REDIS

Redis es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes pero que opcionalmente puede ser usada como una base de datos durable o persistente.

Instalación

El sudo no es necesario si están trabajando en un contenedor docker, ya que en ese caso serían superusuarios por defecto.

```
$ sudo apt-get update
$ sudo apt-get install build-essential wget

$ wget download.redis.io/releases/redis-stable.tar.gz
$ tar -xzf redis-stable.tar.gz
$ cd redis-stable/

$ make
$ sudo make install
```

Servidor-Cliente

```
$ redis-server
```

```
$ redis-cli
```

Instalación

```
$ docker run redis
```

Y gracias a la magia de docker, eso logra que redis se ejecute y levante un servidor. Aunque tal y como está ahora, no es tan útil ya que no nos podemos conectar directamente.

```
$ docker run --name contenedor_redis redis
$ docker exec -it contenedor_redis redis-cli
```

Revisar conexión

redis> PING

CRUD

redis> SET llave valor

redis> GET llave

redis> DEL llave

redis> MSET llave1 valor1 llave2 valor2 ... ...

redis> MGET llave1 llave2 ...

Transactions

Varias operaciones que queremos que se ejecuten juntas sin que nada pueda ejecutarse de forma intermedia.

```
redis> MULTI
...> SET ...
...> SET ...
...> SET ...
...> EXEC
```

Transactions

WATCH se puede utilizar para mantener una o más llaves vigiladas. Si alguna de las llaves cambia antes de que se ejecute el bloque MULTI, no se realiza la transacción.

```
redis> WATCH llave
redis> MULTI
...> SET ...
...> SET ...
...> SET ...
...> EXEC
```

Hashes

Pares llave-valor dentro de las pares llave-valor. En lugar de algo como:

```
redis> MSET hugo:nombre "Hugo Pato" hugo:color "rojo"
```

Podemos juntarlo en un solo objeto:

```
redis> HSET usuario:hugo nombre "Hugo Pato"
redis> HSET usuario:hugo color "rojo"
```

O bien:

```
redis> HMSET usuario:hugo nombre "Hugo Pato" color "rojo"
```

Hashes

Para recuperarlas, puede ser directamente por el dato completo:

redis> HGET usuario:hugo color

Podemos también obtener todos los valores:

redis> HVALS usuario:hugo

todos los campos:

redis> HKEYS usuario:hugo

o bien, todo:

redis> HGETALL usuario:hugo

Listas

REDIS permite trabajar con una estructura de datos ordenada: la lista.

redis> RPUSH donald:sobrinos paco luis
redis> LPUSH donald:sobrinos hugo

Podemos obtener la longitud de las listas:

redis> LLEN donald:sobrinos

Para recuperarlas, necesitamos indicar el rango que queremos recuperar, los índices en REDIS van desde 0. Un índice negativo significa que la cuenta comienza desde el final:

redis> LRANGE donald:sobrinos 0 -1

LREM permite eliminar un elemento de una lista, recibe un número, para saber cuántas copias debe eliminar, si se usa 0, se eliminan todas, Números negativos hacen que la cuenta sea de derecha a izquierda.

redis> LREM donald:sobrinos 0 paco

POPs:

redis> LPOP donald:sobrinos

redis> RPOP donald:sobrinos

Lamentablemente, la CLI no soporta que se guarde el valor que se obtiene, para recuperarlo y usarlo de nuevo se necesita ya entrar a un programa.

Lo que si se puede hacer es sacar el valor de una lista para guardarlo en otra.

redis> RPOPLPUSH lista:origen lista:destino

Esta es la única opción que se tiene, no existen las otras combinaciones de los lados para sacar e ingresar datos de las listas.

Conjuntos

```
redis> SADD rojo manzana hugo "vino tinto" fresa
redis> SMEMBERS rojo

redis> SADD frutas manzana uva pera fresa

redis> SINTER rojo frutas
redis> SDIFF rojo frutas
redis> SUNION rojo frutas

redis> SINTERSTORE rojo_y_fruta rojo frutas
redis> SDIFFSTORE rojo_nofruta rojo frutas
redis> SUNIONSTORE rojo_o_fruta rojo frutas
```

Conjuntos
Modificaciones

SREM permite eliminar elementos de un conjunto, no recibe número.

redis> SREM rojo hugo "vino tinto"

SMOVE permite hacer movimientos de un conjunto a otro.

redis> SMOVE conjunto:origen conjunto:destino valor

SCARD (de cardinalidad) es el equivalente de LLEN para listas.

redis> SCARD conjunto

SPOP saca un valor al azar del conjunto.

