Redis Notes

Table of Content

Chapter 1. Introduction to Redis

Redis is an acronym for Remote dictionary server. It’s an open source, in-memory and persistent key-value database / store which stores data as key-value pairs and also doubles up as a message broker. Redis supports a wide array of data structures including sets, lists, hashes, strings, HyperLogLogs and so many more.

What is a key-value pair?

A key-value pair is a set or pair of two linked items. Consider the following:

car = Mercedes

In this case, car is the key and Mercedes is the value. In a Redis database, this information can be written using the syntax:

```
SET  "key1" "value1"
```

Our example will translate to:

```
SET "car" "Mercedes"
```

1.1. Benefits of using Redis

- Unlike relational databases like MySQL, Redis is a NoSQL database that stores data as a key value pair. This makes it simple and flexible as there’s no need of creating any tables , columns and rows which are associated with relational databases. Feeding data to Redis is simple and straightforward.
- One of the apparent uses of Redis is its use as a caching system. While it does so, it also offers persistence to data being written on it.
- The in-memory architecture of Redis makes it super-fast in storage and retrieval of data.
- The redis cache system is quite robust and has the capacity of withstanding failures and interruptions.
- Redis ships with a Master-Slave replication feature. When changes are made to the master node, they are automatically replicated on the slave nodes to ensure high availability.
- Redis has the ability to store large key & value pairs of up to 512 MB.
- Given its small footprint, Redis can be installed in IoT devices such as Raspberry Pi and Arduino to support IoT apps.
- Redis is a cross-platform database and caching system which can be installed in Windows , mac and Linux.

Chapter 2. Install Redis on CentOS 8

2.1. Update system repositories

Login to your CentOS 8 / RHEL 8 system and update system packages and repositories using the command:

```
$ sudo dnf update -y
```

2.2. Install Redis with dnf

Redis version 5.0.x is now included in CentOS 8 AppStream repository and installing it is a walk in the park. Simply run the command:

```
$ sudo dnf install redis -y
```

[illegible]:

Once installed, you can verify the version of Redis installed by running the command:

```
[arafat@server ~]$ rpm -q redis
redis-5.0.3-2.module_el8.2.0+318+3d7e67ea.x86_64
```

[illegible]:

From the output , it is clear that we have installed Redis version 5.0.3. To retrieve more information about Redis such as the version, architecture, license and a brief description, run the command:

```
$ rpm -qi redis
```

[illegible]

To start Redis:

```
$ sudo systemctl start redis
```

[illegible]:

If you want to automatically start Redis in each system boot, you need to enable Redis:

```
$ sudo systemctl enable redis
```

□□□□□□□ □□□□□□:

To see status:

```
$ sudo systemctl status redis
```

□□□□□□□□□□, □□□□□ □□□□ □□□□□□ □□□□□ □□□□ `netstat` □□□□ □□□□□□□ □□□ □□□□□□□ □□□□ □□□□:

By default, Redis runs on port 6379. This can be confirmed by running the `netstat` command:

```
$ sudo netstat -pnltu | grep redis
```

2.3. Configure Redis

2.3.1. Allow Remote Access

The default installation only allows connections from localhost or Redis server and blocks any external connections. We are going to configure Redis for remote connection from a client machine.

Access the configuration file as shown:

```
$ sudo vim /etc/redis.conf
```

Locate the `bind` parameter and replace `127.0.0.1` with `0.0.0.0`

```
bind 0.0.0.0
```

Save and close the configuration file. For the changes to come into effect, restart Redis.

```
$ sudo systemctl restart redis
```

To log in to Redis shell, run the command:

```
$ redis-cli
```

Try to ping redis server. You should get a 'PONG' response as shown.

```
[arafat@server ~]$ redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

2.3.2. Securing Redis Server

Our Redis setup allows anyone to access the shell and databases without authentication which poses a grave security risk. To set a password, head back to the configuration file `/etc/redis.conf`

Locate and uncomment the `requirepass` parameter and specify a strong password.

