# Giới thiệu

## Tổng quan

Redis là một phần mềm mã nguồn mở (BSD licensed) lưu trữ dữ liệu toàn bộ trong RAM, được sử dụng như một database, cache, message broker.

Redis cung cấp các cấu trúc dữ liệu như string, hashes, lists, sets, sorted sets với “range queries”, bitmaps, hyperloglogs, geospatial indexes và stream.

Redis có sẵn tính năng “replication”, Lua scripting, LRU cache, transactions và nhiều mức độ “on-disk persistence” và cung cấp High Availability với Redis Sentinel và automatic Partitioning với Redis Cluster

để chạy redis cũng rất đơn giản bạn có thể tải và làm theo hướng dẫn [ở đây](https://redis.io/download), sau đó có thể chạy với lệnh sau

redis-server /path/to/config

## Thị trường

Theo DB Engine Ranking dựa trên kết quả tìm kiếm Google, Stack Overflow thì Redis là database nằm trong

top 4 NoSQL database dựa trên thị trường,

top 1 database được sử dụng trên container,

top 4 datastore được chia sẻ trên stackshare.io,

và nắm giữ danh hiệu database được yêu thích nhất năm 2017, 2018, 2019, 2020 và 2021 theo khảo sát của Stack Overflow Developer Survey

Theo Wikipedia Redis được sử dụng bởi Twitter, Amazon Web Services với dịch vụ Elasticache, Microsoft với Azure Cache, Alibaba với ApsaraDB

Do vậy có thể thấy Redis được sử dụng rất rộng rãi và có những ứng dụng rất quan trọng trong hệ thống nhờ tốc độ cũng như sự đơn giản của mình

## So sánh

Redis được thiết kế để việc đọc, ghi dữ liệu sẽ xảy ra trực tiếp trên RAM, redis cũng lưu dữ liệu trên đĩa nhưng chỉ sử dụng để xây dựng lại dữ liệu vào trong RAM khi restart, do vậy Redis cũng có data model khác hoàn toàn với RDBMS khi truy vấn dữ liệu, thay vì người dùng phải viết câu truy vấn (SQL) sau đó database sẽ phân tích câu truy vấn thành các lệnh của database để truy xuất dữ liệu thì với Redis người dùng sẽ viết trực tiếp lệnh để Redis thực thi trực tiếp

trên bộ dữ liệu do Redis không hỗ trợ vấn đề trên

Ngoài ra Redis đặc biệt sử dụng fork system call để nhân đôi process cùng chứa dữ liệu, nhờ đó process cha có thể tiếp tục phục vụ các client, còn các child process sẽ làm các công việc khác như lưu dữ liệu xuống file đĩa, phục vụ các kết nối replication của slave,...

## Ngôn ngữ

Đến hiện tại 2021 Redis gần như hỗ trợ bất kỳ ngôn ngữ phổ biến lẫn ít phổ biến

C, C++, C#, Dart, Erlang, Go, Haskell, Java, Node.js, Lua, Objective-C, Perl, PHP, Python, R, Ruby, Rust, Scala, Smalltalk, Swift, …

## Các tính năng

* Persistence

chi tiết persistence xem [tại đây](#_4w7h8ptdnp48)

redis lưu toàn bộ dữ liệu trên RAM, nhưng vẫn hỗ trợ lưu trữ lâu dài xuống file như database phổ thông phục vụ cho việc đảm bảo dữ liệu backup and restore cho các nghiệp vụ nhất định

* Replication

chi tiết replication có thể xem [tại đây](#_wbl4v5fpm2fi)

đảm bảo an toàn dữ liệu với nhiều node cùng chứa dữ liệu, hoạt động với mô hình master-slave, leader-follower, master (leader) sẽ sync dữ liệu bằng cách gửi các lệnh tới slave để các slave cập nhật dữ liệu sau đó

tuy nhiên chưa có tự động failover khi master die thì cả cụm trở thành read only do vậy cần [sentinel](#_rhqw0vkpshgd) phía dưới để làm việc trên

* Sentinel

chi tiết sentinel xem [tại đây](#_s00xbwehik5e)

do trên các cụm redis master-slave (leader-follower) của việc sync dữ liệu kể trên không tự động chuyển master khi master die do vậy sentinel được dùng để monitor, tự động failover chuyển đổi master khi có master die, crash, network partition, …

sự kết hợp giữa replication và sentinel là khá an toàn với mọi mô hình hoạt động đơn giản tuy nhiên với nghiệp vụ có nhiều write và muốn cân bằng tải thì replication và sentinel sẽ không đáp ứng được do vậy sẽ cần [cluster](#_baj6jbfqyxw4)

* Cluster

chi tiết cluster xem [tại đây](#_cictjhz5mb7)

clsuter là một mode hoạt động của redis mà các instance redis-server (không có sentinel) có thể trao đổi với nhau nhằm tự động điều phối, load balance các key giữa các node trong cluster, và tự động việc failover, promote một slave của master bị die làm master của partition đó

## Khởi tạo instance

redis-server /etc/redis/redis.conf

trong đó

redis-server là binary từ package redis

/etc/redis/redis.conf là đường dẫn tới config của redis cụ thể các config sẽ được đề cập sau

//TODO

# Persistence

## Tổng quan

(redis persistence là cơ chế cho phép lưu trữ lâu dài)

<https://redis.io/topics/persistence>

<http://antirez.com/post/redis-persistence-demystified.html>

Tuy redis lưu trữ toàn bộ trên RAM, vốn không phải vùng nhớ lâu dài do vậy nếu redis instance stop, shutdown, crash, mất điện thì dữ liệu trên Redis sẽ mất hết, do vậy redis hỗ trợ chúng ta một vài cơ chế “persistent” để đảm bảo “data durability”, lưu trữ lâu dài trên disk, giúp redis có thể khôi phục dữ liệu khi restart, crash,...

các cơ chế redis hỗ trợ:

* [RDB](#_vkpavx1ui2h0) (redis database): redis cung cấp “point-in-time snapshots” của toàn bộ datasets sau một khoảng thời gian nhất định
* [AOF](#_pt7eoz7zkcbr) (appendonly file): aof persistence sẽ “log” lại các lệnh ghi dữ liệu (cập nhật, thêm mới,... ) mà nhận được bởi server, log này sẽ được “replay” lại khi server startup để xây dựng lại bộ dữ liệu, format của file log sử dụng Redis protocol, và nếu file log kích thước lớn redis sẽ tự động rút gọn file lại
* No Persistence: nếu muốn bạn có thể tắt toàn bộ persistence của redis tức data sẽ nằm trên RAM khi tắt Redis sẽ mất hết dữ liệu, việc này trên lý thuyết có cải thiện tốc độ
* [RDB + AOF](#_6psiwga3w4z5): có thể kết hợp RDB và AOF, trong trường hợp này Redis sẽ sử dụng AOF do dữ liệu của AOF sẽ đầy đủ hơn so với RDB

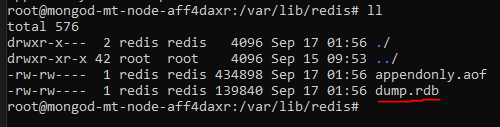
## RDB

Redis Database

<https://redis.io/topics/persistence#rdb-advantages>

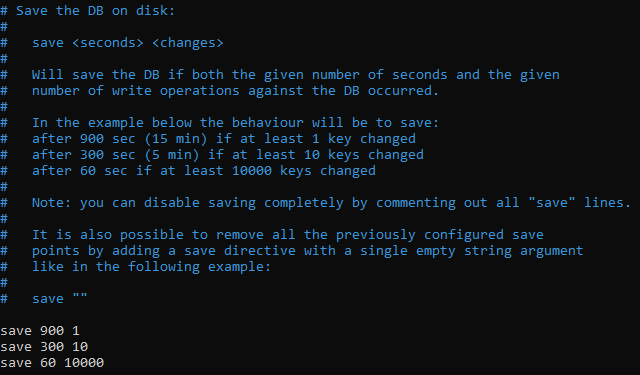
<https://redis.io/topics/persistence#rdb-disadvantages>

redis database cho phép tạo snapshot của toàn bộ dữ liệu vào một file đã được nén file này nằm trên ổ đĩa, sau đó khi restart redis sẽ load dữ liệu từ snapshot để khôi phục dữ liệu tại phiên làm việc trước đó của redis, dưới là hình ảnh file snapshot của redis



snapshot là một file đã được nén nên rất dễ khi trao đổi file giữa các datacenter và các dịch vụ backup như S3

để bật tính năng RDB chúng ta cần cài đặt trong config của redis, ảnh dưới là là một ví dụ config



save 900 1 có ý nghĩa sau 900s nếu có ít nhất 1 key được thay đổi thì việc snapshot sẽ xảy ra

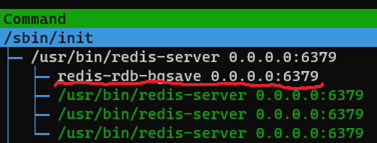
redis cho phép nhiều rules save, khi kiểm tra nếu rule match thì sẽ thực thi snapshot

do vậy ta có thể tạo snapshot theo phút, giờ, ngày, tháng để phục vụ các mục đích backup khác nhau

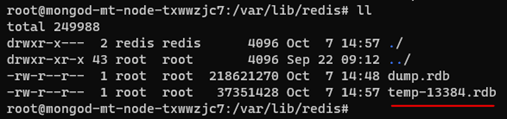
ngoài ra ta có thể tạo snapshot bằng lệnh trực tiếp SAVE hoặc BGSAVE

quá trình snapshot sẽ xảy ra như sau

B1: process cha sẽ fork process con để phục vụ việc tạp snapshot



B2: process con ghi liên tiếp snapshot ra một file tạm temp-xxx.rdb



B3: sau đó đổi tên về file gốc (mặc định dump.rdb)

có thể thấy việc tạo snapshot sẽ không ảnh hưởng tới process cha, process cha có thể phục vụ client mà không bị block

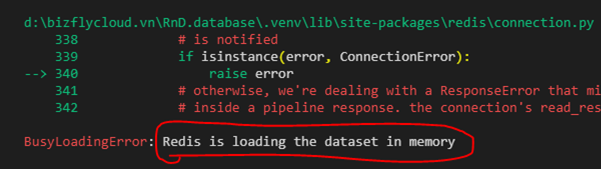
tuy nhiên việc fork với bộ dữ liệu lớn sẽ tốn thời gian và block process cha trong khoảng thời gian có thể tới vài giây nếu dữ liệu lớn

ngoài ra RBD hỗ trợ việc partial resync của slave khi restart hoặc failover

nhược điểm của snapshot là việc dữ liệu có thể mất nếu giữa 2 lần snapshot có sự cố như chập điện, cháy nổ xảy ra thì dữ liệu bị mất là điều có xảy ra

tuy nhiên thực tế với trường hợp reboot server, restart redis, nếu bật RDB thì redis sẽ tạo snapshot trước khi tắt

trong quá trình load dữ liệu từ snapshot nếu client đọc ghi dữ liệu sẽ trả về lỗi, phía dưới là hình ảnh thực tế



Tổng hợp lại các config liên quan tới RDB

save 300 10

bật tính năng snapshot và các rule snapshot interval

dbfilename "dump.rdb"

chỉ định tên file sẽ được lưu xuống disk, mặc định “dump.rdb”

dir "/var/lib/redis"

chỉ định thư mục làm việc, nơi sẽ lưu RDB file và AOF file

rdbcompression yes

nén snapshot khi ghi file ra disk nhằm tiết kiệm không gian lưu trữ và transfer data giữa các server

## AOF

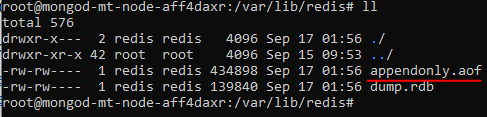
Append Only File

<https://redis.io/topics/persistence#aof-advantages>

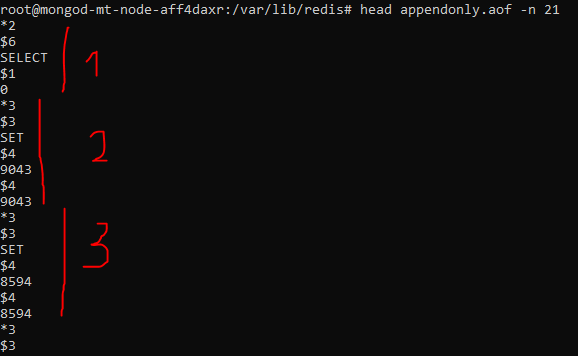
<https://redis.io/topics/persistence#aof-disadvantages>

nếu muốn dữ liệu an toàn hơn nữa Redis hỗ trợ log lại toàn bộ những command tới một redis instance vào một file và sau đó khi khởi động, redis có thể “replay” lại những command mà nó nhận được cho tới thời điểm Redis bị shutdown, crash, …

