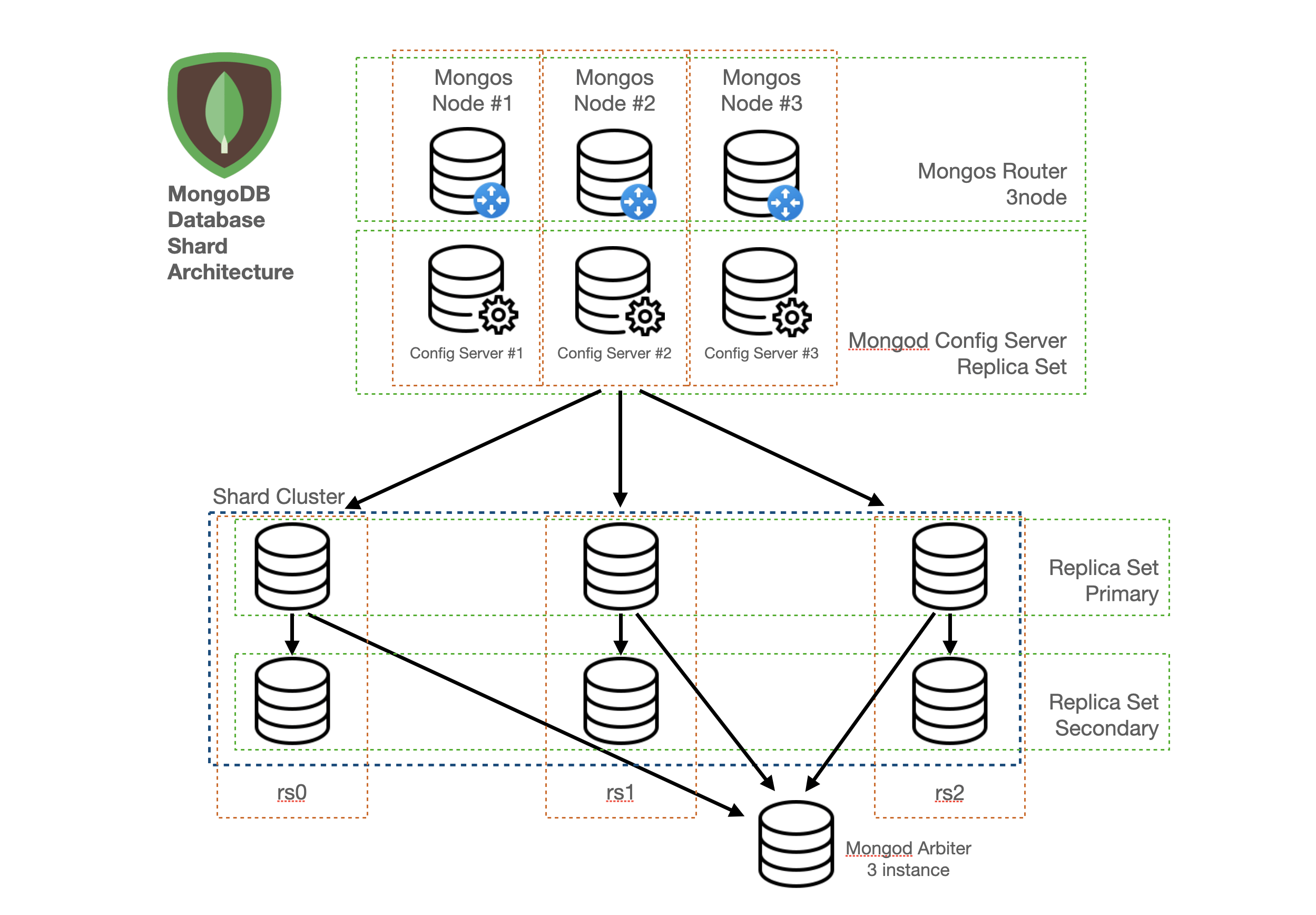
**MongoDB 샤딩(sharding) 환경 구성**



위 그림과 같은 환경으로 셋팅 진행

각 서버마다 mongo가 설치되어 있어야 한다.

