rockPLACE

Redis Introduction

rockPLACE Inc.

NoSQL

2022.08.30

목차

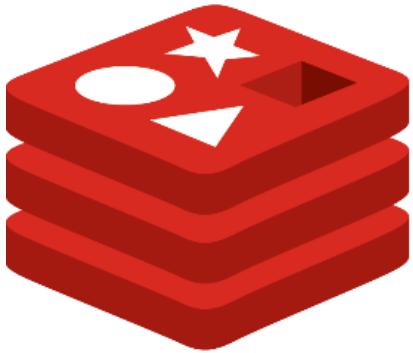
REDIS

What is NoSQL?

Redis?

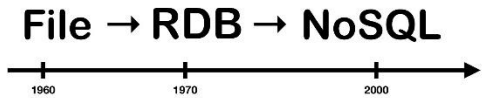
주요기능

Use Case



What is NoSQL?

- **NoSQL(Not Only SQL)란?**
  - 표준 SQL 인터페이스를 사용하지 않는 오픈소스 데이터베이스를NoSQL(Not Only SQL)지칭
  - 최근에는 대용량의 데이터를 처리하거나 다양한 요구사항에 따라 확장성을 고려하여 출시되는 제품



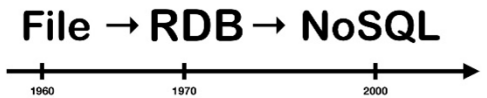
File → RDB

문제점

1. 동시 접근 제한
2. 중복 데이터 관리의 어려움
3. 데이터 분실 가능성
4. 보안
5. …

보완

1. 동시 접근 가능
2. 중복 데이터 제거
3. 접근 권한을 주어 데이터 분실과 보안 문제 해결
4. …



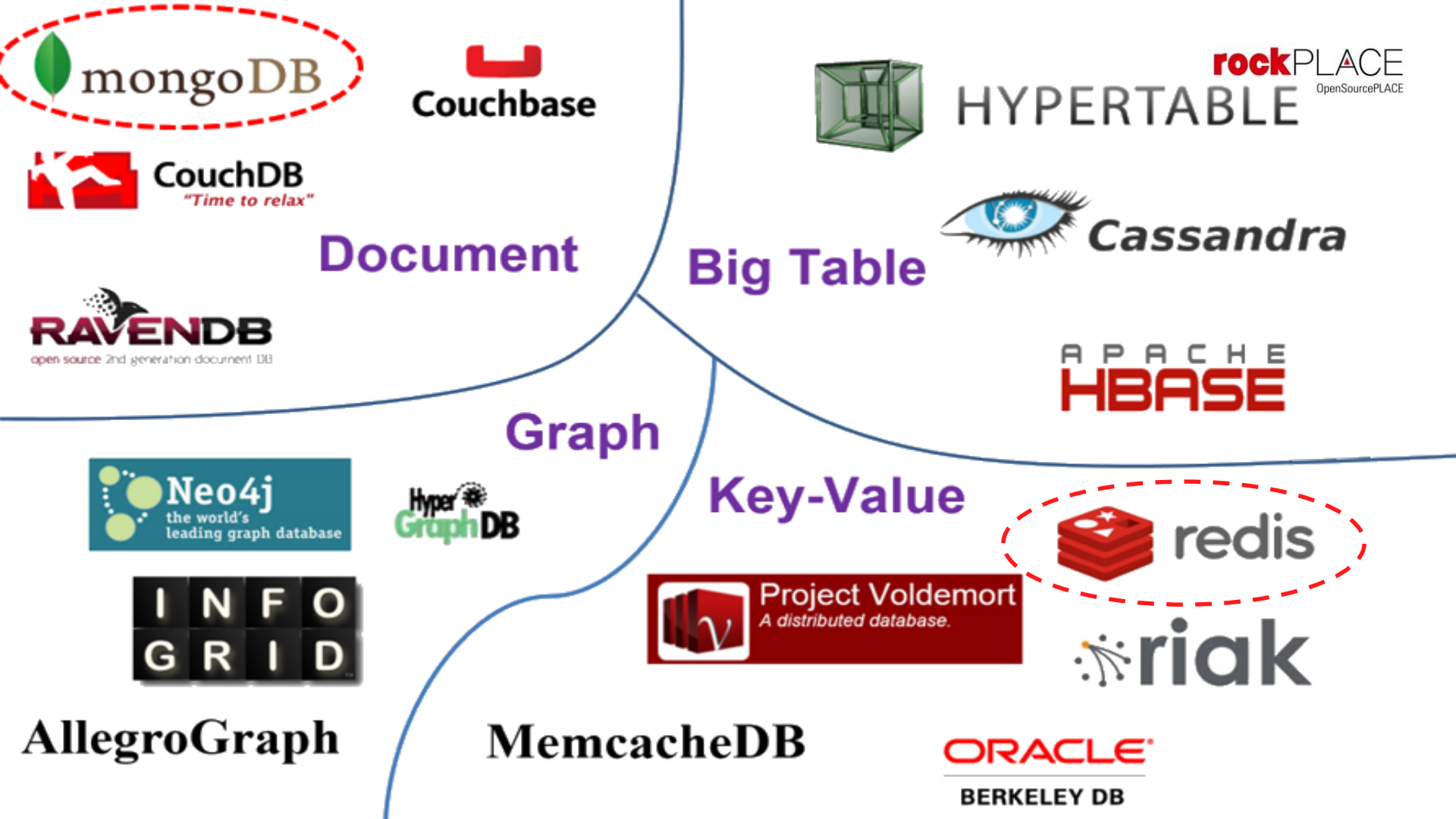
RDB → NoSQL

문제점

1. 폭발적으로 증가하는 데이터
2. 구조가 복잡하다
3. 고정된 **schema**
4. **table** 재구성의 어려움
5. …

보완

1. 높은 확장성
2. 간단한 구조
3. **schema-less**
4. **table X**
5. …


mongoDB
Couchbase
rockPLACE
OpenSourcePLACE
HYPERTABLE
CouchDB
"Time to relax"
Cassandra
Document
Big Table
RAVENDB
open source 2nd generation document DB
APACHE
HBASE
Graph
Key-Value
Neo4j
the world's
leading graph database
HyperGraphDB
redis
INFO
GRID
Project Voldemort
A distributed database.
riak
AllegroGraph
MemcacheDB
ORACLE
BERKELEY DB

Redis

REmote **DI**ctionary **S**erver

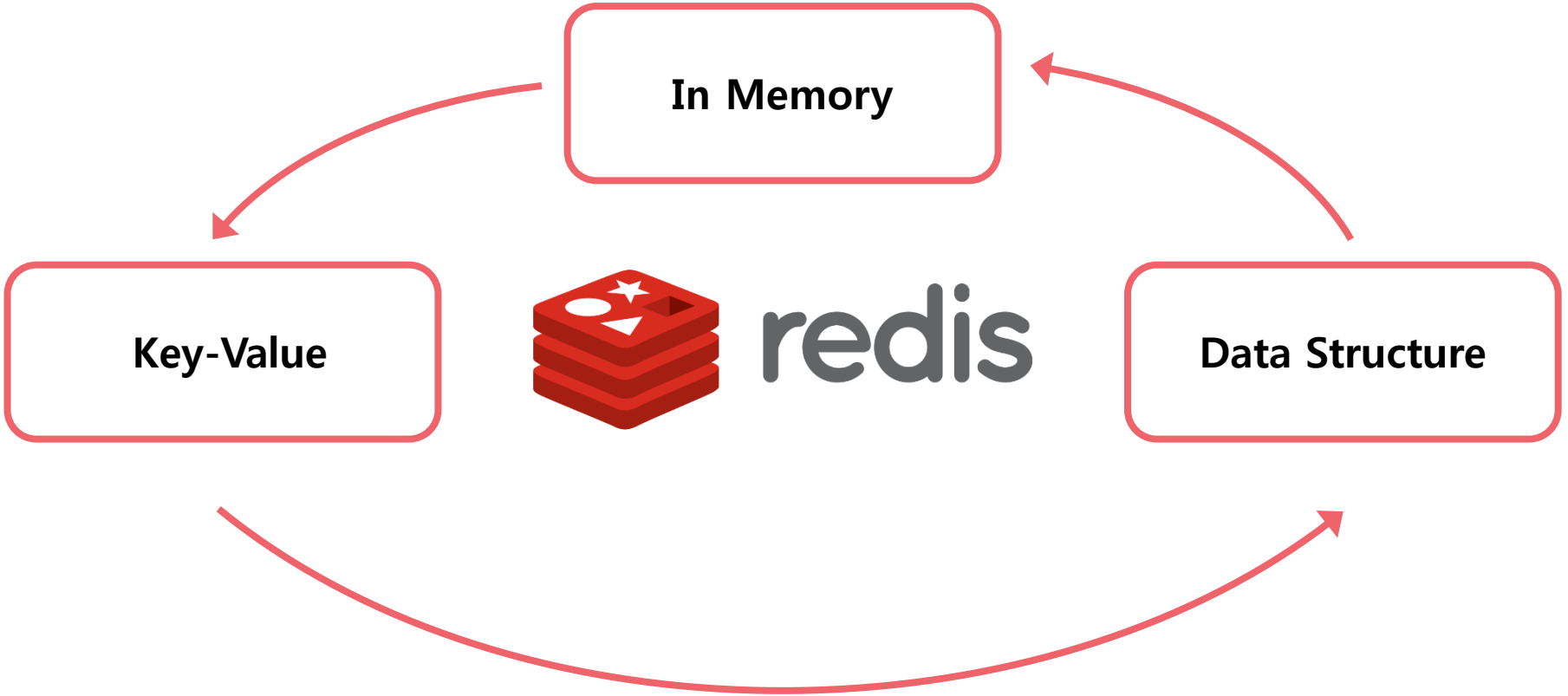
key-value 구조의 비정형 데이터를 저장하고 관리하기 위한
오픈 소스(BSD 라이선스)기반의 비관계형 데이터베이스

Redis

- DB-Engines Ranking(http://db-engines.com/en/ranking, August 2022)

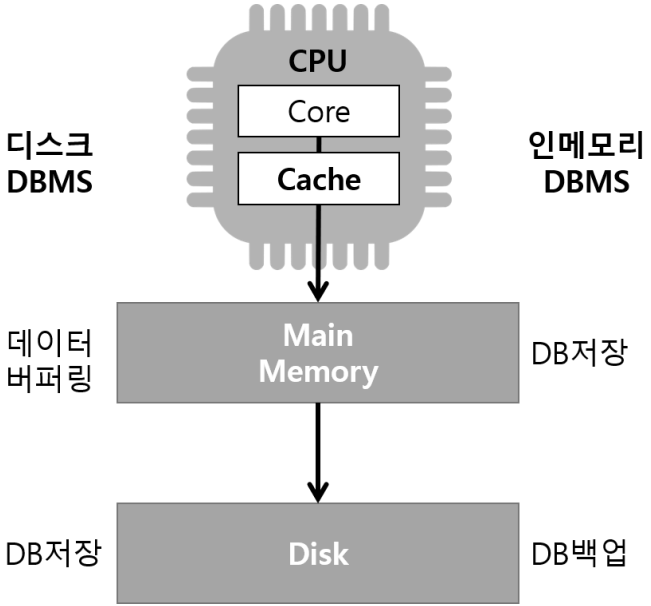
Rank			DBMS	Database Model
Aug 2022	Jul 2022	Aug 2021		
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	6.	Redis	Key-value, Multi-model
7.	7.	7.	IBM Db2	Relational, Multi-model
8.	8.	8.	Elasticsearch	Search engine, Multi-model
9.	9.	↑ 10.	Microsoft Access	Relational
10.	10.	↓ 9.	SQLite	Relational

Redis



Redis

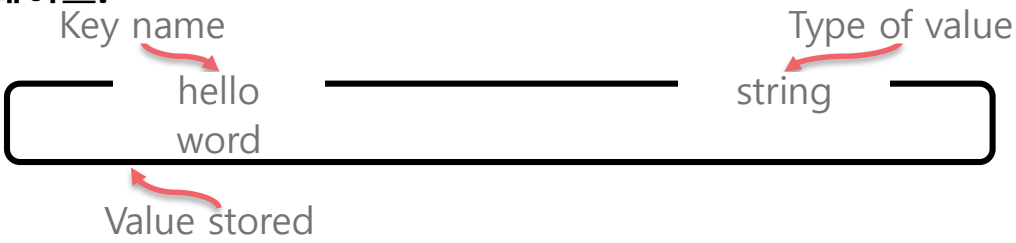
- **NoSQL DBMS로 분류되며 동시에 인메모리 솔루션으로 분리**
  - 인 메모리 DB는 디스크나 SSD에 데이터를 저장하는 기본적DB와 달리 저장을 위해 주로 메모리에 의존하는 목적에 맞게 구축 된 데이터 베이스 유형
- **Database로도 사용할 수 있고, Cache로도 사용 가능**



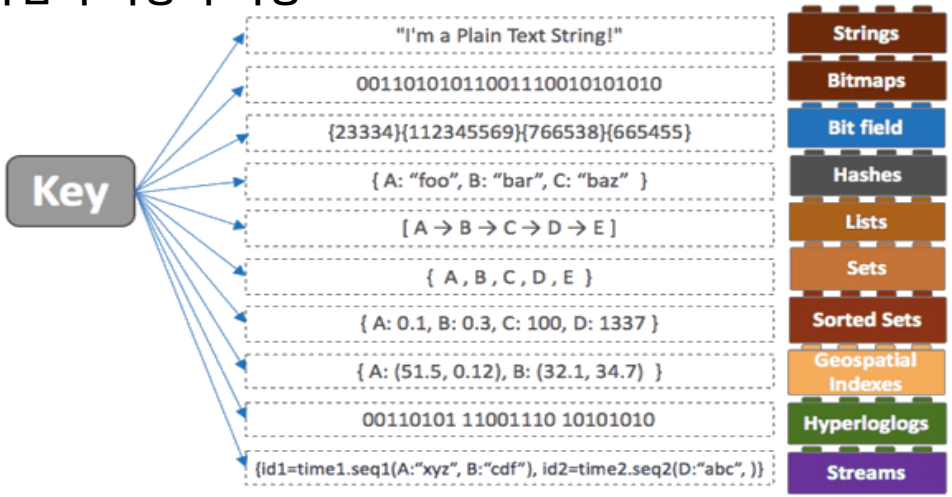
Redis

- **key-value 구조의 비정형 데이터를 저장하고 관리하기 위한 오픈 소스(BSD 라이선스)기반의 비관계형 데이터베이스.**

Key-value Ex) string



- **다양한 데이터 타입의 사용이 가능.**



Redis 기능

In-memory Data Structures

문자열, 해시, 목록, 집합, 스트림 등을 지원하는 "Data Structure Server"

Clustering

hash기반 sharding을 통한 수평 확장성, cluster 확장 시 자동 re-partitioning으로 수백만 개의 노드로 확장 가능

Programmability

Lua 및 Redis 함수로 Redis 확장, 서버 자체에서 사용자 정의 스크립트를 실행할 수 있는 프로그래밍 인터페이스 제공

Persistence

안정성을 제공하기 위해 특정 시점 RDB(스냅샷)과 데이터가 변경될 때마다 이를 디스크에 저장하는 Append Only File(AOF)을 지원

Extensibility

다양한 오픈 소스 클라이언트를 사용할 가능, Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go 외 다수의 언어 지원

High Availability

Replication 과 Clustered 구조에서 Auto Failover 제공, Cluster를 통해 데이터 분산 처리도 수행 가능

Redis

- **Redis는 개발과 운영을 좀 더 쉽고 빠르게 수행할 수 있는 여러 가지 도구를 제공**

  Ex) pub/sub, TTL, modules API ...