điều này đảm bảo dữ liệu đầy đủ nhất sau khi redis bị crash, shutdown,… , phía dưới là hình ảnh file log



nội dung của file appendonly.aof chính là một seri các command (log) đã được thực thi



trong hình ảnh trên các lệnh đã được thực thi là

SELECT 0

SET 9043 9043

SET 8594 8594

để bật tính năng AOF sửa appendonly thành yes trong file config



trong quá trình load dữ liệu (replay) từ file AOF, client đọc ghi sẽ báo lỗi

fsync

do AOF sẽ ghi lại command với mỗi lệnh mà redis-server nhận được do vậy việc ghi file xuống đĩa redis cũng hỗ trợ tuỳ chỉnh với các cấp độ khác nhau

appendfsync always với một lần write một lần fsync, việc này sẽ giảm tuổi thọ SSD (theo [redis.com](https://redis.com/))

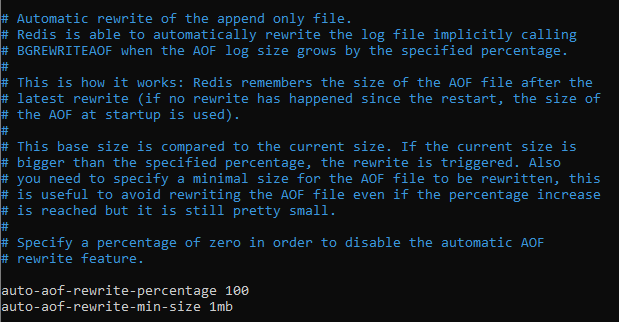
appendfsync everysec cứ sau 1s gọi syscall fsync() một lần

appendfsync no để OS, kernel quyết định (~30s)

vậy để mặc định appendfsync everysec cân bằng tốc độ cũng như độ an toàn

khi file command log lớn dần theo thời gian redis có thể "rewrite" log files ở background để thu gọn file giúp quá trình restart được nhanh hơn, việc này thực hiện song song, luồng chính vẫn tiếp tục append vào log files cũ, khi log files mới sẵn sàng redis "switch" và append vào file log mới, kích thước của file appendonly.aof sau "rewrite" sẽ giảm ít nhiều tuỳ vào độ lớn của file và tính chất của log

để bật tính năng tự động rút gọn file AOF cần sửa trong config như sau



trong đó

auto-aof-rewrite-percentage y khi aof\_current\_size (kích thước file aof) bằng x phần trăm so với aof\_base\_size (size khởi tạo)

size cơ sở - khởi tạo được lấy từ kích thước file lần cuối rewrite nếu không có sẽ lấy kích thước file khi redis khởi động

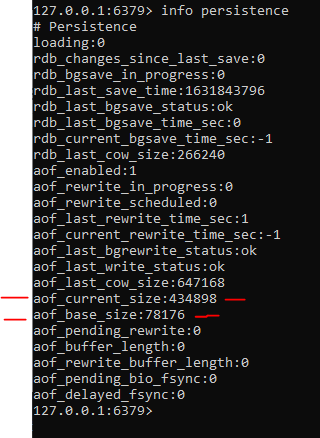
nếu x phần trăm này vượt so với config tức y phần trăm thì việc rewrite sẽ kiểm tra điều kiện phía dưới

sau đó tuỳ kết quả điều kiện sẽ trigger hay không trigger rewrite

VD: base size là 64mb thì với y=100 thì size hiện tại phải trên 64 + 64 \* y = 128mb thì mới trigger việc rewrite

auto-aof-rewrite-min-size z để tránh việc rewrite khi base size nhỏ (không cần thiết rewrite) thì khi current size nhỏ hơn z thì dù việc vượt % kích thước ở trên xảy ra thì redis cũng không trigger rewrite và chỉ khi đáp ứng đủ cả 2 tiêu chí vượt % và vượt min-size thì việc rewrite mới xảy ra

phía dưới là ảnh của current size và base size phục vụ cho AOF



ngoài ra do AOF dạng command log nên sẽ có thể rollback manual bằng cách chỉnh sửa, thêm bớt các lệnh trong file này

xét trường hợp khi xóa nhầm toàn bộ data bằng lệnh FLUSHALL, ta có thể khôi phục bằng cách xoá lệnh FLUSHALL ở cuối file AOF và sau đó khởi động lại redis từ đó mọi dữ liệu sẽ được xây dựng lại tơi thời điểm trước lệnh FLUSHALL

## RDB + AOF

việc restart dữ liệu với RDB nhanh hơn so với restart từ AOF do vậy redis hỗ trợ cơ chế “lai” giữa snapshot và command logs bằng cách

kết hợp phần đầu là snapshot và phần sau là command log, điều này cho phép chỉ replay command từ snapshot gần nhất do vậy sẽ cải thiện tốc độ restart

đề bật tính năng trên cần sửa aof-use-rdb-preamble thành yes trong file config

aof-use-rdb-preamble yes

nếu giá trị của aof-use-rdb-preamble là no thì file AOF sẽ chỉ chứa commands và không chứa snapshot

## Persistence Config

//TODO

## Reference

với nghiệp vụ không quá quan trọng dữ liệu an toàn như cache, thì với RDB - snapshot là quá đủ với mọi nghiệp vụ do kể cả việc bị tắt đột ngột redis vẫn sẽ tạo snapshot trước khi down, ngoài ra việc restart với snapshot nhanh hơn đáng kể so với AOF với cùng kích thước dữ liệu

với nghiệp vụ rất quan trọng về dữ liệu và muốn cải thiện tốc độ restart thì sử dụng RDB + AOF là hợp lý và đảm bảo an toàn dữ liệu cao nhất

trong quá trình load file từ snapshot hoặc AOF thì client đọc ghi dữ liệu sẽ trả về lỗi đang load datasets, việc này sẽ cần phải chú ý khi kích thước của dữ liệu lớn hoặc tốc độ ổ đĩa, CPU chậm

một lưu ý nhỏ trong replication với snapshot sẽ hỗ trợ các slave partial resync, thay vì full resync, còn AOF thì không hỗ trợ partial resync, do vậy trong mô hình replication, master-slave cũng cần chú ý tới điều này

# Replication

## Tổng quan

khi vận hành hệ thống không thể tránh khỏi thất thoát dữ liệu như server crash, delete nhầm volume, cháy server, cháy datacenter,... do vậy nếu như chỉ chạy một instance redis là không đủ, điều này đặt ra bài toán về an toàn dữ liệu khi dữ liệu được lưu ở nhiều nơi thay vì một chỗ, giống như các database phổ thông khác redis cũng hỗ trợ mô hình replicate dữ liệu ra nhiều node từ một node mô hình này chính là master-slave hay leader-follower

nếu chỉ với replication cụm redis master-slave (leader-follower) không có cơ chế tự động failover (chọn ra master mới khi master hiện tại die) do vậy sẽ redis hỗ trợ [sentinel](#_s00xbwehik5e) để tự động hoá cơ chế này

để tạo redis master bạn sẽ không cần chỉnh sửa nhiều, redis mặc định chạy standalone, hay chính là master

redis-server /etc/redis/redis.conf

để tạo một redis slave bạn sẽ cần chỉnh sửa thêm một vài config, sau đó mới chạy instance này

replicaof 192.168.1.1 6379

trong đó 192.168.1.1 là địa chỉ của master, có thể sử dụng hostname và 6379 là port của master sau đó chạy redis

redis-server /etc/redis/redis.conf

chi tiết các config có thể có của redis các bạn có thể xem thêm ở [mục config](#_u47zd8sjsdjv)

## Khởi tạo replication

sau đây là các bước để tạo một cụm master-slave hay leader-follower

tất cả các config các bạn có thể xem thêm [tại đây](#_u47zd8sjsdjv)

B1: khởi tạo master

điều kiện bắt buộc

- đã cài đặt package redis

bước này khá đơn giản do mặc định redis-server khi được chạy sẽ là standalone (master)

redis-server /etc/redis/redis.conf

cụ thể các config xem thêm ở mục [replication config](#_u47zd8sjsdjv)

B2: khởi tạo slaves

điều kiện bắt buộc

- địa chỉ master + port master

- requirepass của master (nếu master bật requirepass)

- master network cho phép slave kết nối tới

- đã cài đặt package redis

cách 1: cấu hình ngay lúc khởi tạo

với mỗi slave cần thêm config replicaof trong file config thì khi startup redis-server sẽ có role là slave

VD: replicaof 192.168.1.1 6379

ngoài ra nếu master có config requirepass tức yêu cầu mật khẩu để kết nối thì chúng ta cũng cần phải sửa config

VD: master-auth “your-master-requirepass”

trùng với requirepass của master để slave có thể sync hợp lệ từ master

redis-server /etc/redis/redis.conf

cách 2: khởi tạo xong sau đó cấu hình trỏ tới master (cách này có thể dùng để chuyển master)

khởi chạy redis-server với mode standalone như bình thường

redis-server /etc/redis/redis.conf

sau đó kết nối tới redis instance này và chạy lệnh

REPLICAOF your-master-ip your-master-port hoặc SLAVEOF với redis < 5.0

sau khi chạy lệnh trên redis instance sẽ chuyển thành slave là thực hiện init sync (full resync) tới master

và phục vụ tiếp như bình thường

B3: kiểm tra kết quả

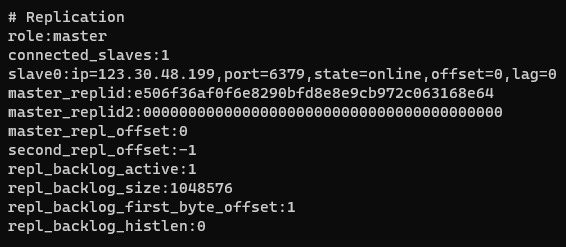
điều kiện bắt buộc

- địa chỉ + port của instance (master hoặc slave)

- requirepass của instance

- network allow kết nối tới instance

kết nối vào master và chạy lệnh INFO REPLICATION để kiểm tra

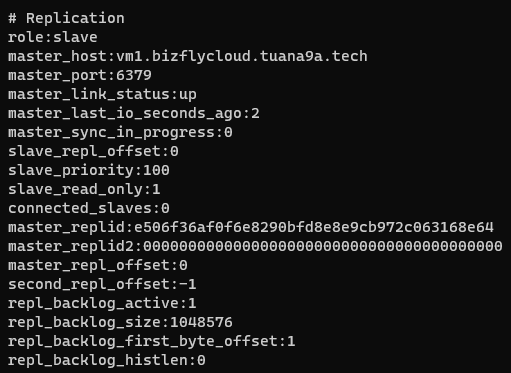


có thể thấy đã có một slave0 kết nối tới master vậy là chúng ta đã thành công

đối với các client từ các ngôn ngữ lệnh kiểm tra cũng sẽ tương tự

VD: client.info(“replication”) trong python, ...

vậy còn phía client thì sao, kết nối vào slave và chạy lệnh INFO để kiểm tra



điều tương tự xảy ra với role giờ là slave và có master tương ứng

## Cơ chế hoạt động

redis master sẽ nhận các write operation và các lệnh write operation chỉ có thể áp dụng trên master, sau đó sẽ gửi những lệnh này (stream command) tới các slave mà đã kết nối tới master, các slave từ đó sẽ lan truyền những lệnh này tới các sub-slave (nếu có) và từ đó các slave sẽ cập nhật dữ liệu ứng với master mà chúng trỏ tới

các slave sẽ chỉ nhận read operation do vậy có thể giảm bớt tải cho master bằng cách đọc dữ liệu từ slave và ghi dữ liệu vào master, từ đó các yêu cầu sẽ được phục vụ bởi cả master và slave do đó cái thiện tốc độ của toàn hệ thống

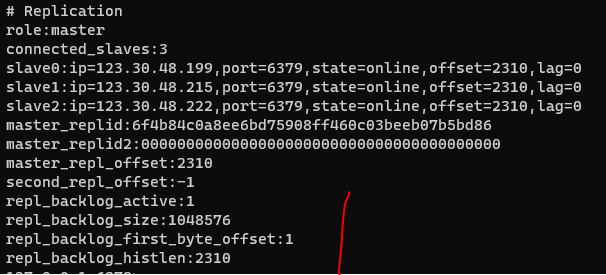
quá trình replication của redis là async tức master sẽ gửi command theo hình thức “fire and forget” mà không quan tâm slave có nhận và thực thi được số command đó không

redis replication hoạt động và kiểm soát việc replicate dữ liệu bằng các chỉ số, tính năng sau

replication-id là id của master thể hiện duy nhất trên một bộ dataset, 2 node chung repl-id phải chung dữ liệu, được sinh ngẫu nhiên khi master restart, slave được promote làm master

offset là số nguyên tăng dần thể hiện tiến độ sync của slave so với master, từ con số này slave liên lục cập nhật offset tới master để sau đó master gửi “stream command” tới slave với những command còn thiếu tính từ offset

backlog là nơi chứa toàn bộ lịch sử các commands mà master nhận được, nhờ có backlog master có thể gửi lại những command đã nhận từ trước đó, tuy nhiên kích cỡ của backlog là có thể điều chỉnh được (mặc định 1mb) và nếu slave có offset nằm ngoài khoảng của backlog (trước backlog, sau backlog) thì master sẽ coi đó là không hợp lệ và master sẽ yêu cầu slave thực hiện full resync



diskless replication là cơ chế mới đang được thử nghiệm tại bản 5.0, tính năng này cho phép master gửi trực tiếp snapshot tới slave thay vì tạo một bản snapshot ở local sau đó mới gửi tới slave, điều này sẽ tăng tốc cho full resync, hạn chế snapshot được ghi xuống disk

