**REDIS Introduction**

**RE**mote **DI**ctionary **S**erver: Redis is an open-source, in-memory data structure(Key Value) store that can be used as a database, cache, or message broker. It is a NoSQL database.

Key features of Redis include:

1. **In-Memory Storage**: Redis primarily stores data in memory, which enables fast read and write operations. It offers high-performance data access and is well-suited for use cases that require low latency and high throughput.
2. **Data Persistence**: Redis provides options for data persistence, allowing you to **save data to disk** and recover it even after restarting the server. This feature ensures durability and reliability of data.
3. **Rich Data Structures**: Redis supports a wide range of data structures, including strings, hashes, lists, sets, sorted sets, bitmaps, hyper logs, and geospatial indexes. These data structures come with their own set of operations and provide flexibility in modeling and manipulating data.
4. **Pub/Sub Messaging**: Redis supports publish/subscribe messaging, enabling real-time communication between different parts of an application. It allows publishers to send messages to channels, and subscribers to receive messages from those channels.
5. **Distributed Caching**: Redis is commonly used as a cache due to its fast in-memory storage and support for key expiration. It can significantly improve application performance by caching frequently accessed data and reducing the load on the primary data store.
6. **Advanced Functionality**: Redis offers various advanced features such as transactions, Lua scripting, remote server administration, data eviction policies, and more. These capabilities make it a versatile tool for a wide range of use cases.

Redis is known for its simplicity, speed, and versatility, and it is widely used in web applications, real-time analytics, caching layers, job queues, messaging systems, and many other scenarios where fast, scalable, and flexible data storage is required.

**Single Instance Redis Configuration**: A single instance Redis configuration refers to a setup where Redis is deployed as a standalone server. In this configuration, there is only one Redis instance running, typically on a single server or node.

**High Availability (HA) Redis Configuration**: High Availability (HA) Redis configuration is a setup that provides redundancy and fault tolerance by using multiple Redis instances in a clustered or replicated fashion. HA configurations are designed to ensure that Redis remains available even in the event of failures or outages.

There are a few common approaches to implementing HA with Redis:

**Redis Sentinel**: Redis Sentinel is a built-in solution for high availability in Redis. It involves deploying multiple Redis instances, where one acts as a master and the others as slaves. Sentinel nodes monitor the Redis instances and handle automatic failover if the master fails. The Sentinel nodes elect a new master and redirect clients to the new master.

**Redis Cluster**: Redis Cluster is another built-in solution for distributed and highly available Redis deployments. It partitions data across multiple nodes and provides automatic sharding and failover. Redis Cluster supports horizontal scalability and fault tolerance by distributing the data and workload across multiple Redis instances.

In an HA Redis configuration, multiple Redis instances are deployed, and mechanisms are in place to ensure replication, failover, and load balancing. The goal is to provide increased reliability, scalability, and resilience to handle failures and handle high traffic loads.

**Getting Started with Redis**

Navigate to the url - <https://redis.io/>

Then Navigate to Download - <https://redis.io/download/>

Redis is not officially supported on Windows. However, we can install Redis on Windows for development, To install Redis on Windows, we'll first need to enable WSL2 (Windows Subsystem for Linux). WSL2 lets us run Linux binaries natively on Windows. For this method to work, we'll need to be running Windows 10 version 2004 and higher or Windows 11.

The Microsoft Project could be found at - <https://github.com/MicrosoftArchive/redis/>

The binaries and MSI could be found here - <https://github.com/microsoftarchive/redis/releases>

Download the ZIP Redis-x64-3.0.504.zip and extract.