- **속도가 빠르고 사용이 간편하여 최고의 성능이 필요한 웹, 모바일, 게임, 광고 기술 및 IoT 애플리케이션에서 사용**
  - Trusted by 1000s of Customers
  - Twitter, GitHub, Weibo, Pinterest, Snapchat, Craigslist, Digg, StackOverflow, Flickr, Naver Line, KaKao Talk

Redis

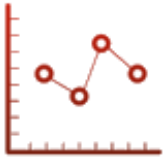
실시간 분석

사용자 세션 저장소

실시간 데이터 수집

고속 거래

작업 및 대기열 관리



시계열 데이터

복잡한 통계 분석

알림

분산 잠금

콘텐츠 캐싱

지리 공간 데이터

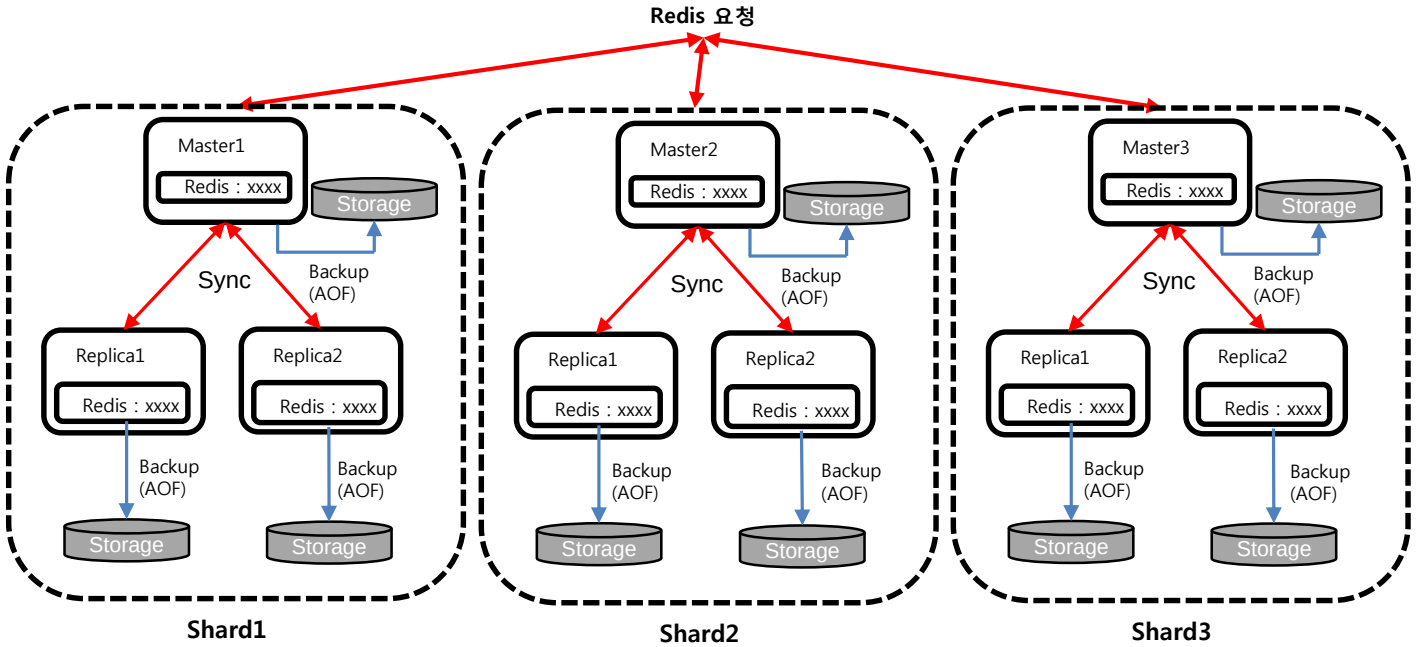
스트리밍 데이터

기계 학습

Redis Use Case

A社 (Redis cluster - 9 node)

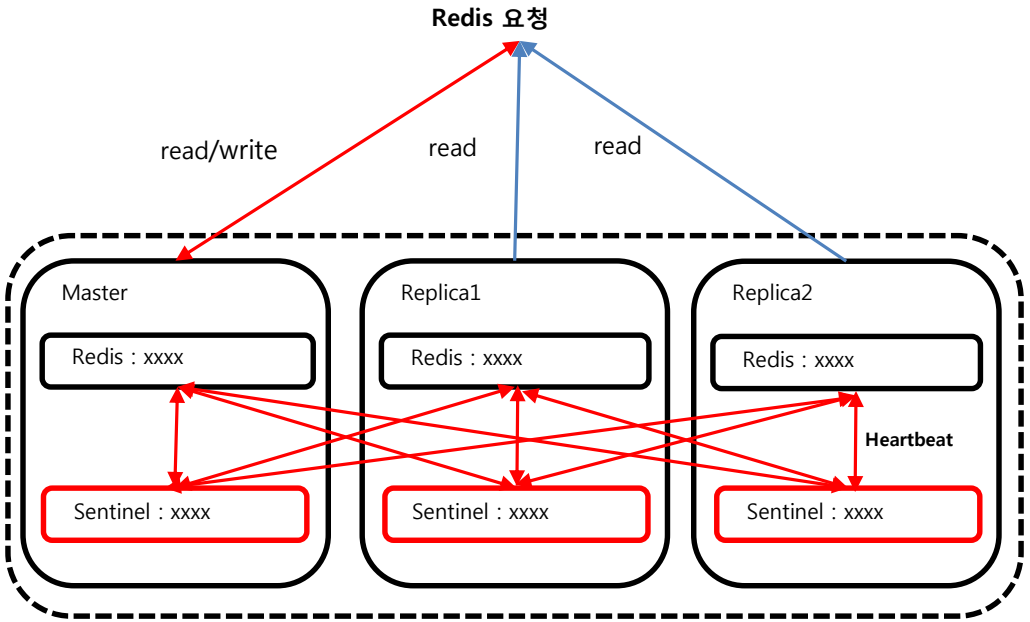
- VOD 추천 서비스



Redis Use Case

B社(replication & Sentinel - 3 node)

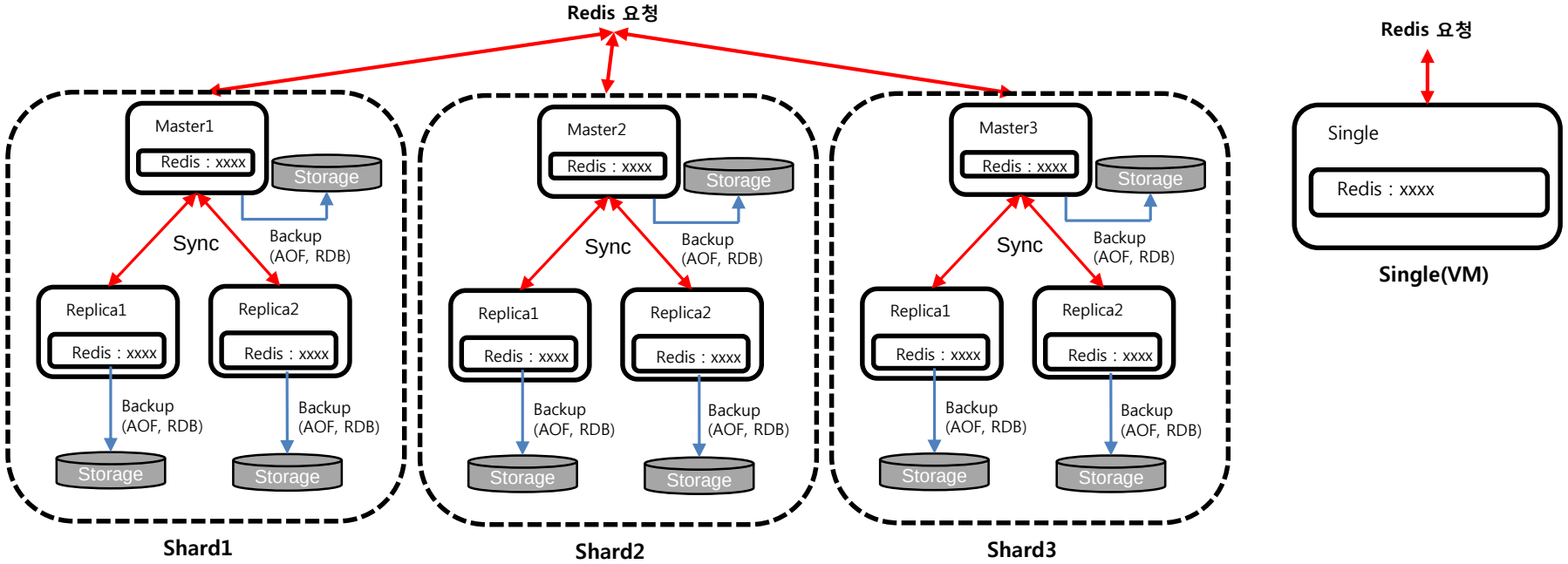
- 상품서비스(마이페이지), 위치기반 관련 서비스 등



Redis Use Case

C社 (Redis cluster 9 node + Single 1 node)

- 운송 서비스

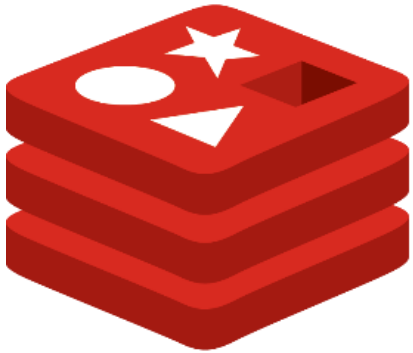


rockPLACE
OpenSourcePLACE

REDIS 시작하기

사전 OS설정

Redis 설치



사전 OS설정

1. TCP back log

TCP backlog: Redis 서버의 초당 클라이언트 연결 개수

Redis 서버 설정 값은 511이나, OS 설정 기본값이 128이라 Redis에서도 128로 설정되기 때문에 리눅스 설정 값 증가 필요

accept()를 기다리는 소켓 개수(Somaxconn)와 backlog사이즈(tcp_max_syn_backlog)를 증가 설정

Somaxconn 설정 후 확인

```
# sysctl -w net.core.somaxconn=4096
4096
# cat /proc/sys/net/core/somaxconn
4096
```

tcp_max_syn_backlog 설정 후 확인

```
# sysctl -w net.ipv4.tcp_max_syn_backlog=4096
net.ipv4.tcp_max_syn_backlog = 4096
# cat /proc/sys/net/ipv4/tcp_max_syn_backlog
4096
```

사전 OS설정

2. Overcommit_memory

Overcommit_memory: 가상 메모리 사용 관련 커널 파라미터

Redis 서버는 메모리 데이터베이스이기 때문에 1로 설정

Overcommit_memory 설정 후 확인

```
# sysctl -w vm.overcommit_memory=1
vm.overcommit_memory = 1
# cat /proc/sys/vm/overcommit_memory
1
```

사전 OS설정

3. Transparent Huge Pages (THP)

THP: huge pages의 생성, 관리, 사용의 대부분을 자동화하는 추상화 계층

리눅스에서는 대량 메모리를 할당하기 위해서 부팅 시 할당하는 Huge page 사용

메모리를 많이 사용하는 Redis의 성능을 저하시키는 요인 중 하나이므로 THP를 disable 설정

THP 설정 후 확인

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
# cat /sys/kernel/mm/transparent_hugepage/defrag
always madvise [never]
```

사전 OS설정

4. Max num of openfile, processes

클라이언트 수가 많이 증가하거나 한 서버 내에서 Redis를 여러 개 띄우다 보면 에러가 발생할 수 있음

에러에 대비해 max open files max processes 값을 변경

Max num 변경 후 확인

```
# vi /etc/security/limits.conf
* soft nofile 65535 (* = 모든유저)
* hard nofile 65535
* soft nproc 4096
* hard nproc 4096
# ulimit -n => 확인
# ulimit -u  => 확인
# ulimit -a  => 전체 확인
```

사전 OS설정

5. Selinux

Selinux 기본정책으로 Redis의 기본 포트인 6379포트 차단

Redis server가 Selinux 정책에 의해 차단 당할 수 있기 때문에 비활성화 권장. Selinux 사용시 6379포트를 연결할 수 있도록 설정 필요.

```
# vi /etc/sysconfig/selinux
SELINUX=disabled
```

6. Swappiness

Swappiness: 스왑 메모리 관련 파라미터

Redis 서버는 인메모리 데이터베이스이기 때문에 스왑을 사용하지 않는 것을 권장

```
# vi /etc/sysctl.conf
vm.swappiness = 0
```

사전 OS설정

7. 리부팅 후 유효 설정

리부팅 후에도 변경한 값이 적용될 수 있도록 sysctl.conf 파일을 수정

```
# vi /etc/sysctl.confnet.core.somaxconn = 4096
net.ipv4.tcp_max_syn_backlog = 4096
vm.overcommit_memory = 1
```

리부팅 후에도 변경한 값이 적용될 수 있도록 /etc/rc.local 파일 적용

```
# vi /etc/rc.local
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

CentOS7에서는 부팅 시 rc.local이 자동 실행되지 않아 추가로 아래와 같이 설정필요

```
# chmod u + x /etc/rc.d/rc.local
# systemctl start rc-local
```

Redis 설치

1. Redis 소스파일 다운로드

Redis를 설치하는 방법은 운영 체제에 따라 다르지만 소스파일을 이용하면 Linux 및 macOS를 포함한 다양한 플랫폼 및 운영 체제의 소스에서 Redis를 컴파일하고 설치 가능

최신 버전의 Redis 소스파일 다운로드

```
# wget https://download.redis.io/redis-stable.tar.gz
```

이전 버전의 Redis 소스파일 다운로드

```
# wget https://download.redis.io/redis-<<version>>.tar.gz
```

Redis의 모든 이전 버전은 아래의 사이트에서 확인가능

- https://download.Redis.io/releases/

Redis 설치

2. Redis 소스파일 컴파일

Redis를 컴파일하려면 먼저 압축을 해제 후 해당 디렉토리로 이동

```
# tar -xzvf redis-stable.tar.gz
# cd redis-stable
```

해당 디렉토리에서 컴파일을 진행

```
# make
```

컴파일에 성공하면 ~/src 하위에 다음을 포함한 여러 Redis 바이너리 확인가능

- Redis-server : Redis 서버 자체
- Redis-cli : Redis와 통신하기 위한 명령 줄 인터페이스 유틸리티

해당 바이너리들을 /usr/local/bin에 설치하려면 아래 명령어 수행

```
# make install
```

보안 및 계정설정

수신가능 IP 설정

Redis는 디폴트 값으로 서버 내부에서만 접속이 가능한 상태

외부에서 접근 시 외부 접속 IP, Redis 서버가 설치된 서버에 Redis 서버가 사용할 IP를 bind에 등록

```
#bind <IP>
```

- 서버에 있는 사용가능한 네트워크 인터페이스IP를 등록
  - 해당 IP로 접속하는 클라이언트만 받아들임

- 파라미터가 지정되지 않은 경우
  - Redis는 서버에서 사용할 수 있는 모든 네트워크 인터페이스에서 연결을 수신

- 해당 파라미터는 최대 16개까지 선택된 인터페이스를 수신 가능

보안 및 계정설정

Protected mode

Redis Config에서 설정 값을 변경하여 protected mode로 설정 가능

```
# protected-mode yes
```

- 인터넷에 열러 있는 Redis 인스턴스에 액세스하여 악용되지 않도록 하기 위해 보안 보호 계층
- 해당 모드 사용시 Redis는 루프 백 인터페이스의 쿼리에만 응답
- 다른 주소에서 연결하는 클라이언트에게는 오류메시지로 응답

보안 및 계정설정

Password 설정

Redis Config에서 Require pass설정 값을 변경하여 Password를 설정할 수 있음

```
# requirepass <password>
```

Redis6버전이상 부터는 Requirepass는 ACL시스템의 상위계층으로 기본 사용자의 비밀번호를 설정
설정 후에는 다른 명령을 실행하지 전에 "AUTH <PASSWORD>" 명령을 통해 접근 필요

```
127.0.0.1:6379> role
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth <password>
OK
```

보안 및 계정설정

접속 IP별 보안설정

protected-mode	bind	requirepass	접속ip
yes	지정ip	Null	지정ip
yes	Null	Null	로컬ip
yes	Null	지정pw	모든ip (지정 pw입력 필요)
no	지정ip	Null	지정ip
no	Null	Null	모든ip

보안 및 계정설정

계정설정

Redis 6.0 버전 이상부터 사용자 관리 명령어로 사용자를 만들고 password를 지정할 수 있는 ACL(Access Control List)명령 사용가능

사용자마다 실행할 수 있는 명령, 액세스 할 수 있는 키, 특정 연결 제한 가능

- ACL SETUSER

```
> ACL SETUSER username [rule [rule ...]]
```

```
127.0.0.1:6379> ACL SETUSER <username> +@all

OK
```

보안 및 계정설정

계정설정

ACL파일을 로드 하여 ACL 규칙을 파일에 정의된 규칙으로 설정가능
파일의 모든 행이 유효하면 모든 ACL규칙이 로드
하나 이상의 행이 유효 하지 않으면 아무것도 로드되지 않으며 기존에 정의된 ACL규칙이 계속 사용

- ACL LOAD

```
> ACL LOAD
```

```
127.0.0.1:6379> ACL LOAD
OK
```

보안 및 계정설정

ACL 규칙

사용자 활성화 및 금지	
on	사용자를 활성화합니다: 이사용자로 인증할 수 있습니다.
off	사용자를 비 활성화합니다: 더 이상 이 사용자로 인증할 수 없지만 이미 인증된 연결은 계속 작동합니다.
명령 허용 및 금지	
+<command>	해당 명령의 실행을 허용합니다.
-<command>	해당 명령의 실행을 허용하지 않습니다.
+@<category>	@admin, @set, @sortedset, ... 등과 같은 유효한 범주가 있는 모든 명령 실행을 허용합니다 ACL CAT 명령 을 호출하여 전체 목록을 확인 할 수 있습니다.특수 범주 @all은 모든 명령을 의미하지만 서버에 현재 존재하며 향후 모듈을 통해 로드 됩니다.
-@<category>	+@<category>클라이언트가 호출 할 수 있는 명령 목록에서 명령을 제거합니다.
+<command>|subcommand	사용하지 않도록 설정된 명령의 특정 하위 명령을 허용합니다.이 양식은 -DEBUG|SEGFUT와 같이 음수로 허용되지 않으며, "+"로 시작하는 가법만 허용됩니다.
allcommands	+@all의 별칭으로 모듈 시스템을 통해 로드되는 모든 향후 명령을 실행할 수 있는 능력을 의미합니다.
nocommands	-@all의 별칭입니다.