đêt bật tính năng diskless replication thì cần sửa repl-diskless-sync thành yes trong file config



ngoài ra nếu tính năng diskless được bật thì trong quá trình phục vụ một slave thì các slave khác sẽ phải chờ tới lượt mình

do vậy có thể cài đặt delay để cho phép trong thời gian này có thể nhiều redis cùng muốn full resync từ đó một process có thể gửi cho nhiều slave



tính năng diskless phù hợp với tình uống disk chậm nhưng network băng thông lớn

### Partial Sync

là yêu cầu của slave khi slave có thông tin của việc sync dữ liệu và muốn sync tiếp tục với bộ dữ liệu trước đó trong các trường hợp

slave được promote lên làm master và các slave khác có thể tiếp tục sync

slave restart và có thể tiếp tục sync từ thời điểm nó bị shutdown

quá trình của partial sync như sau

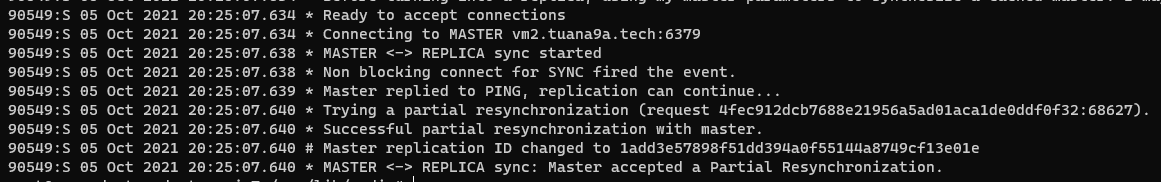
slave gửi meta data gồm replication-id và offset của phiên data sync trước đó

nếu metadata là hợp lệ, master sẽ gửi phần offset còn thiếu tới slave (một tập các command từ offset của slave tới hiện tại)

nếu không hợp lệ master sẽ trả về mã và sau đó slave sẽ gửi yêu cầu [full resync](#_25x760nlfpf7) tới master

sau đó slave tiếp tục sync như bình thường

dưới là hình ảnh partial sync thành công và slave cập nhật lại id của master mới lên



### Full Resync

là yêu cầu của slave khi không thể partial sync, thường xảy ra khi

slave lần đầu connect tới master

slave requests sync với id không tồn tại

slave request với offset ngoài backlog

master restart

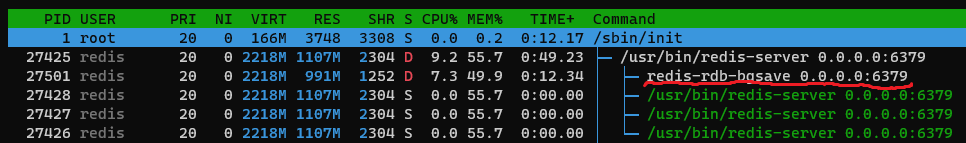
offset của slave lớn hơn của master

B1: master start background process (lệnh BGSAVE) tạo RDB file xuống disk

lệnh bgsave sẽ tạo một process và sau đó process này sẽ đảm nhiệm toàn bộ việc đẩy snapshot xuống file

trong quá trình này master non-blocking, có thể tiếp tục phục vụ các client, master sẽ buffer write command trong quá trình ghi RDB file

sau quá trình ghi file RDB, master thực thi command trong buffer và gửi những command này tới slave sau đó



B2: master gửi file RDB tới slave

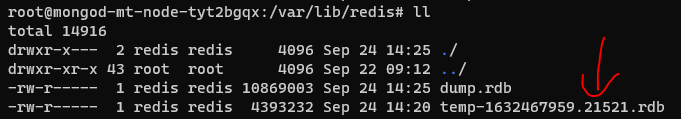
quá trình này sẽ mất rất nhiều thời gian và chiếm hết băng thông nếu kích thước file RDB lớn và tốc độ mạng chậm

? repl-timeout 60

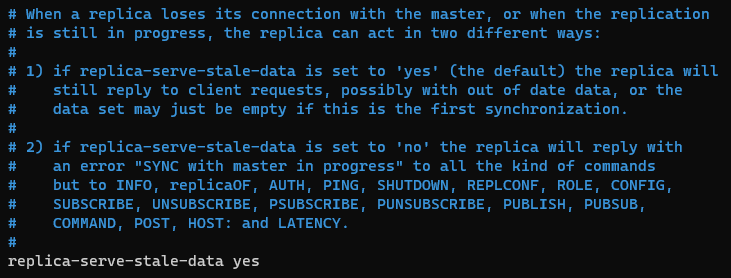
B3: slave lưu dần file RDB xuống disk

file sẽ được ghi liên tục xuống đĩa vào file tạm thời temp-xxx.rdb

sau đó sẽ được đổi lại tên về file gốc dump.rdb



trong suốt quá trình ghi file RDB, slave có thể phục vụ client với dữ liệu cũ mà không bị block



cho phép client đọc dữ liệu cũ

replica-serve-stale-data no để không cho client đọc dữ liệu cũ

B4: slave flush (xóa) hết dữ liệu cũ và load dữ liệu mới từ snapshot

trong quá trình load redis sẽ block tất cả, nếu client yêu cầu đọc ghi sẽ trả về lỗi

(AOF enable) B5: thì salve sẽ gọi lệnh BGREWRITEAOF ngay sau đó

## Manual Failover

(không dùng sentinel)

manual chọn master mới cho cụm

B1: chọn một slave để lên làm master bằng cách chạy lệnh

REPLICAOF NO ONE

hoặc SLAVEOF NO ONE với các redis < 5.0

lệnh này sẽ biến node thành standalone (master)

nhưng vẫn lưu thông tin của master cũ nhằm tiếp tục sync từ vị trí master cũ die

B2: trỏ các slave còn lại tới master mới lên bằng lệnh

REPLICAOF master\_address port VD: REPLICAOF vm2.tuana9a.tech

hoặc SLAVEOF master\_address port

sau lệnh này các slave sẽ có master mới và tiếp tục sync như bình thường

NOTE: sẽ không có full resync xảy ra, slave sẽ tiếp tục sync từ vị trí master cũ bị down

B3: cập nhật lại dns trỏ tới master mới (nếu cần)

CAUTIONS:

nếu offset của master mới lên nhỏ hơn offset của slave thì full resync sẽ xảy ra

do việc offset "chạy trước ô tô" sẽ bị master detect là không hợp lệ

vì vậy tốt nhất nên chọn slave có offset cao nhất để làm master

khi master cũ quay lại

B1: cấu hình password để master cũ làm slave của master mới trong config /etc/redis/redis.conf

masterauth master\_password

CAUTIONS: việc này có thể dễ quên nếu từ ban đầu chỉ config trên slave

B2: trỏ master cũ tới master mới để cập nhật dữ liệu mới nhất

TH1: bật lên với role là master sau đó step down làm slave cho master mới

REPLICAOF master\_address port

hoặc SLAVEOF master\_address port

cách này chắc chắn sẽ có full resync khi master cũ join lại với role là slave bất kể có snapshot hay không

TH2: có snapshot và config làm slave cho master mới trước khi start trong /etc/redis/redis.conf

replicaof master\_address port

cách này thì không full resync mà nó sẽ tiếp tục sync từ điểm bị die như một slave đổi chủ

do vậy nếu restart master bị die khuyến khích TH2 do thời gian load sẽ được cải thiện

CAUTIONS:

ví dụ có 4 node: node1, node2, node3, node4

ban đầu node 1 là master, sau đó manual failover node2 làm master

nếu làm node1 master trở lại bằng cách

cho node1 join lại với role slave để cập nhật dữ liệu mới nhất

promote master1 làm master

sau đó trỏ lần lượt lại các node khác về node1

thì full resync sẽ xảy ra và rất có thể các node khác sẽ mất dữ liệu

lý do vì offset của node1 giờ nhỏ hơn offset của các node còn lại

do vậy nếu việc đưa master cũ lên làm master không cần thiết thì cứ để ở role slave

Việc thay master "thủ công" như vậy có thể lặp đi lặp lại mà không bị full resync

miễn sao phải đảm bảo offset của master mới phải lớn hơn hoặc bằng các slave còn lại

## NAT, Docker

với công nghệ container đang sử dụng rất phổ biến hiện tại, và việc redis được sử dụng trong container rất phổ biến, tuy nhiên cùng với container là tầng ảo hoá về network, điều này có thể dẫn tới một vài hạn chế khi các instance redis muốn trao đổi với nhau, cụ thể là [sentinel](#_s00xbwehik5e) do sentinel phải kết nối tới các instance redis thì mới có thể monitor các instance redis, do vậy cần có sự điều chỉnh về địa chỉ + port sao cho các instance khác có thể kết nối tới redis

để điều chỉnh địa chỉ + port của các instance redis khi nó tự “giới thiệu” với master và sau đó sentinel khi kết nối tới master có thể lấy những thông tin này để kết nối sau đó, chỉnh sửa những config sau trong file config để đạt được mong muốn

replica-announce-ip 5.5.5.5 với 5.5.5.5 là địa chỉ bạn mong muốn

replica-announce-port 1234 với 1234 là port bạn mong muốn

## Max memory

trong quá trình vận hành config maxmemory trên slave sẽ bị ignore, còn trên master sẽ hoạt động bình thường do việc loại key ở slave sẽ gây dữ liệu không nhất quán giữa master và slave, vi phạm mô hình replication, do vậy việc loại key sẽ do master quản lý, các slave sẽ được lệnh DEL key tương ứng, phía dưới là config liên quan

replica-ignore-maxmemory yes

mặc định của config này là yes tức sẽ ignore maxmemory trên các slave, nếu muốn tắt sửa thành no, tuy nhiên việc này không khuyến khích do dữ liệu sẽ không nhất quán so với master

## Replication Config

để bật tính năng replicate dữ liệu ta chỉ việc sửa config của các slave (follower) để trỏ tới địa chỉ của master như sau

replicaof 192.168.1.1 6379

192.168.1.1 là địa chỉ của master, có thể sử dụng hostname

6379 là port của master

bật config này biến node thành slave

tắt config này sẽ khiến node trở thành master

nếu master có bật authentication với password thì ở phía slave chúng ta cũng cần “nhập mật khẩu” để có thể kết nối tới master

masterauth “my-secret-password”

“my-secret-password” là giá trị requirepass của master

dấu nháy kép có thể bỏ qua

đoạn mã đơn thuần là một string tuy nhiên do redis rất nhanh nên độ dài của password phải đủ dài

để tin tặc không thể brute force password với redis server của mình

để bật tính năng diskless đổi giá trị sau thành yes, mặc định giá trị này là no, cùng với đó là thời gian delay trước khi diskless sync xảy ra

repl-diskless-sync yes

repl-diskless-sync-delay 5

trong đó 5 là số giây delay của master

như đã nói ở trên backlog là nơi chứa lịch sử các commands, kích thước backlog càng lớn thì cho phép các slave có thể downtime càng lâu, phía dưới là config để sửa kích thước của backlog

repl-backlog-size 1mb

trong đó 1mb là kích thước, redis cũng hỗ trợ các đơn vị khác như kb, mb, gb, ...

trong quá trình load snapshot như khởi động, full resync, các slave có thể tiếp tục phục vụ clients với dữ liệu cũ trước khi dữ liệu mới được load xong

để bật tính năng đó sửa giá trị sau trong config thành yes

replica-server-stale-data yes

với giá trị no, slave sẽ block client đọc ghi thông tin và trả về lỗi

để dữ liệu được an toàn hơn redis cũng hỗ trợ đảm bảo tối thiểu dữ liệu được ghi trên nhiều node trên một lệnh ghi dữ liệu, tuy nhiên do redis sync trên cơ chế async nên redis tính toán trên số slave online và trả lỗi nếu số slave online không đủ, nếu số slave online đủ số lượng redis sẽ trả về thành công, số lượng đó được điều chỉnh ở config sau

min-replicas-to-write 2

2 là giá trị chỉ số lượng slave online tối thiểu để các lệnh ghi dữ liệu được trả về thành công

nếu số lượng online bé hơn giá trị này redis sẽ trả về lỗi tới client

để master detect được slave đang online hay không, redis dựa vào thời gian trong cài đặt phía dưới, do slave sẽ ping master sau một khoảng thời gian nào đó (mặc định 1s) dựa vào thời gian này master sẽ quyết định là slave có đang bị “lag” hay không

min-replicas-max-lag 10

dựa vào chỉ số lag, nếu slave < 10s lag thì được coi là online, ready, …

chỉ số này có ảnh hưởng tới min-replica-to-write ở trên

client-output-buffer-limit normal 0 0 0

// TODO

client-output-buffer-limit replica 256mb 64mb 60

// TODO

client-output-buffer-limit pubsub 32mb 8mb 60

// TODO

replica-announce-ip 5.5.5.5

replica-announce-port 1234

nếu các redis instance nằm phía sau NAT, docker remap thì sentinel có thể sẽ không kết nối được tới các slave

do vậy có thể điều chỉnh lại địa chỉ + port sao cho sentinel có thể connect được tới sau này

replica-priority 100

độ ưu tiên của slave chỉ được dùng khi [sentinel](#_s00xbwehik5e) muốn chọn ra master mới

