Adatbázis-kezelő rendszerek alkalmazása

MongoDB beadandó feladat

Programtervező informatikus szak

2016.

Tartalomjegyzék

A feladat

A feladat egy olyan MongoDB cluster létrehozása, teszt adatokkal feltöltése, üzemeltetése, fejlesztői dokumentációjának elkészítése és bemutatása, amely magában ötvözi az adatok többszörös tárolásának (*Replica set*) és a vertikális skálázásnak (*sharding*) az alkalmazását.

Az elkészítés menete

FIGYELEM! Mivel a beadandó feladat mélyebb szintű MongoDB ismereteket igényel, ebből kifolyólag a feladatot **NEM kötelező elkészíteni**. A feladat kiírása abból a célból történt, hogy aki szeretne megajánlott jegyet szerezni a MongoDB témakörből, avagy szeretné megmutatni szakmai rátermettségét az adatbázis témakörből, az lehetőséget kapjon rá.

A beadandó munkát csoportokban lehetséges megoldani, amely csoportok csakis **1** vagy **2 fősek** lehetnek. 2 fős csapatok esetén célszerű az egyes csapattagok együttes kooperálása a feladat elvégzése végett, hiszen két hatalmas funkciót (*Replica set* és *Sharding*) ír le a beadandó témaköre.

Leadási határidő

A beadandó leadási határideje a 11. hét vége, **április 24. vasárnap,** 24:00.

Az elkészült megoldásokat összetömörítve (.zip, .rar, .tar.gz) a **kari Moodle rendszerbe** kell feltölteni a tárgyhoz kurzusához:

http://oktatas.mik.uni-pannon.hu/

Honoráció, érdemjegy

A beadandó leadásával leggyorsabban végző első 3 csapat *összes tagja* megajánlott 5-ös nagyZH érdemjegyet kap, így a MongoDB-s nagyZH dolgozat megírása alól ők felmentésre kerülnek.

Elérhetőség, egyéb információk

Ha bármely hallgatónak kérdése merül fel, az alábbi e-mail címen érheti el az előadót:

hugyak@dev.mik.uni-pannon.hu

MongoDB-vel kapcsolatos egyéb információk a következő weboldalon érhetőek el:

http://desoft.hu/oktatas/mongodb/tartalom

Ajánlott irodalom

A MongoDB felhasználói kézikönyve:

https://docs.mongodb.org/manual/

A szoftver letöltése

A MongoDB az alábbi weboldalról tölthető le:

https://www.mongodb.org/downloads#production

Ajánlás

A beadandó elkészítéséhez a hallgatóknak feltétlenül szükséges a JSON (*JavaScript Object Notation*) objektumok jelölésének elsajátítása és megértése. A MongoDB JSON objektumokkal kooperál és működik, a lekérdező és manipuláló nyelve is a JSON objektumok jelölését használja, az adatok tárolását ennek a bináris alakja, a BSON valósítja meg. A MongoDB dokumentációban, illetve az egyes kiadandó parancsokban is JSON objektumok fordulnak elő. A feladat elkészítéséhez a JavaScript nyelv ismerete nem szükséges, a dokumentációban minden egyes példakód értelemszerűen és magától értetődően lett leírva.

Kötelező technológia

Kötelező a MongoDB 3.2-es *production* verzióját használni.

Alkalmazható és szabadon választható technológiák

A cluster bármely operációs rendszeren (Windows, Macintosh, Linux, stb.) bármilyen technológiával (pl. virtualizáció [VMware, VirtualBox, stb.], konténerek [Docker, Vagrant, stb.]) elkészíthető és bemutatható. Lehetséges akár elosztott módon (pl. virtuális gépeken, konténerekben), akár egyetlen egy operációs rendszeren, esetleg egy távoli (dedikált vagy VPS) szerveren, netán *überfun* módon egy cloud-ban létrehozni a cluster-t.

