Seminarski rad

- Sistemi za upravljanje bazama podataka -

Tema: *Cluster rešenja (database clusters) kod MongoDB*

Student:
Marko Oblak

Mentor:
Aleksandar Stanimirović

Sadržaj

Uvod

MongoDB je vodeća NoSQL baza podataka i pripada dokument orjentisanim bazama podataka (engl. document oriented). Napisana je u C++ jeziku i otvorenog je koda, izdata pod kombinacijom GNU Affero General Public License i Apache License. MongoDB čuva podatke kao JSON dokumente sa dinamičkim šemama. JSON (JavaScript Object Notation) je otvoreni standard zasnovan na tekstu, osmišljen za razmenu podataka koji su pogodni za čitanje ljudima. MongoDB je prihvaćen kao backend softver brojnih značajnih veb-sajtova i servisa, uključujući Craigslist, eBay, Foursquare, SourceForge i New York Times.

Osnovne karakteristike:

- smeštanje je usmereno na dokumenta
- apsolutna podrška indeksiranju
- automatsko skaliranje – horizontalno skaliranje bez ugrožavanja funkcionalnosti
- moćni upiti bazirani na dokumentima
- brzi update
- fleksibilna agregacija i obrada podataka
- GridFS – čuva fajlove bilo koje veličine

Klasterizacija baza podataka

Klasterizacija baza podataka predstavlja proces kombinovanja više od jednog servera ili instanci koje su povezane na jedinstvenu bazu podataka. Kada jedan server nije u mogućnosti da upravlja velikom količinom podataka ili velikim brojem zahteva koji pristižu, onda je potrebno iskoristiti klasterizaciju podataka.

Redundantnost podataka

Redundatnost označava nešto suvišno, tj. u ovom slučaju nešto što se ponavlja. Kada više računara radi zajedno, u cilju skladištenja velike količine podataka i pružanja istih podataka jedan drugom, lako može doći

do redundantnosti. Svi računari - čvorovi klastera su sinhronizovani, što znači da će svaki imati isti skup podataka. Potrebno je izbeći vrste ponavljanja koje dovode do nejasnoća podataka, a to se javlja upravo zbog sinhronizacije. U slučaju da računar ne radi, imaćemo na raspolaganju sve podatke (replicirati podatke).

Balansiranje opterećenja

Balansiranje opterećenja ili skalabilnost ne dolazi sama po sebi kada se radi o bazi podataka. To se može realizovati klasterizacijom tako što se vrši raspoređivanje radnog opterećenja između različitih računara koji su deo klastera. To znači da ako dodje do toga da veliki broj korisnika istovremeno pristupa bazi, tj. dođe do ogromnog skoka u saobraćaju postoji veća sigurnost da će moći da se podrži novi novi saobraćaj - neće se desiti pad. Bez balansiranja opterećenja određena mašina bi mogla biti preopterećena i saobraćaj bi se usporio i najverovatnije bi se skroz obustavio.

Visok stepen dostupnosti baze podataka

Ako nekome, ko ne zna šta znači visoka dostupnost baze podataka, kažemo da je baza 99 posto dostupna, on će reći da je to i više nego dovoljno. Ako u godini ima 365 dana i procenat uspešnog pristupa bazi je 99 procenta, to znači da bazi tokom godine ne možemo pristupiti skoro 4 dana. Sada to ne zvuči bas uspešno, pogotovo ako u obzir uzmemo neke vodeće kompanije kao što su google i slične. Ova karakteristika zavisi od naših potreba koje su promenljive iz situacije u situaciju - nekad nam je potrebno konstantno izvršavanje transakcija ili bilo koja vrsta analitike nad podacima iz baze. Zbog klasterizacije koja nam pruža balans opterećenja i postojanje dodatnih mašina, dostupnost podacima iz baze podataka se može dovesti na zavidan nivo.

Monitoring (praćenje) i automatizacija

Ove zadatke na lak način možemo koristiti i za uobičajenu bazu podataka korišćenjem softvera, pri čemu se ne uočava prava primena i korisnost. Prednost postaje sve očitija kada je prisutan klaster, pri čemu je omogućeno da se veliki broj procesa automatizuje i izvršava i istovremeno, i kada se omogućava postavljanje pravila radi upozoravanja na potencijalne probleme. To sprečava eventualni ručni povratak da bi se uočilo gde su problemi nastali. Takodje pri automatizaciji kod klasterovanja moguće je dobijati obaveštenja ako se od sistema traži previše. Klaster poseduje određenu mašinu koja će se koristiti kao sistem za upravljanje bazama podataka / kontrolna tabla za ceo klaster. Ova odabrana mašina može imati skripte koje se automatski pokreću za čitav klaster i one rade za sve čvorove iz njega.