**1. Config 서버 (1대)**

- mongo config 파일 수정 (default 경로 : /etc/mongod.conf)

-> 여러가지 옵션을 설정해 줄 수 있지만, 꼭 필요한 설정을 나열해보자면

processManagement:

fork: true

net:

port: 27019

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “cfgrepl”

sharding:

clusterRole: configsvr

- mongoDB 구동

$ sudo mongod -f /etc/mongod.conf

- mongoDB 접속해서 replication set 시작

$ mongo localhost:27019

> rs.initiate()

※한 서버에서 여러 개의 config 프로세스를 구동하고 싶은 경우, 기존 mongod.conf 파일을 복사해서 포트만 다르게 개방하고 mongoDB 프로세스를 시작한다. (과정은 아래 설명과 같다.)

**1-1. Config 서버 (3대)**

- config 서버가 여러 대인 경우

- 각 서버에 mongo config 파일 수정 (경로 : /etc/mongod.conf)

-> 아래 옵션 추가

processManagement:

fork: true

net:

port: 27019

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “cfgrepl”

sharding:

clusterRole: configsvr

- 모든 config 서버의 IP 주소가 다르다는 가정 하에 진행

- 각 서버의 mongoDB 구동

$ sudo mongod -f /etc/mongod.conf

- 맨 첫 번째 서버만 레플리카 셋 시작 (나머지 서버에서는 절대 rs.initiate()를 하지 않는다.)

$ mongo localhost:27019

> rs.initiate()

- 첫 번째 서버의 mongo 레플리카 셋(Replication Set)에 나머지 config 서버를 추가

> rs.add(“mongodbs01:27019”)

> rs.add(“mongodbs01:27019”)

- 노란색 부분은 추가할 config 서버 각각의 퍼블릭 IPv4 DNS 서버 주소를 입력해야 한다.

- 레플리카 셋 상태 확인

> rs.status()

- 첫 번째 서버의 mongo 명령줄을 확인했을 때 아래처럼 나오면 성공

cfgrepl:PRIMARY>

- 나머지 서버의 mongo 명령줄은 secondary 로 표시된다.

cfgrepl:SECONDARY>

**2. Router 서버 (mongos, 1대)**

- mongos.conf 파일이 없으면 새로 생성하여 내용 입력 (경로 : /etc/mongos.conf)

sharding:

configDB: “cfgrepl/ip-172-31-23-134:27019”

systemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongos.log

processManagement:

fork: true

net:

port: 27017

bindIpAll: true

- 파란색 부분과 노란색 부분은 config 서버의 mongo 에서 rs.status() 명령어를 실행한 후 각각 “set” 값과 “members”의 “name” 값을 입력해야 한다.

- config 서버가 여러 대인 경우, 콤마(,) 구분자로 서버 추가 가능 (rs.initiate() 하지 않은 서버들)

ex) “cfgrepl/ip-172-31-23-134:27019,ip-172-31-23-134:27020”

- mongos 구동으로 config 서버에 붙는지 확인 (아래 명령어가 잘 실행되면 완료)

$ sudo mongos -f /etc/mongos.conf

- mongo 접속을 해보면 명령줄이 아래처럼 표시

$ mongo

mongos>

**3. Primary, Secondary 서버 (각 1대씩 구성, 다른 세트는 추후에 추가)**

- 이 두 서버는 같은 레플리카 셋(Replication Set)으로 사용

- 이후에 arbiter까지 총 3개의 서버가 한 세트로 구성된다.

- 각 서버의 mongo config 파일 수정 (default 경로 : /etc/mongod.conf)

-> 아래 옵션 추가 (config 서버의 mongod.conf 파일과 유사)

processManagement:

fork: true

net:

port: 27018

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “rs0”

sharding:

clusterRole: shardsvr

- 각 서버의 mongoDB 구동

$ sudo mongod -f /etc/mongod.conf

- primary 서버에서만 replication set 시작

$ mongo localhost:27018

> rs.initiate()

**4. Arbiter 서버 (1대)**

- mongo config 파일 수정 (default 경로 : /etc/mongod.conf)

-> 아래 옵션 추가

processManagement:

fork: true

net:

port: 27021

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “rs0”

sharding:

clusterRole: shardsvr

- mongoDB 구동

$ sudo mongod -f /etc/mongod.conf

- 레플리카 세트 실행은 하지 않는다.