Double Click on the **redis-server.exe** to start the Redis Server

[9540] 12 Jun 10:53:35.191 # Warning: no config file specified, using the default config. In order to specify a config file use C:\Users\a5143522\OneDrive - Renesas Electronics Corporation\Desktop\Software\Redis-x64-3.0.504\redis-server.exe /path/to/redis.conf

\_.\_

\_.-``\_\_ ''-.\_

\_.-`` `. `\_. ''-.\_ Redis 3.0.504 (00000000/0) 64 bit

.-`` .-```. ```\/ \_.,\_ ''-.\_

( ' , .-` | `, ) Running in standalone mode

|`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379

| `-.\_ `.\_ / \_.-' | PID: 9540

`-.\_ `-.\_ `-./ \_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' | http://redis.io

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' |

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

`-.\_ `-.\_\_.-' \_.-'

`-.\_ \_.-'

`-.\_\_.-'

[9540] 12 Jun 10:53:35.197 # Server started, Redis version 3.0.504

[9540] 12 Jun 10:53:35.197 \* The server is now ready to accept connections on port 6379

Double click on **redis-cli.exe** to open redis cli to interact with the redis server.

127.0.0.1:6379> ping

PONG

127.0.0.1:6379> set name Sany

OK

127.0.0.1:6379> get name

"Sany"

127.0.0.1:6379>

To access the redis-server and cli from Windows command prompt, go to Environment Variable for your account, add user variable

Redis-x64-3.0.504

C:\Users\a5143522\Redis

And update path %REDIS\_HOME%\Redis-x64-3.0.504

**Now Open Windows command prompt**

C:\Users\a5143522>redis-server

[29388] 12 Jun 11:12:00.487 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

\_.\_

\_.-``\_\_ ''-.\_

\_.-`` `. `\_. ''-.\_ Redis 3.0.504 (00000000/0) 64 bit

.-`` .-```. ```\/ \_.,\_ ''-.\_

( ' , .-` | `, ) Running in standalone mode

|`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379

| `-.\_ `.\_ / \_.-' | PID: 29388

`-.\_ `-.\_ `-./ \_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' | http://redis.io

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' |

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

`-.\_ `-.\_\_.-' \_.-'

`-.\_ \_.-'

`-.\_\_.-'

[29388] 12 Jun 11:12:00.493 # Server started, Redis version 3.0.504

[29388] 12 Jun 11:12:00.493 \* The server is now ready to accept connections on port 6379

C:\Users\a5143522>redis-cli

127.0.0.1:6379> ping

PONG

127.0.0.1:6379> set num 50

OK

127.0.0.1:6379> get num

"50"

127.0.0.1:6379>

**Redis String command**

A screenshot of a computer

Description automatically generated with low confidence

To see all the keys in the Redis DB

127.0.0.1:6379> keys \*

1) "num"

2) "name2"

127.0.0.1:6379> del num

(integer) 1

127.0.0.1:6379> keys \*

1) "name2"

127.0.0.1:6379> flushall

OK

127.0.0.1:6379> keys \*

(empty list or set)

Key expiry

127.0.0.1:6379> setex name 10 max # set expiry

OK

127.0.0.1:6379> get name

"max"

127.0.0.1:6379> ttl name # time to live

(integer) 1

127.0.0.1:6379> ttl name

(integer) -2

127.0.0.1:6379> get name

(nil)

127.0.0.1:6379> mset num1 10 num2 20 num3 30

OK

127.0.0.1:6379> keys \*

1) "num1"

2) "num3"

3) "num2"

127.0.0.1:6379> mset num1 10 num2 20 num3 30 # Multiple Value set

OK

127.0.0.1:6379>

127.0.0.1:6379> keys \*

1) "num1"

2) "num3"

3) "num2"

127.0.0.1:6379> incrby num1 5

(integer) 15

127.0.0.1:6379> decrby num1 7

(integer) 8

127.0.0.1:6379> set mkey hello

OK

127.0.0.1:6379> get mkey

"hello"

127.0.0.1:6379> append mkey world

(integer) 10

127.0.0.1:6379> get mkey

"helloworld"

127.0.0.1:6379>

**Hashes** are a map between String fields and String values; hence they are perfect data type for Object.

In Redis every Hashes can store up to 4 billion field value pairs.

