REDIS - QUICK GUIDE

REDIS OVERVIEW

Redis is an open source, advanced key-value store and a serious solution for building high-performance, scalable web applications.

Redis has three main peculiarities that set it apart from much of its competition:

- Redis holds its database entirely in memory, using the disk only for persistence.
- Redis has a relatively rich set of data types when compared to many key-value data stores.
- Redis can replicate data to any number of slaves.

Redis Advantages

- **Exceptionally Fast :** Redis is very fast and can perform about 110000 SETs per second, about 81000 GETs per second.
- **Supports Rich data types :** Redis natively supports most of the datatypes that most developers already know like list, set, sorted set, hashes. This makes it very easy to solve a variety of problems because we know which problem can be handled better by which data type.
- **Operations are atomic :** All the Redis operations are atomic, which ensures that if two clients concurrently access Redis server will get the updated value.
- **MultiUtility Tool :** Redis is a multi utility tool and can be used in a number of usecases like caching, messaging-queues *RedisnativelysupportsPublish/Subscribe*, any short lived data in your application like web application sessions, web page hit counts, etc.

REDIS - ENVIRONMENT

To install the Redis on Ubuntu, go to terminal and type the following commands:

```
$sudo apt-get update
$sudo apt-get install redis-server
```

This will install redis on your machine.

Start Redis

```
$redis-server
```

Check if redis is working?

```
$redis-cli
```

This will open a redis prompt, as shown below:

```
redis 127.0.0.1:6379>
```

In the above prompt **127.0.0.1** is your machine's IP address and **6379** is port on which redis server is running. Now type the **PING** command as shown below.

```
redis 127.0.0.1:6379> ping
PONG
```

This shows that you have successfully installed redis on your machine.

Install Redis Desktop Manager on Ubuntu

To install redis dessktop manager on ubuntu, just download the package from http://redisdesktop.com/download Open the downloaded package and install it.

Redis desktop manager will give you UI to manage your redis keys and data.

REDIS - DATA TYPES

Redis supports 5 types of data types, which are described below:

Strings

Redis string is a sequence of bytes. Strings in Redis are binary safe, meaning they have a known length not determined by any special terminating characters, so you can store anything up to 512 megabytes in one string.

Example

```
OK
redis 127.0.0.1:6379> GET name
```

In the above example **SET** and **GET** are redis commands, **name** is the key used in redis and **tutorialspoint** is the string value that is stored in redis.

Hashes

A Redis hash is a collection of key value pairs. Redis Hashes are maps between string fields and string values, so they are used to represent objects

Example

```
redis 127.0.0.1:6379> HMSET user:1 username tutorialspoint password tutorialspoint points
200
OK
redis 127.0.0.1:6379> HGETALL user:1

1) "username"
2) "tutorialspoint"
3) "password"
4) "tutorialspoint"
5) "points"
6) "200"
```

In the above example hash data type is used to store user's object whichh contais basic information of user. Here **HMSET, HEGTALL** are commands for redis while **user:1** is the key.

Lists

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements to a Redis List on the head or on the tail.

Example

```
redis 127.0.0.1:6379> lpush tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> lpush tutoriallist mongodb
(integer) 2
redis 127.0.0.1:6379> lpush tutoriallist rabitmq
(integer) 3
```

```
redis 127.0.0.1:6379> lrange tutoriallist 0 10

1) "rabitmq"
2) "mongodb"
3) "redis"
```

The max length of a list is 232 - 1 elements 4294967295, *morethan4billionofelementsperlist*.

Sets

Redis Sets are an unordered collection of Strings. In redis you can add, remove, and test for existence of members in O1 time complexity.

Example

```
redis 127.0.0.1:6379> sadd tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist mongodb
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist

1) "rabitmq"
2) "mongodb"
3) "redis"
```

NOTE: In the above example rabitmq is added twice but due to unique property of set it is added only once.

The max number of members in a set is 232 - 1 4294967295, *morethan4billionofmembersperset*.

Sorted Sets

Redis Sorted Sets are, similarly to Redis Sets, non repeating collections of Strings. The difference is that every member of a Sorted Set is associated with score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, scores may be repeated.