```
################################## SECURITY ###################################

# 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.
#
# This should stay commented out for backward compatibility and because most
# people do not need auth (e.g. they run their own servers).
#
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you should
# use a very strong password otherwise it will be very easy to break.
#
# requirepass foobared
```

Restart Redis and head back to the server.

```
$ sudo systemctl restart redis
```

If you attempt to run any command before authenticating, the error shown below will be displayed

```
[arafat@server ~]$ redis-cli
127.0.0.1:6379> ping
(error) NOAUTH Authentication required.
127.0.0.1:6379>
```

To authenticate, type ‘auth’ followed by the password set.

```
auth 'PASSWORD'
```

Thereafter, you can continue running your commands.

```
[arafat@server ~]$ redis-cli
127.0.0.1:6379> auth 'PASSWORD'
OK
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

To come out from redis-cli, type `exit`

2.3.3. Configuring the Firewall for Redis

Lastly, we need to configure the firewall to allow remote connections to the Redis server. To do this, we need to open the redis port which is 6379.

So, run the commands below.

```
$ sudo firewall-cmd --add-port=6379/tcp --permanent
$ sudo firewall-cmd --reload
```

To access Redis remotely, use the syntax below.

```
$ redis-cli -h REDIS_IP_ADDRESS
```

Next authenticate and hit ‘ENTER’

The IP address of our Redis server is 192.168.1.5 The command from another client PC will be

```
$ redis-cli -h 192.168.1.5
```

Next, provide the password and hit ‘ENTER’

```
auth 'PASSWORD'
```

2.4. How to perform Redis Benchmark

Redis comes with a built-in tool known as `redis-benchmark` that gives insights on the system’s performance statistics such as data transfer rate, throughput and latency to mention a few.

Some of the command options you can use with Redis include

- `-n`: This defines the number of requests to be made. The default is 100000
- `-c`: Defines the number of parallel connections to be simulated. By default, this value is 50
- `-p`: This is the Redis port which by default is 6379
- `-h`: Used to define the host. By default, this value is set to localhost (127.0.0.1)
- `-a`: Used to prompt for a password if the server needs authentication
- `-q`: Stands for quiet mode. Displays the average requests made per second
- `-t`: Used to run a combination of tests
- `-P`: Used for pipelining for enhanced performance.
- `-d`: Specifies the data size in bytes for GET and SET values. By default, this is set to 3 bytes

Examples:

To confirm the average no. of requests that your Redis server can handle run the command:

```
$ redis-benchmark -q
```

Chapter 3. Hello World in Redis

3.1. PING

Returns PONG if no argument is provided, otherwise return a copy of the argument as a bulk. This command is often used to test if a connection is still alive, or to measure latency.

3.2. ECHO

Returns message.

```
[arafat@server ~]$ redis-cli
127.0.0.1:6379> PING
PONG
127.0.0.1:6379> PING "hello world"
"hello world"
127.0.0.1:6379> ECHO "Hello World!"
"Hello World!"
```

3.3. SET

Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type.

3.4. GET

Get the value of key. If the key does not exist the special value nil is returned. An error is returned if the value stored at key is not a string, because GET only handles string values.

```
127.0.0.1:6379> SET foo 100
OK
127.0.0.1:6379> GET foo
"100"
127.0.0.1:6379> SET bar "Hello World!"
OK
127.0.0.1:6379> GET bar
"Hello World!"
127.0.0.1:6379> GET nonexisting
(nil)
```

3.5. INCR

Increments the number stored at key by one. If the key does not exist, it is set to 0 before performing the operation. An error is returned if the key contains a value of the wrong type or contains a string that can not be represented as integer. This operation is limited to 64 bit signed integers.

3.6. DECR

Decrements the number stored at key by one. If the key does not exist, it is set to 0 before performing the operation. An error is returned if the key contains a value of the wrong type or contains a string that can not be represented as integer. This operation is limited to 64 bit signed integers.

```
127.0.0.1:6379> INCR foo
(integer) 101
127.0.0.1:6379> GET foo
"101"
127.0.0.1:6379> DECR foo
(integer) 100
127.0.0.1:6379> GET foo
"100"
```

3.7. EXISTS

Returns if key exists: - 1 if the key exists. - 0 if the key does not exist.

