Key-value stores
Redis

http://redis.io

The Key-value Abstraction

- (Business) Key → Value
- (twitter.com) tweet id → information about tweet
- (amazon.com) item number → information about it
- (kayak.com) Flight number → information about flight, e.g., availability
- (yourbank.com) Account number → information about it

The Key-value Abstraction (2)

- It’s a dictionary data structure.
  - Insert, lookup, and delete by key
  - E.g., hash table, binary tree
- But distributed.
- Sound familiar?
  - Distributed Hash tables (DHT) in P2P systems

- It’s not surprising that key-value stores reuse many techniques from DHTs.

Key Value Stores

- Key-Valued data model
  - Key is the unique identifier
  - Key is the granularity for consistent access
  - Value can be structured or unstructured
- Gained widespread popularity
  - In house: **Bigtable** (Google), **PNUTS** (Yahoo!), **Dynamo** (Amazon)
  - Open source: **Redis**, **HBase, Hypertable, Cassandra, Voldemort**
- Popular choice for the modern breed of web-applications

Important Design Goals

- **Scale out: designed for scale**
  - Commodity hardware
  - Low latency updates
  - Sustain high update/insert throughput
- **Elasticity – scale up and down with load**
- **High availability – downtime implies lost revenue**
  - Replication (with multi-mastering)
  - Geographic replication
  - Automated failure recovery

Lower Priorities

- **No Complex querying functionality**
  - No support for SQL
  - CRUD operations through database specific API
- **No support for joins**
  - Materialize simple join results in the relevant row
  - Give up normalization of data?
- **No support for transactions**
  - Most data stores support single row transactions
  - Tunable consistency and availability
- ***Avoid scalability bottlenecks at large scale***

System Interface

- Two basic operations:
  - Get(key):
  - Put(key, value)

Redis

- Redis is an open source, advanced **key-value data store**
- Often referred to as a **data structure server** since keys can contain strings, hashes, lists, sets and sorted sets
- The name Redis means Remote Dictionary Server
- Redis works with an **in-memory** dataset
- It is possible to **persist** dataset either by
  - dumping the dataset to disk every once in a while
  - or by appending each command to a log

Who is using Redis?

- Twitter
- GitHub
- Weibo
- Pinterest
- Snapchat
- Craigslist
- Digg
- StackOverflow
- Flickr

I DON'T UNDERSTAND THIS WHOLE REDIS THING.
WHY NOT JUST USE AN RDBMS WITH A TWO-COLUMN TABLE?
FOR THE SAME REASON YOU DON'T DRIVE A TANK TO WORK!
BECAUSE IT'S OUT OF MY PRICE-RANGE?
©2011 ERIC REDMOND
CRUDCOMIC.COM

Configuration

- Configuration file: **`/redis/redis.conf`**

- It is possible to change a port (if you wish):

```
port 6379
```

- For development environment it is useful to change data persisting policy

```
save 900 1
save 300 10
save 60 10000
```