Ajánlott megoldási menet

1. MongoDB-s kézikönyv áttekintése
2. *Replica set* és *Sharding* példa cluster-ek kialakítása megismerés gyanánt
3. Beadandó feladat cluster-ének elkészítése, beüzemelése
4. Fejlesztői dokumentáció megírása
5. Előadás diasorának elkészítése

A feladat részletesen

A beadandó feladat célja egy olyan cluster elkészítése, amely az adatokat többszörösen és elosztva tárolja. Ebből kifolyólag a hallgatóknak a *Replica set* és *Sharding* MongoDB-s témakörökben szükséges elmerülniük.

A *Replica set* szolgál az adat redundancia, a magas rendelkezésre állás és az automatikus failover effektus biztosítására. Segítségével biztonságosan, szerver leállás nélkül üzemeltethetőek MongoDB adatbázis-rendszerek.

A *Sharding* az adatok több szerveren történő elosztását és vertikális skálázását teszi lehetővé. Használatával könnyedén lehet igazodni az adatbázisok egyre növekvő és hatalmas méretéhez, dinamikus bővítéséhez.

Az említett két technológiát szükséges ötvözni egy cluster létrehozásához, amely adatok tárolását és lekérdezését valósítja meg.

A hallgatók feladata az említett technológiák irodalmazásának és működésének áttekintése, majd ezt követően egy összetett rendszer elkészítése. Ajánlott a rendszer elkészítése előtt az egyes technológiák egyenkénti tesztelése – demózása – a működésük megismerése és megértése végett.

Tárolandó információk

Az adatbázis cluster-ben felhasználókról szükséges adatokat tárolnia. A felhasználók a következő attribútumokkal rendelkeznek: név (*name*), kor (*age*) és nem (*sex*).

Az alábbi példa JSON felépítésével megegyező dokumentumok tárolása szükséges:

```
{
    "name" : "Péter"
,   "age"  : 20
,   "sex"  : "male"
```

```
}
```

Az egyes felhasználók adatait a *users* nevezetű collection-ben szükséges majd eltárolni, amely a *cluster* nevű adatbázisban legyen elhelyezve.

Az egyes példa felhasználók adatait a hallgatóknak szükséges *kitalálniuk* és *felvinniük* a cluster-be.

A rendszer áttekintése

A cluster alapvetően a *Sharding* technológiára épül (*sharded cluster*), azonban egyes elemei *Replica set*-tel legyenek megvalósítva.

A *Sharding* megvalósításához 3-féle tag típusra lesz szükség:

- egy router szerverre (*mongos*)
- konfigurációs szerver(ek)re (*Config* szerverek)
- adatok tényleges tárolását megvalósító szerverekre (*Shard*-ok)

A kliens programok és a *mongo* CLI a router-en keresztül tudnak lekérdezni és manipulálni adatokat, ebből kifolyólag minden esetben hozzá kell kapcsolódniuk, nem a többi (*Config* vagy adattároló *shard*) szerverhez. A konfigurációs szerver(ek) feladata az, hogy a router számára biztosítsák a meta adatokat, amelyek alapján az a szükséges adattároló szerverekhez tud kéréseket intézni, azaz le tudja futtatni a kívánt műveleteket. A tényleges adatokat tároló adattároló *shard* szerverek tartományokra osztva tárolják az adatokat. Ezek a tartományok a felhasználók kora alapján 3, egymástól teljesen elkülönült szerveren helyezkednek majd el: az első *shard* szerveren az 1-20 évesek, a második *shard* szerveren a 20-50 évesek, míg a harmadik *shard* szerveren az 50-100 éves felhasználók lesznek tárolva. Ezt az elosztást majd a hallgatóknak kell beállítaniuk kézzel, miután feltöltötték a teszt adatokat a megfelelő *collection*-be.

