* Redis is an open source, **in-memory** **data structure store**, used as database, cache and message broker
* **NoSQL** Key/Value Store
* Redis is similar to MongoDB in that it is **NoSQL** and could be used for similar key/value cases
* MongoDB is a **“disk-based”** document store while Redis is **“memory-based”**
* Redis is used for caching
* It is similar to Memcached in that the data is stored in memory
* Memcached uses volatile cache, therefore the data does not get persisted and is lost with a restart. Redis uses built-in persistence and will not disappear with a restart

**The datatypes that Redis allows is one of it’s best features**

* Strings
* Hashes
* Lists
* Sets
* Sorted Sets
* Bitmaps
* Hyperlogs
* Geospatial Indexes

**Scaling & Partitioning**

* Not the easiest thing to do
* Has gotten better
* Depends on using Redis as cache or data store
* Partitioning: Spit data among multiple Redis instances

**General Security Model**

* Designed to be accessed by trusted clients inside trusted environments
* Not usually a good idea to expose Redis directly
* Clients inside front-end of a web application will query Redis
* Not optimized for max security but for max performance
* Designed to be accessed by trusted clients
* Do not allow any external access if possible
* Simple authentication can be setup
* Redis can be restricted to certain interfaces

**Network Security**

* Deny access to main Redis port
* Firewalled to prevent access
* Loopback can be used

To bind Redis to a single interface (redis.conf):

Bind 127.0.0.1

**Authentication**

* Tiny layer of authentication (Enable in redis.conf)
* Refuses all queries from unauthenticated users
* User must send the AUTH command along with the password
* Password is stored in plain text in the redis.conf file

**Disabling & Renaming Commands**

* Specific commands can be disabled or renamed
* Normal users should not be able to run **CONFIG** or **FLUSHALL**
* To disable, rename to empty string

Commands are renamed in redis.conf

**rename-command-> CONFIG b840fc02d524045429941cc15f59e41cb7be6c52**

**rename-command-> CONFIG ""**

**Other type of Attack**

* Attacks can be triggered from outside even if Redis is secure
* Web form vulnerability – DoS Attacks

**Redis has clients for just about every popular programming language**

* http://redis.io/clients

**Redis as Data Structure Server and how It is different**

* Redis doesn’t serve raw data
* No schemas & column names
* Traditional db’s – Scan entire table or scan entire index
* Redis – Direct data retrieval commands
* No internal query engine
* You decide how to store and retrieve data

**Redis as Data Storage**

* Different than SQL
* Store data you KNOW you will use
* Explicit reading NOT random discovery
* Data stored in live memory

**Basic Commands**

PING --Check if server is running

ECHO hello - Check echo

SET foo 100 - Sets a string variable

GET foo - Get a string variable

INCR foo - Increase the variable by 1

INCR foo - Increase the variable by 1

DECR foo - Decrease the variable by 1

SET bar 200 - Set a String variable

EXISTS bar - Check if value exists

EXISTS bar2 - Check if value exists

DEL bar - delete variable

EXISTS bar - Check if variable exists after delete

INCR foo > commands.txt - Increase by 1 and save to file

FLUSHALL - delete all data

SET server:name myserver

GET server:name

SET server:port 6379

GET server:port

SET resource:foo hello

EXPIRE resource:foo 120 - setting varaible expire time

TTL resource:foo - checking variable expire time

MSET key1 "Hello" key2 "World" -- Sets multiple keys to respective values and Replaces existing values with new ones

MGET key1 key2 -- Returns values of all specified keys and Nill is returned if key doesn’t hold a value

MSETNX key3 "hi" -- Sets multiple keys to respective values as long as none of the keys exist and Will NOT overwrite existing values and Will NOT perform if even a single key already exists

MSETNX key3 "hello" key4 "there"

SET greeting "Hello" --Set a variable

GET greeting -- Get a variable

APPEND greeting " World" -- If key already exists and is a string, the value will be appended at the end of the string and If the key does NOT exist. It works as SET

GET greeting -- Get a variable

APPEND foo "bar" --

GET foo

RENAME greeting greet - Renames a key and Returns error if key doesn’t exist and If does exist, it is overwritten

GET greet

RENAMENX greet greeting Renames key to newkey if newkey does not exist and Returns an error if key does not exist

RENAMENX key1 greting

SET mystring "This is my string"

GETRANGE mystr 0 -1 -- Returns the substring of a string value and Determined by offsets start and end and Negative offsets can be used to start from the end of the string

GETRANGE mystr 0 5

GETRANGE mystr 3 8

GETSET foo -- Automatically sets key to value and returns the old value and Returns an error when key exists and Can be used with INCR for counting with automatic reset

GET foo

SET mykey "hello"

EXPIRE mykey 10

TTL mykey

SETEX mykey 10 "hello" -- Set key to hold a string value and timeout after a given amount of seconds

PSETEX mykey 1000 "hello" -- Same as SETEX except it uses milliseconds instead of seconds

PTTL mykey -- PTTL is used to get the remaining time in milliseconds

SETEX mykey 120 "hello"