→

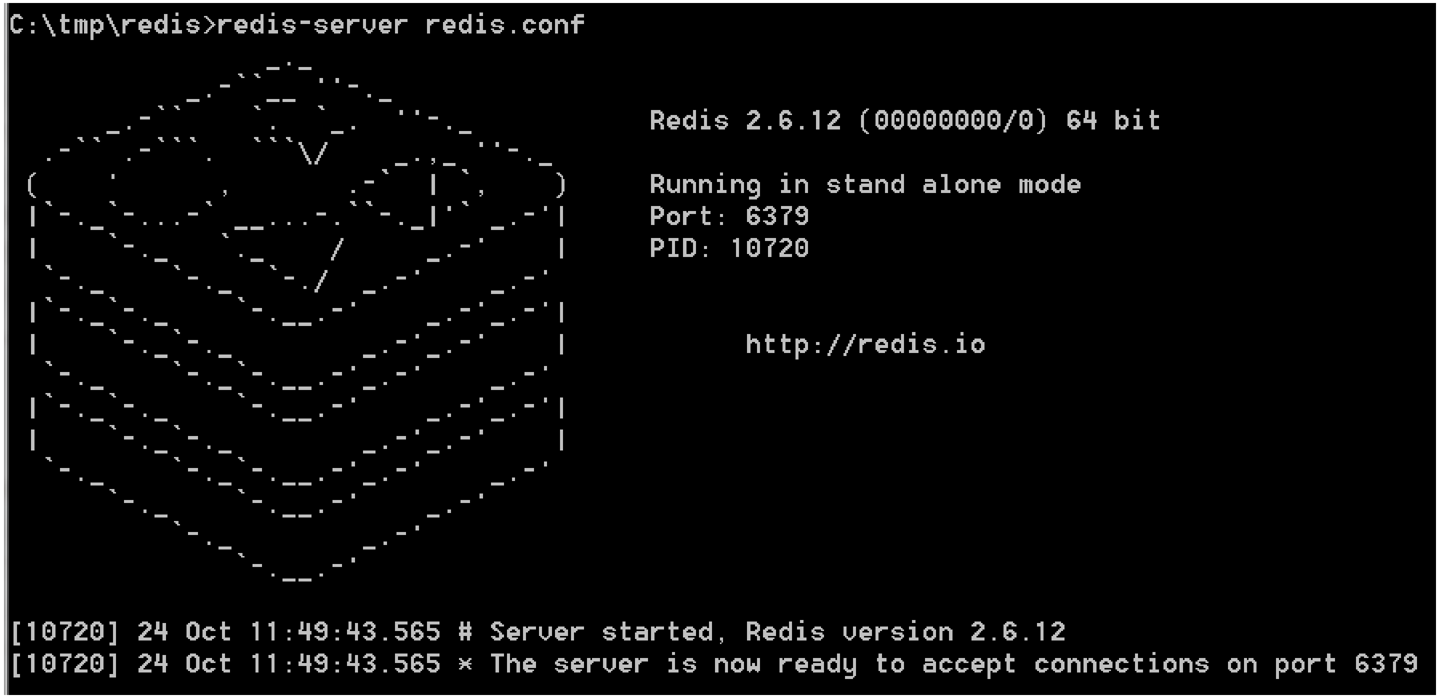
```
save 10 1
```

save after 10 sec if at least 1 key changed

Running Redis Server

- Run **`/redis/bin/redis-server.exe`** and specify configuration file to use

```
redis>redis-server redis.conf
```



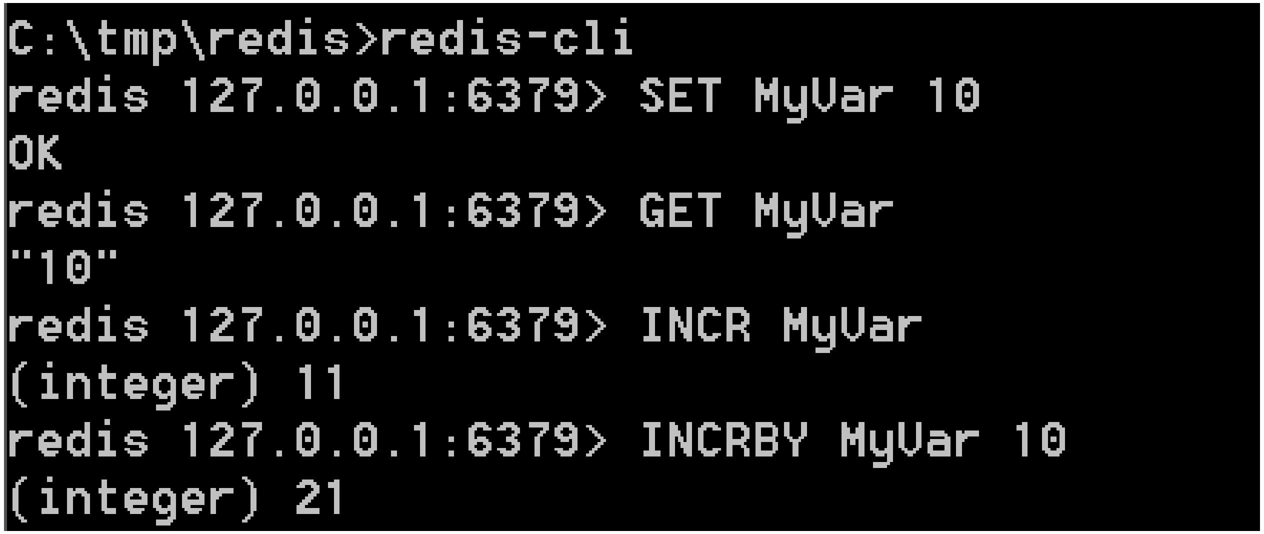
```
C:\tmp\redis>redis-server redis.conf
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 2.6.12 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in stand alone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 10720
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

[10720] 24 Oct 11:49:43.565 # Server started, Redis version 2.6.12
[10720] 24 Oct 11:49:43.565 * The server is now ready to accept connections on port 6379
```