3.8. DEL

Removes the specified keys. A key is ignored if it does not exist.

```
127.0.0.1:6379> EXISTS foo
(integer) 1
127.0.0.1:6379> EXISTS nosuchkey
(integer) 0
127.0.0.1:6379> EXISTS foo bar nosuchkey
(integer) 2
127.0.0.1:6379> DEL bar
(integer) 1
127.0.0.1:6379> EXISTS bar
(integer) 0
127.0.0.1:6379> GET bar
(nil)
```

3.9. FLUSHALL

Delete all the keys of all the existing databases, not just the currently selected one. This command never fails.

```
127.0.0.1:6379> FLUSHALL
OK
127.0.0.1:6379> GET foo
(nil)
127.0.0.1:6379>
```

3.10. EXPIRE

Set a timeout on key. After the timeout has expired, the key will automatically be deleted. A key with an associated timeout is often said to be volatile in Redis terminology.

3.11. TTL

Returns the remaining time to live of a key that has a timeout. This introspection capability allows a Redis client to check how many seconds a given key will continue to be part of the dataset.

```
127.0.0.1:6379> SET greeting "Hello World!"
OK
127.0.0.1:6379> EXPIRE greeting 50
(integer) 1
127.0.0.1:6379> TTL greeting
(integer) 47
127.0.0.1:6379> TTL greeting
(integer) 43
127.0.0.1:6379> TTL greeting
(integer) 37
127.0.0.1:6379> TTL greeting
(integer) 30
127.0.0.1:6379> TTL greeting
(integer) 30
127.0.0.1:6379> TTL greeting
(integer) 26
127.0.0.1:6379> TTL greeting
(integer) 19
127.0.0.1:6379> TTL greeting
(integer) 3
127.0.0.1:6379> TTL greeting
(integer) -2
127.0.0.1:6379> TTL greeting
(integer) -2
```

3.12. SETEX

Set key to hold the string value and set key to timeout after a given number of seconds. This command is equivalent to executing the following commands:

```
SET mykey value
EXPIRE mykey seconds
```

3.13. PERSIST

Remove the existing timeout on key, turning the key from volatile (a key with an expire set) to persistent (a key that will never expire as no timeout is associated).

```
127.0.0.1:6379> SETEX greeting 30 "Hello World!"
OK
127.0.0.1:6379> TTL greeting
(integer) 26
127.0.0.1:6379> TTL greeting
(integer) 21
127.0.0.1:6379> SETEX greeting 130 "Hello World!"
OK
127.0.0.1:6379> TTL greeting
(integer) 125
127.0.0.1:6379> PERSIST greeting
(integer) 1
127.0.0.1:6379> TTL greeting
(integer) -1
127.0.0.1:6379> GET greeting
"Hello World!"
```

3.14. MSET

Sets the given keys to their respective values. MSET replaces existing values with new values, just as regular SET. See MSETNX if you don’t want to overwrite existing values.

3.15. APPEND

If key already exists and is a string, this command appends the value at the end of the string. If key does not exist it is created and set as an empty string, so APPEND will be similar to SET in this special case.

3.16. RENAME

Renames key to newkey. It returns an error when key does not exist. If newkey already exists it is overwritten, when this happens RENAME executes an implicit DEL operation, so if the deleted key contains a very big value it may cause high latency even if RENAME itself is usually a constant-time operation.

```
127.0.0.1:6379> MSET key1 "Hello" key2 "world"
OK
127.0.0.1:6379> GET key1
"Hello"
127.0.0.1:6379> GET key2
"world"
127.0.0.1:6379> APPEND key1 " world!"
(integer) 12
127.0.0.1:6379> GET key1
"Hello world!"
127.0.0.1:6379> RENAME key1 greeting
OK
127.0.0.1:6379> GET key1
(nil)
127.0.0.1:6379> GET greeting
"Hello world!"
```

Chapter 4. Redis Datatypes

NOTE Redis Datatypes

- Strings
- Lists
- Sets
- Sorted sets
- Hashes
- Bitmaps and HyperLogLogs

4.1. Lists

4.1.1. LPUSH

Insert all the specified values at the head of the list stored at key. If key does not exist, it is created as empty list before performing the push operations.