Example

```
redis 127.0.0.1:6379> zadd tutoriallist 0 redis
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 mongodb
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 0
redis 127.0.0.1:6379> ZRANGEBYSCORE tutoriallist 0 1000

1) "redis"
2) "mongodb"
3) "rabitmq"
```

REDIS - KEYS

Redis keys commands are used for managing keys in redis. Syntax for using redis keys commands is shown below:

Syntax

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

Example

```
redis 127.0.0.1:6379> SET tutorialspoint redis
OK
redis 127.0.0.1:6379> DEL tutorialspoint
(integer) 1
```

In the above example **DEL** is the command, while **tutorialspoint** is the key. If the key is deleted, then output of the command will be ***integer* 1**, otherwise it will be ***integer* 0**

REDIS - STRINGS

Redis strings commands are used for managing string values in redis. Syntax for using redis string commands is shown below:

Syntax

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

Example

```
redis 127.0.0.1:6379> SET tutorialspoint redis
OK
redis 127.0.0.1:6379> GET tutorialspoint
```

In the above example **SET and GET** are the command, while **tutorialspoint** is the key.

REDIS - HASHES

Redis Hashes are maps between string fields and string values, so they are the perfect data type to represent objects

In redis every hash can store up to more than 4 billion field-value pairs.

Example

```
OK
redis 127.0.0.1:6379> HGETALL tutorialspoint

1) "name"
2) "redis tutorial"
3) "description"
4) "redis basic commands for caching"
5) "likes"
6) "20"
7) "visitors"
8) "23000"
```

In the above example we have set redis tutorials detail *name*, *description*, *likes*, *visitors* in hash named **tutorialspoint**

REDIS - LISTS

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements in redis lists

in head or tail of list.

Maximum length of a list is 2^{32} - 1 elements 4294967295, *morethan4billionofelementsperlist*.

Example

```
redis 127.0.0.1:6379> LPUSH tutorials redis
(integer) 1
redis 127.0.0.1:6379> LPUSH tutorials mongodb
(integer) 2
redis 127.0.0.1:6379> LPUSH tutorials mysql
(integer) 3
redis 127.0.0.1:6379> LRANGE tutorials 0 10

1) "mysql"
2) "mongodb"
3) "redis"
```

In the above example three values are inserted in redis list named tutorials by the command **LPUSH**.

REDIS - SETS

Redis Sets are an unordered collection of unique Strings. Unique means sets doesn't allow repition of data in a key.

In redis set add, remove, and test for existence of members in O1 *constanttimeregardlessofthenumberofelementscontainedinsidetheSet*. Maximum length of a list is 2^{32} - 1 elements 4294967295, *morethan4billionofelementsperset*.

Example

```
redis 127.0.0.1:6379> SADD tutorials redis
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mongodb
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mysql
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mysql
(integer) 0
redis 127.0.0.1:6379> SMEMBERS tutorials

1) "mysql"
2) "mongodb"
3) "redis"
```

In the above example three values are inserted in redis set named tutorials by the command **SADD**.

REDIS - SORTED SETS

Redis Sorted Sets are similar to Redis Sets with unique feature of values stored in set. The difference is that every member of a Sorted Set is associated with score, that is used in order to take the sorted set ordered, from the smallest to the greatest score.

In redis sorted set add, remove, and test for existence of members in O1 *constanttimeregardlessofthenumberofelementscontainedinsidetheSet*. Maximum length of a list is 2^{32} - 1 elements 4294967295, *morethan4billionofelementsperset*.

Example

```
redis 127.0.0.1:6379> ZADD tutorials 1 redis
```

```
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 2 mongodb
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 3 mysql
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 3 mysql
(integer) 0
redis 127.0.0.1:6379> ZADD tutorials 4 mysql
(integer) 0
redis 127.0.0.1:6379> ZRANGE tutorials 0 10 WITHSCORES

1) "redis"
2) "1"
3) "mongodb"
4) "2"
5) "mysql"
6) "4"
```

In the above example three values are inserted with its score in redis sorted set named tutorials by the command **ZADD**.

REDIS - HYPERLOGLOG

Redis HyperLogLog is a algorithm that use randomization in order to provide an approximation of the number of unique elements in a set using just a constant, and small, amount of memory.

HyperLogLog provides a very good approximation of the cardinality of a set even using a very small amount of memory around 12 kbytes per key with a standard error of 0.81% and there is no limit to the number of items you can count, unless you approach 2^{64} items.