Tipovi klastera baza podataka

Klaster baza podataka je veoma obiman i obuhvata više nivoa i uloga u zavisnosti od zahteva sistema. U nastavku će biti prikazane tri vrste računarskih arhitektura klastera.

1. Failover/visoko dostupni klasteri

Rad mašine može nenadano prestati u bilo kom trenutku. Administratori sistema upravljaju takvim situacijama i efikasno rešavaju probleme. Klasteri tu stupaju na snagu - pripremaju dostupnost servisa repliciranjem servera i rekonfiguracijom softvera i hardvera. Dakle, svaki sistem kontroliše druge i radi na zahtevima kada bilo koji drugi padne. Ove vrste klastera su profitabilne za one korisnike koji u potpunosti zavise od svojih računarskih sistema.

Sistem bi trebalo da bude dovoljno sposoban da zna koji su čvorovi pokrenuti, sa koje IP adrese, koji su zahtevi i šta se treba učiniti u slučaju

pada. Najvažnije je da uvek postoji neki server koji će moći da opslužuje korisničke zahteve.

2. Klasteri visokih performansi

Svrha razvoja visoko efikasnih klastera baza podataka je izrada računarskih sistema visokih performansi. Oni pružaju dodatne programe koji su potrebni za računanje koji iscrpljuju vreme. Takvu raznolikost klastera najčešće preferiraju naučne industrije. Osnovni cilj je inteligentna podela opterećenja.

3. Klasteri sa balansiranim opterećenjem

Ovi klasteri baza podataka služe za distribuciju opterećenja između različitih servera. Nastoje da obezbede povećan kapacitet mreže, konačno povećavajući performanse. Sistemi u ovoj mreži integrišu svoje čvorove uz pomoć kojih su zahtevi korisnika podeljeni podjednako po čvorovima klastera. Sistem ne radi zajedno, već preusmerava zahteve pojedinačno u trenutku kada se pojave.

Slika 1. *Šema klastera*

Kreiranje klastera baze podataka MongoDB

Postoje dve strategije za skaliranje baze podataka: vertikalno skaliranje i horizontalno.

Vertikalno skaliranje uključuje dodavanje više resursa serveru kako bi mogao da upravlja većim skupom podataka. Prednost je što je proces obično jednostavan, ali često dolazi do zastoja, i teško je primeniti automatizaciju.

Horizontalno skaliranje uključuje dodavanje više servera kako bi se povećali resursi, i uglavnom se koriste u konfiguracijama koje koriste dinamičke skupove podataka koji brzo rastu. Zbog toga što je ovaj način zasnovan na konceptu dodavanja više servera, a ne dodavanju više resursa jednom serveru, skupovi podataka često zahtevaju da budu izdeljeni na delove i distribuirani na veći broj servera. Izraz sharding se odnosi na razdvajanje skupa podataka u podskupove kako bi se skladištili na odvojenim serverima baza podataka (a sharded clusters).