độ ưu tiên càng thấp sẽ càng được chọn, nếu priority bằng 0 sẽ không được chọn

## Reference

1. khi có nhiều slave 1 file rdb có phục vụ đc nhiều request full resync ?

redis có phục vụ được nhiều request full resync cùng một lúc chỉ với một process bằng cơ chế copy on write

với mỗi request full resync redis master đều dump snapshot tại thời điểm request (snapshot sau đầy đủ hơn snapshot trước)

với cơ chế copy on write nên mặc dù file snapshot bị ghi lại liên tục thì snapshot của slave nào vẫn là snapshot của slave đó

thậm chí khi đang gửi snapshot tới một slave và snapshot này bị ghi đè lại bởi một request full resync từ slave khác

thì slave trước đó vẫn nhận đúng bản snapshot mà nó đã yêu cầu

2. snapshot của các slave có thể chia sẻ cho nhau không ?

có, snapshot của một slave, của master có thể chia sẻ cho nhau

slave khi load snapshot sẽ lấy thông tin trong snapshot và hoàn toàn tiếp tục sync như các slave khác

mà không cần phải full resync gửi toàn bộ file snapshot từ đầu

xa hơn một xíu thì snap shot chứa thông tin của replication như replication-id và offset

với 2 thông tin này thì một redis slave mới sẽ có thể tiếp tục sync ngày từ offset trong snapshot

mà không cần phải full resync lại toàn bộ từ đầu khi mới join

nếu như snapshot không có thông tin của replication-id và offset, hoặc offset không hợp lệ, replication-id không tồn tại

thì khi slave load snapshot xong và gửi PSYNC

master sẽ detect slave không hợp lệ và 2 bên bắt đầu full resync

3. khi master restart có ảnh hưởng tới sync dữ liệu không ?

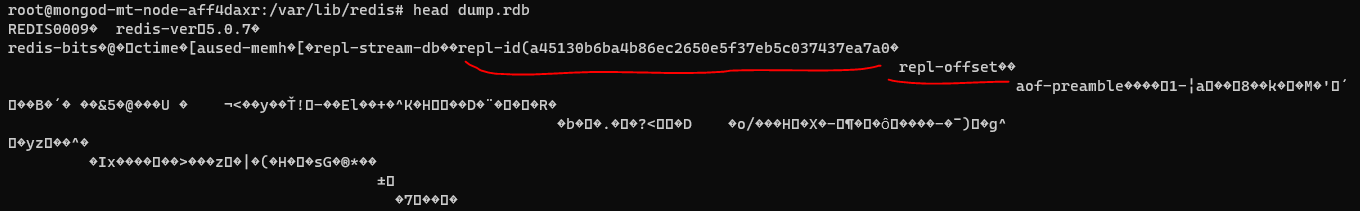
khi master restart bất kể trong snapshot có thông tin của replication trước đó

master sẽ renew hoàn toàn, tất cả slave sau đó sẽ đều bị full resync

điều này sẽ rất tốn thời gian khi kích thước db lớn

4. restart slave sau khi manual failover thì slave chọn master mới hay master cũ ?

do snapshot chỉ chứa thông tin replication-id và offset chứ không chứa địa chỉ của master mới



do vậy khi restart thì slave sẽ sync từ master trong config

điều này phải được handle cẩn thận do nếu master cũ đã restart và không có thông tin sync cũ

thì việc này sẽ dẫn tới full resync (sẽ mất thời gian với dataset lớn)

lý tưởng nhất là sửa lại config replicaof trong config trước khi restart slave

5. khi cập nhật dns của master thì các slave có tự động sync với master mới không ?

ngay khi dns được cập nhật thành công (ngay tại thời điểm có hiệu lực)

thì các slave sẽ connect tới master mới và tiếp tục sync như bình thường

6. với mô hình replication đa cấp có gì khác biệt

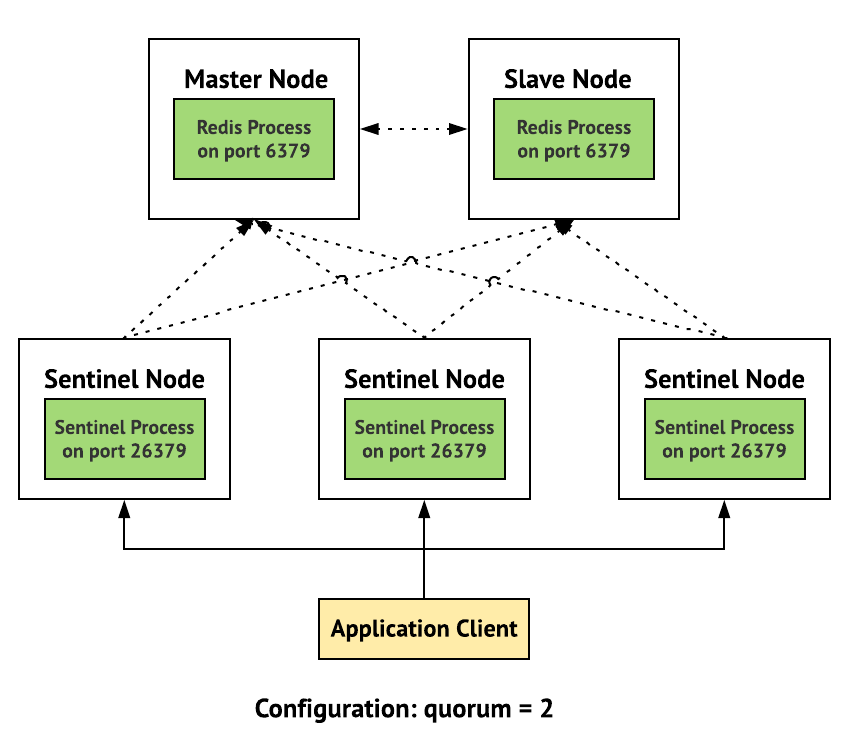
với init sync thì replicaof thằng nào thì lấy snapshot của thằng đó

runtime sync cũng sẽ sync trừ replicaof được chọn

# Redis Sentinel

nếu chỉ cài đặt redis với replication và mô hình master-slave, leader-follower, cụm redis của chúng ta chỉ hỗ trợ lưu trữ dữ liệu trên nhiều nơi, và nếu master của cụm die, thì cụm sẽ trở thành read only, do các slave không có chức năng nhận write operation chính thống, vì vậy redis hỗ trợ cơ chế để tự động hoá failover, detect master die và chọn ra một master mới cho cụm, từ đó cụm của chúng ta sẽ hoạt động bình thường trở lại

tuy nhiên nếu sentinel không thể tự động failover (vì một lí do nào đó) bạn có thể tham khảo [manual failover](#_7q81tz9jhth7) phía dưới, ngoài ra bài viết sẽ giới thiệu các nội dung sau [cơ chế hoạt động của sentinel](#_mtpckrsecjro), [quản lý các node redis, sentinel](#_xekuna7ja7fx)



## Tổng quan

sentinel là một process chạy độc lập với redis-server, sentinel sẽ monitor các instance redis trong mô hình master-slave để tự động detect và chọn master mới khi master cũ down bằng cách đưa ra quyết định dựa trên cơ chế đồng thuận số đông khi có master die các sentinel sẽ “hỏi” nhau rằng liệu master đó có thực sự die, từ đó chúng sẽ bầu ra sentinel leader dựa trên số đông, leader này sau đó sẽ đưa ra quyết định chọn master và các logic sau đó

redis sentinel sẽ được chạy mặc định ở port 26739

các tính năng redis sentinel hỗ trợ:

* Monitoring redis sentinel sẽ điều khiển các node mà nó kết nối tới, các việc mà redis sentinel có thể monitor như

promote slave làm master

reconfig slave (để trỏ sang master mới)

convert master to slave (khi master cũ rejoin sau failover)

* [Automatic failover](#_qt0f3vwz3smr) các sentinel liên tục kiểm tra master, nếu như phát hiện master die thì sentinel bắt đầu quá trình failover, promote một slave làm master, các slave được reconfig tới master mới

* Notification các sentinel có thể gửi thông báo tới System Admin, tới máy khác thông qua API khi có vấn đề xảy ra với redis instance

* Configuration provider các sentinel vừa monitor vừa cung cấp thông tin cho client để cung cấp địa chỉ của master hiện tại cho client nếu như có failover xảy ra sentinel sẽ cung cấp địa chỉ master mới cho client

sentinel sẽ chạy thành một process riêng biệt với master và có thể chạy trên bất kỳ máy nào (kể cả ở phía client)

do vậy một redis sentinel có thể monitor nhiều master

để chạy một sentinel có thể dùng lệnh phía dưới, về bản chất sentinel là redis-server nhưng chạy với mode sentinel và có config riêng, độc lập với redis-server

redis-server /path/to/sentinel.conf --sentinel

do trong quá trình hoạt động sentinel sẽ phải cấu hình lại các redis-server do vậy sentinel phải có permission để write file config ở disk

các trường hợp cần sửa config gồm automatic failover, convert một master thành slave, change master trên các slave

## Khởi tạo sentinel

để monitor một cụm master-slave có sẵn chúng ta có thể khởi tạo các sentinel để monitor các cụm này bất kể từ đâu, tuy nhiên mô hình phổ biến nhất chính làm đặt các sentinel tại chính các node trong cụm

### cách 1: cấu hình trước, start up sau

B1: cấu hình sentinel.conf

điều kiện bắt buộc

- đã cài package redis

- đã có thông tin (các) cụm redis master-slave ip, port

- requirepass của master (nếu master bật requirepass)

cấu hình trong file config của sentinel tới master của cụm cần monitor

chi tiết config và ý nghĩa config có thể xem [tại đây](#_nvxfmm8zqzlo)

sentinel monitor mastername master-address master-ip

sentinel auth-pass mastername "master-requirepass"

config auth-pass trùng với master requirepass nếu master bật requirepass

sentinel down-after-milliseconds mastername 60000

sentinel failover-timeout mastername 180000

sentinel parallel-syncs mastername 1

requirepass "my-secret-password"

đây là password để connect tới sentinel

hiện tại không rõ các sentinel có sử dụng thông tin này để trao đổi không //TODO

do vậy em để giá trị giống với auth-pass

lặp lại với tất cả các sentinel còn lại

B2: start sentinel sau khi đã config

điều kiện bắt buộc

- các cụm này phải allow network từ sentinel sắp khởi tạo

- phải allow port của sentinel (mặc định 26379) để sentinel có thể trao đổi với các sentinel khác

chạy sentinel bằng lệnh

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

lặp lại trên tất cả các sentinel còn lại

B3: kiểm tra

//TODO

### cách 2: start trước, cấu hình sau

(có thể áp dụng với các sentinel đang chạy)

B1: start sentinel

tại bước này không cần điều chỉnh nhiều

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

B2: config sentinel at runtime

điều kiện bắt buộc

- địa chỉ của sentinel + port sentinel

- requirepass của sentinel (nếu sentinel bật requirepass)

kết nối tới sentinel muốn điều chỉnh và chạy lệnh

SENTINEL MONITOR master-name master-ip master-port quorum

lưu ý chỉ có thể sử dụng ip và không thể sử dụng hostname (theo redis docs)

ngoài ra có thể chỉnh sửa thêm các config sau

SENTINEL SET master-name down-after-milliseconds 1000

SENTINEL SET master-name quorum 5

//TODO các lệnh sau đó

B3: kiểm tra

//TODO

## Cơ chế hoạt động

với cài đặt sau một sentinel đã có thể hoạt động

sentinel monitor mymaster 127.0.0.1 6379 2

mymaster à tên của master do một sentinel có thể monitor nhiều redis master

127.0.0.1 là địa chỉ của master, có thể sử dụng hostname nhưng sau đó sentinel sẽ rewrite lại config với IP của master

6379 là port của master

2 là quorum, sẽ được nhắc đến ở [đây](#_yv1kke8ytkpy)

sentinel sẽ kết nối tới master trong config lấy thông tin từ master gồm địa chỉ của các slave do vậy không cần phải chỉ định địa chỉ của các slave trong config từ đây sentinel sẽ tự kết nối tới những slave này và monitor chúng, tuy nhiên điều này có thể cản trở vì các slave có thể nằm sau NAT, Docker port mapping, do vậy cần phải [cài đặt thêm](#_co5axrj91eq3) trên các slave

ngoài ra các sentinel connect tới nhau đôi một và tới master do vậy số lượng connection sẽ “rất nhiều” nếu có nhiều sentinel, các sentinel chỉ kết nối tới master nhưng sau đó chúng tự kết nối tới nhau, vậy làm sao để chúng có thể kết nối với nhau đó chính là nhờ cơ chế [auto discovery](#_h2sz09cc6iwc)

do việc cụm master-slave (leader-follower) được monitor bởi nhiều redis sentinel do vậy nếu một hoặc một vài sentinel die thì việc monitor, automatic failover của cụm vẫn hoạt động bình thường

mỗi sentinel sẽ có một id (được sinh ngẫu nhiên khi hoạt động), do có thể một sentinel vừa out “group” và join lại, khi đó các sentinel khác phải nhận diện được sentinel cũ và tránh tưởng rằng có sentinel mới join vào group, cũng vì lí do này nếu một cách ngẫu nhiên hoặc có chủ ý (tự config) thì nếu có 2 hoặc nhiều sentinel cùng id join vào group thì group chỉ nhận diện có 1 sentinel

