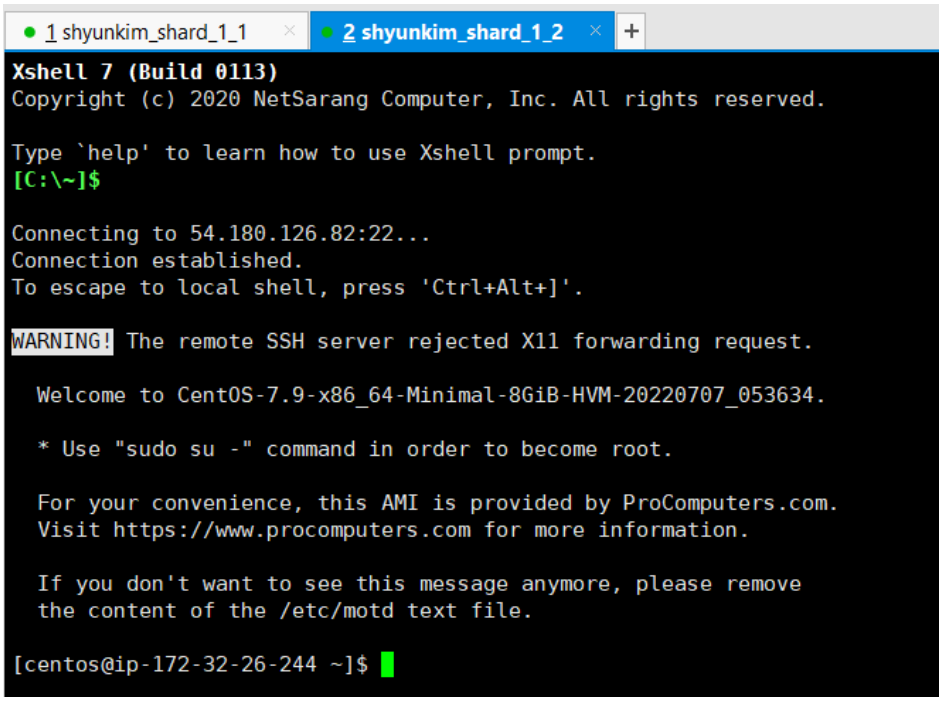
Mongo 6.0 Replication & 3 shard 구성 in AWS EC2 instance

1. AWS EC2 인스턴스 총 13 개 생성 (생성 과정은 생략 하겠다.)

구분	서버 이름	Public IP	Private IP
Config server	shyunkim_configsvr_1_1	3.35.137.32	172.32.222.4
	shyunkim_configsvr_1_2	13.124.176.208	172.32.119.244
	shyunkim_configsvr_1_3	3.39.248.116	172.32.92.11
Router	shyunkim_mongos	3.35.167.189	172.32.114.98
Shard 1	shyunkim_shard_1_1	3.35.149.116	172.32.176.183
	shyunkim_shard_1_2	43.200.252.73	172.32.130.55
	shyunkim_shard_1_3	52.78.61.251	172.32.82.245
Shard 2	shyunkim_shard_2_1	3.36.133.73	172.32.30.9
	shyunkim_shard_2_2	3.34.192.99	172.32.21.124
	shyunkim_shard_2_3	43.201.34.136	172.32.212.60
Shard 3	shyunkim_shard_3_1	3.35.138.220	172.32.162.46
	shyunkim_shard_3_2	13.125.98.206	172.32.51.237
	shyunkim_shard_3_3	13.124.201.107	172.32.42.154