Example

Following example explains how redis HyperLogLog works:

```
1) (integer) 1


1) (integer) 1


1) (integer) 1

redis 127.0.0.1:6379> PFCOUNT tutorials

(integer) 3
```

REDIS - PUB SUB

Redis pub sub implements the messaging system where senders *inredisterminologycalledpublishers* sends the messages while receivers *subscribers* receive them. The link by which messages are transferred is called channel.

In redis a client can subscribe any number of channels.

Example

Following example explains how publish subscriber concept works. In the below given example one client subscribe a channel named **redisChat**

```
redis 127.0.0.1:6379> SUBSCRIBE redisChat

Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "redisChat"
3) (integer) 1
```

Now, two clients are publishing the messages on same channel named redisChat and above subscribed client is receiving messages.

```
(integer) 1


(integer) 1


1) "message"
2) "redisChat"
3) "Redis is a great caching technique"
1) "message"
2) "redisChat"
3) "Learn redis by tutorials point"
```

REDIS - TRANSACTIONS

Redis transactions allow the execution of a group of commands in a single step. Transactions has two properties in it, which are described below:

- All commands in a transaction are sequentially executed as a single isolated operation. It is not possible that a request issued by another client is served in the middle of the execution of a Redis transaction.
- Redis transaction is also **atomic**. Atomic means either all of the commands or none are processed.

Sample

Redis transaction is initiated by command **MULTI** and then you need to pass list of commands that should be executed in transaction and after that whole transaction is executed by **EXEC** command.

```
redis 127.0.0.1:6379> MULTI
OK
List of commands here
redis 127.0.0.1:6379> EXEC
```

Example

Following example explains how redis transaction can be initiated and executed.

```
redis 127.0.0.1:6379> MULTI
OK
redis 127.0.0.1:6379> SET tutorial redis
QUEUED
redis 127.0.0.1:6379> GET tutorial
QUEUED
redis 127.0.0.1:6379> INCR visitors
QUEUED
redis 127.0.0.1:6379> EXEC
```

```
1) OK
2) "redis"
3) (integer) 1
```

REDIS - SCRIPTING

Redis scripting is used to evaluate scripts using the Lua interpreter. It is built into Redis starting from version 2.6.0. Command used for scripting is **EVAL** command.

Syntax

Basic syntax of **EVAL** command is as follows:

```
redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]
```

Example

Following example explains how redis scripting works:

```
1) "key1"
2) "key2"
3) "first"
4) "second"
```

REDIS - CONNECTION

Redis connection commands are basically used to manage client connections with redis server.

Example

Following example explains how a client authenticate itself to redis server and checks whether server is running or not.

```
OK
redis 127.0.0.1:6379> PING
PONG
```

REDIS - BACKUP

Redis **SAVE** command is used to create back up of current redis database.

Syntax

Basic syntax of redis **SAVE** command is shown below:

```
127.0.0.1:6379> SAVE
```

Example

Following example shows that how to create the back up of current database in redis.

```
127.0.0.1:6379> SAVE

OK
```

This command will create dump.rdb file in your redis directory.

Restore Redis Data

To restore redis data just move redis backup file *dump. rdb* into your redis directory and start the server. To get your redis directory use **CONFIG** command of redis as shown below:

```
127.0.0.1:6379> CONFIG get dir

1) "dir"
2) "/user/tutorialspoint/redis-2.8.13/src"
```

In the output of above command **/user/tutorialspoint/redis-2.8.13/src** is the directory, where redis server is installed.

Bgsave

To create redis backup alternate command **BGSAVE** is also available. This command will start the backup process and run this in background

Example

```
127.0.0.1:6379> BGSAVE

Background saving started
```

REDIS - SECURITY

Redis database can be make secured , so any client making connection needs to authenticate before executing command. To secure redis you need to set the password in config file.

Example

Below given example shows the steps to secure your redis instance.

```
127.0.0.1:6379> CONFIG get requirepass
```

By default this property is blank, means no password is set for this instance. You can change this property by executing the following command

```

OK
127.0.0.1:6379> CONFIG get requirepass
```

After setting the password if any client runs command without authentication, then *error* **NOAUTH Authentication required.** error will return to that. So client needs to use **AUTH** command to authenticate himself.

Syntax

Basic syntax of **AUTH** command is shown below:

```
127.0.0.1:6379> AUTH password
```

REDIS - BENCHMARKS

Redis benchmark is the utility to check the performance of redis by running n commands

simultaneously.