PERSIST mykey -- Removes existing timeout on a key

SETNX newkey "foobar" -- Works like SET if the key does NOT exist and If the key already exists, it will not change

GET foobar

SETNX newkey "barfoo”

**Data Persistence Features**

* Datasets are all stored in memory
* Datasets can be saved to disk
* Redis fork – Creating child processes which are an exact copy of the parent
* Copy-On-Write Snapshot

**Persistence Process**

1. Client sends write command to database (Client memory)
2. Database receives the write (Server memory)
3. Database calls system call that writes data on disk (Kernel buffer)
4. The OS transfers the write buffer to the disk controller (Disk cache)
5. Disk controller writes to physical media (Physical Disk)

**Pools:**

Multiple Redis servers running on the same machine using different ports

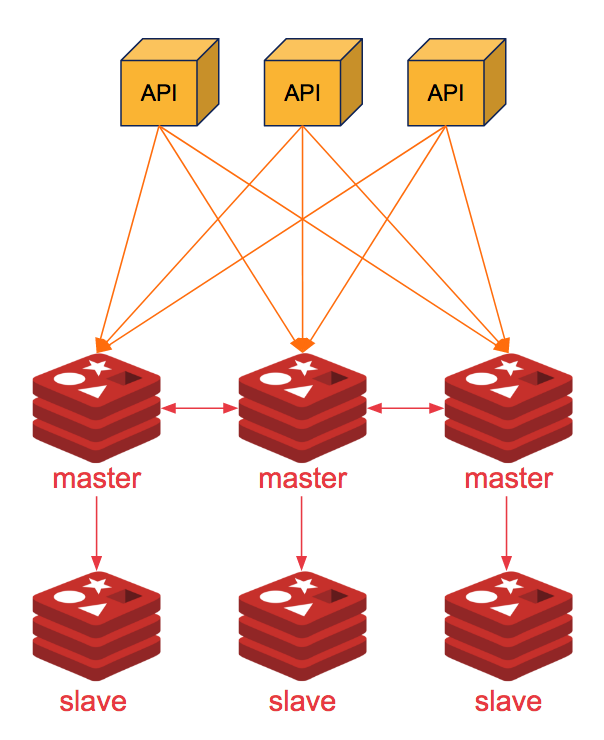
* Efficient memory usage
* More CPUs

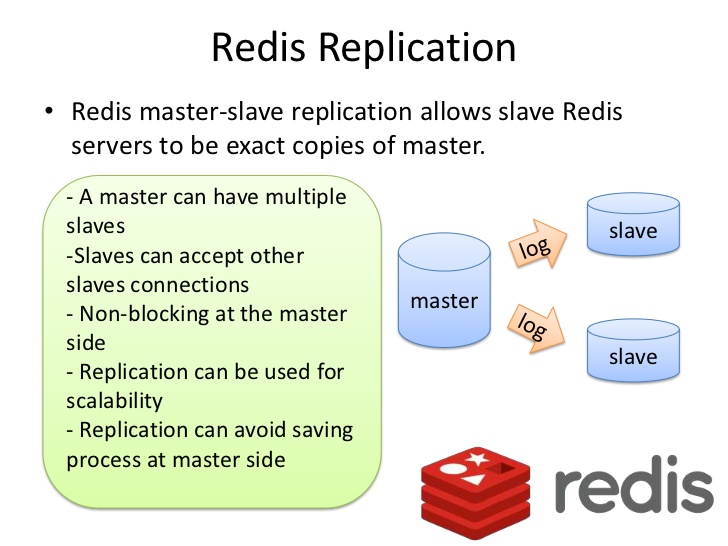
Better fine-tuning

**Replication**

Simple master-slave replication allows slave Redis servers to be copies of master servers

* Asynchronous replication
* Multiple slaves
* Connections from other slaves
* Non blocking on slave side
* Scalability & data redundancy
* Slave read-only



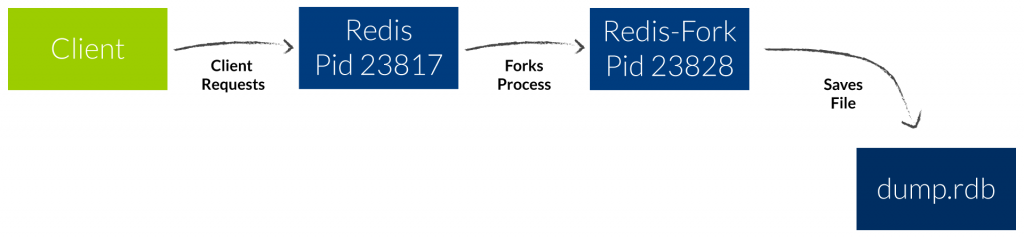


**Persistence Options**

* RDB – Point-in-time snapshots
* AOF – Write operation logging
* Disabled
* Both RDB & AOF

**RDB – Redis Database File**

* Simplest persistence mode
* Enabled by default
* Single-file point-in-time representation
* Uses snapshots
* Controlled by the user
* Can be modified at runtime
* Snapshots are produced as .rdb files
* **SAVE** & **BGSAVE** Commands