Az imént említett cluster architektúra egy szimpla *sharding cluster*-nek felel meg, amelyet még a bővíteni kell magas rendelkezésre állás tulajdonsággal is. Ily módon *Replica set*-eket kell beépíteni az egyes szerverek helyére. Jelen beadandó feladat a router számára nem követel meg magas rendelkezésre állást, ezért az csak egy sima *standalone* szerverként kell, hogy fusson. A *Config* és *shard* szervereket viszont

szükséges ellátni a *high availability* tulajdonsággal, így ezek nem szimpla *standalone* szerverek lesznek, hanem egy-egy *Replica set*-ek. A *Config* szerverek 3 tagból álló *Replica set*-et kell, hogy képezzenek. A *cluster*-ben 3 darab *shard* szerver helyett 3 darab *Replica set*-et kell létrehozni, amely *set*-ek lesznek rendre a 3 darab *shard* szerverek teljes megfelelői. Ez csak annyiban különbözik a *standalone* megvalósítástól, hogy nem egy-egy szerveren lesznek az adatok, hanem több szerveren redundánsan tárolva. Az említett *Config* és *shard Replica set*-ek ismertetése a következő alfejezetben történik.

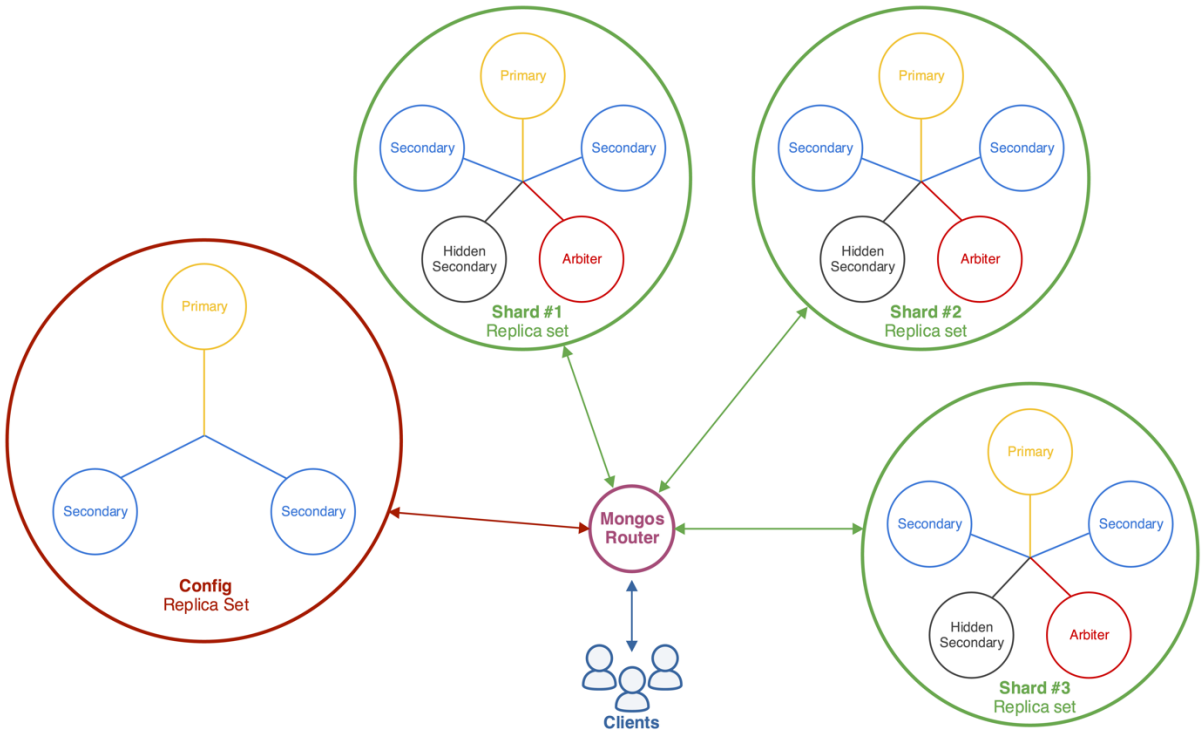

Figure 1 A teljes, elkészítendő cluster

Replica set

A felhasználók adatainak tárolása többszörös meglétükkel valósuljon meg. Ehhez *Replica set*-ekre lesz szükség, amelyek magas rendelkezésre állást biztosítva működnek.