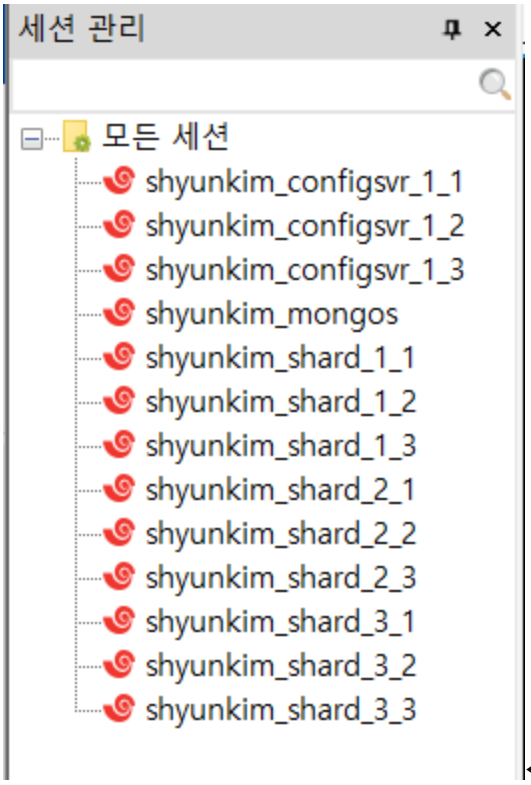
2. 연결 관리를 위해 Xshell 을 설치
(상업적 목적이 아니므로 가정용 무료 버전을 설치 하자)

2-1 . Xshell 에 인스턴스 연결
[새로만들기] → 이름 및 호스트 (IP) 입력 → 사용자 인증 탭 → password 체크박스 해제 및 public key 선택 및 가져오기 하여 .pem 파일 불러오기
→ 사용자 명은 반드시 centos or ec2-user 로 해야함 참고링크



연결 성공 이미지.

2-2.



← 어마어마한 노가다를 모두 완료했다..!

3. 이제 각각의 서버에 mongoDB 를 설치할 것이다. 하지만 이번에는 김수아 주임님이 말씀하신

Percona Server for MongoDB 4.2 버전을 사용해 보려고한다.

하지만 그전에 정확히 기존의 mongoDB 와 대체 무슨 차이가 있길래 앞에 전치사마냥 붙은 건지 알아보자...!!

Percona Server for MongoDB

Percona Server for MongoDB 4.2 이 링크의 document 을 확인해 보면 다운로드 방법을 친절히 알려준다.

```
[centos@ip-172-32-146-184 ~]$ sudo yum list percona-server-mongodb --show
duplicates
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: download.cf.centos.org
 * extras: download.cf.centos.org
 * updates: download.cf.centos.org
psmdb-42-release-x86_64                          | 2.9 kB     00:00
psmdb-42-release-x86_64/7/primary_db               |  79 kB   00:01
Available Packages
Percona-Server-MongoDB.x86_64  3.0.5-rel0.7rc.el7  percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.7-1.0.el7       percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.7-1.1.el7       percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.8-1.2.el7       percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.8-1.3.el7       percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.9-1.4.el7       percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.10-1.5.el7      percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.11-1.6.el7      percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.12-1.7.el7      percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.12-1.8.el7      percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.14-1.9.el7      percona-release-x86_64
Percona-Server-MongoDB.x86_64  3.0.15-1.10.el7     percona-release-x86_64
percona-server-mongodb.x86_64  4.2.0-1.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.1-1.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.2-3.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.3-4.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.5-5.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.6-6.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.7-7.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.8-8.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.9-9.el7         psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.9-10.el7        psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.10-11.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.11-12.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.12-13.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.13-14.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.14-15.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.15-16.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.17-17.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.18-18.el7       psmdb-42-release-x86_64
percona-server-mongodb.x86_64  4.2.19-19.el7       psmdb-42-release-x86_64
```

← 설치 가능한 패키지를 확인후 4.2 버전의 가장 최신 버전인 4.2.22 버전을 설치해보았다. (이미지는 아래가 잘렸다.)

```
Complete!
[centos@ip-172-32-146-184 ~]$ mongo --version
Percona Server for MongoDB shell version v4.2.22-22
git version: d6a3e69b77ba71cfc12454763000bb0fbbea1e54
OpenSSL version: OpenSSL 1.0.2k-fips  26 Jan 2017
allocator: tcmalloc
modules: none
build environment:
    distarch: x86_64
    target_arch: x86_64
[centos@ip-172-32-146-184 ~]$
```

설치가 완료된 모습이다. 자 이제 남은 12 개에 대해서도 반복해 보자.