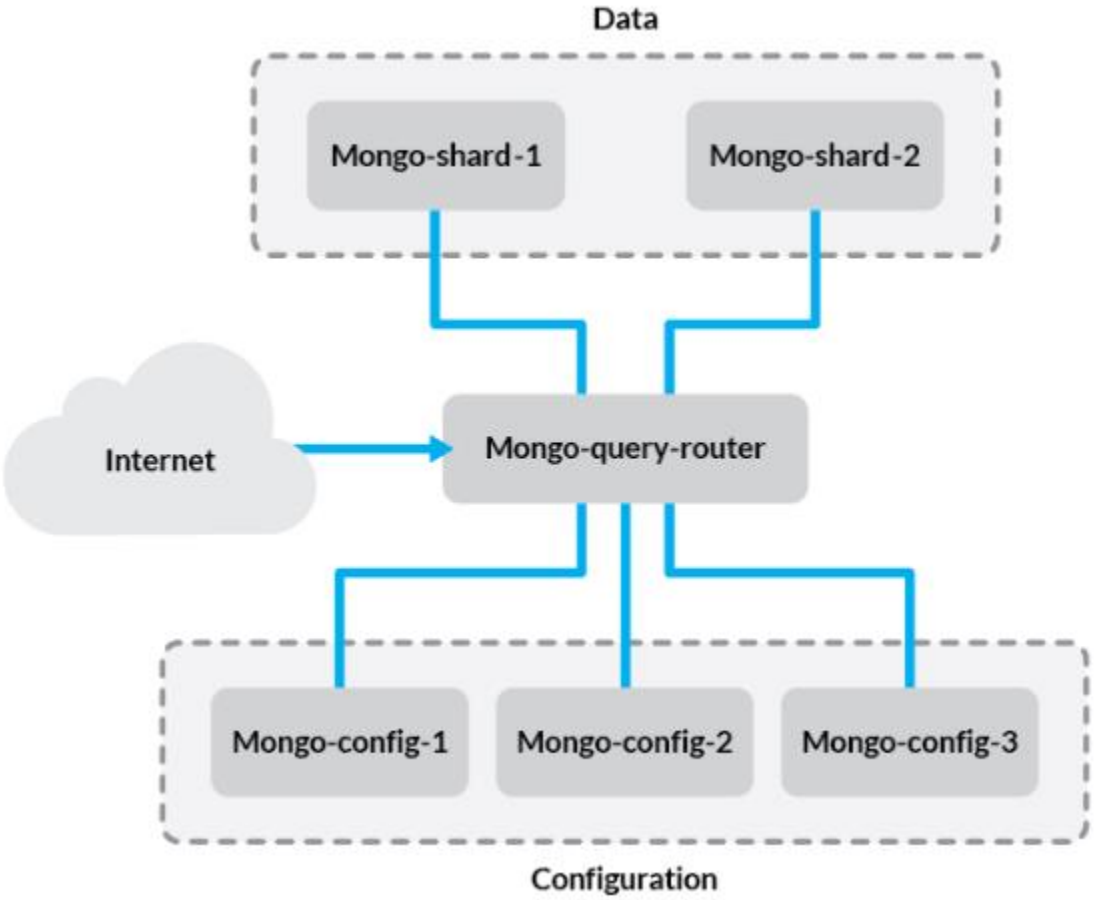
Klaster arhitektura

U nastavku su navedene komponente neophodne za podešavanje baze za klasterizaciju:

Config server - Ovde se čuvaju metapodaci i konfiguraciona podešavanja za ostatak klastera. U ovom radu koristiće se jedan config server radi jednostavnosti, ali u proizvodnim okruženjima to bi trebalo da bude skup replika od najmanje tri Linoda.

Router Query - deluje kao interfejs između klijentske aplikacije i članova klastera. Budući da se podaci distribuiraju na više servera, upiti se moraju preusmeriti na čvor gde se nalazi određeni podatak. Router query se pokreće na aplikacijskom serveru. U ovom radu koristiće se samo jedan router query, i inače (ako postoji veći broj) bi trebalo da bude postavljen na svakom aplikacijskom serveru u klasteru.

Shard - to je jednostavno server baze podataka koji sadrži deo podataka. Radi jednostavnosti, u primeru će se koristiti dva pojedinačna shard-a.



Slika 2. *Dijagram komponenti za klasterizaciju*

Problem u ovoj konfiguraciji je da ako jedan od čvorova koji sadrži određen podskup podataka padne, onda će taj deo podataka biti izgubljen. Da bi se to izbeglo, može se koristiti skupovi replika za svaki deo kako bi se osigurala visoka dostupnost.

Proces klasterizacije

Konfiguracija host fajla

Svi Linodes-i se nalaze u istom data centru, pa se svakom doda privatna IP adresa kako bi se izbegao prenos podataka putem javnog interneta (ako se ne koriste privatne IP adrese, podatke je potrebno šifrovati).

U svakom čvoru klastera se doda sledeće (u /etc/hosts fajlu):

```
192.0.2.1    mongo-config-1
192.0.2.2    mongo-config-2
192.0.2.3    mongo-config-3
192.0.2.4    mongo-query-router
192.0.2.5    mongo-shard-1
192.0.2.6    mongo-shard-2
```

Napomena: Gore navedene IP adrese se zamenjuju sa IP adresama za svaki Linode. Takođe se promene i imena hostova Linodes-a.

Podešavanje MongoDB za proveru autentičnosti

U ovom delu objašnjeno je na koji način se kreira datoteka sa ključevima koja se koristi za obezbeđivanje autentifikacije između članova skupa replika. Iako se u ovom primeru koristi datoteka ključeva generisana sa *openssl*, MongoDB preporučuje korišćenje *X.509* sertifikata za sigurnost veze između proizvodnih sistema.

