## Overview

Key-value store

Stores data in memory by default

Values are structured datatypes

Can be persisted to disk, but designed to be used from system memory

## Architecture

Listens on port for connections from clients

Communicates with clients using REDIS protocol over TCP

Redis stores data in memory by default

Very simple

Built for speed

Sacrificing ease-of-use for flexible storage of data

Simple

Requires design effort to make it work for you

Replication

Scalable

Very simple protocol enables many different clients

## Comparison to other NoSQL db’s

Redis supports many datatypes

Persistence is secondary

Indexing must be done manually

Doc db's view persistence as first order concern.

All access is via key

Data can't be queried outside of by key

## Configuration

redis.conf

Via command line

usefule for trying out config options temporarily

Has sections for configuring security, persistence, replication

## 

## Datatypes

strings

ordered list (can be either prepended or appended)

items can be inserted at beginning or end of list

sets

Hash - multiple fields to contain multiple values

sorted set - sorted when values are added to them

Certain commands are specific to specific datatypes

All types can be expired

Strings

Can contain anything < 512mb

serialized objects

images

Allow bit storage and bit manupulation

Numbers

Commands

SET

GET

APPEND

INCR and DECR

GETRANGE

like a substring

MGET

get multiple strings

MSET

set multiple strings

STRLEN

INCRBY

Key design

Structure your keys so they tell you something about the values your storing

Namespaces,

set user:1 {'name' : 'joe', 'email' : 'joe@joe.com'}

Sequences

set user:id 1

incr user:id

getrange

Userful for extracting data from packed data, structured string format (last 3 chars of string are always something)

getrange user:1 3 5

mset user:1 'mike' user:2 'joe'

mget user:1 user:2

strlen user:1

List

Easy to get things from beginning and end of list

Storing last N items

Not built for getting items from middle, or for printing all the contents

you can do it, but it's not fast

Use as a queue or a stack

Stored as linked list internally

Linear complexity

Allows implementation of queues and stacks in the database

lpush

lpush recentcomments "Comment 1"

lpush recentcomments "Comment 2"

Most recently pushed is first in list

rpush

Most recently rpushed is at end

lrem

lset

lindex

lindex recentcomments 3

Don't use except on small lists

lrange

lrange recentcomments 0 1

llen

lpop

Pull first item off list

lpop recentcomments

lpush/lpop is a stack

rpop

lpush/rpop is a queue

ltrim

ltrim recentcomments 0 4

Set

collection of strings

Dups are ignored (1st is kept)

Set operations a quick

Supports many set mathematical ops

Sets don't keep order

Where useful?

How many people have liked a post

sadd

sadd blog:1:likes "Joe" "bob" "mary"

sadd blog:2:likes "mike" "tim"

scard

scard blog:1:likes

how many members

sdiff

sdiff blog:1:likes blog:2:likes

sinter

sunion

sismember

smembers

smembers blog:1:likes

smove

srem

Hash

Nested key/value store

redis is optimized for hashes

Great if hash doesn't contain more than around 100

useful if you want to reference multiple attributes of something

You don't have to pull the entire thing back, just part of it

Use instead of string if you only want a piece of something as opposed of everything

hset

hset user:1:h name "joe"

hmset

hmset user:1:h email "joe@joe.com" id 1

hget

hget user:1:h name

hmget

hmget user:1:h email id

hgetall

hdel

hexists

hincrby

hkeys

kvals

Sorted sets

A set where each member has a score

Fast at adding, deleting

high-scores

zadd

zadd hs 120 joe 100 bob 150 mary 90 tim

zadd hs 125 joe

doesn't get ignored, will update the score

zcard

zcount

zincrby

zrange

zrange hs 0 4

zrange hs 0 4 withscores

zrank

zrank hs bob

shows where bob ranks (0-index ranking)

xrevrank

zrem

zscore

zscore hs bob

returns 150

## Pub/sub

Very simple

subscribe greetings

publish greetings "hello redis"

subscribe greetings errors (multiple channels)

psubscribe greet\*

pattern-based - useful if you have a structured naming scheme for channels

## Transactions

atomic

MULTI

EXEC

MULTI

DISCARD

There's no rollback

If a command fails as part of executing a trx, the commands that preceded that command won't be rolled back

multi starts queuing commands until exec is invoked, or discard

Use WATCH to prevent executing a trx if something changes before exec

## Persistence

Most redis servers are run with persistence enabled

This is to prevent data loss in the event the server crashes

Not needed if using redis as a cache only

2 ways of persisting db

create RDB snapshots

default behavior

can be tweaked in configuration

when certain number of keys have changed within a period of time

Good for backups, but not great for ensuring all data is persisted immediately after inserted (risk if server crashes)