4. 이제 각 node 의 설정을 수정하고 config Server 의 Replica set 을 구성해보자!...!! (3 번에서 4 번 넘어가는 데 4 시간 걸렸다;;;)

   노드간 (Replica set) 간의 통신을 위한 키를 만들어야한다.

   openssl rand -base64 756 > /etc/keyfile
   위 명령어를 한 노드에서 실행한뒤
   cat /etc/keyfile 을 실행하여 해당 내용을 복사해서 나머지 노드들에 같은 위치에 만든다.

   chmod 400 /etc/keyfile
   chown mongod:mongod /etc/keyfile

   위 명령어를 모든 노드에 실행시켜준다.

   이젠 mongod.conf 파일을 수정할 것이다.

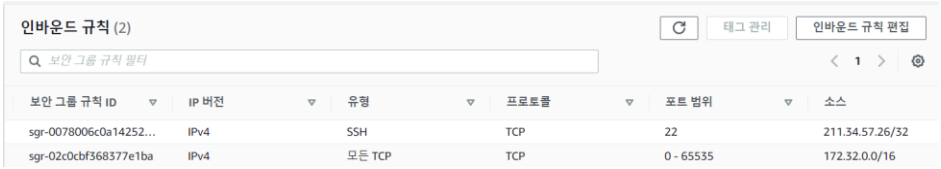
   vi /etc/mongod.conf 을 실행하고

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  keyFile: /etc/keyfile
#operationProfiling:

replication:
  replSetName: cfg
sharding:
  clusterRole: configsvr
## Enterprise-Only Options:

#auditLog:

#snmp:
~
```

줄쳐진 부분처럼 수정한다.

자 이제 telnet 패키지를 이용하여 각 노드간의 통신이 제대로 이루어지고 있는지를 먼저 확인할 것이다. yum install telnet 으로 패키지를 다운받고 telnet <Private IP> 27017 을 해서 확인해보자
* 여기서 만약 연결이 되지 않는다면 AWS 보안 규정을 확인해보자 필자의 경우 보안 규정을 몰라 2 시간 이상 낭비했다...!!

AWS EC2 그룹의 보안 그룹을 수정해 주어야 한다. 각 노드끼리의 통신을 위해서는 같은 보안 그룹으로 묶여있어야 한다



위 이미지와 같은 보안그룹을 새로 만들어 각 노드에 모두 적용해 줬다.
그랬더니..?

```
[root@ip-172-32-146-184 centos]# telnet 172.32.84.144 27017
Trying 172.32.84.144...
Connected to 172.32.84.144.
Escape character is '^]'.
^]
```

연결이 성공적으로 되었다^^

5. 자 이제 mongo 명령어를 이용해 해당 노드에 접속하여 replica set 을 init 해보자

```
[root@ip-172-32-146-184 centos]# mongo
Percona Server for MongoDB shell version v4.2.22-22
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServi
ceName=mongodb
Implicit session: session { "id" : UUID("e190415b-2d26-419c-8d14-8a8aca04e
4c7") }
Percona Server for MongoDB server version: v4.2.22-22
> use admin
switched to db admin
> rs.initiate({_id:"cfg",configsvr:true,members:[{_id:0,host:"172.32.146.1
84:27017"},{_id:1, host:"172.32.84.144:27017"},{_id:2, host:"172.32.216.11
4:27017"}]})
{
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1665476401, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0)
}
cfg:SECONDARY>
```

initiate() 하는 방법은 기존 로컬에서 진행했던 것과 동일하다. 그저 hostname 부분에 Private IP 를 넣어주면 된다.

6. 이번에는 각 Shad 의 Replica set 을 구성해보자

위의 과정과 거의 유사하다.
config 에 사용했던 keyfile 을 그대로 복사하여 똑같은 경로에 복제한다.

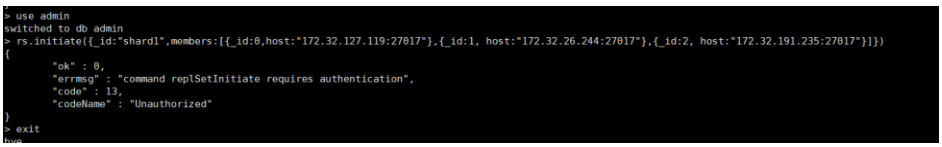
```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  keyFile: /etc/keyfile
#operationProfiling:

replication:
  replSetName: shard1
sharding:
  clusterRole: shardsvr
```

mongod.conf 파일에서 줄친 부분이 다르다는 것은 인지할 필요가 있다.

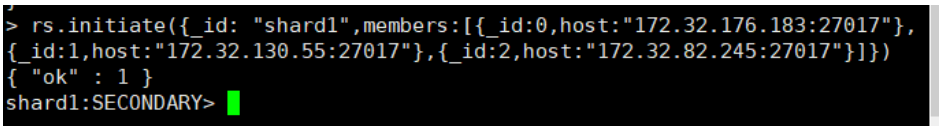
각 3 개의 노드의 설정을 마치고 노드 한개에 mongo 명령어로 접속하여 replica

set 을 init 한다.

rs.initiate({_id:"shard1",members:[{_id:0,host:"172.32.127.119:27017"},{_id:1, host:"172.32.26.244:27017"},{_id:2, host:"172.32.191.235:27017"}]})



에러 발생 .. → 이전 연습하던 와중에 만들었던 auth 와 관련한 오류로 판단된다. 내일 인스턴스를 다시 만들어서 다시 도전해보려고한다.

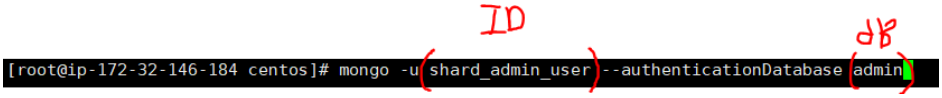


인스턴스를 새로 만들고 위의 과정을 반복했다. 역시 성공했다. 이제 나머지 shard2 , shard3 까지 replicaset 을 구성해보자

7. 모든 replicaset 의 구성을 완료했다. 다음은 각 shard 에 접근 할 수 있는 auth 를 가진 유저를 만들어주자

Primary 노드에서 만들어주어야한다는 것을 잊지 말자.

해당 부분에서 본인은 실수로 Config 서버에도 만들어버렸다. 꼼짝없이 인증에 늪에 빠져버렸지만 걱정하지말자 구글링이 있다.
인증을 가진 상태에서 mongo 로 접근 하기 위해서는



이 명령어를 기억하자

** 큰일 났다. 만들어진 유저를 삭제하기 위해서는 인증이 필요한데 그러기위해 'shard_admin_user' 로 인증하고 'shard_admin_user' 유저를 삭제했다. 그 뒤로는 어떠한 방식으로도 인증이 진행이 안된다.
난 정말 멍청한듯 하다. 모든 config instance 를 삭제하고 다시만들도록 하겠다.

config 노드 3 개를 다시 만들고 재설정 한뒤 다시 시도했다.

8. 라우터 (mongos) 를 위한 config 설정을 진행해보자

일단 mongos 를 위한 .conf 파일을 만들어주어야한다.
여기서 cp 명령어는 복사 명령어니까 알아두자

```
[root@ip-172-32-114-98 centos]# cp /etc/mongod.conf /etc/mongos.conf
```

이제 mongos.conf 파일을 수정할 것이다.

```
# network interfaces
net:
  port: 27018
  bindIp: 0.0.0.0

security:
  keyFile: /etc/keyfile
#operationProfiling:

#replication:

sharding:
  configDB: cfg/172.32.222.4:27017,172.32.119.244:27017,172.32.92.11:27017
```

```
# Where and how to store data.
#storage:
#  dbPath: /var/lib/mongo
#  journal:
#    enabled: true
#  engine: wiredTiger
#  engine: inMemory
```

기존의 .conf 파일과 달라지는 부분만 캡처했다.
여기서 27018 번 포트로 지정해준 이유는 Mongo Document 에서 mongos port 를 27018 번 포트로 지정해야한다고 한다.
그리고 configDB : 의 내용으로는 <config 서버 replSetName> / <config server primary hostname : port> , <config server secondary hostname : port> , ... 이런 식이다.