Kreiranje administrativnog korisnika

1. Kroz čvor, koji će biti primarni, pokrene se mongo shell:

```
mongo
```

2. Izvrši se povezivanje sa admin bazom:

```
use admin
```

3. Kreira se administrativni korisnik sa root privilegijama. (Zameniti „lozinku“ složenijom lozinkom po izboru):

```
db.createUser({user: "mongo-admin", pwd: "password", roles:[{role: "root", db: "admin"}]})
```

Generisanje key fajla

1. Sledećom komandom generiše se key fajl:

```
openssl rand -base64 756 > mongo-keyfile
```

2. Ostatak koraka u ovom odeljku trebalo bi da se izvede za svaki član skupa replika, tako da svi oni imaju datoteku ključeva koja se nalazi u istom direktorijumu, sa identičnim dozvolama. Kreira se direktorijum /opt/mongo za smeštanje datoteke sa ključevima:

```
sudo mkdir /opt/mongo
```

Inicijalizacija config servera

U ovom delu će biti kreiran skup replika config servera. Oni sadrže metapodatke za stanja i organizaciju podataka. To uključuje i informacije o lokaciji podskupova podataka, što je vrlo bitno kada je ceo skup podataka distribuiran na više čvorova u klasteru.

Umesto da se koristi jedan config server, koristiće se set replika da se osigura integritet metapodataka. To omogućava replikaciju master-slave (primary-secondary) između tri servera i automatizuje failover tako da ako vaš primarni config server padne, biće izabran novi i zahtevi će se dalje obrađivati.

Korak u nastavku treba izvoditi na svakom konfiguracijskom serveru pojedinačno, osim ako nije drugačije navedeno.

1. Na svakom config serveru je potrebno da se modifikuju sledeće vrednosti u /etc/mongod.conf:

```
port: 27019
bindIp: 192.0.2.1
```

bindIp treba da odgovara IP adresi koja je konfigurisana za svaki config server u host datoteci u prethodnom odeljku.

2. Odkomentarisati *replication* sekciju i dodati ime skupa replika:

```
replication:
  replSetName: configReplSet
```

3. Odkomentarisati *sharding* sekciju i konfigurisati ulogu hosta u klasteru kao config server:

```
sharding:
  clusterRole: "configsvr"
```

4. Restartovati servis kako bi promene bile primenjene:

```
sudo systemctl restart mongod
```

5. Na jednom od config servera Linodes povezati se, sa MongoDB shell-om preko porta 27019, kao administrativni korisnik:

```
mongo mongo-config-1:27019 -u mongo-admin -p --authenticationDatabase admin
```

6. Inicijalizovati skup replika:

```
rs.initiate( { _id: "configReplSet", configsvr: true, members: [ { _id: 0, host: "mongo-config
-1:27019" }, { _id: 1, host: "mongo-config-2:27019" }, { _id: 2, host: "mongo-config-3:27019"
} ] } )
```

7. Primenom komande:

```
rs.status()
```

dobijaju se sledeće informacije:

```
configReplSet:SECONDARY> rs.status()
{
    "set" : "configReplSet",
    "date" : ISODate("2016-11-22T14:11:18.382Z"),
    "myState" : 1,
    "term" : NumberLong(1),
    "configsvr" : true,
    "heartbeatIntervalMillis" : NumberLong(2000),
    "members" : [
        {
            "_id" : 0,
            "name" : "mongo-config-1:27019",
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
            "uptime" : 272,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "infoMessage" : "could not find member to sync from",
            "electionTime" : Timestamp(1479823871, 1),
            "electionDate" : ISODate("2016-11-22T14:11:11Z"),
            "configVersion" : 1,
            "self" : true
        },
        {
            "_id" : 1,
            "name" : "mongo-config-2:27019",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 17,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "lastHeartbeat" : ISODate("2016-11-22T14:11:17.758Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-22T14:11:14.283Z"),
            "pingMs" : NumberLong(1),
            "syncingTo" : "mongo-config-1:27019",
```

```
            "configVersion" : 1
        },
        {
            "_id" : 2,
            "name" : "mongo-config-3:27019",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 17,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "lastHeartbeat" : ISODate("2016-11-22T14:11:17.755Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-22T14:11:14.276Z"),
            "pingMs" : NumberLong(0),
            "syncingTo" : "mongo-config-1:27019",
            "configVersion" : 1
        }
    ],
    "ok" : 1
}
```

Konfigurisanje Query rutera

U ovom odeljku će se konfigurisati ruter upita (*query router*). Ruter upita dobija metapodatke sa config servera, kešira ih, i koristi za slanje upita za čitanje i upis do traženih čvorova. Svi naredni koraci bi trebalo da se izvode iz Linode rutera upita (zbog toga što konfigurišemo samo jedan usmerivač upita - query ruter, to ćemo učiniti samo jednom, u suprotnom to treba izvesti za svaki Linode usmerivač upita).

