# Redis 2.6

@antirez

vmware®

# Redis 2.6

- Major new features.
- Based on unstable branch (minus the cluster code).

# Why a 2.6 release?

- Redis Cluster is a long term project (*The hurried cat produced blind kittens*).
- Intermediate releases: new features in the hands of developers ASAP.

# Scripting

- Most important feature in this release.
- Changes the game for many applications.
- API is unusual, but ...
- ... practically it is scripts executed server side.

## Scripting 101

- Uses the Lua programming language.
- 99% of commands we would code in C, can now be written as Lua scripts.
- Scripts are atomic like real commands.
- Scripts are fast, minimal overhead.
- Support for JSON and MessagePack.

## What scripting fixes

- Server side manipulation of data.
- Minimizes latency: no RTT payed.
- Maximizes CPU usage: no parsing, syscalls.
- Simpler, faster alternative to WATCH.

## Stored procedures, fixed.

- All the code is client-side.
- We always send full scripts (no commands defined).
- But, we can send SHA1s of scripts!
- Scripting is Redis Cluster friendly.

## Scripting, first example

-
  ```
  EVAL 'redis.call(“SET”,”key”,”somevalue”)' 0
  (nil)
  ```

-
  ```
  GET key
  somevalue
  ```

# Scripting, fixed example

-
  ```
  EVAL 'return redis.call(“SET”,KEYS[1],ARGV[1])' 1 key newvalue
  +OK
  ```

-
  ```
  GET key
  newvalue
  ```
- KEYS / ARGV are cluster friendly.
- Not enforced if you don’t care about cluster.

# Scripting, real example

DECR-IF-GREATER-THAN:

```
local current

current = redis.call('get',KEYS[1])
if not current then return nil end
current = tonumber(current)
if current > tonumber(ARGV[1]) then
    return redis.call('decr',KEYS[1])
else
    return current
end
```

```
EVAL ...body... 1 mykey 10
```

# Scripting using SHA1s

- `EVALSHA 953ed62a3246f2dbd96cdbfc0ec0d92b5cb2f5a8 ...`
- Not defined? `-NOSCRIPT No matching script. Please use EVAL.`
- Defined? The script gets executed.
- No bandwidth wasted nor server-side code.

## Scripting cache ops

- SCRIPT LOAD *loads* a script.
- SCRIPT FLUSH only way to wipe the scripting cache.

## Bit operations

- Redis used to have bit level operations.
- SETBIT: set a bit into a string.
- GETBIT: get a bit from a string.
- Introduced into Redis 2.2.

## Bit operations are awesome.

- They make possible what was impossible.
- Have 100 million users? Store a bit about every user in just 11 megabytes.
- Real time metrics.
- Data mining.

## Bit ops use case

- Million of users in a web app.
- Need to know: who visited app in a specific day.
- At every page view do:
  `SETBIT current_day_key <ID> 1`
- 11M * 365 days = 4GB of RAM.

## But, are we talking 2.2?

- Redis 2.6 adds more indeed ;)
- BITCOUNT for population counting.
- BITOP to AND,OR,XOR and invert bits.

# BITCOUNT key [start end]

- How many users visited the app in a given day? BITCOUNT key_of_this_day.
- With start end: just get ranges, or accumulate sums incrementally.
- BITCOUNT with 11 MB input key: 10 milliseconds.

## BITOP (AND|OR|XOR|NOT) target-key key1 key2 key3 ... keyN

- How many users visited the site day 1 OR day 2?
- BITOP OR temp_key day1_key day2_key
- BITCOUNT temp_key
- Possibilities are infinite!
- BITOP is O(N)! Use a slave if needed.

## Milliseconds EXPIRE

- Much better keys collection algorithm.
- Expires have now millisecond resolution, instead of **one second**.
- New commands to set or inspect expires at millisecond level.
- PEXPIRE, PTTL, PSETEXPIRE, PEXPIREAT.

I DON'T ALWAYS COUNT MILLISECS
BUT WHEN I DO... I USE REDIS
memegenerator.net

# Floats increments

- INCRBYFLOAT and HINCRBYFLOAT commands.
- Reliable output, exponential format never used, same behavior for 32 and 64 bits.
- Reliable replication and AOF: commands translated to SET or HSET.

## Values serialization

- DUMP: turn values into binary blobs.
- RESTORE: restore values into a target instance.
- MIGRATE move keys atomically.
- Speed: 43 milliseconds to dump a 1 million items list (MBA11).

# Better AOF format

- It was like:
  RPUSH mylist a
  RPUSH mylist b
  RPUSH mylist c
- Now it is like:
  RPUSH mylist a b c
- More speed, less space.

## Better ziplists

- Less memory used for:
- Small lists.
- Small hashes.
- Small sorted sets.
- (If very small integers are stored)

# More inside 2.6

- redis-server --test-memory
- Faster with big objects.
- INFO split into sections, with more fields.
- No limit to max number of clients.
- CRC64 checksum in RDB.
- Read only slaves... and a lot more.

# And now the best feature ever.

```
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 2.5.9 (9a8d51ad/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in stand alone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 93034
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'
```

The ASCII logo...

# Status

- We are in freeze.
- Redis 2.6 RC4 will be released in a few days.
- No known bugs currently, but it is new.
- Production? Wait a few more weeks.
- Unless you need features. Many are running 2.6 already.