redis> SPOP conjunto

Conjuntos ordenados

Son estructuras de datos que se comportan como conjuntos ya que no permiten valores repetidos. Pero cada uno de los valores tiene un *score* y en el conjunto los valores están ordenados según dicho valor.

```
redis> ZADD calificacion 8 hugo 10 paco 6 luis
redis> ZINCRBY calificacion 1 luis

redis> ZRANGE calificacion 0 -1
redis> ZREVRANGE calificacion 0 -1 WITHSCORES
redis> ZRANGEBYSCORE calificacion (8 10
redis> ZREMRANGEBYRANK calificacion 0 1
```

Conjuntos ordenados

Los conjuntos ordenados se pueden combinar en un conjunto ordenado de salida. Se tienen varias opciones para mezclar los *scores*.

redis> ZUNIONSTORE conjunto:destino numero:conjuntos conjunto1 [conjunto2 . . .] WEIGHTS peso1 [peso2 . . .] AGGREGATE SUM|MIN|MAX

Expiración

REDIS es comúnmente usado como herramienta temporal de rápido acceso. Muchas veces se requiere que los datos no duren mucho tiempo en memoria y se mantenga en constante liberación.

redis> SET hielo "me derrito..."
redis> EXPIRE hielo 10
redis> EXISTS hielo

Agregar un valor con un tiempo de expiración es bastante común, y tiene su propia instrucción para hacer las dos cosas a la vez.

redis> SETEX hielo 10 "me derrito..."

TTL permite saber cuánto tiempo le queda a un objeto. y PERSIST permite quitar una expiración antes de que se cumpla.

Bases de datos

REDIS cuenta con mas de una base de datos, las cuales se encuentran enumeradas. Por defecto se comienza con la 0, pero se puede cambiar en cualquier momento.

redis> SELECT 1

redis> KEYS *

Comunicacion entre clientes

Diferentes clientes pueden enviar y recibir mensajes entre ellos, hay dos maneras:

- Un cliente que recibe datos a través de una lista por un determinado tiempo

```
redis1> BRPOP lista 0
redis2> LPUSH lista "valor a mandar"
```

- Un cliente manda un mensaje a todos los que se suscriban.

```
redis1> SUBSCRIBE mensajes
redis2> SUBSCRIBE mensajes
redis3> PUBLISH mensajes "valor a mandar"
```

Durabilidad
Configuración

Si no se le pone configuración, REDIS mantiene todo en memoria, es la opción más rápida y solo se guarda si se indica.
redis> SAVE

Por defecto se tienen configuradas tres opciones de guardado en disco en un formato tiempo-minimos_cambios:
save 900 1
save 300 10
save 60 10000

Otra opción es utilizar el *append only file*. Un registro de recuperación de todas las instrucciones que se mandan a la base de datos.
appendonly yes appendfsync everysec|always|no

Para obtener una configuración de replicación entre maestro y replicas, basta con cambiar los puertos y señalar el maestro que se va a replicar. Por defecto 6379.

```
port 6380

replicaof 127.0.0.1 6379
```

Al usar archivos de configuración, es importante además percatarse de dos parámetros que se tienen por defecto: La IP de entrada, y el *protected-mode* si estamos usando docker.

```
# bind 127.0.0.1

protected-mode no
```

Una instancia de REDIS no puede agregarse a un cluster sin la configuración apropiada.

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

El archivo de *cluster-config-file* no se debe editar a mano.
Esta configuración se debe replicar para cada una de las instancias del cluster. Una copia de configuración, cada una en una carpeta diferente y con puertos distintos.

Se tiene que iniciar cada una de las instancias, esto generará un ID para cada nodo y los dejará listos para conectarse, hay que crear por lo menos 6. 3 son el mínimo número de nodos de un cluster, los otros 3 serán réplicas.

```
$ cd redis-7000
$ redis-server redis-7000.conf

* No cluster configuration found, I'm 09886a7e187ebbd2495ccaba348a62ccf215ccbe
```

Ya con las instancias de REDIS en ejecución, se puede armar el cluster:

```
$ redis-cli --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 --cluster-replicas 1
```

Al momento de conectarse desde el CLI, se debe utilizar la opción *-c* para establecer que se conecta a un cluster.

```
$ redis-cli -c -p 7000
```

Actividades

- Desarrollar un sistema de reducción de URLs
- El sistema debe soportar el manejo de usuarios
- La reducción de URLs debe ser por usuario y permitir que se tengan ligas publicas y ligas privadas (solo distinguirlas)
- Cada usuario debe tener una *wishlist* de los sitios que desean visitar en el futuro
- Al igual que la posibilidad de categorizar las URLs

- Para el punto anterior, como administradores deben de ser capaces de hacer consultas sobre las intersecciones entre las categorías de dos usuarios.