Syntax

Basic syntax of redis benchmark is shown below:

```
redis-benchmark [option] [option value]
```

Example

Below given example check the redis by calling 100000 commands.

```
redis-benchmark -n 100000

PING_INLINE: 141043.72 requests per second
PING_BULK: 142857.14 requests per second
SET: 141442.72 requests per second
GET: 145348.83 requests per second
INCR: 137362.64 requests per second
LPUSH: 145348.83 requests per second
LPOP: 146198.83 requests per second
SADD: 146198.83 requests per second
SPOP: 149253.73 requests per second
LPUSH (needed to benchmark LRANGE): 148588.42 requests per second
LRANGE_100 (first 100 elements): 58411.21 requests per second
LRANGE_300 (first 300 elements): 21195.42 requests per second
LRANGE_500 (first 450 elements): 14539.11 requests per second
LRANGE_600 (first 600 elements): 10504.20 requests per second
MSET (10 keys): 93283.58 requests per second
```

REDIS - CLIENT CONNECTION

Redis accepts clients connections on the configured listening TCP port and on the Unix socket if enabled. When a new client connection is accepted the following operations are performed:

- The client socket is put in non-blocking state since Redis uses multiplexing and non-blocking I/O.
- The TCP_NODELAY option is set in order to ensure that we don't have delays in our connection.
- A readable file event is created so that Redis is able to collect the client queries as soon as new data is available to be read on the socket.

Maximum number of clients

In Redis config *redis. conf* there is property called **maxclients**, which describes maximum number of clients that can connect to redis. Basic syntax of command is:

```
config get maxclients

1) "maxclients"
2) "10000"
```

By default this property is set to 10000 *dependinguponmaximumnumberoffiledescriptorslimitofOS*, although you can change this property.

Example

In below given example we have set maximum number of clients to 100000, while starting the server

```
redis-server --maxclients 100000
```

REDIS - PIPELINING

Redis is a TCP server and supports Request/Response protocol. In redis a request is accomplished with the following steps:

- The client sends a query to the server, and reads from the socket, usually in a blocking way, for the server response.
- The server processes the command and sends the response back to the client.

Meaning of Pipelining

Basic meaning of pipelining is, client can send multiple requests to server without waiting for the replies at all, and finally read the replies in a single step.

Example

To check the redis pipelining, just start the redis instance and type following command in terminal.

```
+PONG
+OK
redis
:1
:2
:3
```

In the above shown example we have check redis connection by using **PING** command, after that we have set a string named **tutorial** with value **redis**, after that get that keys value and increment visitor number three times. In the result we can check that all commands are submitted to redis once and redis is giving output of all commands in single step.

Benefits of pipelining

The benefit of this technique is a drastically improved protocol performance. The speedup gained by pipelining ranges from a factor of five for connections to localhost up to a factor of at least one hundred over slower internet connections.

REDIS - PARTITIONING

Partitioning is the process of splitting your data into multiple Redis instances, so that every instance will only contain a subset of your keys.

Benefits of Partitioning

- It allows for much larger databases, using the sum of the memory of many computers. Without partitioning you are limited to the amount of memory a single computer can support.
- It allows to scale the computational power to multiple cores and multiple computers, and the network bandwidth to multiple computers and network adapters.

Disadvantages of Partitioning

- Operations involving multiple keys are usually not supported. For instance you can't perform the intersection between two sets if they are stored in keys that are mapped to different Redis instances.
- Redis transactions involving multiple keys can not be used.

- The partitioning granuliary is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set.
- When partitioning is used, data handling is more complex, for instance you have to handle multiple RDB / AOF files, and to make a backup of your data you need to aggregate the persistence files from multiple instances and hosts.
- Adding and removing capacity can be complex. For instance Redis Cluster supports mostly transparent rebalancing of data with the ability to add and remove nodes at runtime, but other systems like client side partitioning and proxies don't support this feature. However a technique called Presharding helps in this regard.

Types of Partitioning

There are two types of partitioning available in redis. Suppose we have four redis instances R0, R1, R2, R3 and many keys representing users like user:1, user:2, ... and so forth

Range Partitioning

Range partitioning is accomplished by mapping ranges of objects into specific Redis instances. Suppose in our example users from ID 0 to ID 10000 will go into instance R0, while users form ID 10001 to ID 20000 will go into instance R1 and so forth.

Hash Partitioning

In this type of partitioning, a hash function *eg. modulusfunction* is used to convert key into a number and then data is stored in different- different redis instances.