1. Kreirati novi fajl nazvan /etc/mongos.conf, i dodeliti sledeće vrednosti:

```
# where to write logging data.
systemLog:
destination: file
logAppend: true
path: /var/log/mongodb/mongos.log

# network interfaces
net:
port: 27017
```

```
bindIp: 192.0.2.4

security:
keyFile: /opt/mongo/mongodb-keyfile

sharding:
configDB: configReplSet/mongo-config-1:27019,mongo-config-2:27019,mongo-confi
g-3:27019
```

2. Kreirati novu datoteku sistemske jedinice za *mongos* pod nazivom /lib/systemd/system/mongos.service, sa sledećim informacijama:

```
[Unit]
Description=Mongo Cluster Router
After=network.target

[Service]
User=mongodb
Group=mongodb
ExecStart=/usr/bin/mongos --config /etc/mongos.conf
# file size
LimitFSIZE=infinity
# cpu time
LimitCPU=infinity
# virtual memory size
LimitAS=infinity
# open files
LimitNOFILE=64000
# processes/threads
LimitNPROC=64000
# total threads (user+kernel)
TasksMax=infinity
TasksAccounting=false

[Install]
WantedBy=multi-user.target
```

3. Omogućiti mongos.service tako da se automatski pokrene prilikom ponovnog pokretanja, a zatim ga inicijalizovati:

```
sudo systemctl enable mongos.service
sudo systemctl start mongos
```

4. Potvrditi da mongos radi:

```
systemctl status mongos
```

pri čemu treba dobiti sledeći izlaz:

```
Loaded: loaded (/lib/systemd/system/mongos.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2017-01-31 19:43:05 UTC; 10s ago
Main PID: 3901 (mongos)
CGroup: /system.slice/mongos.service
    └─3901 /usr/bin/mongos --config /etc/mongos.conf
```

Dodati čvorove (shard-ove) u klaster

Sada kada je usmerivač upita sposoban da komunicira sa config serverima, treba omogućiti 'sharding' kako bi ruter upita znao na kojim serverima će se nalaziti distribuirani podaci i gde se nalazi bilo koji određeni deo podataka.

1. Logovati svaki od servera i promeniti sledeću liniju u MongoDB konfiguracionom fajlu:

```
bindIp: 192.0.2.5
```

2. Povezati prethodno konfigurisani usmerivač upita sa jednim od kreiranih shard servera:

```
mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
```

3. Iz mongos interfejsa dodati svaki shard (čvor) zasebno:

```
sh.addShard( "mongo-shard-1:27017" )
sh.addShard( "mongo-shard-2:27017" )
```

Opcionalno ako je konfigurisan skup replika za svaki od čvorova one se mogu dodati sledećom komandom:

```
sh.addShard( "rs0/mongo-repl-1:27017,mongo-repl-2:27017,mongo-repl-3:27017" )
```

U ovom formatu, rs0 je naziv replike postavljene za prvi shard, mongo-repl-1 je ime prvog hosta u shard-u (koristeći port 27017), i tako dalje. Treba odvojeno pokrenuti gornju naredbu za svaki pojedinačni skup replika.

Konfiguracija Sharding-a

Poslednji korak je omogućiti sharding. Da bi bilo jasno kako će podaci biti distribuirani, u nastavku su navedene sledeće strukture:

- *Baze podataka* - najširа struktura podataka u MongoDB, koja se koristi za držanje grupa povezanih podataka.
- *Kolekcije* - analogne tabelama u tradicionalnim sistemima relacionihskih baza podataka, kolekcije su strukture podataka koje sadrže baze podataka
- *Dokumenti* - Najosnovnija jedinica za skladištenje podataka u MongoDB. Dokumenti koriste JSON format za skladištenje podataka koristeći parove ključ-vrednost.