보안 및 계정설정

특정 키 허용 및 금지	
~<pattern>	명령의 일부로 언급할 수 있는 키 패턴을 추가합니다. 이 패턴은 KEYS와 같은 글로브 스타일 패턴으로 여러 패턴을 지정할 수 있습니다.
allkeys	~ *의 별칭입니다.
resetkeys	허용된 키 패턴 목록을 비웁니다.
비밀번호 구성	
><password>	사용자의 암호목록에 암호를 추가합니다. 사용자는 원하는 수의 암호를 가질 수 있습니다.
<<password>	사용자의 암호목록에 암호를 제거합니다. 유효하지 않은 값을 삭제할 경우 오류가 발생합니다.
#<hash>	SHA-256 해시 값을 사용자의 유효한 암호 목록에 추가합니다. 이 해시 값은 ACL 사용자에 대해 입력 한 암호의 해시와 비교됩니다. 이를 통해 사용자는 일반 텍스트 암호를 저장하는 대신 파일에 해시를 저장할 수 있습니다.
!<hash>	사용자의 암호목록에 해시 값을 제거합니다.
nopass	사용자의 모든 설정 암호가 제거되고 사용자에게 암호가 필요하지 않은 것으로 표시됩니다. 이 지시문이 기본 사용자에 사용되는 경우 명시적인 AUTH 명령이 필요하지 않고 모든 새 연결이 기본 사용자로 즉시 인증됩니다
resetpass	허용된 비밀번호 목록을 비우고 no pass상태를 제거합니다.

보안 및 계정설정

계정설정

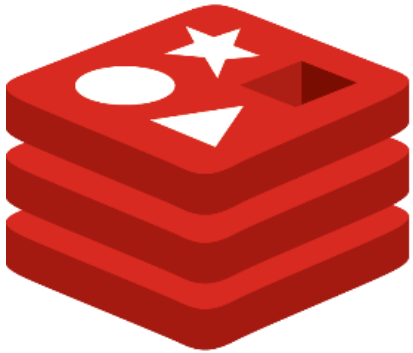
계정 권한제거

```
> ACL SETUSER username reset
```

```
127.0.0.1:6379> ACL SETUSER <username> reset

+OK

127.0.0.1:6379> ACL LIST

1) "user <username> off -@all"
```

REDIS 운영

시작/종료
CRUD
STATUS



Redis 시작하기

Redis Server 실행

Redis 설치 경로의 Redis 실행 파일인 Redis-server를 실행

```
/Redis설치경로/src/Redis-server /Redis설치경로/Redis.conf
```

```
# ./src/Redis-server ./Redis.conf

27271:C 14 Nov 2020 14:16:29.770 # oO0OoO0OoO0Oo Redis is starting

27271:C 14 Nov 2020 14:16:29.770 # Redis version=6.0.8, bits=64, commit=00000000, modified=0, pid=27271, just started

27271:C 14 Nov 2020 14:16:29.770 # Configuration loaded
```

Redis 시작하기

Redis Server 기동 확인

아래 명령어를 통해 Redis Server 기동확인

```
Shell> ps -ef | grep Redis
```

```
# ps -ef|grep Redis
root      27272      1   0 14:16 ?        00:00:00 ./src/Redis-server 127.0.0.1:6379
```

Redis 시작하기

Redis 접속

Redis에서 제공하는 tool(Redis-cli)을 통해 Redis Server에 접속

```
Shell> /Redis설치경로/src/Redis-cli -p <Redis_port>
```

```
# ./src/Redis-cli -p 6379

127.0.0.1:6379> ping

PONG
```

Redis 종료

Redis Server 종료

1) Redis 명령을 이용한 Redis Server 종료

Redis Server 접속 후 Shutdown 명령을 이용하여 Redis Server를 종료 가능

```
> shutdown
```

```
127.0.0.1:6379> shutdown

not connected> exit
```

Redis 종료

Redis Server 종료

2) Kill 명령을 이용한 Redis Server 종료

* OS상에서 Redis 프로세스를 강제 종료 가능

```
Shell> kill <process_num>
```

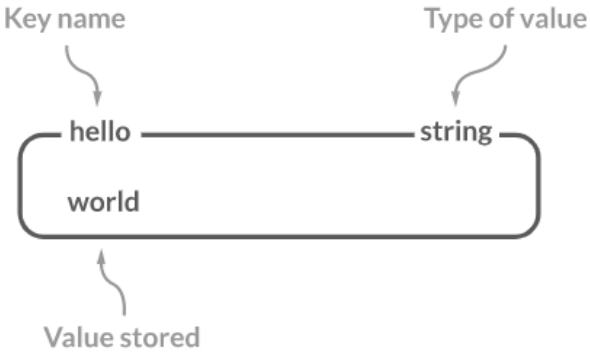
```
# ps -ef|grep Redis
root      27307      1   0 14:19 ?        00:00:00 ./src/Redis-server 127.0.0.1:6379
root      27314 19133   0 14:19 pts/1    00:00:00 grep --color=auto Redis
# kill 27307
```

Redis CRUD

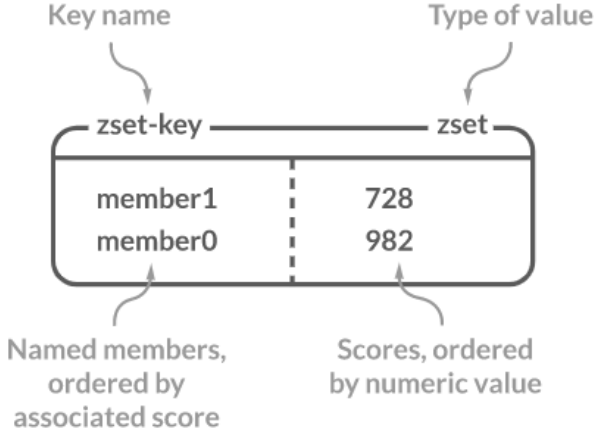
단순한 key-value 데이터 저장소와 달리 Redis 데이터 구조는 최신 애플리케이션의 다양한 사용 사례에 대해 데이터를 모델링하는 유연한 방법 제공

모든 데이터 구조에 대해 다중 유형 작업을 효과적인 방식으로 실행할 수 있도록 전용 명령을 유지 관리 지원

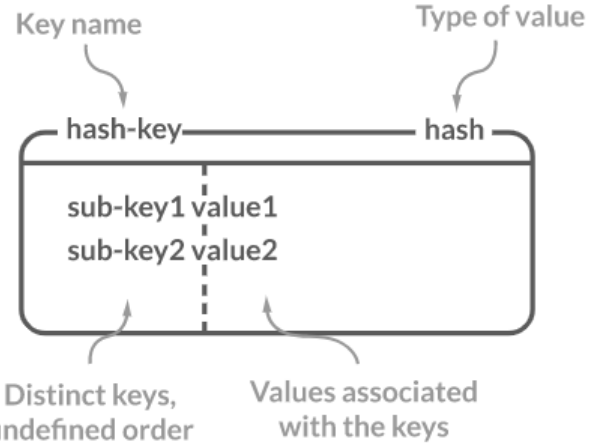
Strings



Sorted Sets



Hashes



●●●

Redis CRUD

String 방식

String 방식은 Redis의 가장 기본적인 데이터 타입이며, 텍스트, 직렬화 된 객체 및 이진 배열을 포함한 바이트 시퀀스를 저장 가능

데이터 삽입

```
>SET key value
```

```
127.0.0.1:6379> set testKEY testVALUE

OK
```

Redis CRUD

String 방식

데이터 조회

>GET key
>'keys *' (select * from table) /모든 키 조회 시

```
127.0.0.1:6379> get testKEY

"testVALUE"

127.0.0.1:6379> keys *

1) "testKEY"

2) "KEY5"

3) "KEY4"

....
```

Redis CRUD

String 방식

데이터 삭제

```
>DEL key
```

```
127.0.0.1:6379> del testKEY

(integer) 1

127.0.0.1:6379> keys *

1) "KEY5"

2) "KEY4"

....
```

Redis CRUD

List 방식

Lists 방식은 단순히 문자열의 목록, 삽입 순서대로 정렬되며 주로 큐(Queue)와 스택(Stack)으로 사용

데이터 삽입(LPUSH: 왼쪽에서 삽입)

```
>LPUSH key value
```

```
127.0.0.1:6379> LPUSH listkey value1
(integer) 1
```

데이터 삽입(RPUSH: 오른쪽에서 삽입)

```
> RPUSH key value
```

```
127.0.0.1:6379> RPUSH listkey value2
(integer) 2
```

Redis CRUD

List 방식

데이터 조회

```
>LRANGE key start stop
```

```
127.0.0.1:6379> LRANGE listkey 0 -1  # listkey list의 모든 값을 출력합니다.
1) "value5"
2) "value4"
....
5) "value1"
127.0.0.1:6379> LRANGE listkey 0 2  # listkey list의 0~2번째 값을 출력합니다.
1) "value5"
2) "value4"
3) "value3"
```

Redis CRUD

List 방식

데이터 삭제

Count > 0 : 왼쪽에서 오른쪽으로 이동하며 요소와 같은 요소를 제거합니다.

Count < 0 : 오른쪽에서 왼쪽으로 이동하며 요소와 같은 요소를 제거합니다.

```
>LREM key count value
```

```
127.0.0.1:6379> LREM listkey 1 value5

(integer) 1
```

Redis CRUD

Sets방식

Sets는 key와 value가 일 대 다 관계로 Value는 입력된 순서와 상관없이 저장되며 중복되지 않음
(Sets 에서는 집합이라는 의미에서 value를 member라고 부름)

데이터 삽입

```
>SADD setkey member
```

```
127.0.0.1:6379> SADD setkey member1
(integer) 1
```

Redis CRUD

List 방식

데이터 조회

```
>SMEMBERS setkey
```

```
127.0.0.1:6379> SMEMBERS setkey
1) "member1"
2) "memberr2"
3) "member3"
```

Redis CRUD

List 방식

데이터 삭제

```
>SREM key member
```

```
127.0.0.1:6379> SREM setkey member1

(integer) 1
```

Redis 상태확인

Redis Info

Redis 서버에 접속하여 info 명령어를 통해 Redis 서버 정보와 통계 값 확인가능

```
>info
```

```
127.0.0.1:6379> info
# Server
Redis_version:6.0.8
...
# Clients
connected_clients:3
....
# Memory
used_memory:1985384
....
```

Redis 상태확인

Redis Info

일부 세션만 선택해서 정보를 확인가능

```
>info <section>
```

```
127.0.0.1:6379> info cluster

# Cluster

cluster_enabled:0


127.0.0.1:6379> info clients

# Clients

connected_clients:3

client_recent_max_input_buffer

.....
```

Redis 상태확인

- **Server**

Redis 서버에 대한 일반적인 정보

redis_version	**Redis 서버의 버전**
redis_git_sha1	Git SHA1
redis_git_dirty	Git dirty flag
redis_build_id	빌드 ID
redis_mode	**서버 모드("standalone", "sentinel", "cluster")**
os	Redis 서버를 호스팅하는 운영 체제
arch_bits	아키텍처 (32 비트 또는 64 비트)
multiplexing_api	Redis에서 사용하는 이벤트 루프 메커니즘
atomicvar_api	Redis에서 사용하는 Atomicvar API
gcc_version	컴파일하는 데 사용되는 GCC 컴파일러의 버전
process_id	서버 프로세스의 PID
run_id	Redis 서버를 식별하는 임의 값 (Sentinel 및 Cluster에서 사용)
tcp_port	TCP / IP 수신 포트

Redis 상태확인

- **Server**

uptime_in_seconds	Redis 서버 시작 이후 경과 된 시간 (초)
uptime_in_days	**Redis 서버 시작 이후 경과 된 시간 (일)**
hz	timer interrupt, 값의 범위는 1 ~ 500 이며 각종 background 작업을 호출하는 주기
configured_hz	서버에 구성된 주파수 설정
lru_clock	LRU 관리를 위해 1 분마다 증가하는 시계
Executable	**서버 실행 파일의 경로**
config_file	**구성 파일의 경로**
io_threads_active	I / O 스레드가 활성 상태인지 여부를 나타내는 플래그

Redis 상태확인

- **Client**

Redis 서버의 클라이언트 연결 정보

connected_clients	**클라이언트 연결 수 (Replica 연결 제외)**
client_recent_max_input_buffer	최근 입력 버퍼 최대 사용량
client_recent_max_output_buffer	최근 출력 버퍼 최대 사용량
blocked_clients	명령으로 대기중인 클라이언트 수
tracking_clients	추적중인 클라이언트 수
clients_in_timeout_table	클라이언트 시간 초과 테이블의 클라이언트 수

Redis 상태확인

- **Memory**

Redis 서버의 메모리 사용에 대한 정보

used_memory	**할당자를 사용하여 Redis에서 할당 한 총 바이트 수** **_human : 사람이 읽기 편한 단위로 표시**
used_memory_rss	실제 real memory를 차지하고 있는 양 **_human : 사람이 읽기 편한 단위로 표시**
used_memory_peak	Redis에서 사용하는 최대 메모리 (바이트) **_human : 사람이 읽기 편한 단위로 표시** **_perc: 백분율 값(**used_memory / used_memory_peak**)**
used_memory_overhead	서버가 내부 데이터 구조를 관리하기 위해 할당 한 모든 오버 헤드의 합계 (바이트)
used_memory_startup	클라이언트 시간 초과 테이블의 클라이언트 수
used_memory_dataset	시작 시 Redis에서 사용한 초기 메모리 양 (바이트) _perc: 백분율 값(used_memory- used_memory_startup)
total_system_memory	Redis 호스트에 있는 총 메모리 양 **_human : 사람이 읽기 편한 단위로 표시**
used_memory_lua	Lua 엔진에서 사용하는 메모리양 _human : 사람이 읽기 편한 단위로 표시
used_memory_scripts	캐시 된 Lua 스크립트에서 사용하는 바이트 수 **_human : 사람이 읽기 편한 단위로 표시**

Redis 상태확인

- **Memory**

maxmemory	maxmemory정책
maxmemory_policy	LRU 관리를 위해 1 분마다 증가하는 시계
allocator_frag_ratio	조각모음을 시작할 최소 조각화 비율
allocator_frag_bytes	조각모음을 시작하기 위한 최소량
mem_fragmentation_ratio	used_memory_rs와 used_memory 사이의 비율
mem_allocator	컴파일 타임에 선택된 메모리 할당 자
active_defrag_running	활성 조각 모음이 활성화되었는지 여부를 나타내는 플래그
lazyfree_pending_objects	해제 대기중인 객체 수

Redis 상태확인

- **Persistence**

RDB와 AOF에 대한 정보

loading	**덤프 파일의로드가 진행 중인지 여부를 나타내는 플래그, 읽어 들이고 있으면 1, 아니면 0**
rdb_changes_since_last_save	RDB 파일을 마지막으로 저장한 이후로 입력된 데이터 건수
rdb_bgsave_in_progress	RDB 저장이 진행 중임을 나타내는 플래그, background save가 진행중일 때 1, 아니면 0
rdb_last_save_time	마지막 성공적으로 RDB 파일을 저장한 시간
rdb_last_bgsave_status	**마지막 RDB 저장 작업의 상태(ok/err)**
rdb_last_bgsave_time_sec	마지막 RDB 저장 작업의 기간 (초)
rdb_current_bgsave_time_sec	진행중인 RDB 저장 작업의 기간
rdb_last_cow_size	마지막 RDB 저장 작업 중 쓰기 시 복사 할당 크기
aof_enabled	**AOF 로깅이 활성화 여부, appendonly yes 이면 1, no 면 0**
aof_rewrite_in_progress	AOF rewrite 작업 진행여부, 진행중이면 1로 표시
aof_rewrite_scheduled	AOF rewrite 예정여부, 예정되어 있으면 1로 표시 RDB 파일을 저장 중이면 동시에 AOF 파일을 저장할 수 없습니다.