**5. 각 서버 연결 작업**

- primary 서버의 mongo 에서 레플리카 셋(Replication Set)에 secondary 서버와 arbiter 서버를 추가

> rs.add(“mongodbs01:27018”)

> rs.addArb(“mongoarb:27021”)

- 노란색 부분은 각 서버(secondary, arbiter)의 퍼블릭 IPv4 DNS 서버 주소와 mongo 포트번호를 입력해야 한다.

※ arbiter는 상관없지만, secondary같은 경우 서버가 다운될 경우 재시작을 해야 할 상황이 있을 수도 있는데, 이때 중지 후 재시작(AWS)을 하면 퍼블릭 주소가 바뀌기 때문에 private ip 주소로 설정을 해도 문제가 없다면 그렇게 하는 것이 좋다.

- 추가된 레플리카 셋 멤버 확인

> rs.status()

- 첫 번째 서버의 mongo 명령줄을 확인했을 때 아래처럼 나오면 성공

rs0:PRIMARY>

- 두 번째 서버의 mongo 명령줄은 secondary 로 표시된다.

rs0:SECONDARY>

- arbiter 서버의 mongo 명령줄은 아래와 같이 표시된다.

rs0:ARBITER>

**6. router 서버에서 샤드 서버 추가**

- arbiter 를 제외하고 primary, secondary 서버 추가

$ mongo

> sh.addShard(“rs0/mongodbp01:27018,mongodbs01:27018”)

- 파란색 부분은 primary 와 secondary 서버에서 입력한 mongod.conf 파일 중 “replication”의 “replSetName” 값 입력

- 노란색 부분은 primary의 mongo 에서 rs.status() 명령어를 실행한 후 “members”의 “stateStr” 값이 각각 primary, secondary인 요소의 “name”값을 입력해야 한다. 한 마디로 주소값들

- 샤드 상태 확인

> sh.status()

**7. 새로운 레플리카 셋(Replication Set) 생성**

- primary02 서버와 secondary02 서버 하나씩 생성

- 각 서버의 mongo config 파일 수정 (default 경로 : /etc/mongod.conf)

-> 아래 옵션 추가

processManagement:

fork: true

net:

port: 27018

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “rs1”

sharding:

clusterRole: shardsvr

- 기존 레플리카 셋과 다르므로 이름을 변경해준다. (rs0 -> rs1)

- 각 서버의 mongoDB 구동

$ sudo mongod -f /etc/mongod.conf

- primary 서버에서만 replication set 시작

$ mongo localhost:27018

> rs.initiate()

- arbiter02 는 기존 서버에서 다른 포트로 하나 개방

- 기존 mongo config 파일을 복사해 새로운 파일 생성 (경로 : /etc/mongod\_arb02.conf)

-> 아래 옵션을 보고 다른 곳 수정

processManagement:

fork: true

net:

port: 27022

bindIp: 0.0.0.0

setParameter:

enableLocalhostAuthBypass: false

replication:

replSetName: “rs1”

sharding:

clusterRole: shardsvr

- 기존 arbiter 의 포트 번호가 27021이므로 다르게 설정해주어야 한다.

- 레플리카 셋 이름도 primary02, secondary02 와 맞게 설정 (rs0 -> rs1)

- mongoDB 구동

$ sudo mongod -f /etc/mongod\_arb02.conf

- 두 arbiter 가 실행이 잘 되고 있는지 확인

$ ps -ef | grep mongo

- 레플리카 셋 실행은 하지 않는다.

**7-1. 각 서버 연결 작업**

- primary02 서버의 mongo 에서 레플리카 셋(Replication Set)에 secondary02 서버와 arbiter02 서버를 추가

> rs.add(“mongodbs01:27018”)

> rs.addArb(“mongoarb:27021”)

- 노란색 부분은 각 서버(secondary, arbiter)의 퍼블릭 IPv4 DNS 서버 주소와 mongo 포트번호를 입력해야 한다.

※ 위에서 언급한 private ip 주소 참고.