### Auto discovery

sentinel có thể kết nối với nhau nhờ cơ chế auto-discovery bằng cách các sentinel sẽ subscribe vào một channel “\_\_sentinel\_\_:hello”

lắng nghe channel này chờ các sentinel khác publish message giới thiệu “bản thân” để các sentinel có thể có thông tin của nhau và có thể kết nối sau đó

các sentinel sẽ gửi thông tin “hello messages” gồm ip, port, runid mỗi 2s

ngoài ra “hello messages” sẽ gồm cả config của master hiện tại và sentinel sẽ cập nhật config của bản thân nếu cũ hơn config nhận được

cũng nhờ runid sentinel sẽ tránh được việc add nhầm các sentinel đã có sẵn bằng việc loại hết runid đã tồn tại

các cấu hình mà các sentinel trao đổi với nhau để đi tới config thống nhất các config này sẽ được dùng để [monitor](#_96iez0e8u2p8) các instance mà chúng quản lý

### Automatic failover

là tính năng giúp một cụm master-slave (leader-follower) có thể tự động chọn ra một master mới khi master cũ bị die và tiếp tục hoạt động sau đó

với cài đặt sau

sentinel monitor mymaster 127.0.0.1 6379 2

mymaster là tên của master do sentinel monitor nhiều redis master

127.0.0.1 là địa chỉ ip của master

6379 là port của master

2 là quorum

quorum là giá trị chỉ số lượng sentinel phát hiện master die nếu số lượng master die lớn hơn giá trị này thì việc automatic failover sẽ xảy ra như sau

B1: master die và số lượng sentinel detect master die lớn hơn hoặc bằng quorum (trong config)

VD: quorum = 2

vậy nếu có 2 sentinel phát hiện master unreachable và chúng đồng ý với nhau như vậy thì một trong 2 sẽ bắt đầu quá trình failover

B2: các sentinel chọn là sentinel master (major)

VD: số lượng sentinel là 5

vậy số lượng sentinel tham gia tối thiểu phải là 3 (> n / 2) thì quá trình chọn sentinel master tiếp diễn

nếu không quá trình dừng tại đây (không có gì thay đổi)

cũng nhờ điều kiện này mà chúng ta an tâm khi có một vài sentinel die thì việc ra quyết định vẫn hoạt động bình thường

ngoài ra cũng hạn chế tình huống network partition khi thiểu số sentinel tự ý quyết định không theo số đông

B3: master sentinel chọn redis master mới, promote redis-server đó làm master

trong quá trình này redis sẽ sửa config của redis-server đó và ghi nó xuống file

thuật toán chọn master mới từ các slave của sentinel như sau

1. sentinel sẽ chọn ra slave có priority (config) cao nhất, nếu bằng nhau chuyển tiếp bước dưới

2. sentinel chọn là slave có offset cao nhất trong số slave cùng priority, nếu không chuyển tiếp

nhằm để dữ liệu của master mới là gần với master cũ nhất

3. sentinel chọn theo bảng chữ cái (random)

chi tiết thuật toán chọn master mới

<https://redis.io/topics/sentinel#replica-selection-and-priority>

B4: các slave khác được reconfig cập nhật master mới

sentinel sẽ “rewrite” config xuống file ở các node mà nó quản lý

nhờ vậy khi restart các slave sẽ tiếp tục connect tới redis master mới thay vì redis master cũ

### Monitoring

các sentinel monitor các redis instance bằng lệnh và sentinel có thể rewrite config (ghi lại file config xuống disk) điều này giúp khi restart thì redis sẽ hoạt động tiếp với phiên làm việc gần nhất, nếu như không ghi lại file config thì sau failover rất có thể khi slave restart nó sẽ trỏ tới master cũ, điều này không đúng và gây khó kiểm soát với cụm

nhờ việc trao đổi các “hello messages” trong message đó có chứa config của cụm hiện tại, các sentinel sẽ “áp đặt” force các redis-server mà chúng quản lý với config mà nó đang giữ (bất kể không có vấn đề gì xảy ra với cụm) và ghi config này xuống disk của node đó, việc ghi lại config sẽ thêm, sửa, xóa các config hiện tại do vậy có thể kiểm tra lại cấu hình trước khi restart một redis-server

từ đó các việc sentinel có thể làm là:

- promote slave làm master, việc này xảy ra khi master hiện tại của cụm die, các sentinel sẽ bầu ra một sentinel master sau đó sentinel master này sẽ chọn và promote một slave lên làm master

- change master, reconfig slave, xảy ra sau khi failover và chọn được master mới hoặc slave connect nhầm master, sentinel sẽ “yêu cầu” các slave cập nhật lại master của mình và sau đó “yêu cầu” tiếp các slave này sẽ phải ghi lại config xuống disk

- convert master to slave, xảy ra khi master cũ join lại cụm mà sentinel đang quản lý, khi master cũ online, sentinel thấy node này “claim” rằng nó là master, không đúng với config hiện tại (do khi failover thì chính master cũ cũng là slave) sau đó các sentinel sẽ force chúng thành slave từ đó master cũ sẽ trở thành slave trong cụm và sẽ được sync với dữ liệu mới nhất

### Configuration Propagation

mỗi khi có failover một config mới sẽ được sinh ra với một giá trị (configuration epoch) lớn hơn config cũ

VD: config hiện tại có epoch là 1 sau đó config có epoch là 2 tức mới hơn so với 1

nhờ có chỉ số config epoch này mà sau khi failover config này sẽ được lan truyền ra toàn bộ sentinel

do sentinel sẽ tự lấy config có config epoch lớn hơn config epoch của nó từ đó config mới nhất sẽ được nhất quán trên toàn bộ các sentinel

### Stale Sentinel

việc force config của các sentinel sẽ là vấn đề khi một sentinel với config cũ rejoin cụm sentinel hiện tại nó sẽ force toàn bộ redis server với config cũ

do vậy sentinel chỉ force config khi config của nó đang giữ đủ lâu, thời gian này phải lớn hơn thời gian của việc broadcast config bằng “hello messages” của các sentinel khác

ràng buộc này đảm bảo khi force config thì config đó sẽ là của số đông và là mới nhất có thể

## Quản lý các members

### Thêm slave

khởi tạo redis slave như bình thường với master là master mà sentinel đang monitor

redis-server /etc/redis/redis.conf

cách làm tương tự như mục [khởi tạo replication](#_a0azr1h0qqc8) khi chỉ cần khởi tạo slave

sau đó các sentinel sẽ lấy thông tin từ master, phát hiện slave mới và tự động monitor slave mới lên

### Xoá instance

sentinel sẽ không “quên” các slave hoặc master kể cả khi chúng đã die được một khoảng thời gian, nhờ vậy các sentinel có thể tiếp tục monitor các node này khi chúng join lại cụm hiện tại

tuy nhiên nếu muốn bạn vẫn có thể remove redis-server khỏi cụm mà sentinel đang quản lý bằng các bước sau

B1: tắt redis instance mà bạn muốn xoá (master, slave)

B2: thực hiện lệnh reset

điều kiện bắt buộc

- địa chỉ + port của các sentinel

- requirepass của các sentinel (nếu sentinel bật requirepass)

kết nối tời từng sentinel, do sentinel vẫn là redis-server nhưng với mode và port khác

do vậy cách kết nối sẽ tương tự như khi kết nối tới redis-server

VD: redis-cli -p 26379

và chạy lệnh sau trên toàn bộ các sentinel

SENTINEL RESET mastername

việc này sẽ khiến sentinel chỉ add những slave của master hiện tại nhờ sử dụng lệnh INFO

B3: kiểm tra kết quả

//TODO

### Thêm sentinel

chỉ cần khởi tạo sentinel như bình thường

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

các bước tương tự khi [khởi tạo sentinel](#_9fem2081k5jg)

sau đó các sentinel khác sẽ phát hiện sentinel mới online và sentinel này cũng sẽ phát hiện các sentinel khác nhờ cơ chế [auto discovery](#_h2sz09cc6iwc) đã nói ở trên

sentinel mới lên sẽ được add vào “group” của cụm sentinel đã tồn tại

### Xoá sentinel

trong quá trình vận hành có thể ta muốn remove một sentinel do cần lý do nào đó

do việc các sentinel phải nhớ id của nhau để tránh bị lặp khi một sentinel rejoin sau khi disconnect, những thông tin về các sentinel khác được ghi vào trong config file do vậy kể cả restart các sentinel vẫn nhớ “mặt” nhau, từ đó việc xoá đi một sentinel khỏi nhóm khá phức tạp và cần phải thực hiện thủ công

B1: dừng sentinel instance mà bạn muốn remove

B2: chạy lệnh reset

điều kiện bắt buộc

- địa chỉ + port của các sentinel

- requirepass của các sentinel (nếu sentinel bật requirepass)

kết nối tời từng sentinel, do sentinel vẫn là redis-server nhưng với mode và port khác

do vậy cách kết nối sẽ tương tự như khi kết nối tới redis-server

VD: redis-cli -p 26379

sau đó chạy lệnh trên toàn bộ các sentinel instance khác

RESET \*

dấu \* có thể thay = id của master để chỉ định remove một cụm master mà sentinel đang quản lý thôi

các lệnh này nên được chạy cách nhau 30s từng redis sentinel một //TODO check

B3: kiểm tra kết quả

chạy lệnh sau trên sentinel

SENTINEL MASTER mastername

mastername là tên của master trong config

lệnh này trả về thông tin liên quan có thể dùng để kiểm tra

//TODO

## Sentinel Config

chú ý các config này nằm khác file với config file so với redis server

sentinel monitor mymaster 127.0.0.1 6379 2

mymaster là tên đặt tuỳ ý dùng để phân biệt với các master khác nếu sentinel monitor nhiều master - nhiều cụm

127.0.0.1 là địa chỉ của master có thể dùng hostname

6379 là port của master

2 là quorum số lượng sentinel đồng thuận rằng master đã down

sentinel sẽ tự detect các slave từ thông tin lấy được từ master

sentinel down-after-milliseconds mymaster 60000

khoảng thời gian down để sentinel chỉ định master die

mymaster là tên ứng với sentinel monitor ở trên

60000 là khoảng thời gian tính theo milisecs ở đây là 60s

sentinel failover-timeout mymaster 180000

? khoảng thời gian tối đa cho việc failover cho slave mới lên // TODO

sentinel parallel-syncs mymaster 1

số lượng

requirepass "my-secret-password"

vì sentinel vẫn là redis-server nhưng với mode sentinel nên client có thể connect tới và điều khiển

bật authen xác thực khi có client kết nối tới

sentinel auth-pass mymaster "my-secret-password"

nếu master yêu cầu xác thực, thì cần config requirepass của master cho sentinel

mymaster tên master ứng với sentinel monitor trong config

“iloveyou” là mật khẩu của master muốn kết nối tới

do các sentinel có thể đứng sau proxy, NAT, docker remap, có thể ảnh hưởng tới việc các sentinel connect tới nhau

do vậy sentinel cho phép tùy chỉnh “lời giới thiệu bản thân” gồm IP và port cho các sentinel khác

sentinel announce-ip <ip>

ip mong muốn sentinel broadcast

sentinel announce-port <port>

port mong muốn sentinel broadcast

## Reference

1. các sentinel khi khởi động chỉ có thông tin master vậy chúng kết nối với nhau như nào ?

khi kết nối tới master sentinel sẽ subscribe vào channel “\_\_sentinel\_\_:hello”

sau đó chúng liên tục publish thông tin của mình (địa chỉ, port, … ) vào channel này

từ đó các sentinel khác có thể lấy được thông tin này và kết nối sau đó

2. sau khi failover và được reconfig, các slave restart có nhận master mới hay master cũ ?

khi failover các sentinel sẽ ghi config (bao gồm cả địa chỉ của master mới) xuống file

do vậy khi restart sẽ lấy được lại thông tin của phiên làm việc trước đó

3. vấn đề liên quan tới việc sử dụng sentinel và dùng dns cho master

nếu dùng sentinel để monitor cụm master-slave thì sentinel sẽ phân giải hostname thành địa chỉ ip và ghi config xuống file

do vậy sẽ không hưởng lợi từ việc đổi master bằng dns, việc đổi master từ dns hoạt động tốt với replication