4.1.2. LRANGE

Returns the specified elements of the list stored at key. The offsets start and stop are zero-based indexes, with 0 being the first element of the list (the head of the list), 1 being the next element and so on.

4.1.3. RPUSH

Insert all the specified values at the tail of the list stored at key. If key does not exist, it is created as empty list before performing the push operation.

```
127.0.0.1:6379> LPUSH people "arafat"
(integer) 1
127.0.0.1:6379> LPUSH people "Jen"
(integer) 2
127.0.0.1:6379> LPUSH people "Tom"
(integer) 3
127.0.0.1:6379> LRANGE people 0 -1
1) "Tom"
2) "Jen"
3) "arafat"
127.0.0.1:6379> LRANGE people 1 2
1) "Jen"
2) "arafat"
127.0.0.1:6379> RPUSH people "Harry"
(integer) 4
127.0.0.1:6379> LRANGE people 0 -1
1) "Tom"
2) "Jen"
3) "arafat"
4) "Harry"
```

4.1.4. LPOP

Removes and returns the first element of the list stored at key.

4.1.5. RPOP

Removes and returns the last element of the list stored at key.

```
127.0.0.1:6379> LRANGE people 0 -1
1) "Tom"
2) "Jen"
3) "arafat"
127.0.0.1:6379> LPOP people
"Tom"
127.0.0.1:6379> LRANGE people 0 -1
1) "Jen"
2) "arafat"
3) "Harry"
127.0.0.1:6379> RPOP people
"Harry"
127.0.0.1:6379> LRANGE people 0 -1
1) "Jen"
2) "arafat"
```

4.1.6. LINSERT

Inserts element in the list stored at key either before or after the reference value pivot.

When key does not exist, it is considered an empty list and no operation is performed.

```
127.0.0.1:6379> LRANGE people 0 -1
1) "Jen"
2) "arafat"
127.0.0.1:6379> LINSERT people BEFORE "arafat" "Tom"
(integer) 3
127.0.0.1:6379> LRANGE people 0 -1
1) "Jen"
2) "Tom"
3) "arafat"
```

4.2. Sets

4.2.1. SADD

Add the specified members to the set stored at key. Specified members that are already a member of this set are ignored. If key does not exist, a new set is created before adding the specified members.

4.2.2. SMEMBERS

Returns all the members of the set value stored at key.

4.2.3. SISMEMBER

Returns if member is a member of the set stored at key.

```
127.0.0.1:6379> SADD cars "Ford"
(integer) 1
127.0.0.1:6379> SADD cars "Honda"
(integer) 1
127.0.0.1:6379> SADD cars "BMW"
(integer) 1
127.0.0.1:6379> SMEMBERS cars
1) "Ford"
2) "BMW"
3) "Honda"
127.0.0.1:6379> SISMEMBER cars "Ford"
(integer) 1
127.0.0.1:6379> SISMEMBER cars "Chevy"
(integer) 0
```

4.2.4. SCARD

Returns the set cardinality (number of elements) of the set stored at key.

4.2.5. SMOVE

Move member from the set at source to the set at destination. This operation is atomic. In every given moment the element will appear to be a member of source or destination for other clients.

4.2.6. SREM

Remove the specified members from the set stored at key. Specified members that are not a member of this set are ignored. If key does not exist, it is treated as an empty set and this command returns 0.

```
127.0.0.1:6379> SMEMBERS cars
1) "Ford"
2) "BMW"
3) "Honda"
127.0.0.1:6379> SCARD cars
(integer) 3
127.0.0.1:6379> SMOVE cars mycars "Ford"
(integer) 1
127.0.0.1:6379> SMEMBERS cars
1) "BMW"
2) "Honda"
127.0.0.1:6379> SMEMBERS mycars
1) "Ford"
127.0.0.1:6379> SREM cars "BMW"
(integer) 1
127.0.0.1:6379> SMEMBERS cars
1) "Honda"
127.0.0.1:6379> FLUSHALL
OK
```

4.3. Sorted sets

4.3.1. ZADD

Adds all the specified members with the specified scores to the sorted set stored at key. It is possible to specify multiple score / member pairs. If a specified member is already a member of the sorted set, the score is updated and the element reinserted at the right position to ensure the correct ordering.