```
[root@ip-172-32-114-98 centos]# mongos --config /etc/mongos.conf
about to fork child process, waiting until server is ready for connections.
forked process: 10231
child process started successfully, parent exiting
```

이제 mongos 라우터 서버를 해당 .conf 파일로 접속해보자.

9. 이제 router 를 관리할 root 사용자를 만들어주어야한다.

```
mongos> use admin
mongos> db.createUser(
... ... {
... ... user:"cluster_admin_user",
... ... pwd:"124578",
... ... roles:[
... ... {role:"dbAdminAnyDatabase",db:"admin"},
... ... {role:"userAdminAnyDatabase",db:"admin"},
... ... {role:"readWriteAnyDatabase",db:"admin"},
... ... {role:"clusterAdmin",db:"admin"}
... ... ]
... ... })
```

```
mongos> db.auth("cluster_admin_user","[redacted]")
1
```

인증은 이 명령어다 기억하자

10. addShard() 명령어를 이용하여 shard 를 합치자...!

```
mongos> sh.addShard("shard1/172.32.176.183:27017")
{
        "shardAdded" : "shard1",
        "ok" : 1,
        "operationTime" : Timestamp(1665542337, 7),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665542337, 7),
                "signature" : {
                        "hash" : BinData(0,"v+pi52HRMNZZb07iZd6glYBYexo="),
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
mongos> sh.addShard("shard2/172.32.30.9:27017")
{
        "shardAdded" : "shard2",
        "ok" : 1,
        "operationTime" : Timestamp(1665542410, 7),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665542410, 7),
                "signature" : {
                        "hash" : BinData(0,"PKyhQ/vdp6y+JUJ2gI4AZZRL0iU="),
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
```

```
mongos> sh.addShard("shard3/172.32.162.46:27017")
{
        "shardAdded" : "shard3",
        "ok" : 1,
        "operationTime" : Timestamp(1665542437, 25),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665542437, 25),
                "signature" : {
                        "hash" : BinData(0,"ns6CQlfY36a9Il7chnbZGQzJvYE="),
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
```

이제 sh.status() 명령어 입력시 현재 Sharded cluster 의 상태가 나올것이다.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("63462280660f2dfca2199e2d")
  }
  shards:
        {  "_id" : "shard1",  "host" : "shard1/172.32.130.55:27017,172.32.176.183:27017,172.32.82.245:27017",  "state" : 1 }
        {  "_id" : "shard2",  "host" : "shard2/172.32.21.124:27017,172.32.212.60:27017,172.32.30.9:27017",  "state" : 1 }
        {  "_id" : "shard3",  "host" : "shard3/172.32.162.46:27017,172.32.42.154:27017,172.32.51.237:27017",  "state" : 1 }
  active mongoses:
        "4.2.22-22" : 1
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                24 : Success
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shard1  1000
                                shard2  19
                                shard3  5
                        too many chunks to print, use verbose if you want to force print
```

완벽하다!!!!!!!!

TEST 를 위해 1000 개의 더미 데이터를 생성하고 정확히 Sharding 이 되는지 확인해 보자

```
mongos> for (var i=0;i<1000;i++) db.item.insertOne({"item":i})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("63464bd77722e26150681269")
}
```

```
mongos> db.item.find().count()
1000
```
1000 개의 데이터를 넣어주었다

MongoDB 는 자기 마음대로 sharding 을 하지 않고, sharding 하라는 명령어가 주어져야 시작한다.

1. 제일 먼저 해당 데이터베이스를 샤딩 가능 상태로 만든다.

```
mongos> sh.enableSharding("test")
{
        "ok" : 1,
        "operationTime" : Timestamp(1665551449, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665551449, 3),
                "signature" : {
                        "hash" : BinData(0,"ORDxpd5dwup8OEK7OfinHqBoPfs=")
,
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
mongos>
```

2. Index 를 만든다.

기본적으로 _id:1 인덱스가 생성되는데, 아래 명령어를 통해 다른 인덱스를 만들 수 있다.

_id : 1 과 -1 은 오름차순, 내림차순 정렬 인덱싱을 의미하고

hashed 는 hash 값으로 정렬 인덱싱을 의미한다.