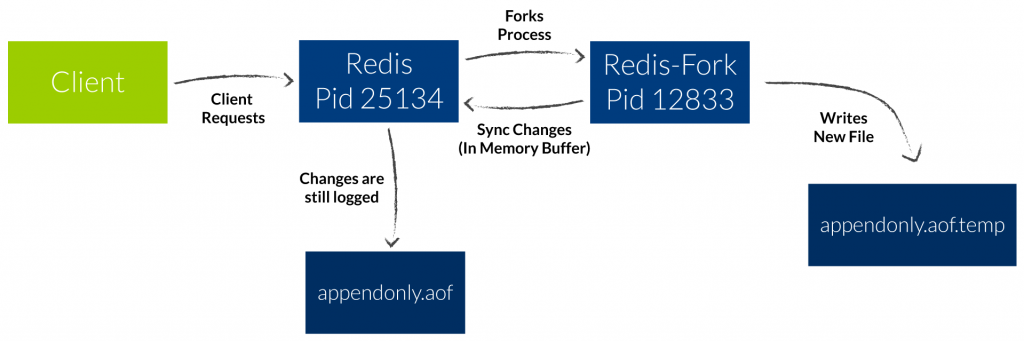


**AOF – Append Only File**

* Main persistence option
* Every operation gets logged
* Log is the same format used by clients
* Can be piped to another instance
* Dataset can be reconstructed
* Used when AOF file gets too big
* Rewrite database from scratch
* Directly access data in memory
* No need for disk access
* Once written, the temp file is synched on to disk

It works in **fsync Policies**

* No fsync – Done by OS. Usually every 30s or so
* fsync every second (default)
* fsync at every query (slow)



Command Executed for Rbd and Aof

sudo locate \*rdb

redis-cli

SET user:john "John Doe"

SET user:mary "Mary Williams"

SET user:jill "Jill Jackson"

SET user:paul "Paul Harris"

SET user:mike "Mike SMith"

SET user:derek "Derek Hanlon"

SAVE

exit

sudo cp /var/lib/redis/dump.rdb /home/sandeep/redis-backup

sudo nano /home/sandeep/redis/redis-backup

sudo apt-get install -y rdiff-backup

sudo rdiff-backup --preserve-numerical-ids /var/lib/redis /home/sandeep/redis

sudo crontab -e

2

//SETUP DAILY BACKUP

0 0 \* \* \* rdiff-backup --preserve-numerical-ids --no-file-statistics /var/lib/redis /home/sandeep/redis

redis-cli

SET user:mike "Mike Wells"

exit

sudo rdiff-backup --preserve-numerical-ids /var/lib/redis /home/sandeep/redis

sudo nano /home/sandeep/redis/dump.rdb

//ENABLE AOF

BGREWRITEAOF

info

//Scroll to the Persistence section, and check that the aof entries match what's shown here. If aof\_rewrite\_in\_progress is 0, then the recreation of the AOF file has completed

exit

ls /var/lib/redis

sudo service redis-server stop

sudo nano /etc/redis/redis.conf

appendonly yes

sudo service redis-server start

sudo rdiff-backup --preserve-numerical-ids /var/lib/redis /home/sandeep/redis

redis-cli

SET user:tom "Tom Doe"

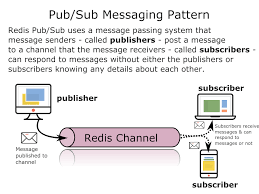
exit

sudo rdiff-backup --preserve-numerical-ids /var/lib/redis /home/sandeep/redis

**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.



**Pros**

* Easy to implement.
* Works well when the data sources and processors are distributed geographically.

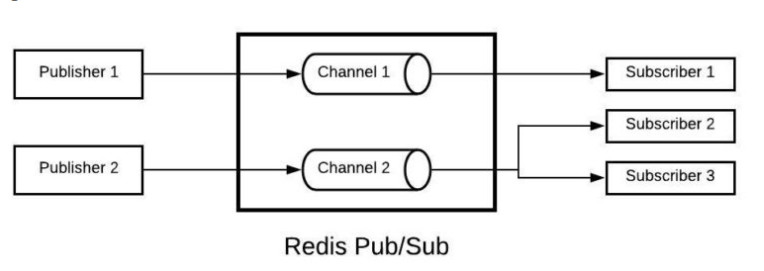
**Cons**

* The solution requires the publishers and subscribers to be up all the time. Subscribers lose data when stopped, or when the connection is lost.
* It requires more connections. A program cannot publish and subscribe to the same connection, so each intermediate data processor requires two connections – one to subscribe and one to publish. If running Redis on a DBaaS platform, it is important to verify whether your package or level of service has any limits to the number of connections.

**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**

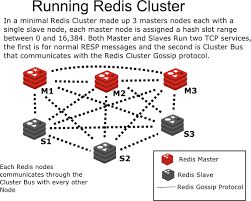
**Redis Clustering:**

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>`