **LPUSH**(inserting element at head or left)

// **RPUSH** (inserting element at tail or right

2) Show Element from list //**LRANGE** listname range

3) Count element from list **LLEN** listname

4 **LPOP** // Left POP

5 **RPOP** // Right POP

6) **LINSERT** before

2)**Sets**: // Unordered collections of String//Can add remove test for existence for element//Do not allow repeating element//Support server side commands to commpute sets starting from existing sets

1)**SADD**- add given values to a set// already value will be ignorned///

sadd people 1970 "John Deo"

2)**SREM**- Remove the element from the list

3)**SISMEMBER** - Tests ifgiven value is in the set

4)**SMEMBERS** - Return the list of all the members of the set

5)**SCARD** - return the count

6)**SMOVE** - Move element from one set to another

7)**SUNION** - Combine two or more set and return a list of members

8)**SDIFF** - Returns the members of the set resulting from the difference between the first and all successive sets

9)**SRANDMEMBER**- Returns random number from set

10)**SPOP** - Remove and return a random member from a specific set

3)**Sorted Set** : Score

1)**ZADD**: Adding a member into Sorted Set

2)**ZREM**: Removing a member from Sorted Set

3)**ZRANGE** : Fetch Value with the specified range. Order lowest to highest by score

4)**ZREVRANGE** : Reverse the sorted Set . Order higest to lowest score

5)**ZRANGEBYSCORE**: Return score based on score

6)**ZCARD**: Returns the number of member in sorted set

7)**ZCOUNT**: Return the number of element in Sorted set score between min and max

8)**ZINCRBY** : Increase score by number provided

4)**Hashes** : Maps b/w string field and string value.. Perfect for representing objects.....very compact...like JSON Object

1)**HSET**: Set a field in hashes // HSET user1:name "John"

2)**HMSET** : Set multiple field to their respoective values... // HMSET user2:name "jill" email "jill@abc.com" age 25//HMSET user:kate name "kaye Smith" email "kate@gmail.com" age 45

3)**HGET** : Get a value associated with a field HGET user1 name

4)**HMGET** : Get Multiple values HMGET user2 name age email

5)**HGETALL**: Get all values from field HGETALL user2

6)**HDEL**: Remove specified fields from hash : HDEL user2 age

7)**HEXISTS**: Hexists name user3

8)**hincrby** : hincrvy

9)**Hkeys** : hkey user1

10)**HLEN**

11)**HVALs**

**Redis Persistence**

Redis provides a different range of persistence options:

* The RDB persistence performs point-in-time snapshots of your dataset at specified intervals.
* The AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log in the background when it gets too big.
* If you wish, you can disable persistence completely, if you want your data to just exist as long as the server is running.
* It is possible to combine both AOF and RDB in the same instance. Notice that, in this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.

**Persistence setting in redis.conf:**

The filename where to dump the DB

**dbfilename dump.rdb**

**dir ./** ---path to save dump.rdb and appendonly.aof file

**appendonly yes**

# The name of the append only file (default: "appendonly.aof")

**appendfilename "appendonly.aof"**

Create a cronjob to save the dump.rdb & appendonly.aof file to another location everyday.

**Securing a Redis instance**

Require clients to issue AUTH <PASSWORD> before processing any other commands. This might be useful in environments in which you do not trust

others with access to the host running redis-server.

3 ways to secure redis instance:

1. **requirepass foobared**
2. **rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52**
3. Only allow the port of redis from authorized domains and IPs.

Setting password using command Line : **CONFIG set requirepass "sandeep"**

**Redis - Publish Subscribe**

Redis Pub/Sub implements the messaging system where the senders (in redis terminology called publishers) sends the messages while the receivers (subscribers) receive them. The link by which the messages are transferred is called channel.

In Redis, a client can subscribe any number of channels.

**SUBSCRIBE redisChat**

**PUBLISH redisChat "Redis is a great caching technique"**

**PUBLISH redisChat "Learn redis by tutorials point"**

**Eviction Policy :**

Set a memory usage limit to the specified amount of bytes. When the memory limit is reached Redis will try to remove keys according to the eviction policy selected.

**MAXMEMORY POLICY**: how Redis will select what to remove when maxmemory

is reached. You can select among five behaviors:

**volatile-lru** -> Evict using approximated LRU among the keys with an expire set.

**allkeys-lru** -> Evict any key using approximated LRU.

**volatile-lfu** -> Evict using approximated LFU among the keys with an expire set.

**allkeys-lfu** -> Evict any key using approximated LFU.

**volatile-random** -> Remove a random key among the ones with an expire set.

**allkeys-random** -> Remove a random key, any key.

**volatile-ttl** -> Remove the key with the nearest expire time (minor TTL)

**noeviction** -> Don't evict anything, just return an error on write operations.

**LRU means Least Recently Used**

**LFU means Least Frequently Used**

Both LRU, LFU and volatile-ttl are implemented using approximated randomized algorithms.