Running Redis Client

- Run **`redis-cli.exe`**

- Now we can play with Redis a little bit



```
C:\tmp\redis>redis-cli
redis 127.0.0.1:6379> SET MyVar 10
OK
redis 127.0.0.1:6379> GET MyVar
"10"
redis 127.0.0.1:6379> INCR MyVar
(integer) 11
redis 127.0.0.1:6379> INCRBY MyVar 10
(integer) 21
```

Useful Commands

- Print all keys:

```
KEYS  *
```

- Remove all keys from all databases

```
FLUSHALL
```

- Synchronously save the dataset to disk

```
SAVE
```

Redis keys

- Keys are binary safe - it is possible to use any binary sequence as a key
- The empty string is also a valid key
- Too long keys are not a good idea
- Too short keys are often also not a good idea ("`u:1000:pwd`" versus "`user:1000:password`")
- Nice idea is to use some kind of schema, like: "`object-type:id:field`"

Redis data types

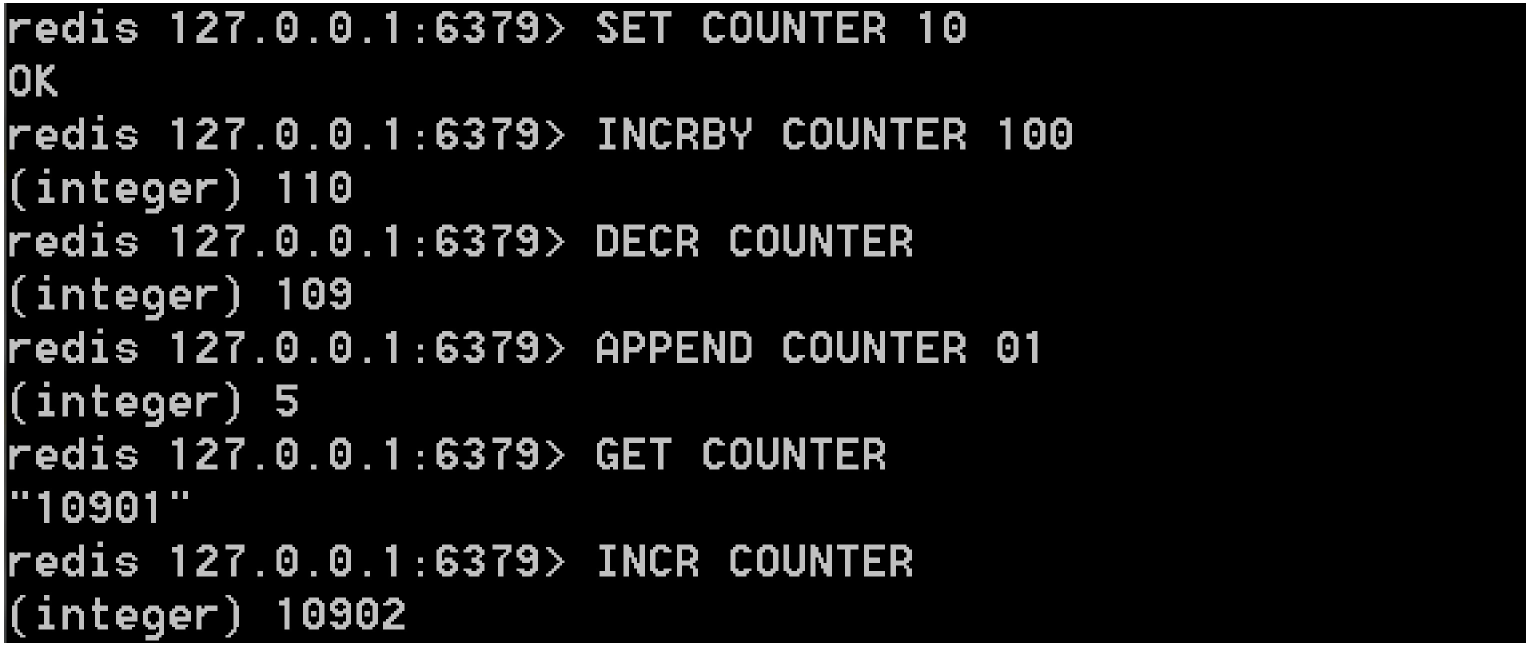
Redis is often referred to as a **data structure server** since keys can contain:

- Strings
- Lists
- Sets
- Hashes
- Sorted Sets

Redis Strings

- Most basic kind of Redis value
- Binary safe - can contain any kind of data, for instance a JPEG image or a serialized Ruby object
- Max 512 Megabytes in length
- Can be used as atomic counters using commands in the INCR family
- Can be appended with the APPEND command