- 추가된 레플리카 셋 멤버 확인

> rs.status()

- 첫 번째 서버의 mongo 명령줄을 확인했을 때 아래처럼 나오면 성공

rs1:PRIMARY>

- 두 번째 서버의 mongo 명령줄은 secondary 로 표시된다.

rs1:SECONDARY>

- arbiter 서버의 mongo 명령줄은 아래와 같이 표시된다.

rs1:ARBITER>

**7-2. router 서버에서 샤드 서버 추가**

- arbiter02 를 제외하고 primary02, secondary02 서버 추가

$ mongo

> sh.addShard(“rs1/mongodbp01:27018,mongodbs01:27018”)

- 파란색 부분은 primary02 와 secondary02 서버에서 입력한 mongod.conf 파일 중 “replication”의 “replSetName” 값 입력

- 노란색 부분은 primary02의 mongo 에서 rs.status() 명령어를 실행한 후 “members”의 “stateStr” 값이 각각 primary, secondary인 요소의 “name”값을 입력해야 한다. 한 마디로 주소값들

- 샤드 상태 확인

> sh.status()

**8. 샤드 DB 및 컬렉션 명시**

- 먼저 router 서버 mongo 에서 데이터를 저장할 데이터베이스 및 컬렉션 생성

(여기서 데이터베이스 이름은 final, 컬렉션 이름은 v1 으로 진행)

- router 서버 mongo 환경에서 샤딩할 데이터베이스와 컬렉션, 샤딩기법 설정

> sh.enableSharding("final")

> sh.shardCollection("final.v1", { \_id: "hashed" })

- 파란색 부분은 데이터베이스 이름, 노란색 부분은 “데이터베이스이름.컬렉션이름” 형식으로 입력

- 회색 부분은 데이터를 어떤 필드를 기준으로 어떻게 분산 저장할지 명시 (샤드 키)

( \_id 는 데이터 필드 이름, hashed 는 기법 )

- 추가된 데이터베이스, 컬렉션 및 상태 확인

> sh.status()

**9. 테스트**

- final 데이터베이스의 v1 컬렉션에 10000개의 더미데이터 삽입

> use final

> for (var i = 0; i < 10000; i++) db.v1.insert({test: i})

- 두 레플리카 셋이 잘 샤딩되어 있다면, primary 서버와 primary02 서버에 final 데이터베이스와 v1 컬렉션이 존재하지 않아도 router 서버에서 데이터를 삽입한다면 자동적으로 두 서버에 데이터베이스와 컬렉션이 생성이 되고 데이터가 분산되어 저장된다.

- 두 primary 서버에 데이터가 분산되었는지 확인

> db.v1.getShardDistribution()

- 거의 절반으로 분산되어 저장

**10. 샤드 삭제**

샤드를 어쩔 수 없이 삭제해야 하는 경우, 일단 모든 샤드 서버에 접속이 되는지 확인

(모든 primary 및 secondary 서버)

- router 서버에 접속하여 샤드 구성 확인

> use config

> db.shards.find({})

- 샤드 제거

> use admin

> db.runCommand({ removeShard: “rs1” })

- 노란색 부분은 제거하고 싶은 샤드 replica set의 이름을 입력해준다.

- 이름은

- 위에 샤드 구성 확인 명령어로 “\_id” 필드의 값을 확인하거나,

- mongod.conf 파일을 확인하거나,

- 해당 mongo 서버에서 rs.status() 명령어를 입력했을 때 “set” 필드의 이름을 확인

- 제거 과정 확인

> use admin

> db.runCommand({ removeShard: “rs1” })

- 같은 명령어를 다시 입력해주면 확인 가능

- “remaining” 필드에 “chunks” 값이 계속 내려가는 것을 확인할 수 있다. 0 까지 내려가야 한다.

- 만약 “dbsToMove”에 데이터베이스 목록이 존재한다면 해당 데이터베이스의 서버를 다른 set의 primary로 수동으로 옮겨줘야 한다.

> db.runCommand({ movePrimary: “db\_name”, to: “rs3” })

- “chunks” 값이 0이 됐고, “dbsToMove” 데이터베이스 리스트도 빈 리스트가 되면

- 위의 removeShard 명령어를 입력했을 경우 removeShard completed successfully 메시지가 뜬다.

- 샤드 제거 후, 상태 확인

> use config

> db.shards.find()

- 목록에 해당 샤드 set이 사라진 것 확인

- 삭제된 샤드 set에 있던 데이터는 자동으로 다른 set으로 이관된다.