Redis 상태확인

- **Persistence**

aof_last_rewrite_time_sec	마지막 AOF rewrite 소요 시간
aof_current_rewrite_time_sec	AOF rewrite가 진행중이면 시작부터 현재까지 시간(초)가 표시
aof_last_bgrewrite_status	마지막 AOF background rewrite 작업의 상태(ok/err)
aof_last_write_status	**마지막 AOF write 상태(ok/err)**
aof_last_cow_size	마지막 AOF rewrite 작업 중 쓰기 복사 할당 크기

Redis 상태확인

- **Stats**

Redis서버에 대한 일반 통계 정보

total_connections_received	**서버에서 수락 한 총 연결 수**
total_commands_processed	서버에서 처리 한 총 명령 수
instantaneous_ops_per_sec	**초당 처리 된 명령 수**
total_net_input_bytes	네트워크에서 읽은 총 바이트 수
total_net_output_bytes	네트워크에 기록 된 총 바이트 수
instantaneous_input_kbps	네트워크의 초당 읽기 속도 (KB / 초)
instantaneous_output_kbps	네트워크의 초당 쓰기 속도 (KB / 초)
rejected_connections	**maxclients 제한으로 거부된 접속 수**
sync_full	**마스터 노드일 때 슬레이브 노드와 full sync 횟수**
sync_partial_ok	**마스터 노드일 때 슬레이브 노드와 partial sync ok 횟수**

Redis 상태확인

- **Replication**

Master/Replica에 대한 정보

total_connections_received	**서버에서 수락 한 총 연결 수**
total_commands_processed	서버에서 처리 한 총 명령 수
instantaneous_ops_per_sec	**초당 처리 된 명령 수**
total_net_input_bytes	네트워크에서 읽은 총 바이트 수
total_net_output_bytes	네트워크에 기록 된 총 바이트 수
instantaneous_input_kbps	네트워크의 초당 읽기 속도 (KB / 초)
instantaneous_output_kbps	네트워크의 초당 쓰기 속도 (KB / 초)
rejected_connections	**maxclients 제한으로 거부된 접속 수**
sync_full	**마스터 노드일 때 슬레이브 노드와 full sync 횟수**
sync_partial_ok	**마스터 노드일 때 슬레이브 노드와 partial sync ok 횟수**

Redis 상태확인

- **Replication**

-Master에서만 나오는 정보

connected_slaves	**master/slave 중 하나로 표시, slave지만 다른 slave의 master인 경우 slave로 표시**
slave0	연결된 Replica의 정보, 연결된 Replica의 수만큼 표시 (Ip,Port,state)

-Replica에서만 나오는 정보

master_host	**마스터의 호스트 또는 IP 주소**
master_port	연결된 Replica의 정보, 연결된 Replica의 수만큼 표시 (Ip,Port,state)
master_link_status	**Master노드와 연결 상태 정상 up, 비정상 down**
…	

Redis 상태확인

- **CPU**

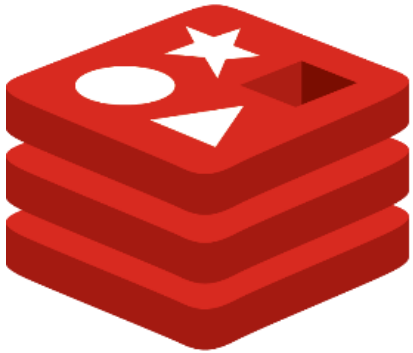
Redis 서버의 CPU소비 통계

used_cpu_sys	**시스템 모드에서 사용한 CPU 시간**
used_cpu_user	사용자 모드에서 사용한 CPU 시간
used_cpu_sys_children	백그라운드 프로세스에서 사용하는 시스템 CPU 시간
used_cpu_user_children	백그라운드 프로세스에서 사용하는 사용자 CPU 시간

PERSISTENCE

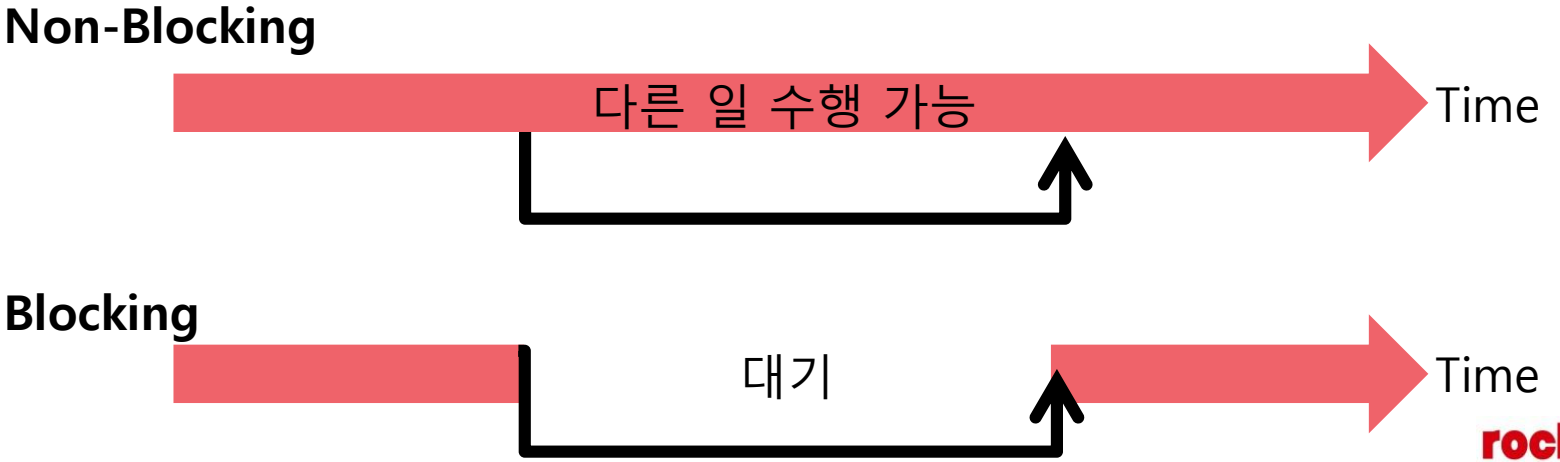
AOF

RDB



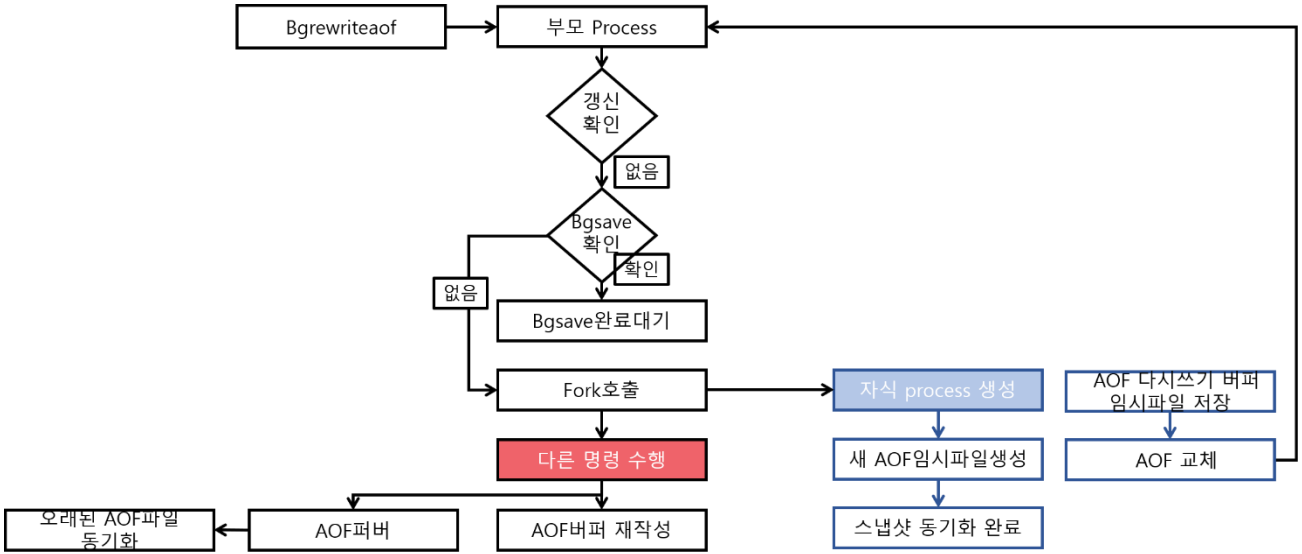
Redis 기능 - Persistent

- **Redis는 Memcached와 달리 데이터를 디스크로 저장할 수 있는 Persistent 기능을 제공**
  디스크에 저장되어 있는 데이터를 기반으로 장애 시 다시 복구가능
- **Append On File(AOF) 방식**
  - 입력/수정/삭제 명령이 실행될 때마다 연산 자체를 모두 log 파일에 기록하는 방식
  - 연산이 발생할 때마다 매번 기록하기 때문에, RDB 방식과는 달리 특정 시점이 아니라 항상 현재 시점까지의 로그를 기록가능, 기본적으로 non-blocking 방식



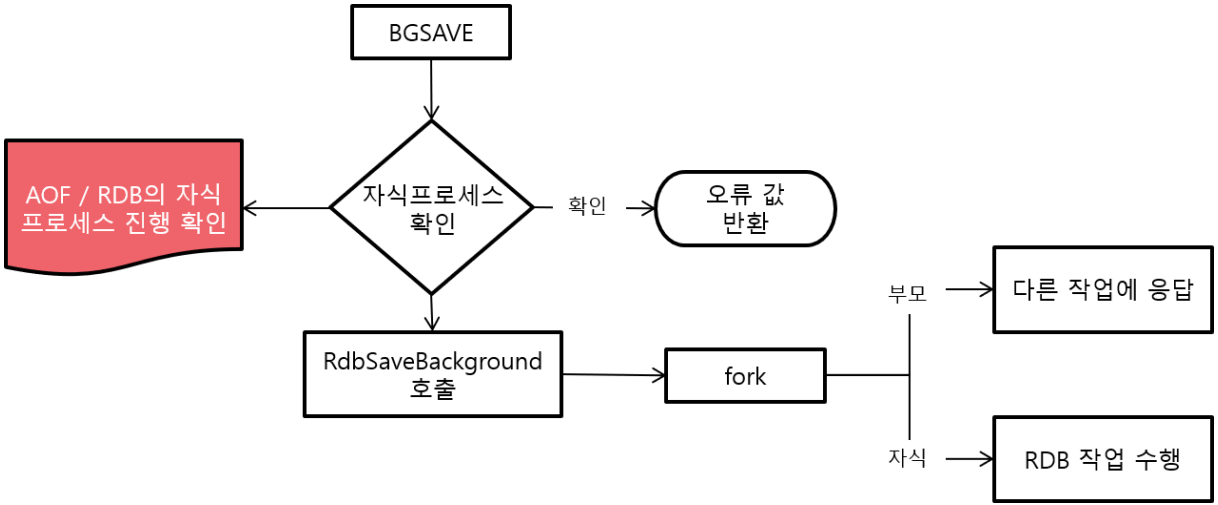
Redis 기능 - Persistent

- **Append On File(AOF) 방식**
  - 서버가 restart될 때 기록된 연산을 순차적으로 재실행하여 데이터를 복구
  - 파일의 사이즈가 계속 커지면 OS파일 제한에 걸릴 수 있고, 로드 시간이 많이 소요되어 AOF파일 사이즈가 100%커지면 rewrite 진행



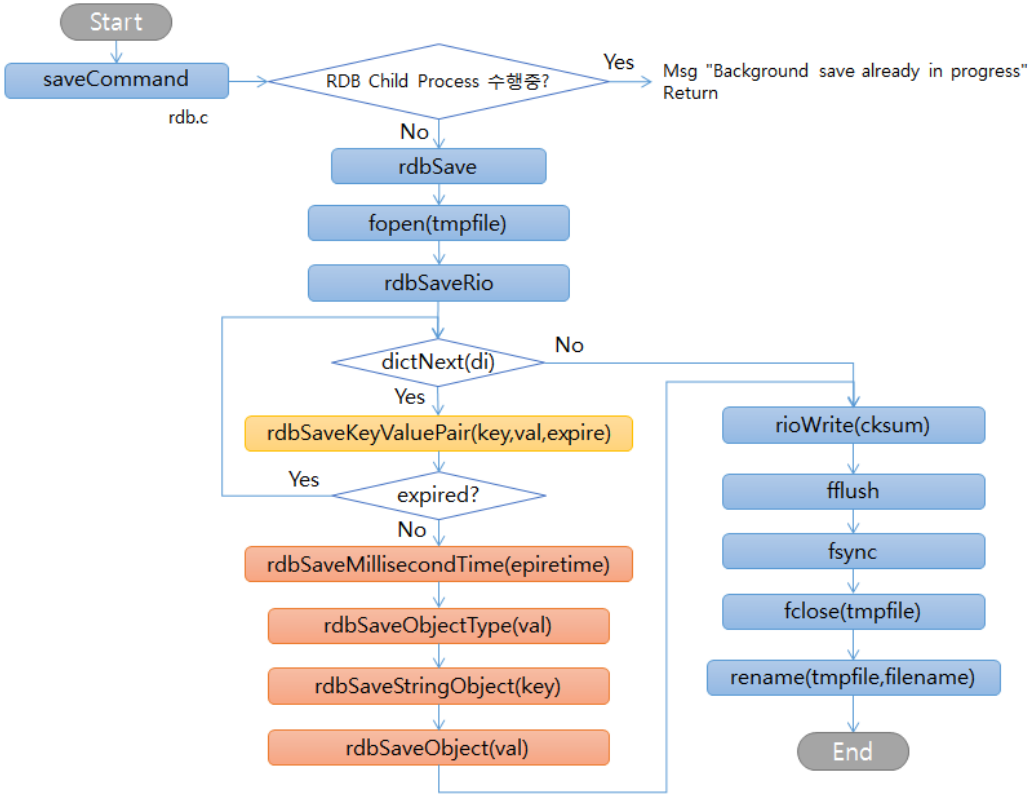
Redis 기능 - Persistent

- **RDB 방식**
  - 메모리에 있는 내용을 디스크에 옮겨 담는 방식
  - RDB 저장을 위한 명령으로 SAVE 와 BGSAVE 두가지 존재
    - BGSAVE: non-blocking 방식으로 별도의 process를 띄운 후, 명령어 수행 당시의 snapshot을 디스크에 저장하여 Redis는 동작을 멈추지 않고 정상적으로 동작가능



Redis 기능 - Persistent

- SAVE: blocking 방식, 순간적으로 Redis의 모든 동작을 정지시키고 snapshot을 디스크에 저장



Redis 기능 - Persistent

- **RDB & AOF**

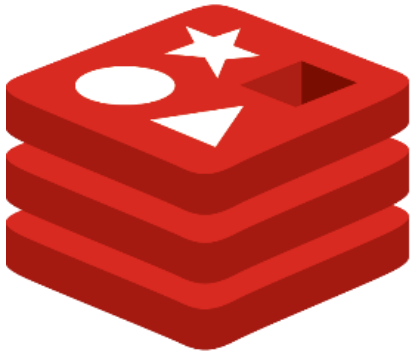
RDB	AOF
시스템 자원이 최소한으로 요구 (특정 시점에 쓰기 작업 발생)	시스템 자원이 집중적으로 요구 (지속적 쓰기 작업이 발생)
특정 시점마다 저장이 이루어져 지속성이 떨어짐	마지막 시점까지 데이터 복구가 가능
복구 시간이 빠름	대용량 데이터 파일로 복구 작업 시 복구 성능이 떨어짐
저장공간이 압축되기 때문에 최소 필요 (별도의 파일 압축이 필요 없음)	저장 공간이 압축되지 않기 때문에 데이터 양에 따라 크기 결정

REPLICATION

Replication?