# Redis Cluster

## Tổng quan

nhìn chung với [redis replication](#_wbl4v5fpm2fi) và [redis sentinel](#_s00xbwehik5e) thì đáp ứng được tính an toàn dữ liệu và cả việc xử lý failover khi có master die thì cụm vẫn có thể tiếp tục hoạt động bình thường, tuy nhiên với nghiệp vụ mà có lượng ghi dữ liệu lớn, từ nhiều client thì bài toán trở nên phức tạp hơn với yêu cầu cân bằng tải mà vẫn đáp ứng được tự động failover

do vậy redis hỗ trợ tính năng redis cluster, với redis cluster chúng ta có thể vừa cân bằng tải bằng việc tự động phân bổ (shards) các key trên nhiều master (mà không cần phải cài đặt hay proxy gì cả) và việc tự động failover khi có một master die, một slave của master đó sẽ lên làm master, từ đó cluster của chúng ta sẽ hoạt động bình thường kể cả khi có một (một vài) redis master die

tuy nhiên nếu cụm master-slave của một hash slot cùng lúc die hoặc network partition với phần còn lại thì cả cụm sẽ ngừng hoạt động

(error) The cluster is down

lúc này client không thể đọc ghi dữ liệu kể cả đọc ghi phần dữ liệu có thể tiếp cận trên các master còn lại

nhưng nếu cụm master-slave đó không chứa hash slot nào thì khi die cụm sẽ không bị ảnh hưởng

redis cluster hoạt động mà không cần tới sentinel, redis cluster là một mode hoạt động của redis-server

khi redis-server hoạt động với mode này, mỗi một instance redis gồm 2 thành phần chính là

- normal redis phục vụ dữ liệu cho client, đây là thành phần redis thông thường như standalone, replication

- cluster bus là thành phần đảm nhiệm việc trao đổi thông tin giữa các node trong cluster để đưa ra quyết định giống [sentinel](#_s00xbwehik5e)

mặc định cluster bus hoạt động ở port có giá trị là port của redis-server + 10000

VD: port redis-server là 7000 thì port cluster bus là 17000

redis cluster không hỗ trợ nhiều database trên 1 instance (tức lệnh SELECT sẽ không hoạt động)

việc này không cần để ý do khi sử dụng redis thường chỉ cần dùng database mặc định là 0 (SELECT 0)

từ đó cụm redis của chúng ta sẽ gồm nhiều redis master, mỗi redis master sẽ có nhiều redis slave, với mỗi redis instance bất kể master, slave đều có cluster bus để trao đổi thông tin với nhau mà không cần một process ngoài monitor như sentinel

để khởi tạo redis-server với mode cluster chỉ cần các config sau là đã có thể bật redis với mode cluster

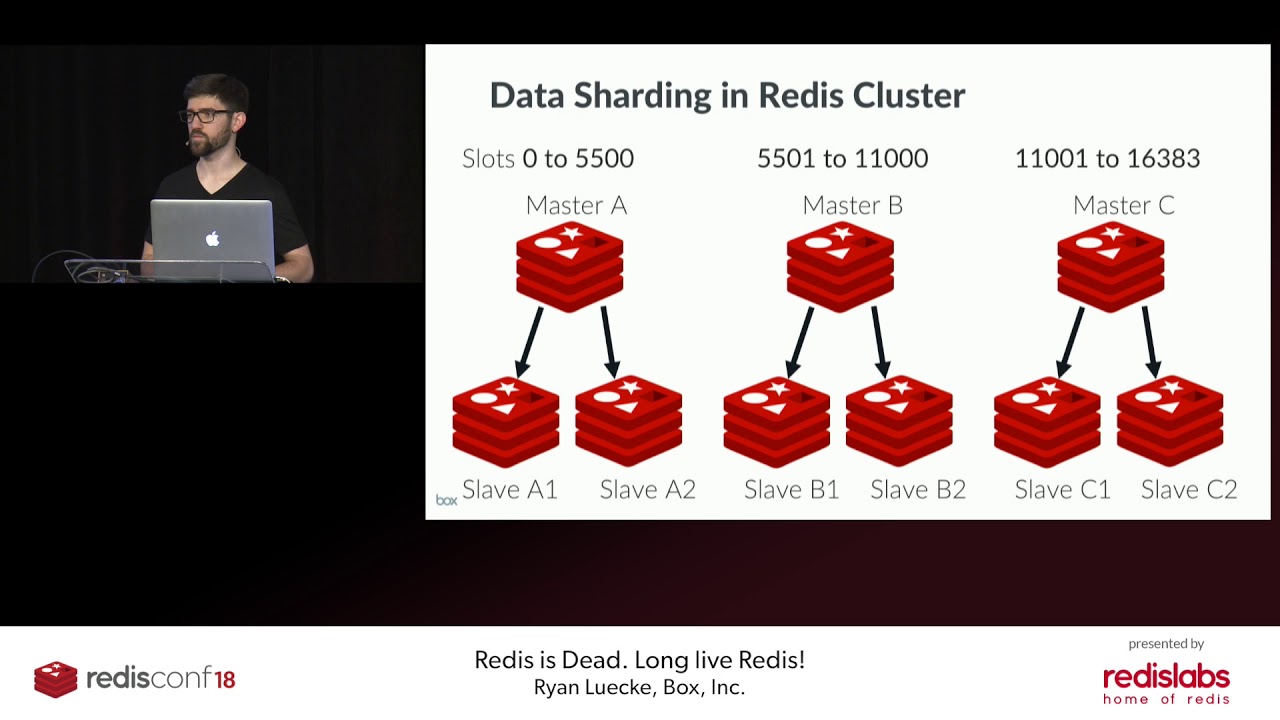
port 7000

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

tuy nhiên để khởi tạo một cluster sẽ cụm sẽ cần được khởi tạo lần đầu bởi client (client tools) sau đó cụm của chúng ta sẽ tiếp tục hoạt động mà không cần phải có một client nào monitor cả



## Ngôn ngữ

redis cluster hiện tại ở phía client chưa được hỗ trợ phổ biến, tuy nhiên các ngôn ngữ phổ thông thì đã có, phía dưới là các danh sách hỗ trợ

Ruby [redis-rb-cluster](http://github.com/antirez/redis-rb-cluster)

Python [redis-py-cluster](https://github.com/Grokzen/redis-py-cluster)

PHP [predis](https://github.com/nrk/predis)

Java [jedis](https://github.com/xetorthio/jedis)

C# [StackExchange.Redis](https://github.com/StackExchange/StackExchange.Redis)

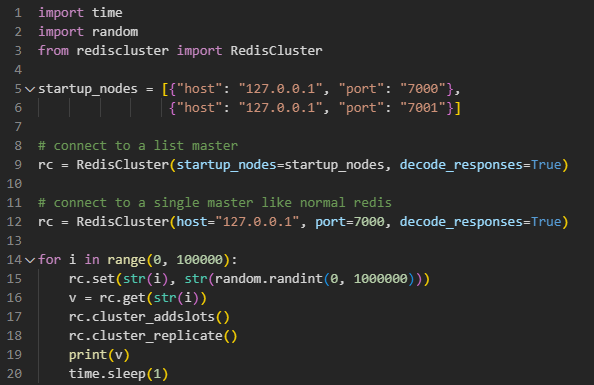
Go [redis-go-cluster](https://github.com/chasex/redis-go-cluster)

Node.js [thunk-redis](https://github.com/thunks/thunk-redis) và [ioredis](https://github.com/luin/ioredis)

ngoài ra với redis-cli là tool đi kèm với package redis cũng hỗ trợ sẵn redis cluster với “-c” để switch sang mode cluster

một tính năng của cluster là “điều phối” key tới master tương ứng, việc này được hỗ trợ bởi các thư viện phía client khá tốt, thậm chí chúng còn cải thiện tốc độ bằng cách cache lại map giữa hash slots và địa chỉ của các node từ đó thay vì phải tạo lại kết nối tới node tương ứng, client sẽ dùng trực tiếp kết nối tới master trước đó, đoạn cache này sẽ được refresh khi có thay đổi của cluster config như failover hoặc admin cập nhật lại config

ví dụ với python

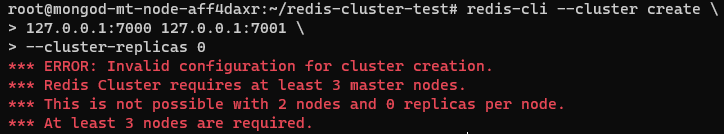


## Khởi tạo cluster

### cách 1: khởi tạo bằng tools

điều kiện bắt buộc

- cần phải có ít nhất 3 master



B1: khởi tạo các redis-server với mode cluster

điều kiện bắt buộc

- đã cài package redis

- requirepass của các master trong cụm

- các master, slave phải có requirepass giống nhau (vì nếu khác thì sẽ không thể failover)

sửa các config sau trong file config của redis

chi tiết các config và ý nghĩa có thể xem ở [đây](#_yd6a62j6d8y)

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

nếu muốn bật bảo mật

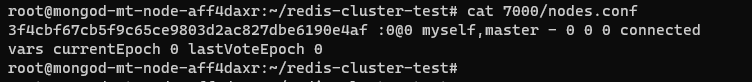
masterauth your-master-requirepass

requirepass your-requirepass

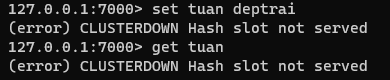
sau đó start redis như bình thường

redis-server /etc/redis/redis.conf

ban đầu các node sẽ chỉ có thông tin của mình cluster đó



nếu chỉ đến đây và với thông tin như vậy cụm sẽ không hoạt động



và sẽ trả về lỗi do chưa khởi tạo “phân lô” hash slot do vậy phải tới bước 2

B2: từ một client khởi tạo config và đẩy config này tới các redis instance

điều kiện bắt buộc

- các redis instance có thể connect tới port của cluster bus của nhau

- redis-cli phải kết nối được tới các thằng redis còn lại

- các redis-master phải có requirepass giống nhau do nếu khác nhau thì redis-cli với switch -a không thể hoạt động

từ đó không thể khởi tạo cluster

VD: có một cụm gồm 6 redis instance trên cùng một host nhưng chạy ở port 7000 tới 7005

với redis 5.0 trở lên tools tạo cluster có sẵn trong redis-cli

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

ngoài ra bạn có thể gõ lệnh trên với --cluster-yes để mặc định đồng ý và redis-cli sẽ tạo luôn cho bạn 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 --cluster-yes

nếu cụm có bật authen thì chỉ cần thêm -a your-requirepass là thành công

redis-cli --cluster create \

127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \

--cluster-replicas 0 -a rXNYq+tpWo6u8i1

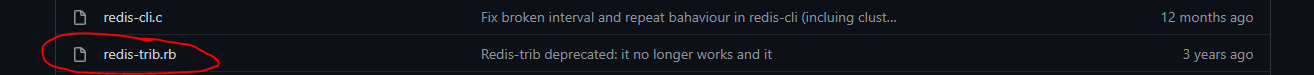
trong đó

create để tạo cluster và sau đó là danh sách ip:port của các node trong mạng

--cluster-replicas 1 ý nghĩa mỗi master sẽ có một slave nếu là 0 thì các master sẽ không có slave

--cluster-yes để mặc định đồng ý với mọi prompt console

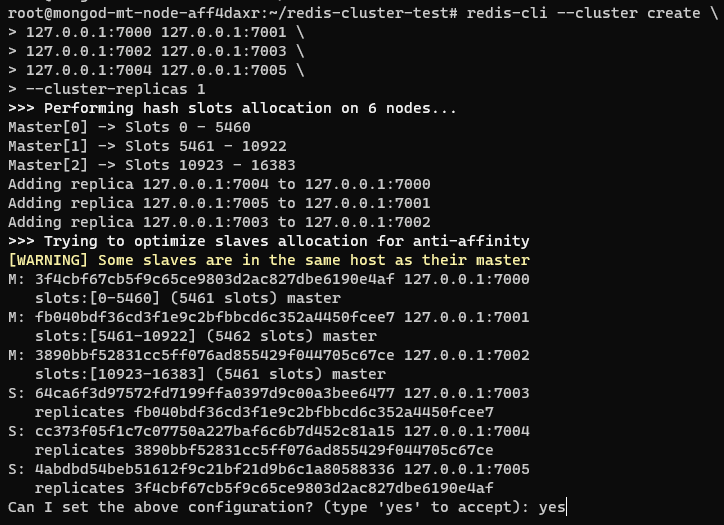
với redis 3.0 và 4.0 thì sẽ cần phải tải một tool khác từ source redis được viết bằng ruby tuy nhiên đã deprecated



để sử dụng phải cần phải install redis từ package của ruby

gem install redis

đối với trường hợp bạn không có --cluster-yes ở cuối câu lệnh thì redis yêu cầu xác nhận config, bạn chỉ cần gõ yes

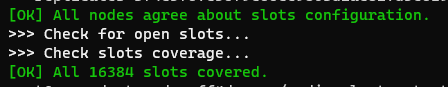


với ảnh trên chúng ta sẽ có một cụm gồm 3 master và mỗi master có 1 slave tương ứng