11. 기타

- primary 서버가 mongo에서 secondary로 잡혀있고, secondary 서버가 mongo에서 primary로 잡혀있는 경우

- primary로 잡혀있는 서버의 mongo에 들어가서 rs.stepDown() 명령어 사용

> rs.stepDown()

- primary 서버 mongo 프로세스를 sudo mongod -f /etc/mongod.conf 명령어로 올리려고 해도 안올라가지고, 긴 시간 기다려서 올라갔는데 막상 mongo 접속해보니 rs0:ROLLBACK> 으로 뜨는 경우

-> DB가 깨진 것으로 추정됨… mongod.conf 파일에서 db path 확인한 후, 원래 db path 디렉토리를 백업해놓고 새로운 디렉토리를 만들어서 다시 프로세스 시작하면 정상적으로 작동한다.

- config 서버 용량도 확인해야 할 것으로 보임.

**12. 샤드 이관 (aws -> azure)**

**router, config, primary, secondary 신규 작업**

**primary, secondary는 기존 셋과 동일 개수로 작업**

- primary, secondary, config, arbiter

1. 신규 생성
2. 기존 셋과 동일 개수 필요

- router

1. 신규 생성
2. 신규 replica set addShard 작업
3. 기존 db hashed 작업

- 각각의 백업할 primary에서 dump 작업

> mongodump --port=27018

- dump 후 폴더 router 서버로 파일 전송

- router 서버에서 입력 \*(dump 폴더 내에 config 폴더 삭제 후 입력)

> mongorestore ./dump폴더

***- 끝 -***

번외1) 참고문헌

https://rastalion.me/mongodb-shard-cluster-%EA%B5%AC%EC%84%B1%ED%95%98%EA%B8%B0/

번외2) 계속해서 레플리카 셋을 추가하고 싶을 때 7, 7-1, 7-2번 처럼 세트를 계속 만들어서 추가해주면 된다.

번외3) 레플리카 셋을 구성할 때 ip나 포트번호를 신중하게 입력해서 진행해야 나중에 꼬이지 않고 한번에 구성할 수 있다.

번외3) 레플리카 셋 시작(rs.initiate())을 실수로 했거나 다시 초기화 하고 싶은 경우, 해당 서버에 mongod.conf 파일에 있는 storage의 dbPath 경로 (default : /var/lib/mongodb) 폴더 내용을 모두 지워버리고 mongo 서버를 껐다 키면 초기화된다. (router 서버 제외)

번외4) 샤드 키의 뜻 (댓글에 hashed 사용 이유까지)

https://blog.seulgi.kim/2014/08/mongodb-sharding-3-shard-key.html

번외5) 샤딩은 데이터를 특정 기준으로 그룹핑해서 관리하는 것이며, 그 그룹을 청크(chunk) 라고하는 단위로 관리 (<https://mysterico.tistory.com/7>)

- chunksize 확인

> use config

> db.settings.find()

- chunksize 수정

> db.settings.save({\_id: “chunksize”, value: “16”})

- 단위 : MB

final DB 테스트 현황

v2

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v2”, {\_id: “hashed”})

for (var i = 0; i<10000; i++) db.v2.insert({name: "final" + i});

🡪 primary01, primary02에 분산저장

v3

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v3”, {\_id: “hashed”})

for (var i = 0; i<10000; i++) db.v3.save({name: "final" + i}); // insert를 save로 변경

🡪 primary01, primary02에 분산저장

v4

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v4”, {\_id: 1}) // shard key 를 1로 변경

for (var i = 0; i<10000; i++) db.v4.insert({name: "final" + i});

🡪 primary02에만 저장

v5

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v5”, {\_id: “hashed”})

for (var i = 0; i<1000000; i++) db.v5.insert({name: "final" + i}); // 개수를 100만개로 변경

🡪 primary01, primary02에 분산저장. 15분 정도 소요

v6

db 세팅 변경 – chunksize : 16

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v6”, {\_id: “hashed”})

for (var i = 0; i<1000000; i++) db.v6.insert({name: "final" + i});

🡪 primary01, primary02에 분산저장. 15분 정도 소요. 분산 데이터 크기엔 별 차이가 없다.

v7

chunks - rs0 : 2, rs1 : 2

sh.shardCollection(“final.v7”, {\_id: “hashed”})

컬렉션 v7 에 csv import로 데이터 삽입

🡪 primary01, primary02에 분산저장