Redis Strings: Example



```
redis 127.0.0.1:6379> SET COUNTER 10
OK
redis 127.0.0.1:6379> INCRBY COUNTER 100
(integer) 110
redis 127.0.0.1:6379> DECR COUNTER
(integer) 109
redis 127.0.0.1:6379> APPEND COUNTER 01
(integer) 5
redis 127.0.0.1:6379> GET COUNTER
"10901"
redis 127.0.0.1:6379> INCR COUNTER
(integer) 10902
```

Transactions

- Redis's MULTI block atomic commands are a similar concept to transactions. Wrapping two operations like SET and INCR in a single block will complete either successfully or not at all.
- We begin the transaction with the MULTI command and execute it with EXEC (rollback with DISCARD).

```
redis 127.0.0.1:6379> MULTI
redis 127.0.0.1:6379> SET foo bar
redis 127.0.0.1:6379> INCR counter
redis 127.0.0.1:6379> EXEC
```

Redis Hashes

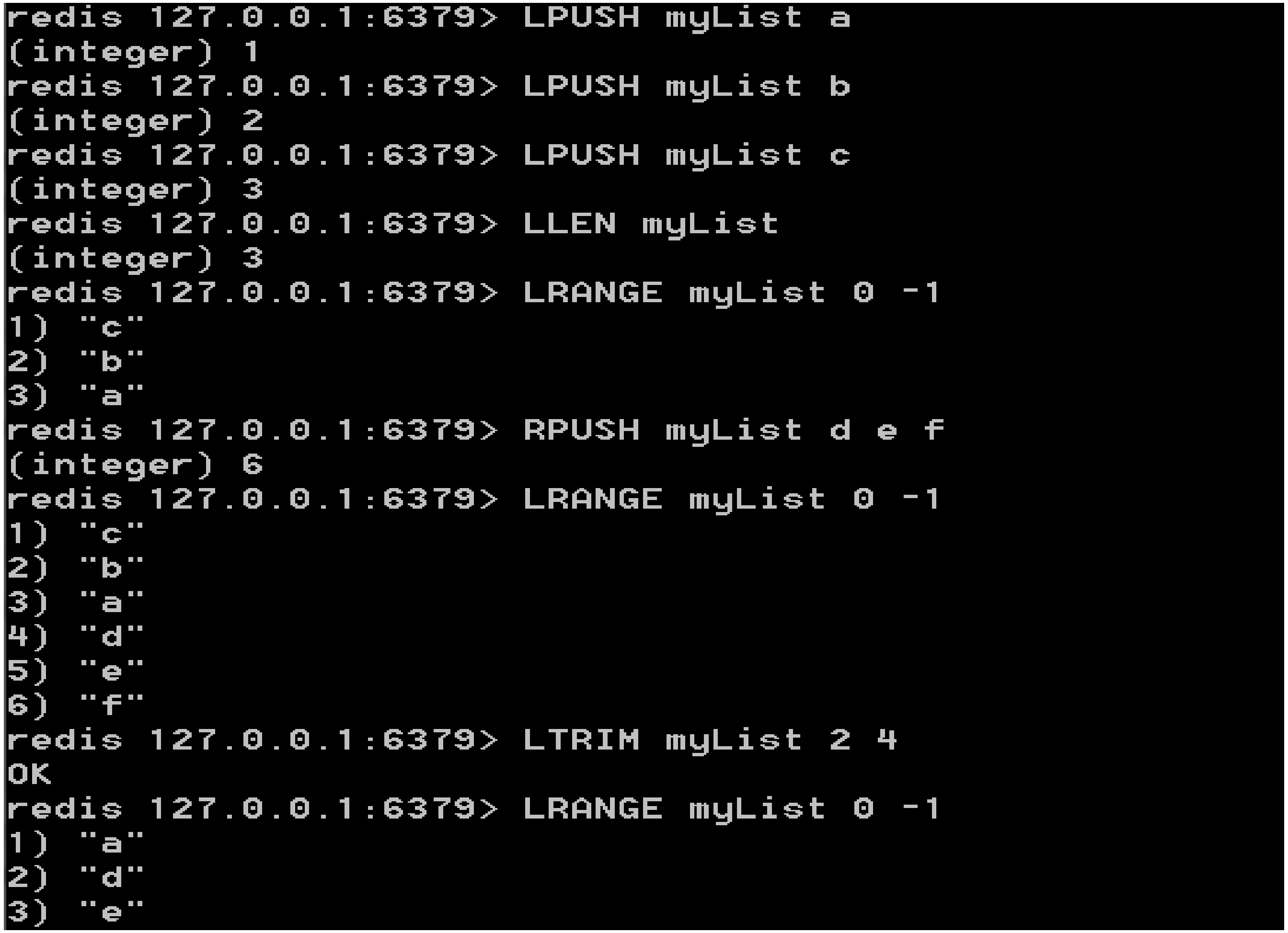
- Redis objects that can take any number of key-value pairs
- Map between string fields and string values
- Perfect data type to represent objects

```
HMSET user:1000 username gomez password P1pp0 age 34
HGETALL user:1000
HVALS user:1000
HKEYS user:1000
HSET user:1000 password 12345
HGETALL user:1000
HGET user:1000 username
```