4.3.2. ZRANK

Returns the rank of member in the sorted set stored at key, with the scores ordered from low to high. The rank (or index) is 0-based, which means that the member with the lowest score has rank 0.

4.3.3. ZRANGE

Returns the specified range of elements in the sorted set stored at key.

4.3.4. ZINCRBY

Increments the score of member in the sorted set stored at key by increment. If member does not exist in the sorted set, it is added with increment as its score (as if its previous score was 0.0). If key does not exist, a new sorted set with the specified member as its sole member is created.

```
127.0.0.1:6379> ZADD users 1981 "Arafat Hasan"
(integer) 1
127.0.0.1:6379> ZADD users 1975 "John Doe"
(integer) 1
127.0.0.1:6379> ZADD users 1990 "Mike Smith"
(integer) 1
127.0.0.1:6379> ZADD users 1990 "Kate Rogers"
(integer) 1
127.0.0.1:6379> ZRANK users "Mike Smith"
(integer) 3
127.0.0.1:6379> ZRANK users "John Doe"
(integer) 0
127.0.0.1:6379> ZRANK users "John Do"
(nil)
127.0.0.1:6379> ZRANK users "Arafat Hasan"
(integer) 1
127.0.0.1:6379> ZRANGE users 0 -1
1) "John Doe"
2) "Arafat Hasan"
3) "Kate Rogers"
4) "Mike Smith"
127.0.0.1:6379> ZINCRBY users 1 "John Doe"
"1976"
127.0.0.1:6379> ZINCRBY users 10 "John Doe"
"1986"
127.0.0.1:6379> FLUSHALL
OK
```

4.4. Hash

4.4.1. HSET

Sets field in the hash stored at key to value. If key does not exist, a new key holding a hash is created. If field already exists in the hash, it is overwritten.

4.4.2. HGET

Returns the value associated with field in the hash stored at key.

4.4.3. HGETALL

Returns all fields and values of the hash stored at key. In the returned value, every field name is followed by its value, so the length of the reply is twice the size of the hash.

```
127.0.0.1:6379> HSET user:arafat name "Arafat Hasan"
(integer) 1
127.0.0.1:6379> HSET user:arafat email "arafat@example.com"
(integer) 1
127.0.0.1:6379> HGET user:arafat email
"arafat@example.com"
127.0.0.1:6379> HGETALL user:arafat
1) "name"
2) "Arafat Hasan"
3) "email"
4) "arafat@example.com"
```

4.4.4. HMSET

Sets the specified fields to their respective values in the hash stored at key.

4.4.5. HKEYS

Returns all field names in the hash stored at key.

4.4.6. HVALS

Returns all values in the hash stored at key.

```
127.0.0.1:6379> HMSET user:john name "John Doe" email "doe@example.com" age "25"
OK
127.0.0.1:6379> HGETALL user:john
1) "name"
2) "John Doe"
3) "email"
4) "doe@example.com"
5) "age"
6) "25"
127.0.0.1:6379> HKEYS user:john
1) "name"
2) "email"
3) "age"
127.0.0.1:6379> HVALS user:john
1) "John Doe"
2) "doe@example.com"
3) "25"
```

4.4.7. HINCRBY

Increments the number stored at field in the hash stored at key by increment.

4.4.8. HDEL

Removes the specified fields from the hash stored at key.

4.4.9. HLEN

Returns the number of fields contained in the hash stored at key.

```
127.0.0.1:6379> HINCRBY user:john age 1
(integer) 26
127.0.0.1:6379> HDEL user:john age
(integer) 1
127.0.0.1:6379> HGETALL user:john
1) "name"
2) "John Doe"
3) "email"
4) "doe@example.com"
127.0.0.1:6379> HLEN user:john
(integer) 2
```

Chapter 5. Redis Persistence

NOTE Read Here