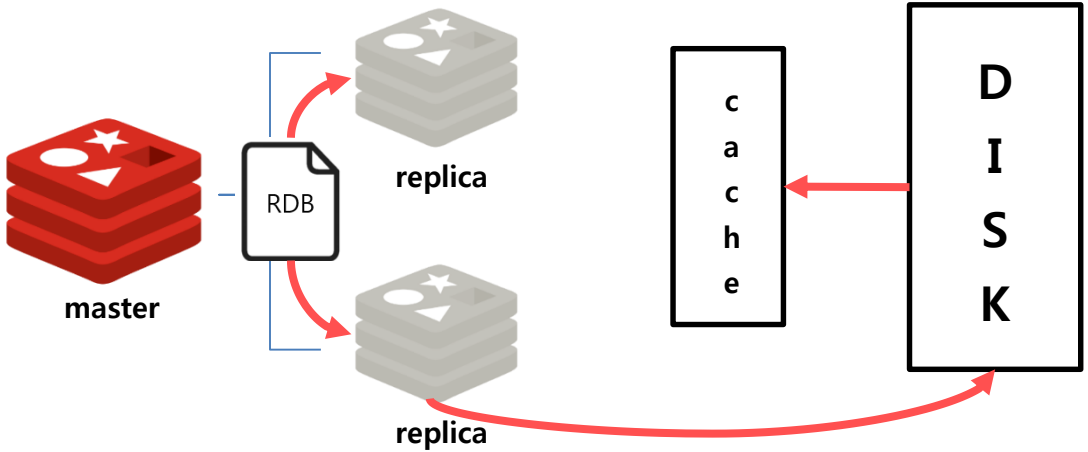
Replication 구성

운영



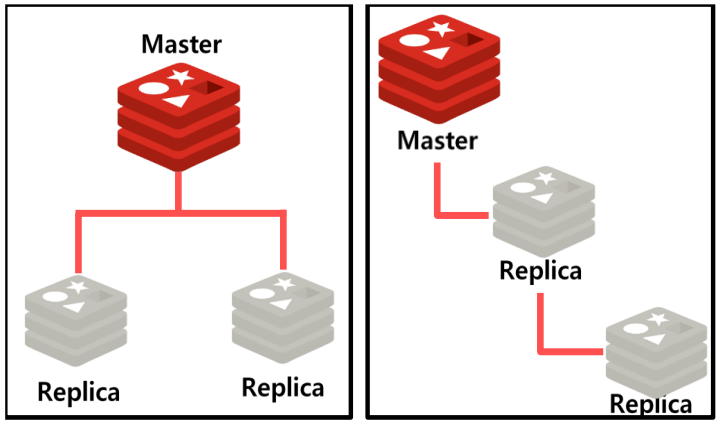
Redis Replication

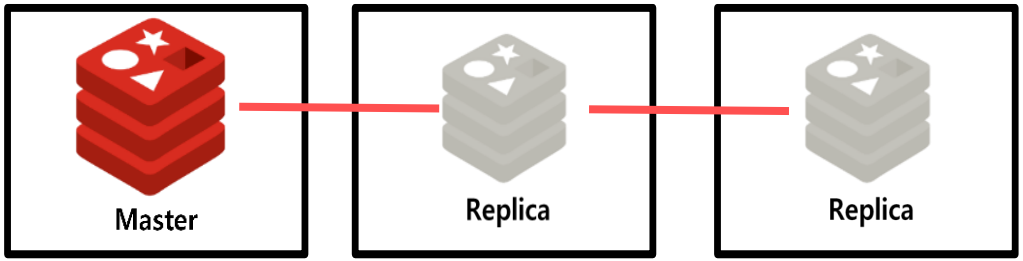
- Redis는 Replication을 구성하여 failover 상황에 대비 가능
- 기본적으로 Replication은 Master/replica 구조로 비동기 복제를 사용
  Master는 Read/Write 모두 수행이 되고, replica는 Master의 데이터를 복제하고 있으며 read 전용
- Master는 replica에 해당 RDB 파일을 전달, replica는 디스크에 저장한 후에 메모리로 로드
- Auto-failover를 지원하지 않으므로 장애 발생시 사용자가 수동으로 설정을 해줘야 하는 단점
  -> auto-failover를 지원하는 기능인 sentinel사용 권장



Redis Replication

- Replication은 기본적으로 Master – Replica 구조, Replica 노드는 다수로 구성가능.
- Replication 구성시에는 최소 1 Master & 2 Replica 구성으로 총 3노드로 구성하는 것을 권장
- 데이터의 손실을 최소화 하기위해서 Replica는 Master와 물리적으로 분리된 3개의 서버에 2노드 이상으로 구성하는 것을 권장





Redis Replication 구성

- **Replicaof**

```
# replicaof <masterip> <masterport>
```

Redis인스턴스를 다른 Redis서버의 Replica로 만듭니다. Master의 정보를 적으면 해당 서버의 데이터를 실시간으로 복사

- **Masterauth**

```
# masterauth <master-password>
```

Master가 "requirepass"구성 지시문을 사용하여 비밀번호로 보호되어 있는 경우 복제 동기화 프로세스를 시작하기 전에 Replica에 인증하도록 지시가능. 그렇지 않으면 Master가 Replica의 요청을 거부

- **repl-diskless-sync**

```
# repl-diskless-sync <yes/no>
```

Replica서버가 시작할 때 또는 재 접속할 때 Master에서 Replica로 전체 동기화.이때 Master서버는 RDB파일을 생성해 Replica 서버로 전송하며 두가지 방식으로 전송가능

Redis Replication 구성

•repl-ping-replica-period

```
# repl-ping-replica-period<time>
```

Replica는 미리 정의된 간격으로 Master에게 Ping을 전송. 기본 설정 값은 10초이며 repl-ping-replica-period파라미터를 사용하여 이 간격을 변경 가능.

•repl-timeout

```
# repl-timeout <time>
```

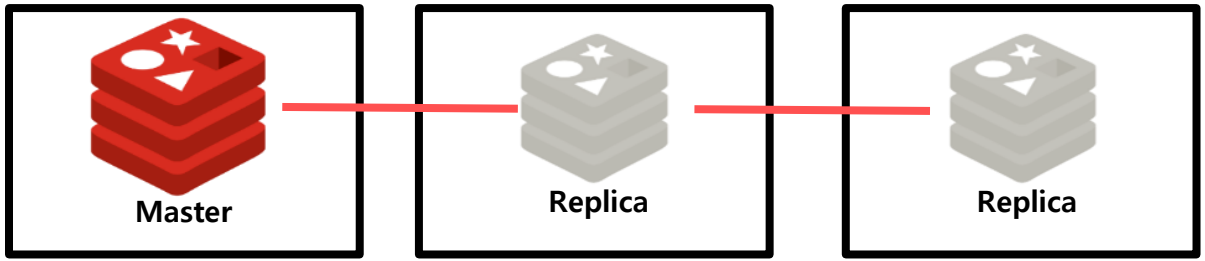
Master와 Salve의 연결이 끊겼다고 인식하는 시간

Repl-timeout 값이 더 크다면 Master와 Replica간에 트래픽이 적을 때마다 시간 초과가 감지되기 때문에 반드시 이 값이 repl-ping-replica-period에 지정된 값보다 큰지 확인해야 함

해당 파라미터는 Master와 Replica관점으로 나누어 봐야함.

Redis Replication 운영

Redis Server 실행



Replication 구조에서는 데몬 구동은 반드시 **Master->Replica** 순으로 구동

```
/Redis설치경로/src/Redis-server /Redis설치경로/Redis.conf
```

Redis Replication 운영

•**Redis Sentinel 구성 확인**

Master 에서 구성확인

```
Ip:port> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=<n.n.n.n>,port=<nnnn>,state=online,offset=1051667111,lag=0
slave1:ip=<n.n.n.n>,port=<nnnn>,state=online,offset=1051667111,lag=0
master_failover_state:no-failover
```

Master를 바라보고 있는 Replica정보를 모두 보여줌

```
Ip:port> role
1) "master"
2) (integer) 1051689019
3) 1) 1) "n.n.n.n" #replica ip
      2) "6000" #replica port
      3) "1051688880"
   2) 1) "n.n.n.n"
      2) "nnnn"
      3) "1051688880
```

Redis Replication 운영

•Redis Sentinel 구성 확인

Replica에서 구성확인

```
Ip:port> info replication
# Replication
role:slave
master_host:n.n.n.n
master_port:nnnn
master_link_status:up
....
```

Replica가 바라보고있는 master의 정보를 보여줌

```
Ip:port> role
1) "slave"
2) "n.n.n.n" #master IP
3) (integer) nnnn #master port
4) "connected"
5) (integer) 1051724411
```

Redis Replication 운영

•repl-timeout

- **Master**

Master관점에서 해당 파라미터는 "Replica시간 초과(REPLCONF ACK ping)"

Master는 1초마다 Replica에 replconf 명령을 보내며 일반적으로 lag 값이 0이나 1이나 Replica로부터 ack 가 오지 않으면 그 시간만큼 lag값이 증가하게 됨

```
Ip:port> info replication
# Replication
role:master
connected_Replicas:1
Replica0:ip=10.64.40.30,port=6379,state=online,offset=1299803,lag=1
```

이 lag값이 timeout을 초과하게 되면 Master는 Replica에 대한 연결을 해제하고 정보를 지움

Redis Replication 운영

•repl-timeout

-Replica

Replica 관점에서 해당 파라미터는 "SYNC 중 대량 전송 I/O"그리고"Master 시간 초과"

Replica는 repl-ping-replica-period 간격으로 ping을 보내며 timeout시간동안 응답이 없거나 Master로부터 timeout시간동안 데이터가 오지 않으면 Replica는 Master와 연결이 다운된 것으로 인식 함

기본 상태

```
Ip:port> info replication
# Replication
...
master_link_status:up
master_last_io_seconds_ago:0
```

연결이 다운된 경우

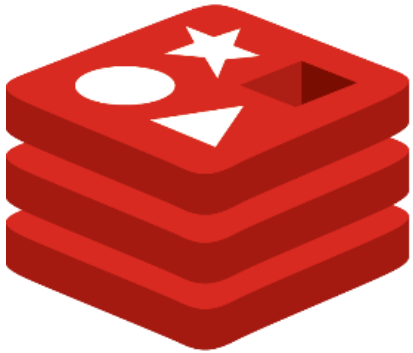
```
Ip:port> info replication
# Replication
...
master_link_status:down
master_last_io_seconds_ago:-1
```

SENTINEL

Sentinel?

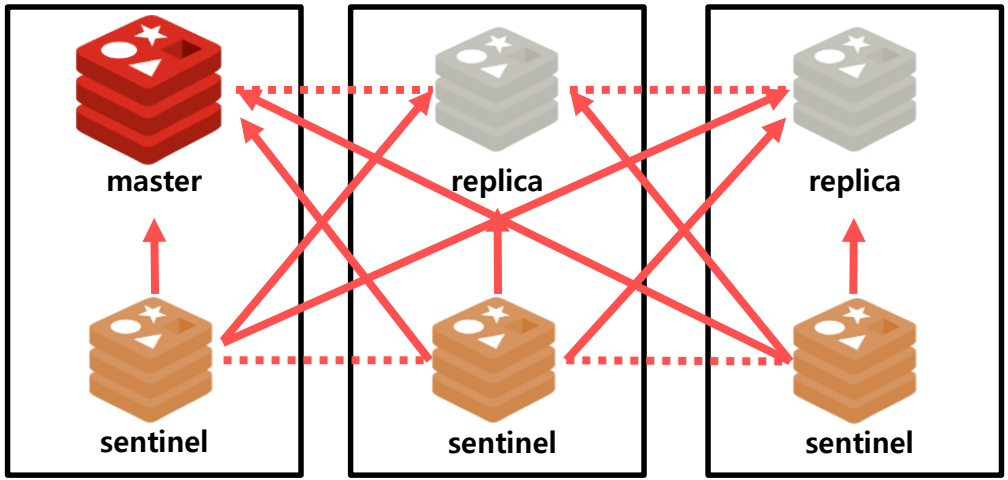
Sentinel 구성

운영



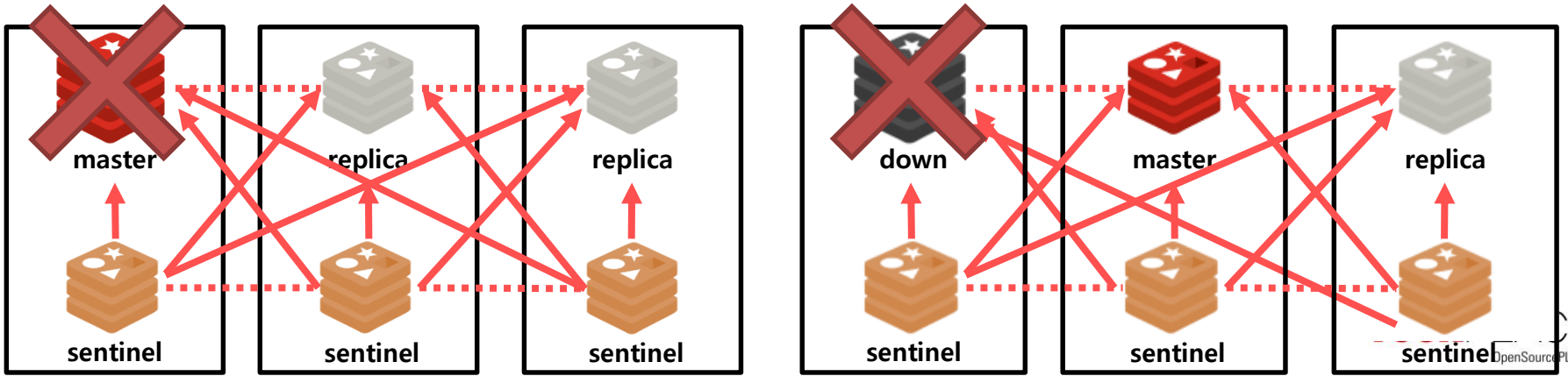
Redis Sentinel

- Redis sentinel은 master와 replica를 감시하다가, master가 다운되면 이를 감지해서 자동으로 replica를 master로 승격
- Redis sentinel은 Redis 상태 체크 시 다수결에 의해 결정되어 sentinel은 홀수로 구성
  - Redis sentinel 기본 구성 요소



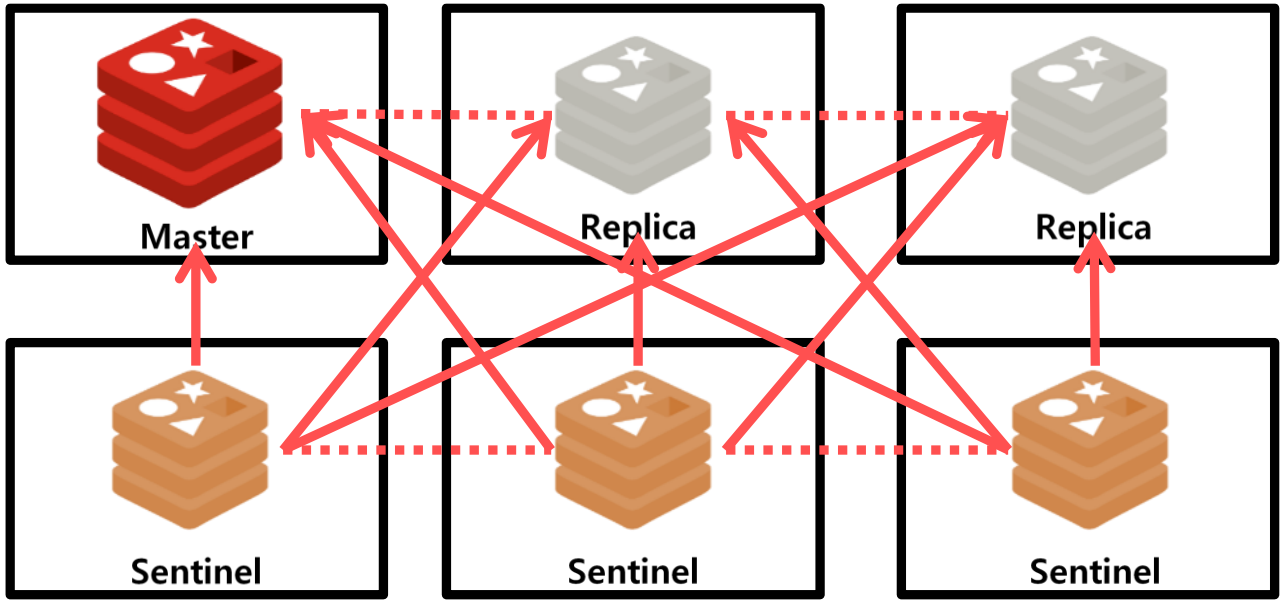
Redis Sentinel

- **Redis Sentinel 기능**
  - **Monitoring :** master와 replica 인스턴스가 제대로 동작하는지 지속적으로 감시
  - **Notification :** 모니터링 중인 Redis 인스턴스들 중에 문제 발생 시 API를 통해 시스템 관리자, 다른 컴퓨터 프로그램에게 알림
  - **Automatic failover** : 만약 master가 동작하지 않는다면, replica를 master로 승격시키고 다른 replica들은 새로운 master로 부터 데이터를 받을 수 있도록 재구성. 다운된 master가 재 시작 시 replica로 전환되어 새로운 master를 바라보게 구성