Note: with any of the above policies, Redis will return an error on write operations, when there are no suitable keys for eviction.

The default is:

**maxmemory-policy noeviction**

**maxmemory <bytes>**

**maxmemory-samples 5** : LRU, LFU and minimal TTL algorithms are not precise algorithms but approximated.algorithms (in order to save memory), so you can tune it for speed or accuracy. For default Redis will check five keys and pick the one that was used less recently, you can change the sample size using the following configuration directive.

The default of 5 produces good enough results. 10 Approximates very closely

true LRU but costs more CPU. 3 is faster but not very accurate.

**replica-ignore-maxmemory yes** : replica doesn't have eviction policy master will send del command for specigic key which it has evicted.

**maxmemory 2000**

**HSET USER name Sandeep**

(**error) OOM command not allowed when used memory > 'maxmemory'.**

**Redis latency monitoring framework**

**CONFIG SET latency-monitor-threshold 100**

**Redis Clustering:**

Redis Cluster provides a way to run a Redis installation where data is automatically sharded across multiple Redis nodes.

Redis Cluster also provides some degree of availability during partitions, that is in practical terms the ability to continue the operations when some nodes fail or are not able to communicate. However the cluster stops to operate in the event of larger failures (for example when the majority of masters are unavailable).

So in practical terms, what do you get with Redis Cluster?

* The ability to automatically split your dataset among multiple nodes.
* The ability to continue operations when a subset of the nodes are experiencing failures or are unable to communicate with the rest of the cluster.

Configuration for Redis Cluster :

Download From http://download.redis.io/releases/redis-5.0.8.tar.gz

tar xzf redis-5.0.8.tar.gz

cd redis-5.0.8

make

src/redis-server

mkdir cluster-test

cd cluster-test

mkdir 7000 7001 7002 7003 7004 7005

cp redis-5.0.8 to 7000 7001 7002 7003 7004 7005

port 7000

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

cd /home/sandeep/redis-cluster-test/7000

/home/sandeep/redis-cluster-test/7000/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7000/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7001

/home/sandeep/redis-cluster-test/7001/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7001/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7002

/home/sandeep/redis-cluster-test/7002/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7002/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7003

/home/sandeep/redis-cluster-test/7003/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7003/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7004

/home/sandeep/redis-cluster-test/7004/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7004/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7005

/home/sandeep/redis-cluster-test/7005/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7005/redis-5.0.8/redis.conf daemonize yes

cd /home/sandeep/redis-cluster-test/7006

/home/sandeep/redis-cluster-test/7006/redis-5.0.8/src/redis-server /home/sandeep/redis-cluster-test/7006/redis-5.0.8/redis.conf daemonize yes --dbfilename dump.rdb dir /tmp/dump.rdb

/home/sandeep/redis-cluster-test/7006/redis-5.0.8/src/redis-cli -p 7006

sudo apt install ruby

gem install redis

To run the cluster : /home/sandeep/redis-cluster-test/7000/redis-5.0.8/src/redis-cli -p 7000 --cluster create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 127.0.0.1:7006 --cluster-replicas 2

/home/sandeep/redis-cluster-test/7002/redis-5.0.8/src/redis-cli -p 7002

/home/sandeep/redis-cluster-test/7004/redis-5.0.8/src/redis-cli -p 7004

/home/sandeep/redis-cluster-test/7005/redis-5.0.8/src/redis-cli -p 7006

/home/sandeep/redis-cluster-test/7000/redis-5.0.8/src/redis-cli -p 7000 cluster nodes | grep myself

To know the nodeId: redis-cli -p 7000 cluster nodes | grep myself

To know the master slave nodes hash slots: redis-cli --cluster check 127.0.0.1:7000

To know Cluster master : redis-cli -p 7000 cluster nodes | grep master

To know Cluster slave : redis-cli -p 7000 cluster nodes | grep slave

Manual Failover: redis-cli -p 7002 debug segfault

To know the nodes connected: redis-cli -p 7000 cluster nodes

Adding new Node Manually : redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 After restaring the server

Adding a new node as a replica : redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave

Specify exactly what master you want to target with your new replica: redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave --cluster-master-id 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e

Adding new Replica of slave : redis 127.0.0.1:7006> cluster replicate 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e

Verify nre replica to cluster node: redis-cli -p 7000 cluster nodes | grep slave | grep 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e

Removing a node : redis-cli --cluster del-node 127.0.0.1:7000 `<node-id>`