Omogućiti sharding na nivou baze podataka

1. Pristupiti mongos shell-u preko query rutera. To se može učiniti preko bilo kog servera u cluster-u:

```
mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
```

2. Odatle kreirati novu bazu podataka (npr. exampleDB):

```
use exampleDB
```

3. Dozvoliti sharding za novu bazu:

```
sh.enableSharding("exampleDB")
```

Strategije shardinga

1. *Range-based sharding*

Ova strategija deli podatke na osnovu određenih raspona vrednosti - ako postoji kolekcija kupaca i njihovih adresa, poštanski broj može biti dobar izbor za ključ (ovo bi kupce na dobar način rasporedilo na odredjene čvorove). Ovakav raspored bi bio pogodan da se nadju kupci koji su jedni blizu drugih, ali ako kupci nisu geografski ravnomerno raspoređeni onda se može desiti da jedan čvor ima previše a drugi nijedan podatak za skladištenje. Zato je neophodno pre svega dobro analizirati podatke.

2. *Hash-based sharding*

Vrši se distribuiranje podataka korišćenjem hash funkcije za shard ključ, za ravnomernu raspodelu medju čvorovima. Ako sada razmatramo situaciju sa kupcima, kao shard ključ možemo odabrati matični broj kupca. Ovaj broj se transformiše pomoću hash funkcije. Ovo je dobra strategija kada aplikacija uglavnom obavlja operacije pisanja ili jednostavne operacije čitanja.

Omogućiti sharding na nivou kolekcija

Sada kada je baza dostupna za sharding i strategija je odabrana (hash-based), treba dozvoliti sharding na nivou kolekcija. To znači da dokumenti iz kolekcija budu distribuirani na različitim čvorovima.

1. Kreirati novu kolekciju

```
db.exampleCollection.ensureIndex( { _id : "hashed" } )
```

2. Konačno, rasporediti kolekciju (sharding):

```
sh.shardCollection( "exampleDB.exampleCollection", { "_id" : "hashed" } )
```

Testiranje rada klastera

Da bi osigurali ravnomernu distribuciju podataka u primeru baze podataka i kolekcije koja je konfigurisana gore, generisaće se osnovni testni podaci i izvršiće se provera podele podataka između čvorova.

1. Prebaciti se na exampleDB bazu:

```
use exampleDB
```

2. Pokrenuti sledeći kod kako bi se generisalo 500 prostih dokumenata i dodalo u exampleCollection:

```
for (var i = 1; i <= 500; i++) db.exampleCollection.insert( { x : i } )
```

3. Proveriti distribuciju podataka:

```
db.exampleCollection.getShardDistribution()
```

Konačno izlaz bi trebalo da izgleda ovako:

```
Shard shard0000 at mongo-shard-1:27017
 data : 8KiB docs : 265 chunks : 2
 estimated data per chunk : 4KiB
 estimated docs per chunk : 132

Shard shard0001 at mongo-shard-2:27017
 data : 7KiB docs : 235 chunks : 2
 estimated data per chunk : 3KiB
 estimated docs per chunk : 117

Totals
 data : 16KiB docs : 500 chunks : 4
 Shard shard0000 contains 53% data, 53% docs in cluster, avg obj size on shard : 33B
 Shard shard0001 contains 47% data, 47% docs in cluster, avg obj size on shard : 33B
```

Odeljci koji počinju sa Shard, daju informacije o svakom shard-u u klasteru. Koliko se shard-ova doda, toliko će biti odeljaka. Ovde možemo primetiti da distribucija nije savršeno podjednaka. Hash funkcija ne garantuje apsolutno ravnomernu distribuciju, ali uz pažljivo izabrani ključ za deljenje, obično će biti prilično blizu.

Zaključak

Sve u svemu, može se reći da je klasterizacija MongoDB baze podataka jednostavna (ali je potrebna dobra priprema - analiza celokupnog skupa podataka koji se skladišti) i pomaže u izgradnji složenih aplikacija. U zavisnosti od potreba same aplikacije, koriste se različite strategije kako ne bi došlo do zagušenja saobraćaja ili gubitka skupa podataka padom servera.

Literatura

https://ndimensionz.com/kb/what-is-database-clustering-introduction-and-brief-explanation/#:~:text=Database%20Clustering%20is%20the%20process,a%20Data%20Cluster%20is%20needed.

https://www.linode.com/docs/databases/mongodb/build-database-clusters-with-mongodb/

Slike