Redis Sentinel

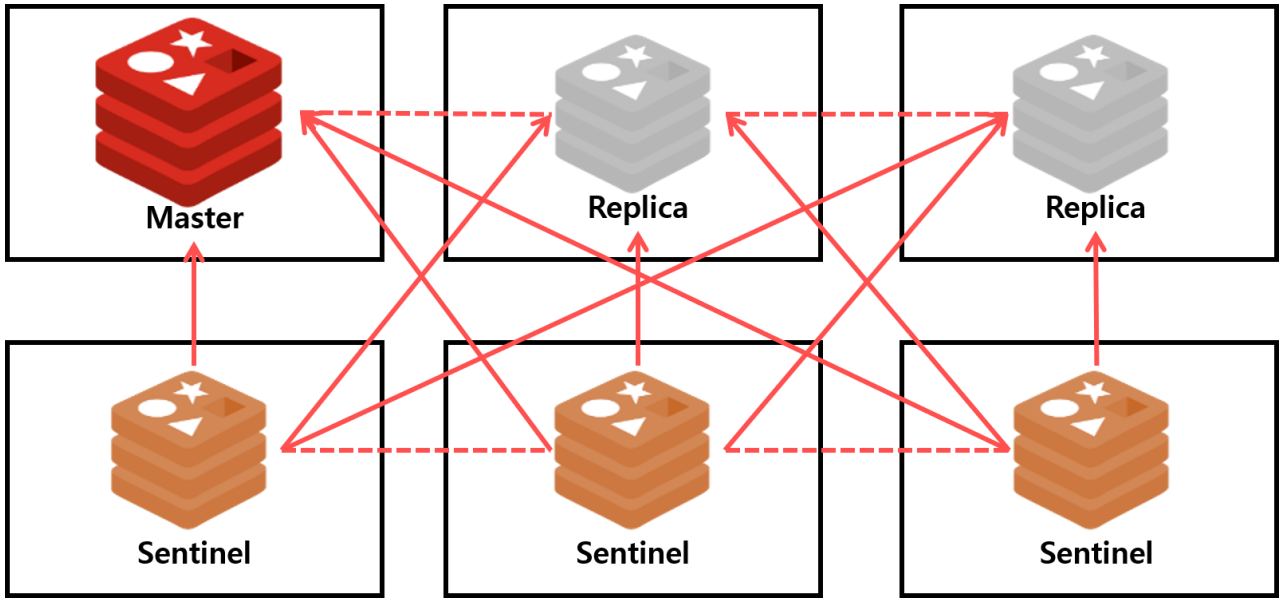
- Sentinel 구성시에는 Auto Failover 기능사용을 위해 최소 1master & 2replica & 3Sentinel 구성으로 총 6노드를 물리적으로 분리된 서버로 구성하는 것을 권장.
  Sentinel 노드 수는 오 탐지(Split Brain 등) 확률을 낮추기 위해 3노드 이상으로 구성 하는 것을 권장



[Sentinel 구성도]

Redis Sentinel 시작하기

Redis Server 실행



자동장애조치(Auto-Failover)를 위해 기존 Replication 구성(Master/Replica)에 Sentinel을 추가

Sentinels는 기본적으로 TCP 포트 26379에 대한 연결을 수신 대기하므로 Sentinel이 작동하려면 26379번 포트가 열려 있어야 함

Redis Sentinel 시작하기

Redis Server 실행

Redis 설치 경로의 Redis Sentinel 실행 파일인 Redis-sentinel을 실행

```
> /Redis설치경로/src/Redis-sentinel /Redis설치경로/sentinel.conf
```

```
# ./src/Redis-sentinel ./sentinel.conf

# oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo

# Redis version=6.0.8, bits=64, commit=00000000, modified=0, pid=27341, just started

# Configuration loaded
```

Redis Sentinel

•sentinel monitor

```
sentinel monitor <master-name> <ip> <redis-port> <quorum>
```

모니터링할 Master 설정,Replica노드의 정보는 Master에서 가져오기 때문에 Master의 정보만 설정.

Master세트의 이름을 각 다르게 설정할 수 있기 때문에 Sentinel은 서로 다른 Master 및 Replica 세트를 동시에 모니터링 가능.단,Master set 이름은 중복으로 사용불가

Quorum은 Master서버 다운 시 몇 개의 Sentinel이 다운되었는지 인지해야 객관적으로 다운되었다고 판정하는 기준으로 sentinel 개수보다 작은 짝수 값으로 설정 권장

•sentinel auth-pass

```
sentinel auth-pass <master-name> <password>
```

Master가 "requirepass"구성 지시문을 사용하여 비밀번호로 보호되어 있는 경우 Master에 접근하기 위해 Master의 암호지정 필요

Redis Sentinel

•sentinel down-after-milliseconds

```
# sentinel down-after-milliseconds <master-name>  <milliseconds>
```

Sentinel이 모니터하는 노드(Master, Replica, 다른 Sentinel)가 다운되었다고 인지 시 인스턴스에 연결할 수 없는 시간(PING에 응답하지 않거나 오류로 응답하는 경우). 이 상태를 주관적 다운(Subjectively Down, SDown)라고 함

※ Sentinel 장애 조치 단계

1단계: 주관적 다운(Sdown)

2단계: 객관적 다운(Odown)

3단계: Sentinel 리더 선출(elected-leader)

4단계: Replica선정(selected-Replica)

5단계: 선정된 Replica를 Master로 승격(promoted-Replica)

6단계: Replica들이 새 Master에서 데이터를 받도록 REPLICAOF 명령 수행(Replica-reconf-done)

7단계: Sentinel 내 정보 갱신(failover-end, switch-master, update_config)

Redis Sentinel

•sentinel failover-timeout

```
sentinel failover-timeout <master-name> <milliseconds>
```

장애 조치 시간 지정

•sentinel parallel-syncs

```
sentinel parallel-syncs <master-name> <numreplicas>
```

장애 조치 후 동시에 새 Master를 사용하도록 재구성 할 수 있는 Replica 수를 설정

장애조치 단계 중 5단계(클론들이 새 Master에서 데이터를 받도록 REPLICAOF 명령 수행)에서 동시에 몇 개의 Replica에서 동기화를 수행할지 정하는 파라미터

기본값은 1이며 한번에 하나의 Replica에서 명령을 실행하고 완료 후 다음 Replica에서 명령 수행

Redis Sentinel 운영

•Redis Sentinel 구성 확인

Redis 전체 구성확인

```
Ip:port>info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=ip:port,slaves=1,sentinels=3
```

Master-2Replica 3Sentinel 구성을 확인 가능

Redis Sentinel 운영

•Redis Sentinel 구성 확인

Master 정보 확인

Master의 이름을 따로 설정해주지 않았다면 기본적으로 mymaster로 설정

```
IP:Port>sentinel master <mastername>
        1) "name" # master이름
        2) " master이름"
        3) "ip" # master ip
        4) "n.n.n.n"
        5) "port" # master port
        ....
        9) "flags" #Master 서버의 상태
        10) "master" # 정상 작동 시
        10) "s_down,master,disconnected" #master s_down시
        11) "link-pending-commands" #완료되지 않은 명령의 개수
        12) "0"
        13) "link-refcount" #링크 참조 개수
        13) failover-state #failover_in_progress 일 때 상세한 상태
        ....
```

Redis Sentinel 운영

•Redis Sentinel 구성 확인

Replica 정보확인

Sentinel이 바라보고 있는 Master의 Replica정보를 모두 확인 가능

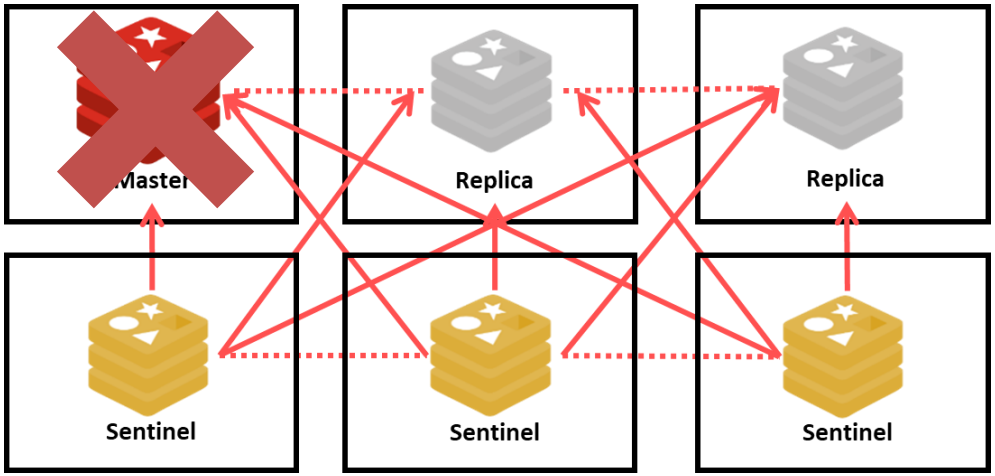
```
Ip:port>sentinel replicas <masternamе>
1)  1) "name" # replica이름
    2) "n.n.n.n:port"
    3) "ip" # replica ip
    4) "n.n.n.n"
    ....
    29) "master-link-status" " #replica와 master의 연결상태
    29) "master-link-down-time" #replica와 master가 연결이 끊긴 시간
    ....
2)  1) "name" # replica이름
    2) "n.n.n.n:port"
    3) "ip" # replica ip
    4) "n.n.n.n"
    ....
```

Redis Sentinel 운영

•Sentinel Auto Failover 기능 확인

Master-2Replica 3Sentinel 구성에서 마스터 노드를 다운

Master ip:port>shoutdown

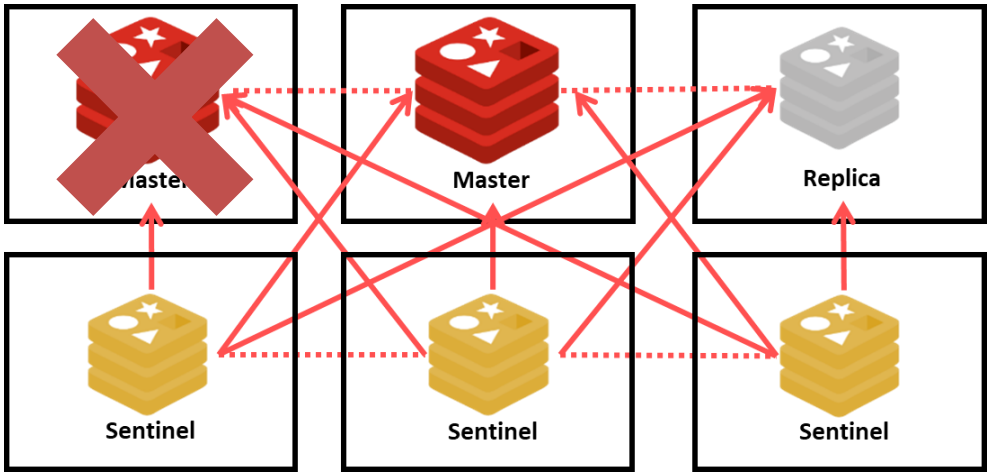



```
10.64.40.140:6000> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=10.64.40.150,port=6000,state=online,offset=4234928,lag=0
slave1:ip=10.65.50.70,port=6000,state=online,offset=4234651,lag=1
master_failover_state:no-failover
```

Redis Sentinel 운영

•**Sentinel Auto Failover 기능 확인**

Replica 노드 중 한 노드가 Master로 승격



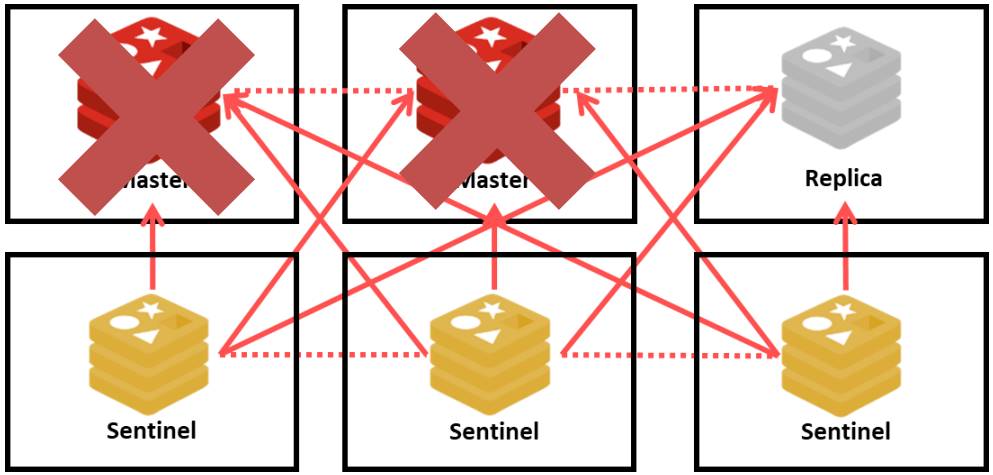

```
10.65.50.70:6000> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=10.64.40.150,port=6000,state=online,offset=4267194,lag=1
master_failover_state:no-failover
```

Redis Sentinel 운영

•**Sentinel Auto Failover 기능 확인**

승격한 마스터 노드를 다운

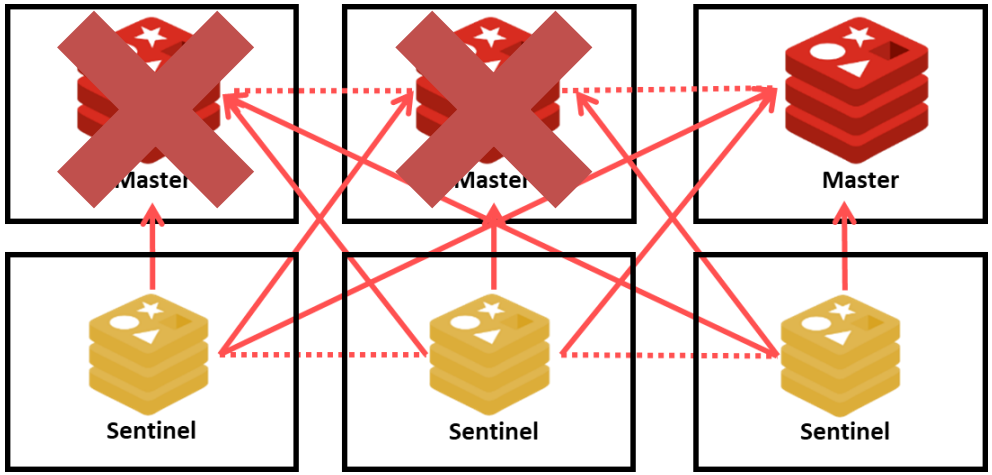
```
Master ip:port>shoutdown
```



Redis Sentinel 운영

•Sentinel Auto Failover 기능 확인

Replica 노드 중 한 노드가 Master로 승격

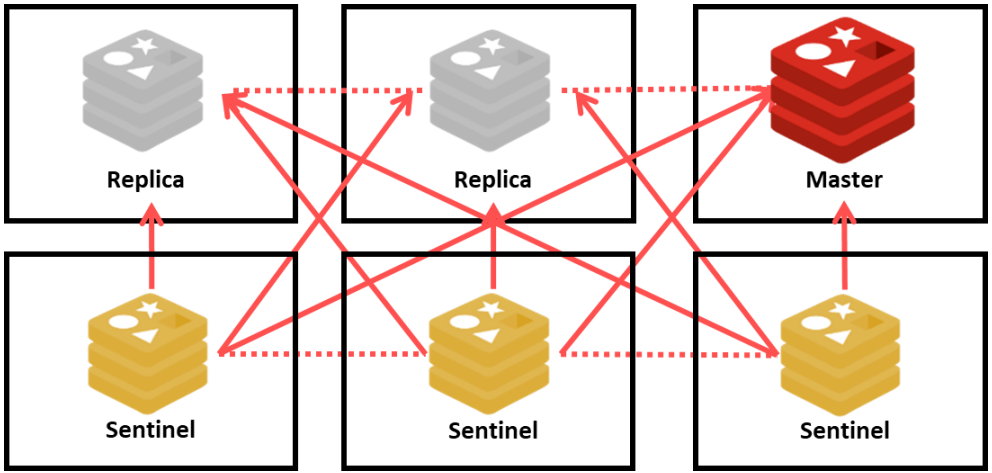



```
10.64.40.150:6000> info replication
# Replication
role:master
connected_slaves:0
master_failover_state:no-failover
```

Redis Sentinel 운영

•Sentinel Auto Failover 기능 확인

다운 시켰던 노드들을 복구하면 Master-2Replica 3Sentinel 구성으로 올라오는 것을 확인 가능



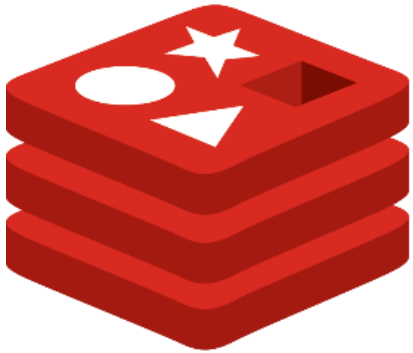

```
10.64.40.150:6000> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=10.64.40.140,port=6000,state=online,offset=4391996,lag=1
slave1:ip=10.65.50.70,port=6000,state=online,offset=4391996,lag=0
master_failover_state:no-failover
```

CLUSTER

Cluster?