A picture containing text, screenshot, font, number

Description automatically generated

127.0.0.1:6379> hmset student1 name max age 15 class 8 # has set multiple values

OK

127.0.0.1:6379> hget student1 name

"max"

127.0.0.1:6379> hgetall student1

1) "name"

2) "max"

3) "age"

4) "15"

5) "class"

6) "8"

127.0.0.1:6379> hexists student1 surname

(integer) 0

127.0.0.1:6379> hexists student1 name

(integer) 1

127.0.0.1:6379> hdel student1 class

(integer) 1

127.0.0.1:6379> hgetall student1

1) "name"

2) "max"

3) "age"

4) "15"

127.0.0.1:6379> hmset student1 class 8

OK

127.0.0.1:6379> hgetall student1

1) "name"

2) "max"

3) "age"

4) "15"

5) "class"

6) "8"

127.0.0.1:6379> hsetnx student1 name tom

(integer) 0

127.0.0.1:6379> hsetnx student1 surname muller

(integer) 1

127.0.0.1:6379> hgetall student1

1) "name"

2) "max"

3) "age"

4) "15"

5) "class"

6) "8"

7) "surname"

8) "muller"

127.0.0.1:6379> hkeys student1

1) "name"

2) "age"

3) "class"

4) "surname"

127.0.0.1:6379> hincrby student1 age 5

(integer) 20

127.0.0.1:6379> hget student1 age

"20"

127.0.0.1:6379> hmget student1 name age

1) "max"

2) "20"

127.0.0.1:6379>

Redis **List** are simple list of strings sorted by their insertion order.

A list would have head on top and tail on bottom. An element can be inserted from top or bottom.

If we see a list in the horizontal way then the head will be on left hand side and tail on the right-hand side, hence comes the command as **lpush** – left push or head push. **rpush** – right push or bottom push.

127.0.0.1:6379> redis-cli monitor # Open a duplicate session and execute this command to monitor the timestamp of all the events executed.

A picture containing text, screenshot, font, diagram

Description automatically generated

127.0.0.1:6379> lpush num 1 2 3 4

(integer) 4

127.0.0.1:6379> lrange num 0 10 # iterates the list from 0th index to 10th index

1) "4"

2) "3"

3) "2"

4) "1"

127.0.0.1:6379> lpop num

"4"

127.0.0.1:6379> lrange num 0 10

1) "3"

2) "2"

3) "1"

127.0.0.1:6379> rpush num 4

(integer) 4

127.0.0.1:6379> lrange num 0 10

1) "3"

2) "2"

3) "1"

4) "4"

127.0.0.1:6379> rpop num

"4"

127.0.0.1:6379> lrange num 0 10

1) "3"

2) "2"

3) "1"

127.0.0.1:6379> lindex num 1

"2"

127.0.0.1:6379> lset num 0 7 # Store the number 7 at the 0th index.

OK

127.0.0.1:6379> lrange num 0 10

1) "7"

2) "2"

3) "1"

127.0.0.1:6379> lrange num 0 -1 # list all the values in the list

1) "7"

2) "2"

3) "1"

127.0.0.1:6379> lpushx num 9 # push data into list if the key num exists

(integer) 4

127.0.0.1:6379> lrange num 0 -1

1) "9"

2) "7"

3) "2"

4) "1"

127.0.0.1:6379> linsert num before 2 55

(integer) 5

127.0.0.1:6379> lrange num 0 -1

1) "9"

2) "7"

3) "55"

4) "2"

5) "1"

127.0.0.1:6379>

**Redis sets** are unordered collection of unique strings.

A picture containing text, screenshot, font, design

Description automatically generated

**Redis Publish Subscribe**

Messaging system where the sender is the publisher and receiver are the subscriber.

There can be multiple subscribers to the same channel.

127.0.0.1:6379> **subscribe** redis

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "redis"

3) (integer) 1

127.0.0.1:6379> **publish** redis hi

(integer) 1

127.0.0.1:6379>

A screenshot of a computer screen

Description automatically generated with medium confidence

127.0.0.1:6379> **PSUBSCRIBE** \*

Reading messages... (press Ctrl-C to quit)

1) "psubscribe"

2) "\*"

3) (integer) 1

**PSUBSCRIBE takes a pattern which** basically means any publisher matching the pattern. Example \* means subscribe to all the channels on the Redis Server connection, R\* means any subscribe to any channel whose name starts with R.