< index 만들기(ranged) >

db.[데이터베이스 이름].createIndex({ [콜렉션의 키 이름] : 1 })

예) db.mydatabase.createIndex({ _id: 1 })

< index 만들기(ranged) >

db.[데이터베이스 이름].createIndex({ [콜렉션의 키 이름] : "hashed" })

예) db.mydatabase.createIndex({ _id: "hashed" })

< compound index 만들기 >
db.[데이터베이스 이름].createIndex({ [콜렉션의 키 이름] : 1, [콜렉션의 키 이름] :1, ... })

예) db.mydatabase.createIndex({ _id: 1, name : 1})

< index 가져오기 >
db.[데이터베이스 이름].getIndexes()

예) db.mydatabase.getIndexes()

< index 제거하기 >
db.[데이터베이스 이름].dropIndex({ [콜렉션의 키 이름] : 1 })

예) db.mydatabase.dropIndex({ name : 1 })

3. 데이터베이스 내에서 샤딩하고 싶은 콜렉션을 샤딩시킨다.

방법은 두 가지가 있는데, hashed 샤딩, ranged 샤딩이 있다.

ranged 로 sharding 하려면 ranged index 가 있어야 하고,

hashed 로 sharding 하려면 hashed index 가 있어야 한다.

ranged sharding : sh.shardCollection("[데이터베이스 이름].[콜렉션 이름]", { [콜렉션의 키 이름] : 1 })

hashed sharding : sh.shardCollection("[데이터베이스 이름].[콜렉션 이름]", { [콜렉션의 키 이름] : "hashed" })

예)

ranged sharding : sh.shardCollection("mydatabase.mycollection", { _id : 1 })

hashed
sharding : sh.shardCollection("mydatabase.mycollection", { _id : "hashed" })

```
mongos> db.item.createIndex({"key":"hashed"})
{
        "raw" : {
                "shard3/172.32.162.46:27017,172.32.42.154:27017,172.32.51.
237:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1665552627, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665552627, 2),
                "signature" : {
                        "hash" : BinData(0,"R8Wr7WuoY7ZHjZAkKEKbXO0vxd0=")
,
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
```

index 를 만들고 ,

```
mongos> sh.shardCollection("test.item",{"key":"hashed"})
{
        "collectionsharded" : "test.item",
        "collectionUUID" : UUID("0352fb00-1a81-4f72-bd24-f6afc330d28d"),
        "ok" : 1,
        "operationTime" : Timestamp(1665552640, 10),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1665552640, 10),
                "signature" : {
                        "hash" : BinData(0,"rNGRgyAHY49B0qazVaNTcOwxNT8=")
,
                        "keyId" : NumberLong("7153442991275769888")
                }
        }
}
```

sharding 을 했다..! 근데 문제 발생...

3000 개 정도 씩 나누어질 것이라고 예상했지만 실제로는 shard3 에 10000 개의 doc 모두 저장되며 예상과 전혀 달랐다.

```
mongos> db.item.getShardDistribution()

Shard shard3 at shard3/172.32.162.46:27017,172.32.42.154:27017,172.32.51.2
37:27017
 data : 341KiB docs : 10000 chunks : 1
 estimated data per chunk : 341KiB
 estimated docs per chunk : 10000

Totals
 data : 341KiB docs : 10000 chunks : 1
 Shard shard3 contains 100% data, 100% docs in cluster, avg obj size on sh
ard : 35B
```

해결 방안 (1) Chunk Size 를 줄여보자

collection 을 여러 조각으로 파티션하고 각 조각을 여러 샤드 서버에 분산해서 저장하는데 이 데이터 조각을 chunk 라고 한다.
이러한 chunk 는 각 샤드서버에 균등하게 저장되어야 좋은 performance 를 낼 수 있는데 균등하기 저장하기 위해
mongodb 에서는 큰 chunk 를 작은 chunk 로 chunk split 하고, chunk 가 많은 샤드에서 적은 샤드로 chunk migration 을 수행한다

혹시 Chunk Size 가 커서 한개의 Chunk 안에 다담겨 버린게 아닐까?

Modify Chunk Size in a Sharded Cluster ← 여기서 chuck size 를 바꾸는 명령어를 확인할 수 있다.