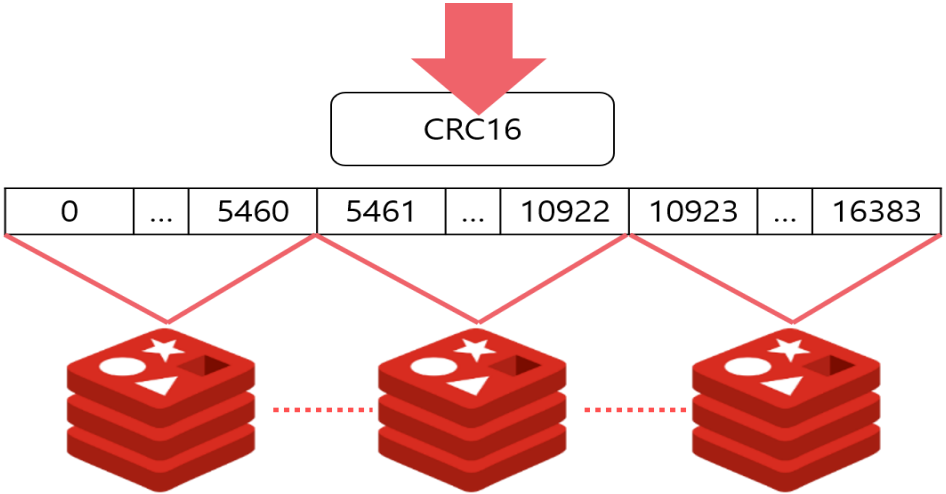
Cluster 구성

운영



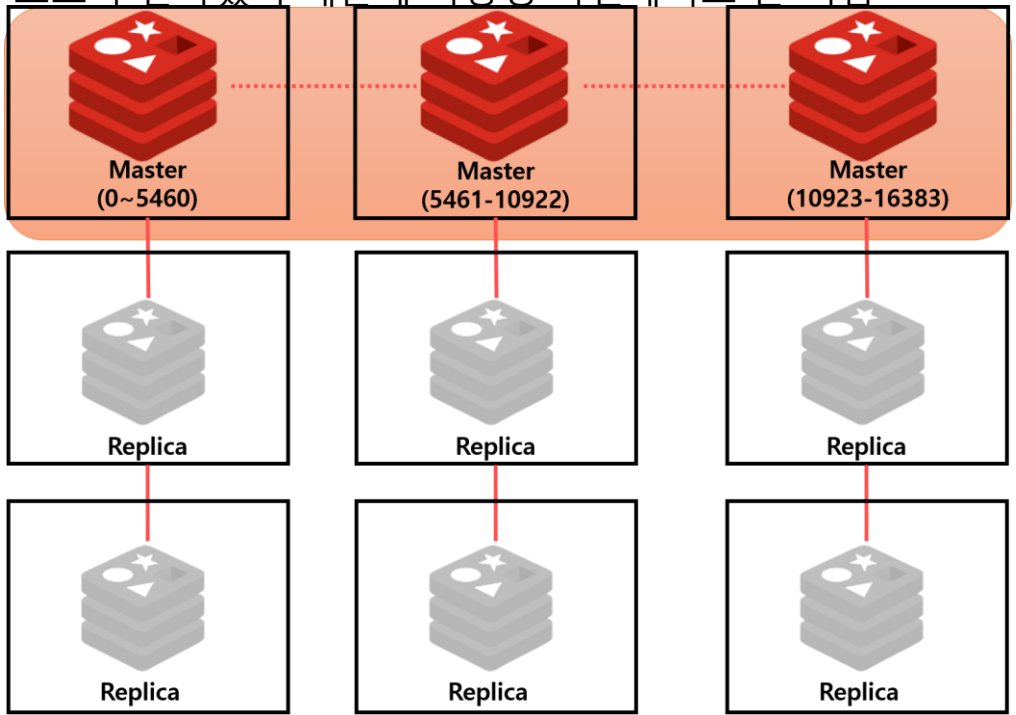
Redis Cluster

- Redis의 Partitioning 구성을 위해 Redis 3.0 부터 Redis Cluster를 지원하고, 최소 3개의 노드가 필요
- Cluster 모드로 구성된 각 Redis 서버에 데이터를 균등하게 분산 저장
- Sentinel 서버없이 Cluster 구성만으로 auto-failover기능을 사용 가능
- 최대 1000개의 노드까지 확장이 가능
- Redis Cluster는 총 16384개(0~16383)의 hash slot을 사용하며, Redis cluster는 모든 master서버에 slot 정보를 할당



Redis Cluster

- 물리적으로 분리된 서버에 3 master & 6 replica 구성으로 총 9노드의 구성을 권장
- 이는 각 master마다 2개의 replica를 가짐으로써 원활한 failover 기능을 지원, 1개의 노드에 장애가 발생했을 때에도 2개의 노드가 살아있기 때문에 가용성 측면에서도 큰 이점



Redis Cluster 구성

Cluster에 필요한 파라미터들은 Redis.conf 파일에 주석으로 처리

Cluster 구성을 위해 필요한 파라미터들 수정 필요

•cluster-enabled

```
# cluster-enabled <yes/no>
```

특정 Redis 인스턴스에서yes로 하면 cluster mode로 시작, no로 설정 시 standalone mode로 시작

•cluster-config-file

```
# cluster-config-file <configfile>
```

이 파일은Cluster 노드가 변경 될 때마다 Cluster 구성 (기본적으로 상태)을 자동으로 유지하는 파일, 즉 Cluster의 상태를 기록하는 바이너리 파일

시작할 때 다시 읽을 수 있으며 파일에는 Cluster의 다른 노드, 상태, 영구 변수 등이 나열

종종 이 파일은 일부 메시지 수신의 결과로 디스크에 다시 쓰이고 flushed

Redis Cluster 구성

•cluster-node-timeout

```
# cluster-node-timeout <millisecond>
```

Redis 노드가 다운되었는지 판단하는 시간, 단위는 millisecond

이 파라미터는 Redis Cluster를 제어하며 지정된 시간 동안 Master노드에 도달 할 수 없는 모든 노드의 쿼리 수신을 중지. 디폴트 값인 15000은 체감상 시간이 길어 보다 적은 값으로 설정하는 것을 권장

•cluster-require-full-coverage

```
# cluster-require-full-coverage <yes/no>
```

yes로 설정 시 키 공간의 일부가 노드에 포함되지 않으면 Cluster가 쓰기 허용을 중지.

Cluster가 부분적으로 다운 시 (예 : 해시 슬롯 범위가 더 이상 포함되지 않음) 모든 Cluster를 사용불가

모든 슬롯이 다시 포함되면 자동으로 사용 가능한 상태로 가동

No로 설정 시 키 하위 집합에 대한 요청만 처리 할 수 있는 경우에도 Cluster가 쿼리를 계속 제공

Redis Cluster 구성

Redis Cluster 생성

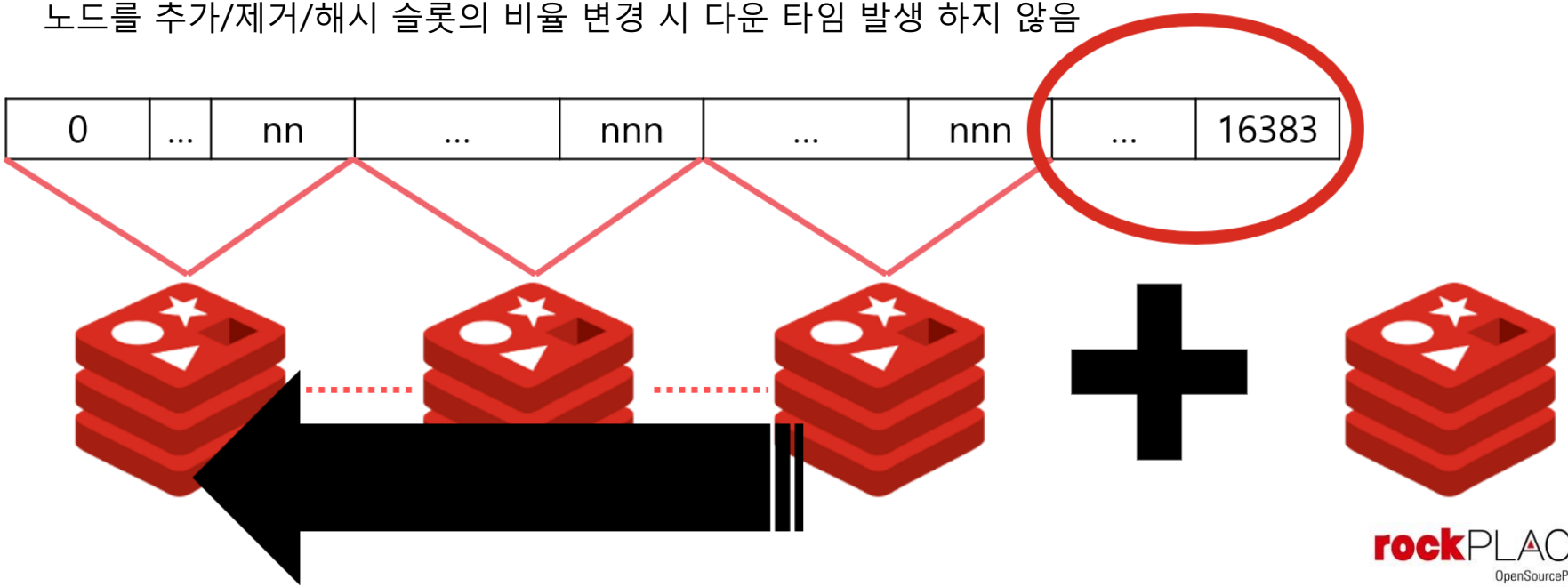
최소 3개이상의 노드를 하나의 Cluster로 생성

```
> /Redis설치경로/src/Redis-cli --cluster create <IP1><IP2><IP3>
```

```
# .../src/Redis-cli –cluster create <Master1_IP> <Master2_IP><Master3_IP>

>>> Creating cluster

..... 중간 생략 .....

Can I set the above configuration? (type 'yes' to accept): yes (입력)

..... 중간 생략 .....

[OK] ALL 16384 slots coverd
```

Redis Cluster 구성

- 새로운 노드 추가 시 새 노드로 일부 해시 슬롯을 이동 필요.
- Cluster에서 기존 노드 제거시에도 제거하려는 노드가 제공하는 해시 슬롯을 이동필요. 제거하려는 노드가 비어 있을 때 Cluster에서 완전히 제거 가능
- 노드를 추가/제거/해시 슬롯의 비율 변경 시 다운 타임 발생 하지 않음



Redis Cluster 구성

Redis Cluster node 추가

추가할 노드는 Cluster mode로 시작되어 있어야 하고, 데이터가 없어야 추가 가능

Master

```
> .../src/Redis-cli --cluster add-node new_ip:port existing_ip:port
```

Replica

```
> .../src/Redis-cli --cluster add-node new_ip:port existing_ip:port --cluster-replica
```

```
# --cluster add-node new_ip:port existing_ip:port
>>> Adding node new_ip:port to cluster existing_ip:port
>>> Performing Cluster Check (using node existing_ip:port)
...
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Send CLUSTER MEET to node new_ip:port to make it join the cluster.
[OK] New node added correctly.
```

Redis Cluster 구성

Redis Cluster node 재분배

구성 되어있는 cluster 노드 재분배가능 ,재분배 시 슬롯과 데이터를 같이 분배

- 노드의 개수를 지정해서 재분배

```
> .../src/Redis-cli --cluster reshard ip:port node-id
```

- 노드의 개수를 지정하지 않고 균등하게 재분배

```
> .../src/Redis-cli --cluster rebalance ip:port options
```

Redis Cluster 운영

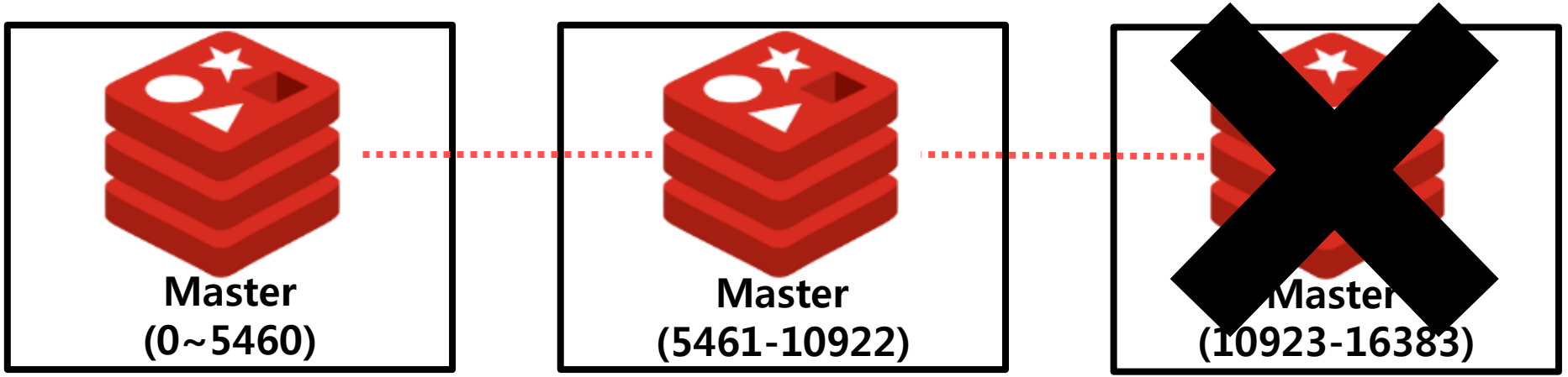
Redis Cluster 구성 확인

Redis 전체 구성확인

```
[root@sql_140 redis]# redis-cli -p 5000 cluster nodes
e8315c4e964d8b447d0f43c9f649311ad368d043 10.64.40.150:5000@15000 master - 0 1627368953610 2 connected 5461-10922
fba3567283c1d5b21dc78b2094458740fa548dbe 10.64.40.140:5000@15000 myself,master - 0 1627368954000 1 connected 0-5460
6e76b551eb706cb27da96bc91ff547d7c65149a8 10.64.40.40:5300@15300 slave e8315c4e964d8b447d0f43c9f649311ad368d043 0 1627368954614 2 connected
cdcbdb96f61efd4d1f6ab6405c524df460f480a9 10.65.50.70:5000@15000 master - 0 1627368954000 3 connected 10923-16383
da6fd5b685fbe668adb32c79c1c474d8feaa6f46 10.64.40.30:5000@15000 slave fba3567283c1d5b21dc78b2094458740fa548dbe 0 1627368955620 1 connected
ec017cd18803559829a386daac693108a492ebd8 10.65.50.71:5000@15000 slave cdcbdb96f61efd4d1f6ab6405c524df460f480a9 0 1627368954000 3 connected
```