B3: kiểm tra kết quả

sau đó nếu output có kết quả như sau là bạn đã thành công

[OK] All 16384 slots covered



### cách 2: khởi tạo bằng cách kết nối trực tiếp tới các instance

(tự tìm hiểu, ý tưởng, chưa test)

B1: vẫn khởi tạo các redis server như trên

B2: sau đó kết nối tới các redis instance vả chạy lệnh lần lượt trên các node

cluster meet

B3: phân lại hash slot tới các instance mong muốn

cluster addslots

B4: kiểm tra

## Cơ chế hoạt động

redis cluster hoạt động nhờ kết hợp nhiều cụm master-slave vào thành một cluster, mỗi cụm master-slave này sẽ quản lý một hash slot nói nôm na là một khoảng giá trị kết quả sau khi băm key, và tự động trao đổi dữ liệu sao cho key nào vào đúng hash slot của key đó

để quản lí và trao đổi với nhau redis cluster sử dụng các thông tin

node-id là định danh cho một node được sinh tự động khi lần đầu start một redis-server với mode cluster, node-id sẽ không đổi khi restart

config-epoch cũng giống sentinel để đi tới thống nhất các cluster bus sẽ sử dụng config-epoch là một số nguyên dương tăng dần mỗi khi có cập nhật config từ đó config mới nhất sẽ luôn được lan truyền tới tất cả các cluster bus trong cụm để ra quyết định tiếp theo

hash-slot là range giá trị thu được sau khi băm key để có thể điều phối các key tới đúng cụm master-slave đang quản lý

mỗi instance redis sẽ lưu các thông tin này vào file nodes.conf (có thể điều chỉnh trong cài đặt) để sau đó khi restart chúng có thể join lại vào cụm

phía dưới là nội dung của nodes.conf với nội dung có format

id ip:port flags master pings pongs epoch link slot ... slot

trong đó

id: id của node đó

ip:port

flags: master, replica, myself, fail, ...

master if it is a replica, the Node ID of the master

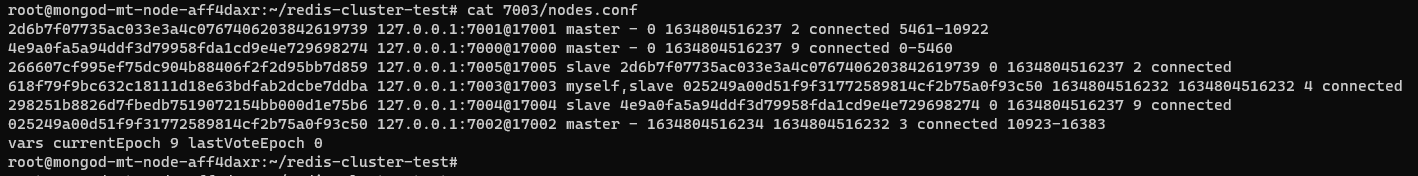
pings Time of the last pending PING still waiting for a reply.

pongs Time of the last PONG received.

epoch Configuration epoch for this node (see the Cluster specification).

link Status of the link to this node.

slot...slot Slots served...

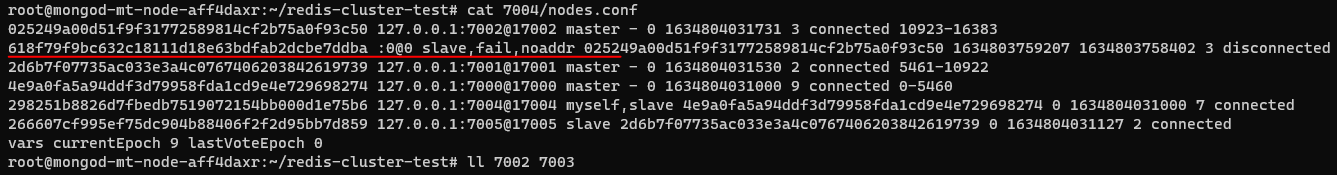


trong hình trên các bạn có thể thấy được các thông tin như địa chỉ, role, hash-slot, config epoch như mình đã nêu ở trên

### Cautions

nếu như lỡ xoá nhầm file nodes.conf này thì khi start redis instance sẽ không join được cluster cũ vì khi start chúng sẽ tự sinh id mới và không chứa thông tin phiên làm việc trước đó, ở phía cụm node này sẽ được đánh fail vì thông tin của instance join lại không hợp lệ và chỉ có thể join lại thành công khi sửa lại thông tin nodes.conf chuẩn với config của cluster hiện tại sau đó restart

phía dưới là ảnh khi node rejoin với file nodes.conf biến mất, từ đây cũng có thể thấy được redis cluster yếu hơn sentinel ở khoản quản lý việc rejoin, với sentinel thì khi rejoin thì slave, master sẽ được áp config, còn với cluster các instance redis sẽ phải tự quản lý config của mình, nếu mất thì sẽ cần phải có sự can thiệp thủ công



### Data sharding

redis không sử dụng consistent hashing mà sử dụng hash slot, mỗi redis cluster có 16384 hash slots được chia đều trên n master

VD: có 3 master A, B, C thì A sẽ handler 0 tới 5500, B sẽ handler từ 5501 tới 11000 và C sẽ handler từ 11001 tới 16383

mỗi redis master có thể chứa nhiều khoảng hash slot, xem hình dưới



như hình trên redis master đang chứa 3 slot

khi client kết nối tới một master nào đó và sử dụng lệnh write nào đó

redis sẽ băm key và được bao nhiêu redis sẽ “điều phối” key đó tới đúng master đang quản lý hash slot tương ứng

tuy nhiên do việc phân các hash slots cho các master và slave của nó quản lý, do vậy bằng một cách nào đó toàn bộ hash slot gồm master và slave của nó không thể truy cập với phần còn lại của cụm thì cụm sẽ không thể hoạt động được

(error) The cluster is down

như vậy toàn bộ cụm sẽ không thể hoạt động được kể cả khi phần còn lại đều online, nếu như sau đó

việc phân lô cho phép thêm, xóa node trong cluster dễ dàng bằng cách chuyển các hash slot qua lại trong mạng, ví dụ để xóa một node ta cần phải chuyển toàn bộ hash slots của chúng sang những node khác, sau đó thì node này có thể xoá một cách an toàn, và cũng chính vì lý do có thể move hash slots giữa các node một cách dễ dàng nên sẽ không có downtime khi remove hay add thên node vào cluster, chi tiết xem trong mục [quản lý cluster](#_xvpq1t969kxv)

### Automatic failover

cũng như sentinel khi phát hiện một master die, các sentinel sẽ promote một slave của master đó lên làm master mới à master mới lên sẽ đảm nhiệm hash slot của master đã die trước đó, quá trình này xảy như sau

tuy nhiên nếu vì lý do nào đó bạn muốn promote một slave lên làm master thay master hiện tại thì bạn xem mục [manual failover](#_16lv86d54jw8)

## Quản lý cluster

### README

nếu cluster có bật authen thì chỉ cần thêm swith -a your-requirepass vào cuối các lệnh, còn với lib client thì tùy lib sẽ có cách điền tương ứng

VD: redis-cli --cluster add-node 127.0.0.1:7003 127.0.0.1:7000 --cluster-slave -a BmJ5rIisn/fPWv6a1RskK27n6mE16IKA078O+Vz

### resharding

tại một thời điểm nào đó sau khi chạy một thời gian bạn muốn đi chuyển một (một vài) hash slot từ node này tới node khác hoặc bạn muốn chuyển từ nhiều node tới một node đây chính là reshard, trong quá trình reshard phía client sẽ không bị ảnh hưởng (transparency)

B1: lấy thông tin của cluster gồm node-id, hash slot và kích thước hash slot của node master đó

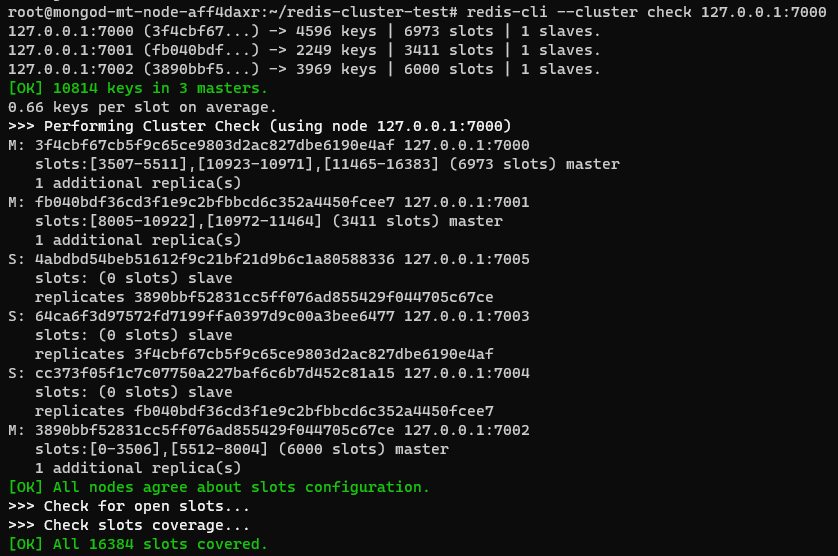
điều kiện bắt buộc

- địa chỉ + port của một node trong cluster

- requirepass của node đó (nếu bật requirepass)

sau đó chạy lệnh sau để lấy thông tin

redis-cli --cluster check 127.0.0.1:7000



B2: sau khi đã có thông tin thì tiến hành reshard cluster tuỳ theo mục đích với lệnh sau

điều kiện bắt buộc

- node id của node mà bạn muốn chuyển tới

- node id của node mà bạn muốn gửi (tuỳ nhu cầu có thể gửi toàn bộ với all hoặc một vài)

sau đó chạy lệnh dưới

redis-cli --cluster reshard 127.0.0.1:7000 \

--cluster-from 3f4cbf67cb5f9c65ce9803d2ac827dbe6190e4af \

--cluster-to fb040bdf36cd3f1e9c2bfbbcd6c352a4450fcee7 \

--cluster-slots 3000 \

--cluster-yes

trong đó

127.0.0.1:7000 là địa chỉ + port của một node bất kì trong cluster

--cluster-from là danh sách các id của node bị chuyển mất hash slot (node gửi hash slot)

VD: --cluster-from 3f4cbf67cb5f9c65ce9803d2ac827dbe6190e4af 3890bbf52831cc5ff076ad855429f044705c67ce

hoặc có thể để giá trị là all để lấy từ tất cả các node khác

--cluster-to là node id của node muốn chuyển hash slot tới

VD: --cluster-to fb040bdf36cd3f1e9c2bfbbcd6c352a4450fcee7

--cluster-slots số lượng hash slot mà bạn muốn chuyển

VD: --cluster-slots 3000

tức chuyển 3000 slot từ --cluster-from

--cluster-yes giống -y trong lệnh sudo apt install -y some-package

để confirm với mọi prompt

ngoài ra bạn có thể chỉ cần chạy phần đầu của lệnh là redis-cli --cluster reshard 127.0.0.1:7000 để vào interactive mode, với mode này sẽ có tương tác và trả lời từng câu hỏi của việc reshard và ấn “yes” với các lần prompt của console

### add instance

B1: chuẩn bị instance muốn add

điều kiện bắt buộc

node mới là empty hoàn toàn (nếu node này được remove trước đó thì cần xóa file cấu hình cụm VD: nodes.conf)

do nếu không xóa khi add sẽ có lỗi do node mới thêm vẫn chứa cấu hình trước đó



B2: add instance mới tạo

điều kiện bắt buộc

để add master chạy command như sau

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000

trong đó

127.0.0.1:7000 là địa chỉ của một node bất kì trong cluster hiện tại

127.0.0.1:7006 là địa chỉ mới của node muốn thêm

để add slave chạy command như sau

add slave vào random master

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave

add slave vào một master nhất định

redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000 --cluster-slave \

--cluster-master-id 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e

trong đó

127.0.0.1:7006 là địa chỉ + port của instance mới tạo

127.0.0.1:7000 là địa chỉ + port của instance bất kỳ trong cluster

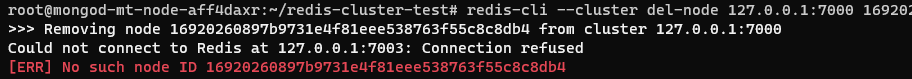
--cluster-slave để chỉ định node muốn thêm là slave

3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e là id của master trong cluster

### remove instance

điều kiện bắt buộc

không được tắt instance muốn xóa, nếu không sẽ có lỗi sau



lý do vì nếu shutdown trước thì cấu hình hiện tại của cluster vẫn được lưu ở node bị tắt

do vậy gây nhập nhằng, phức tạp khi node đó được bật lên và join lại vào cluster

nếu là master phải move toàn bộ hash slot của chúng với việc [reshard](#_no2vclb62klr) ở trên

hoặc có thể convert chúng thành slave với [manual failover](#_yuvhs4nvwac1) phía dưới