Redis Lists

- Lists of ordered values (insertion order)
- Can act as queues or stacks (or just lists)
- Add elements to a Redis List pushing new elements on the head (on the left) or on the tail (on the right) of the list
- Max length: (2^32 - 1) elements
- Model a timeline in a social network, using LPUSH to add new elements, and using LRANGE in order to retrieve recent items
- Use LPUSH together with LTRIM to create a list that never exceeds a given number of elements

Redis Lists: Example



```
redis 127.0.0.1:6379> LPUSH myList a
(integer) 1
redis 127.0.0.1:6379> LPUSH myList b
(integer) 2
redis 127.0.0.1:6379> LPUSH myList c
(integer) 3
redis 127.0.0.1:6379> LLEN myList
(integer) 3
redis 127.0.0.1:6379> LRANGE myList 0 -1
1) "c"
2) "b"
3) "a"
redis 127.0.0.1:6379> RPUSH myList d e f
(integer) 6
redis 127.0.0.1:6379> LRANGE myList 0 -1
1) "c"
2) "b"
3) "a"
4) "d"
5) "e"
6) "f"
redis 127.0.0.1:6379> LTRIM myList 2 4
OK
redis 127.0.0.1:6379> LRANGE myList 0 -1
1) "a"
2) "d"
3) "e"
```

More list functions

- LREM removes from the list given value

```
LREM myList 0 a
```

- LPOP removes from the left (head) of the list

```
LPOP myList
```

- RPUSH/RPOP add/remove from the right of the list
- RPOPLPUSH pop a value from the tail of one list and push it to the head of another

```
RPOPLPUSH myList yourList
```

Blocking lists

- Producer-consumer example.
- Open another redis client, one client (the consumer) just listens for new comments and pop them as they arrive.

```
BRPOP comments 300
```

- The command will block until a value exists to pop. Timeout in seconds is set to five minutes.
- Now the producer should push a message to comments.

```
LPUSH comments “ModernDB is a great class!"
```

- Switch back to the consumer console, two lines will be returned: the key and the popped value. The console will also output the length of time it spent blocking.

Sets

- Unordered collections with no duplicate values, supports unions and intersections

```
SADD myPref movies reading walking
```

- SMEMBERS retrieves the whole set.

```
SMEMBERS myPref
SADD yourPref running painting reading fishing
```

- To find the intersection use the SINTER command.

```
SINTER myPref yourPref
```

- Remove any matching values in one set from another:

```
SDIFF myPref yourPref
```

- Union is a set, any duplicates are dropped.

```
SUNION myPref yourPref
```

- That set of values can also be stored directly into a new set:

```
SUNIONSTORE hobbies myPref yourPref
```

Redis Sorted Sets

- 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

```
ZADD scoreboard 500 me 9 you 15 him
```

- To increment a score, we can either re-add it with the new score, which just updates the score but does not add a new value, or increment by some number, which will return the new value.

```
ZINCRBY scoreboard 1 you
```

Sorted Set Ranges

- To get scores from the sorted set:

```
ZRANGE scoreboard 0 1
```

- To get scores append WITHSCORES
- ZREVRANGE gets them in reverse
- ZRANGEBYSCORE allow to provide score range (inclusive by default)

```
ZRANGEBYSCORE scoreboard 100 500
ZRANGEBYSCORE scoreboard (100 500
ZRANGEBYSCORE scoreboard (100 inf
```

Sorted Set Unions

```
ZUNIONSTORE destination numkeys key [key ...]
   [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]
```

- destination is the key to store into
- numkeys is simply the number of keys you're about to join
- key is one or more keys to union
- weight [optional] is the number to multiply each score of the relative key by (if you have two keys, you can have two weights, and so on).
- aggregate is the optional rule, sum is default

For today

- Add a sorted set called scoreboard with the values 50 me, 30 you, 15 her, and 10 him
- Add the gamesWon sorted set for three players: you (3), me (4), and him (8)
- Multiply by 10 the number of wins as the points for each player and add them to the scoreboard (can be done in one or more commands)
- Retrieve the scoreboard in reverse order
- Submit your commands and outputs to ICON