Will lead to data loss if server crashes, most likely

Amount of data loss depends on how often you have the snapshots configured to be created

RDB snapshots are created by a forked process

parent process never writes to disk

restarting from an RDB file is faster than from an AOF log file

append only mode

each command gets saved to disk before the next command is run

The most you can lose is one command worth of data

Definitely comes with a performance penalty

If using append only mode, it's good idea to enable RDB snapshots as well

Adds little overhead and gives you a backup in case redis has a problem restoring from appended logfiles (perhaps a bug)

Both AOF and RDB files are saved in the working directory, specified in the conf file

Configuring persistence

## Replication

Master

Slave

By default, slaves are readonly

Slaves really should be in read-only because if writes are enabled, the master doesn't get updated

A reason to enable writes is if you just want to allow a client to write some ephemeral data which doesn't need to be available to other servers

Usage

CLI

cli has auto-complete help

COMMANDS

flushdb

clears all keys of current database

flushall

clears all keys of all databases

SAVE

synchronously save db to disk

bgsave

asynchronously save db to disk

client list

show all current client connections

config rewrite

generate a configuration file based on the current in-memory configuration

dbsize

list number of keys in current db

info

list server info

lastsave

show last unixtimestamp of last successful save

shutdown

shuts down the server

Enable a password for client connections

requirepass <password>

Authenticate to server

auth <password>

Obscurify commands

rename-command CONFIG sfdawjvel;wajvwavsda

Remove commands

rename-command CONFIG ""

Print help

/usr/local/Cellar/redis/3.0.2 [chef:production] (master) $ redis-server --help

Usage: ./redis-server [/path/to/redis.conf] [options]

./redis-server - (read config from stdin)

./redis-server -v or --version

./redis-server -h or --help

./redis-server --test-memory <megabytes>

Examples:

./redis-server (run the server with default conf)

./redis-server /etc/redis/6379.conf

./redis-server --port 7777

./redis-server --port 7777 --slaveof 127.0.0.1 8888

./redis-server /etc/myredis.conf --loglevel verbose

Sentinel mode:

./redis-server /etc/sentinel.conf --sentinel

connect to redis server

./redis-cli -h [server-host]

start redis-server with given redis config file

./redis-server [redis-config file]

shutdown server

./shutdown

disable savepoints

CONFIG SET SAVE ""

change savepoint configuration

CONFIG SET SAVE "900 1"

save every 900 seconds if at least 1 key has changed

configure instance to be a slave of master

slaveof <masterip> <masterport>

prevent slave from servicing requests if its connection to master is severed

slave-serve-stale-data no

enable writes on a slave

slave-read-only no

start server as slave of master

redis-server --port 7777 --slaveof 127.0.0.1 6379

connect to slave instance

redis-cli -p 7777

Start monitor utility

monitor

Set temporary configuration

config set <foo>

Configur slow-logging for commands taking longer than X

config set slowlog-log-slower-than .000001

Show slow logs

slowlog get

Show last two slowlog entries

slowlog get 2

set

set greeting "hello world"

get

get greeting

del

del greeting

Michaels-Air-2:~ grudkowm$ brew install redis

==> Downloading https://homebrew.bintray.com/bottles/redis-3.0.2.yosemite.bottle.tar.gz

######################################################################## 100.0%

==> Pouring redis-3.0.2.yosemite.bottle.tar.gz

==> Caveats

To have launchd start redis at login:

ln -sfv /usr/local/opt/redis/\*.plist ~/Library/LaunchAgents

Then to load redis now:

launchctl load ~/Library/LaunchAgents/homebrew.mxcl.redis.plist

Or, if you don't want/need launchctl, you can just run:

redis-server /usr/local/etc/redis.conf

==> Summary

🍺 /usr/local/Cellar/redis/3.0.2: 9 files, 892K

Michaels-Air-2:~ grudkowm$

Start redis-server

redis-server

## Security

It's not a very robust security model

Ideally, this won't be a problem if database doesn't need to be exposed

## Monitoring

Allows you to monitor commands executed by clients

For every command processed on the server by auth'd client

shows execution time, remote ip/port, and command

Slow log will log commands taking a long time

## Concepts

Document databases

RavenDB, MongoDB

Docs are persisted to disk and indexed

Indexes can be created ahead of time

no-sql database

Data is conceived completely differently from relational

No explicit concept of relationships as defined in relational db

Relationships can be modeled, but they're not strictly enforced

Often, all data is stored together

relational database

Data is stored in tables that are related to each other via foreign keys

Tables are structured and have unique id's

We're trying to relate rows of data

Reference

redis.io/topic/quickstart

redis admin ui -

Terms

RDB Snapshot

fsync

unix function used to write modified buffer state to persistent storage

RDB

Simple compact snapshot representation of a database at a single point in time

log rewrite

TODO