** 결과적으로는 달라지지 않았다.. 잉?

해결 방안 (2) 데이터 크기 자체가 작아서 그런게 아닐까?
데이터를 10000 개가 아닌 50000 개를 넣어보자.. 라고 생각하고 for (var i=0;i<40000;i++) db.item.insertOne({"key":i}) 를 실행했다.

당연히 50000 개의 데이터가 조회될 줄 알았지만

```
mongos> db.item.find().count()
73298
```

아니 이게 무엇인가 ... ? ;;;;;;;;;;;;;;;;;;;;;;;;;;

일단 OK , Sharding 은 어떻게 되는지 보자

```
mongos> db.item.getShardDistribution()

Shard shard1 at shard1/172.32.130.55:27017,172.32.176.183:27017,172.32.82.245:27017
 data : 854KiB docs : 25012 chunks : 1
 estimated data per chunk : 854KiB
 estimated docs per chunk : 25012

Shard shard3 at shard3/172.32.162.46:27017,172.32.42.154:27017,172.32.51.237:27017
 data : 1.61MiB docs : 48286 chunks : 1
 estimated data per chunk : 1.61MiB
 estimated docs per chunk : 48286

Totals
 data : 2.44MiB docs : 73298 chunks : 2
 Shard shard1 contains 34.12% data, 34.12% docs in cluster, avg obj size on shard : 35B
 Shard shard3 contains 65.87% data, 65.87% docs in cluster, avg obj size on shard : 35B
```

일단 희망적인 소식은 분배되는 shard 가 한개 늘었다. (짞짞짞짞 ..)
절망적인 소식은 여전히 Shard2 는 사용되지 않고 있다는 것이다. 대체 왜일까,...?

새롭게 도전하는 의미에서 더 많은 데이터를 넣어보았다. 사실 document 가 { _id : 1~1000000} 뿐인 매우 작은 데이터이기 때문에 사실 100 만개라도 상관없을 것이다.

그래서 실행했다. (Thanks to 수아 주임님)

```
mongos> for( i =0;i<1000000;i++) db.item.insertOne({"item":i})

{
        "acknowledged" : true,
        "insertedId" : ObjectId("63465ffc7722e26150783f09")
}
mongos>
```

무식하게 한다고 욕하지 않았으면 좋겠다. 나도 알고있으니까... 할

또 이번에는 chunk 사이즈를 6mib 로 정했다.

```
switched to db config
mongos> db.settings.find()
{ "_id" : "chunksize", "value" : 1 }
mongos> db.settings.save({"_id":"chunksize
2022-10-12T06:38:06.892+0000 E  QUERY    [js] SyntaxError: "" literal not terminated before end of script :
@(shell):1:34
mongos> db.settings.save({"_id":"chunksize","value":6})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongos> db.settings.find()
{ "_id" : "chunksize", "value" : 6 }
```

```
mongos> db.item.getShardDistribution()

Shard shard3 at shard3/172.32.162.46:27017,172.32.42.154:27017,172.32.51.237:27017
 data : 34.33MiB docs : 1000000 chunks : 12
 estimated data per chunk : 2.86MiB
 estimated docs per chunk : 83333

Shard shard1 at shard1/172.32.130.55:27017,172.32.176.183:27017,172.32.82.245:27017
 data : 11.99MiB docs : 349523 chunks : 12
 estimated data per chunk : 1023KiB
 estimated docs per chunk : 29126

Shard shard2 at shard2/172.32.21.124:27017,172.32.212.60:27017,172.32.30.9:27017
 data : 10.99MiB docs : 320397 chunks : 11
 estimated data per chunk : 1023KiB
 estimated docs per chunk : 29127

Totals
 data : 57.33MiB docs : 1669920 chunks : 35
 Shard shard3 contains 59.88% data, 59.88% docs in cluster, avg obj size on shard : 36B
 Shard shard1 contains 20.93% data, 20.93% docs in cluster, avg obj size on shard : 36B
 Shard shard2 contains 19.18% data, 19.18% docs in cluster, avg obj size on shard : 36B
```

..? 성공이다. 와우 이유를 정확히 파악 할 수 없으니 같은 과정을 바꿔보고 chunk 사이즈도 바꿔볼 생각이다