3Master-3Replica Cluster 구성을 확인 가능

Redis Cluster 운영



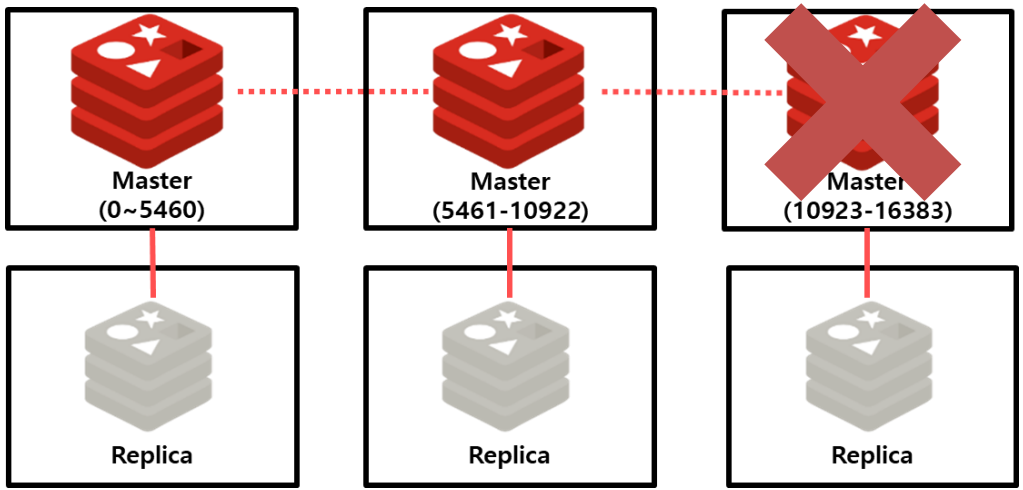
- 기존 노드에서 한 노드가 다운되면 해당 노드가 할당하고 있는 범위의 해시 슬롯을 제공 할 방법이 없기 때문에 Cluster를 계속할 수 없음
  - 가용성을 유지하기 위해 모든 해시 슬롯에 N 개의 Replica가 있는 모델 사용 권장

Redis Cluster 운영

•Sentinel Auto Failover 기능 확인

3Master-3Replica Cluster 구성에서 마스터 노드를 다운

```
Master ip:port>shoutdown
```

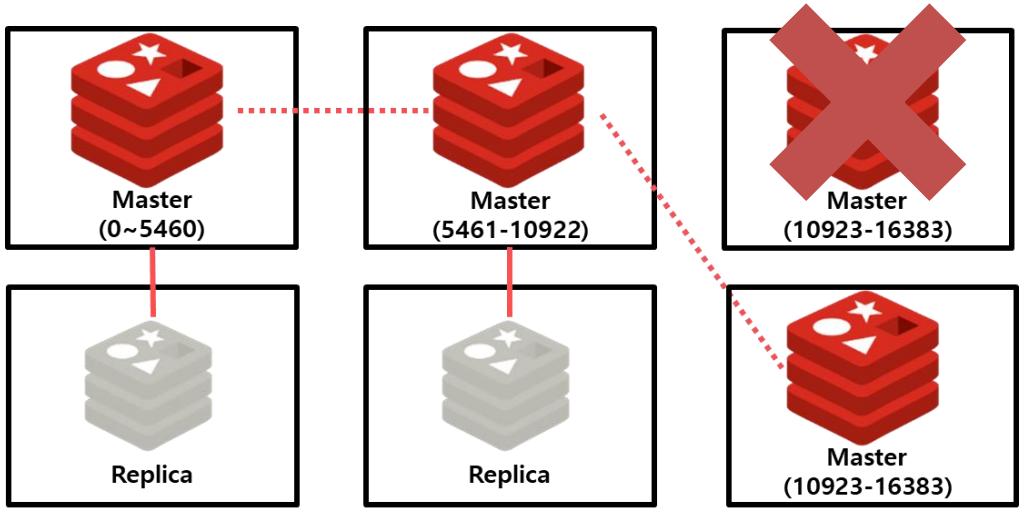


Redis Sentinel 운영

•Sentinel Auto Failover 기능 확인

Cluster가 생성 될 때 모든 Master에 Replica노드를 추가하여 구성하면 Replica노드가 Master노드를 복제하고 Master노드가 다운되면 Cluster는 해당 Master의 Replica를 새 Master로 승격

-> 노드 C가 다운되어도 시스템이 계속 작동 가능



Redis Cluster 운영

•Cluster mget 사용

Redis Cluster의 경우, slot에 의해 키 값들이 분산 저장되기 때문에 서로 다른 노드의 데이터를 적재하고 가져와야 하는 'mset', ' mget'과 같은 멀티 키 명령어는 기본 형태로는 사용할 수 없음

```
127.0.0.1:6379> mset key1 Hello key2 Redis
(error) CROSSSLOT Keys in request don't hash to the same slot

127.0.0.1:6379> set key1 Hello
OK

127.0.0.1:6379> set key2 Redis
-> Redirected to slot [2388] located at 10.65.50.70:6379
OK

10.65.50.70:6379> mget key1 key2
(error) CROSSSLOT Keys in request don't hash to the same slot
```

Redis Cluster 운영

•Cluster mget 사용

멀티 키를 사용하려면 아래와 같이 키 값에 특정 값을 동일하게 부여하여 해당 key들을 같은slot으로 적재

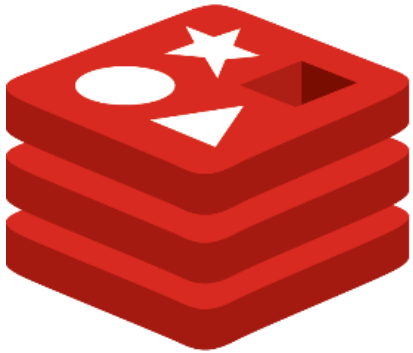
단, 해당 방식으로 적재 시 데이터 편중현상이 발생하여 특정 shard만 데이터 사이즈가 늘어나기 때문에

여러 이슈들이 생길 수 있음

```
127.0.0.1:6379> set key1{hash} value1
OK

127.0.0.1:6379> > set key2{hash} value2
OK

127.0.0.1:6379> > set key3{hash} value3
OK

127.0.0.1:6379> mget key1{hash} key2{hash} key3{hash}
1) "value1"
2) "value2"
3) "value3"
```

TRUBLESHOTTING

Backup & Restore
Monitoring
Redis 장애 사례



Backup & Restore

RDB AOF 전환

RDB를AOF로 전환할 때는 Redis서버를 재시작 할 필요없이 간단히 전환 가능
안전한 데이터보호를 위해서는 두가지 방법 모두를 사용하는 것을 권장

- 기존에 사용하던 dump.rdb 파일 백업, 최대한 안전한 장소로 백업하는 것을 권장

```
#cp dump.rdb /redis/dump2.rdb
```

- AOF를 활성화하는 명령어와 스냅샷의 지속성을 끄는 명령어를 실행

```
# redis-cli config set appendonly yes
# redis-cli config set save ""
```

- Redis서버에 접속해 DB에 이전과 데이터 사이즈가 동일한지 확인

```
# redis-cli -p port
Ip:port> dbsize
```

- 쓰기가 올바르게 AOF파일에 추가되었는지 확인

```
# vi   appendonly.aof
```

Backup & Restore

복구절차

- 백업파일 확인

서버에서 나와 백업할 AOF나 RDB파일을 확인

```
#ll
Total 0
-rw-r--r-- 1 root 192 Nov 22 15:40 dump.rdb
-rw-r--r-- 1 root 106   Nov   5 17:35 radis.aof
```

- 백업파일 이동

준비된 백업파일을 해당 Redis 서버가 백업파일을 읽어오는 경로로 이동

```
#cp dump.rdb /redis/data/dump2.rdb
```

- Redis 서버 다운

Redis 서버에 재 접속하여 서버를 다운

```
# redis-cli -p port
Ip:port>SHUTDOWN
not connected> exit
```

Backup & Restore

복구절차

- 기존 백업 파일 변경

기존에 있던 백업파일과 이동해온 백업파일의 이름을 변경

```
# mv dump.rdb dump_old20200722.rdb
# mv dump2.rdb dump.rdb
# ll
-rw-r--r-- 1 root root 192 Nov 22 15:40 dump.rdb
-rw-r--r-- 1 root root 92 Nov 22 15:35 dump_old20200722.rdb
-rw-r--r-- 1 root root 106 Nov 5 15:35 radis.aof
```

쓰기가 올바르게 AOF파일에 추가되었는지 확인

```
#redis-server /redis-6.0.8/redis.conf
```

- 백업/복구 확인

백업/복구를 진행한 Redis 서버의 data를 확인하여 잘 이루어졌는지 확인

```
# redis-cli -p port
Ip:port> dbsize
```

Redis Monitoring

Redis SlowLog

서버 성능을 분석 또는 문제(trouble) 발생시 명령의 수행시간이 설정 시간 이상이면 추적하는 도구

설정시간은 Redis Config파일에서 설정 가능, 단위는 마이크로초(microseconds)

```
# slowlog-log-slower-than <Time>
```

조회

```
>slowlog get<count>
```

```
127.0.0.1:637> slowlog get <count>
1) 1) <id>
   2) <timestamp>
   3) <time in microseconds>
   4) <arguments array>
   5) <client IP and port>
   6) <client name>
```

Redis Monitoring

Redis-Benchmark

내장 되어있는 벤치마킹 유틸리티로 Redis설치에 포함

```
#Redis-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests>] [-k <boolean>]
```

여러 옵션으로 사용자가 원하는 대로 벤치마킹이 가능

- 주로 사용하는 옵션

option	Description	Default
-h	서버 호스트 이름	127.0.0.1
-p	서버 포트	6379
-s	서버 소켓	
-c	클라이언트 수	50
-n	실행 횟수	10000
-d	value 크기	2
-k	1 = 생존, 0 = 다시 연결	1
-r	key 또는 value를 랜덤 생성	
-p	서버 포트	1
-h	서버 호스트	
...		

Redis 장애 사례

메모리 부족

메모리 확보를 위해 사용하지 않는 데이터(key)를 삭제

Redis 서버에 접속한 후 설정되어 있는 max memory를 확인

```
127.0.0.1:6379> config get maxmemory
1)"maxmemory"
2) "0"
//2는 maxmemory의 값 확인
```

- max memory 변경

```
127.0.0.1:6379> config set maxmemory [메모리 사이즈]gb
```

Redis 장애 사례

I/O 대기발생

실행 시간이 오래 걸리는 'keys *' 와 같은 명령 대신 scan 명령 사용을 권장

Keys는 키 스페이스 레이아웃 변경과 같은 특수 작업 및 디버깅을 위한 것으로 극도의 주의를 기울여 프로덕션 환경에서만 사용해야하는 명령으로 간주

해당 명령을 대용량 데이터베이스에서 실행하면 성능이 저하될 수 있어 일반 애플리케이션 코드에서 사용을 지양

키스페이스의 하위 집합에서 키를 찾는 방법을 찾고 있다면 scan 또는 sets 사용 권장

- Scan 사용 예시

```
redis 127.0.0.1:6379> scan 0
1) "17"
2)  1) "key:12"
    2) "key:8"
    3) "key:4"
    4) "key:14"
    5) "key:16"
    6) "key:17"
....
```

Redis 장애 사례

I/O 대기발생

Sets 조회 명령어

comands	Syntax	Description
SMEMBERS	key	한 set에서 하나 또는 여러 임의의 구성원 조회
SCARD	key	집합에서 member의 수 조회
SISMEMBER	key member	집합에 member가 존재하는지 확인
SRANDMEMBER	key [count]	집합에서 무작위로 member를 조회
SSCAN	cursor [MATCH pattern] [COUNT count]	member를 일정 단위 개수만큼 씩 조회

Redis 장애 사례

Sentinel monitoring 장애

- Redis 상태

Redis 노드 전체가 Replica role 로 되어 있었고 특정 노드 하나를 master 로 promote 하였으나 일정시간 후 다시 전체 노드가 Replica 로 바뀌는 문제 발생

- Redis Sentinel log

```
# -sdown master *-Redismaster node1 port
# -odown master *-Redismaster node1 port
# -failover-abort-not-elected master *-Redismaster node1 port
# -sdown replica node3-Private:port node3-Private port @ *-Redismaster node1 port
# -sdown replica node2:port node2 port @ *-Redismaster node1 port
# -sdown replica node2-Private:port node2-Private port @ *-Redismaster node1 port
# -sdown replica node3:port node3 port @ *-Redismaster node1 port
# +config-update-from sentinel f5fae74b39fc7cf09638dc4925a2f64cac35ee39 sentinel2 port @ *-Redismaster node1 port
# +switch-master *-Redismaster node1 port node2-Private port
...
```

- Log분석 결과

고객사에서 대역 대 IP 추가 후,네트워크 설정이나 sentinel 설정을 변경 적용하는 업데이트 과정을 별도로 수행하지 않아 해당 이슈가 발생했을 가능성 제시

Redis 장애 사례

Sentinel monitoring 장애

- 해결방안 제시

사용자 임의로 master변경 시 replica의 경우 online으로 해당 파라미터를 변경 할 수 있지만, sentinel의 경우에는 온라인으로 변경이 불가능 하기 때문에 수동으로 수정 후 재시작이 필요

Sentinel 수정이 이루어지지 않으면 monitoring에 장애가 발생해 fail over가 비정상 동작

Replica 파라미터

```
# replicaof <masterip> <masterport>
```

Sentinel 파라미터

```
#sentinel monitor <master-name> <ip> <redis-port> <quorum>
```

Master 나 replica의 경우에도 쓰레기 값 등과 같은 변수가 있을 수 있으므로 확실하게 하기 위해서는 Redis의 재 기동을 권장

재 기동 시에는 master->replica->sentinel 순으로 재 기동

Redis 장애 사례

빈번한 fullsync 발생

Replica 사용시 유입되는 데이터의 양이 많아지면 master의 경우에는 제한이 없어 모든 데이터를 유입
replica는 제한이 있어 데이터의 읽기 속도가 설정 값 이하로 떨어지게 되면 client의 연결이 해제
이로 인해 빈번한 fullsync가 발생하게 되어 성능에 문제를 발생 가능

이슈 발생 시 Client연결 해제 제한과 replica buffer 제한 값 수치를 상향

-Client-out-put-buffer-limit

```
# client-output-buffer-limit <class> <hard limit> <soft limit> <soft seconds>
```

해당 파라미터는 클라이언트 출력 버퍼 제한을 사용하여 서버에서 데이터를 빠르게 읽지 않는 클라이언트의 연결을 강제로 끊음
세 가지 등급의 클라이언트에 대해 한도를 다르게 설정가능

normal -> MONITOR 클라이언트를 포함한 일반 클라이언트
replica -> replica clients
pubsub -> 하나 이상의 Pubsub 채널 또는 패턴에 가입한 고객

Redis 장애 사례

빈번한 fullsync 발생

백로그는 Replica가 한동안 연결이 끊겼을 때 Replica 데이터를 축적하는 버퍼

-repl-backlog-size

replica 백로그 크기 설정

```
# repl-backlog-size <data_size>
```

- 재 연결 시 backlog-buffer의 데이터를 동기화진행
- 연결이 끊긴 동안 누락된 데이터 부분을 전달하기만 하면 전체동기화대신 부분 동기화로 충분

0으로 설정하면 서버 시작 시 에러메시지 출력 후 멈추기 때문에 해당 파라미터는 0으로 설정불가

해당 파라미터들은 각 conf파일 수정 후 master-replica 순으로 재 기동을 진행필요

Redis 장애 사례

간헐적 서비스 지연 현상

서비스에 따라 피크타임 시에 설정되어 있는 maxclients 이상의 커넥션 발생 시 초과되는 커넥션으로 인해 서비스 지연 이슈 발생 가능

메모리의 용량에 무리가 되지않는 상황에서 해당 이슈가 발생 시 maxclients 수치 상향이 필요
해당 파라미터는 online으로 변경이 가능

```
redis 127.0.0.1:6379> config get maxclients
1) "maxclients"
2) "4064"

redis 127.0.0.1:6379> config get maxclients <clients num>
ok
```


Q&A


rockPLACE
감사합니다 _☺
nosql@rockplace.co.kr