B1: sau đó chạy lệnh sau

redis-cli --cluster del-node 127.0.0.1:7000 <node-id>

trong đó

127.0.0.1:7000 là địa chỉ của instance bất kỳ trong cluster

node-id là id của node đó trong cluster

B2: kiểm tra kết quả

chạy lệnh sau

redis-cli --cluster check 127.0.0.1:7000

hoặc ở code client (các lib khác tương tự)

rc.cluster\_nodes()

node bị remove sau khi restart sẽ không join lại cluster cũ, tuy nhiên cấu hình trước đó node này vẫn lưu do vậy nếu muốn tái sử dụng như việc add vào một cluster khác thì phải xóa file config này đi (default nodes.conf)

### change master

(lệnh này cũng kiêm luôn convert master thành slave trong cluster)

tuy nhiên để convert được thì bắt buộc master đó không chứa hash slot

redis cũng hỗ trợ việc cân bằng số lượng các slave của các master nhằm đảm bảo không có master “mồ côi”

trong lúc runtime có thể thay đổi master của một slave hoặc convert master thành slave bằng lệnh sau

B1: kết nối tới đúng instance muốn sửa đổi

redis-cli -c -p 7000

kết nối với mode bình thường

redis-cli -p 7000

kết nối với mode cluster

cả 2 cách kết nối với mode bình thường và mode cluster đều hỗ trợ

B2: chạy lệnh trên instance đó

điều kiện bắt buộc

để master convert thành slave thì master đó không được nắm giữ bất kỳ hash slot

với việc dùng redis-cli

CLUSTER REPLICATE <master-node-id>

với lib không có cluster phía client (tương tự với các lib client khác)

r.cluster(“REPLICATE fb544ffb6451b81b578bb6cee5e2ac5259ac6030”) //TODO test

với lib cluster phía client

rc.cluster\_replicate(“fb544ffb6451b81b578bb6cee5e2ac5259ac6030”) //TODO check

convert slave to master hiện tại chưa rõ do vậy nếu muốn làm thì cần reset và add lại vào cluster thì role này lên master

B3: kiểm tra

chạy lệnh sau

redis-cli --cluster check 127.0.0.1:7000

hoặc ở code client (các lib khác tương tự)

rc.cluster\_nodes()

### manual failover

B1: kết nối tới instance mà bạn muốn thực thi

redis-cli -p 7000

B2: chạy

với trường hợp master online, chạy lệnh

CLUSTER FAILOVER

với lib client

r.cluster(“FAILOVER”)

rc.cluster\_failover()

với trường hợp master fail hoặc start slave trước và muốn slave lên làm master vì lý do nào đó

CLUSTER FAILOVER FORCE

với lib client

r.cluster(“FAILOVER FORCE”)

rc.cluster\_failvoer(“FORCE”)

B3: kiểm tra

chạy lệnh sau

redis-cli --cluster check 127.0.0.1:7000

hoặc ở code client (các lib khác tương tự)

rc.cluster\_nodes()

## Cluster Config

bể bật / tắt mode cluster sửa config sau trong file config

cluster-enabled yes

yes để bật tính năng cluster của redis

no để tắt tính năng này (mặc định)

nơi để các instance lưu trữ cấu hình của cluster, bạn không nên chỉnh sửa file này nếu cluster hoạt động bình thường

cluster-config-file nodes.conf

file config sẽ được redis ghi ra mặc định là nodes.conf

file này sẽ chứa tất cả metadata liên quan tới cluster

(lưu ý) nếu xoá file này thì redis instance sẽ không thể join lại cụm trước đó, nếu xoá nhầm vẫn có thể sửa lại bằng tay được

cluster-node-timeout 5000

5000 là thời gian để detect một node die, unreachable

khi master bật requirepass thì để các slave có thể sync bắt buộc phải có pass của master, config này tương tự ở trên

masterauth "master-requirepass"

dấu nháy có thể bỏ qua

nếu master bật requirepass thì các slave phải có config này

để bật xác thực cho một instance sửa config sau, tương tự các config ở trên

requirepass "your-secret-password"

dấu nháy có thể bỏ qua

# So sánh

## redis sentinel vs redis cluster

về mặt đọc ghi dữ liệu

redis sentinel chỉ có nhiệm vụ handle việc tự động failover của một cụm redis replication (master-slave)

quá trình ghi vẫn chỉ diễn ra trên một master

quá trình đọc có thể đọc trên slave

redis cluster gồm nhiều master-slave và chia dữ liệu làm nhiều hash slot

do đó có thể split luồng ghi dữ liệu trên nhiều master một cách tự động

đương nhiên để làm được điều như vậy cần sự support từ client (cli, lib, … )

về mặt triển khai

redis sentinel triển khai khá đơn giản và có thể triển khai từ một node khác độc lập cụm master slave

tuy nhiên số lượng redis tham gia sẽ tăng lên (ngoài các node trong redis replication, giờ thêm các process sentinel)

redis cluster triển khai phức tạp hơn

do sẽ có nhiều master-slave và phải quản lý các hash slot và các vị trí của chúng

nhưng cluster chạy không cần process ngoài do vậy sẽ đơn giản hơn trong việc quản lý

về mặt high availability

redis sentinel chạy độc lập với redis server

do vậy khi các node trong master-slave chết thì các sentinel vẫn có thể tiếp tục monitor chọn ra master mới

nếu triển khai cùng trên các node chứa dữ liệu thì nếu số lượng node die quá bán thì sentinel sẽ trở nên vô nghĩa

redis cluster cũng có khả năng failover để chọn master mới cho cụm

tuy nhiên cụm hash slot master-slave nào thì chỉ có thể failover trên cụm master-slave đó

redis cluster không thể chọn một instance từ hash slot khác lên làm thay master hiện tại

do vậy nếu toàn bộ master, slave của một hash slot die thì toàn bộ cụm sẽ không thể hoạt động được

về mặt kết nối, hỗ trợ client

redis sentinel hỗ trợ thông tin cấu hình giúp tìm master, slave từ đó các lib client khác có thể kết nối tới

redis cluster có các lib hỗ trợ ít tuy nhiên các lib cluster hỗ trợ sharding tự động

# Sử dụng redis

## Cache

<https://redis.io/topics/lru-cache>

LRU (Least Recently Used) là một trong những thuật toán "cache replacement" phổ biến nhất và được sử dụng rộng rãi

loại bỏ những phần tử mà lần cuối được sử dụng là xa nhất

LFU (Least Frequently Used) cũng là một thuật toán phổ biến trong "cache replacement"

loại bỏ phần tử có tần suất sử dụng thấp nhất

cái này hơn LRU trong kịch bản

các phần tử mới được thêm vào gần như không được sử dụng sau đó

còn phần tử sử dụng nhiều lần thì giữa 2 lần sử dụng bị "chen" bởi các phần tử mới thêm ở trên

do vậy tỉ lệ phần tử bị dùng nhiều bị xoá cao hơn rất nhiều (cache miss)

trong khi đó nếu dùng tỉ lệ sẽ tăng tỉ lệ (cache hit)

Các config liên quan:

giới hạn bộ nhớ cache trong config /etc/redis/redis.conf

maxmemory 100mb

mỗi một lệnh mà khiến kích thước bộ nhớ tăng, redis check bộ nhớ đã vượt config

nếu vượt thì sẽ tuỳ maxmemory-policy thì key nào sẽ bị "evict" tức loại bỏ khỏi db

khi bộ nhớ tới hạn cần loại bỏ bớt key theo policy /etc/redis/redis.conf

maxmemory-policy X

trong đó X nằm trong các giá trị sau

noeviction không loại key nào khi vượt giới hạn, sẽ báo lỗi khi vượt hạn

volatile-lru loại bỏ xấp xỉ LRU với keys có EXPIRE, với keys không có EXPIRE sẽ không bị chọn

allkeys-lru loại bỏ xấp xỉ LRU với toàn bộ keys

volatile-lfu loại bỏ xấp xỉ LFU với keys có EXPIRE, với keys không có EXPIRE sẽ không bị chọn

allkeys-lfu loại bỏ xấp xỉ LFU với toàn bộ keys

volatile-random loại bỏ random với keys có EXPIRE, với keys không có EXPIRE sẽ không bị chọn

allkeys-random loại bỏ random trên toàn bộ keys

volatile-ttl loại bỏ key với EXPIRE gần nhất (tức TTL - time to live nhỏ nhất)

redis hỗ trợ LRU, LFU xấp xỉ khi chỉ "cố gắng" lấy ra ứng viên tốt /etc/redis/redis.conf cụ thể phía dưới

maxmemory-samples 5

lấy tập ứng viên đủ tốt có kích thước k (với config trên tức là 5)

sau đó chọn ứng viên tốt nhất trong số các ứng viên được chọn

điều này giảm bộ nhớ sử dụng

tăng maxmemory-samples sẽ tăng độ chính xác

## Distributed Lock

<https://redis.io/topics/distlock>

sử dụng redis làm distributed lock trong các hệ thống multi-user, chia sẻ tài nguyên, ...

# Reference

## Links

blog về RDB, AOF khá chi tiết của pháp sư trung hoa

<https://www.fatalerrors.org/a/persistence-mechanism-of-redis-rdb-and-aof.html>

ebook của [redis.com](https://redis.com/) nói về AOF

<https://redis.com/ebook/part-2-core-concepts/chapter-4-keeping-data-safe-and-ensuring-performance/4-1-persistence-options/4-1-2-append-only-file-persistence/>

## Commands

SAVE

lưu database xuống file dump.rdb bằng luồng chính, luồng chính block cho tới khi xong

BGSAVE

lưu database xuống file như bằng background thread, luồng chính không bị block

BGREWRITEAOF

ghi đè lại file appendonly.aof rút gọn lại nhằm giảm thời gian "replay" khi restart redis bằng luồng background

REPLICAOF NO ONE

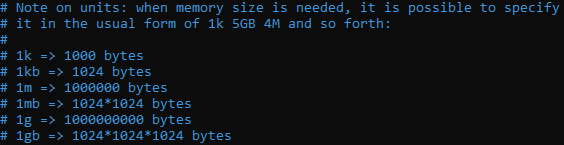
promote slave làm master

BGREWRITE

chủ động rewrite thu gọn lại file AOF

## Config

đơn vị bộ nhớ trong các config



requirepass yourSuperSecretPassword

yêu cầu client cung cấp password để thực thi command

VD: requirepass iloveyou

supervised

If you run Redis from upstart or systemd, Redis can interact with your

supervision tree. Options:

supervised no - no supervision interaction

supervised upstart - signal upstart by putting Redis into SIGSTOP mode

supervised systemd - signal systemd by writing READY=1 to $NOTIFY\_SOCKET

supervised auto - detect upstart or systemd method based on

UPSTART\_JOB or NOTIFY\_SOCKET environment variables

Note: these supervision methods only signal "process is ready."

They do not enable continuous liveness pings back to your supervisor.

## FAQ

1. trong quá trình transfer rdb file thì add thêm dữ liệu vào master ?

add được thôi, file gửi tới slave là snapshot trước khi add dữ liệu

những lệnh add dữ liệu này sẽ thực thi ở master

sau đó khi slave load xong dump thì nó nhận tiếp command từ master như bthg

2. việc tắt chủ động bằng lệnh shutdown thì redis có tự động save (dump snapshot) ?

nếu trong config không có save thì phải có lệnh shutdown save còn mặc định shutdown là không save

nếu trong config có save thì lệnh shutdown sẽ tự động save OK

3. tại sao offset ở slave nó cứ tự dài ra dù không có write operation nào, liệu có phải có một command nào đó không

nếu có command write nào đó thì có vẻ như đây là một command được sử dụng internal

do vậy sẽ không xuất hiện trong những file như AOF

việc offset này liên tục được ghi như một cách để cho thấy slave đang bắt kịp với master

xét trường hợp sau

## TODO

bảng so sánh giữa replication, sentinel, cluster

test lệnh CLIENT PAUSE có như mongo fsyncLock() ?

test cập nhật dns thì slave với config đã bị thay đổi có cập nhật theo không ?

sau đó test là dns trỏ tới một master không nằm trong cụm quản lý của sentinel

(DONE) thử tạo redis cluster với --cluster-replicas = 0 tức chỉ toàn master

được có thể tạo cluster với chỉ toàn master mà không có một slave nào

(DONE) không recreate lại với cụm đã tạo với cùng lệnh trước đó được

test lib ở phía client với sentinel, replication, cluster với python xem ?

thử test an toàn dữ liệu, thất thoát dữ liệu với failover của các mode như sentinel, replication, và cluster

thử không dùng redis-cli mà dùng cluster meet, để khởi tạo cluster ?

sau đó dùng cluster add slots trên các node để tạo sau đó

rc.cluster\_addslots("node id")

(done) tắt hết slave > tắt master > start slave trước

cluster sẽ down toàn bộ, slave không được tự động promote làm master

nếu muốn hoạt động thì có thể start master, hoặc nếu muốn và phải promote slave lên (không tự động promote)

master với trường hợp promote bằng tay sau khi join lại sẽ được convert thành slave

(done) remove file nodes.conf (cluster-config-file) của instance

là toang node đó luôn, tuy nhiên có thể manual fix bằng file được !

do thông tin nếu fresh start sẽ không hợp lệ vì tự gen id mà

cụm sẽ báo lỗi

(done) config epoch để biết config nào là mới nhất từ các node khác