Kétféle *Replica set*-re lesz szükség: a *Config* szerverek létrehozásához egyre, illetve a felhasználók (azaz tényleges adatok) tárolásához 3-ra – utóbbiak konkrétan az egyes *shard* „szerverek” lesznek *Replica set*-ekkel megvalósítva.

1. Replica Set a felhasználói adatok tárolására (shard-ok)

A felhasználók adatait *shard*-ok tárolják több szerveren történő elosztás végett. A *shard*-ok valójában *Replica set*-ek, amelyek 5 tagból kell, hogy álljanak:

- 1 elsődleges (*primary*) tag
- 2 másodlagos (*secondary*) tag
- 1 rejtett (*hidden secondary*) tag
- 1 *arbiter* tag

Mind az 5 tag szavazati joga 1 legyen. A prioritásuk a rejtett és az *arbiter* tagon kívül az összesnek 1 kell, hogy legyen. Az elsődleges és másodlagos tagok gondoskodnak a lekérdezések és manipulációk futtatásáról, míg az *arbiter* tag csak a szavazáson, azaz új elsődleges választása esetén, tevékenykedik. A rejtett tag feladata az adatok biztonsági másolatának (*backup*) tárolása, és a szavazáson való részvétel 1-es szavazati joggal.

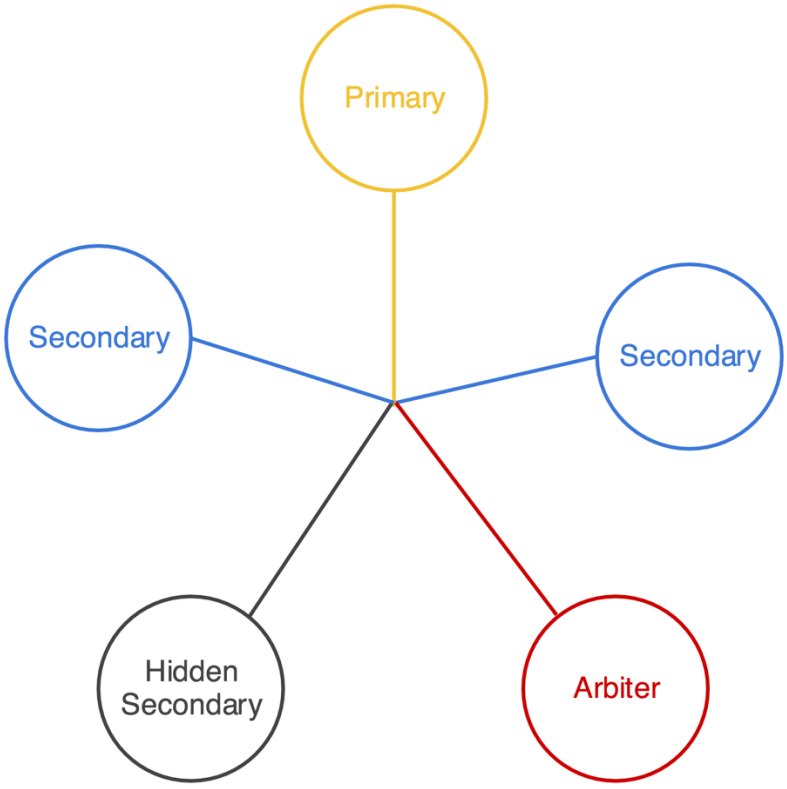

Figure 2 Egy *Shard* „szerver” - egy teljes *Replica Set*

2. Replica set a Config szerverek létrehozásához

A *Config* szerverek feladata a meta adatok tárolása a cluster számára. A *Config* szerverek számára is egy *Replica set*-et szükséges létrehozni, amely biztosítja a magas rendelkezésre állásukat. Ezen *Replica set* 3 tagból kell, hogy álljon:

- 1 elsődleges (*primary*) tag
- 2 másodlagos (*secondary*) tag

Az egyes tagok szavazati joga és prioritása az alapértelmezett 1 legyen.

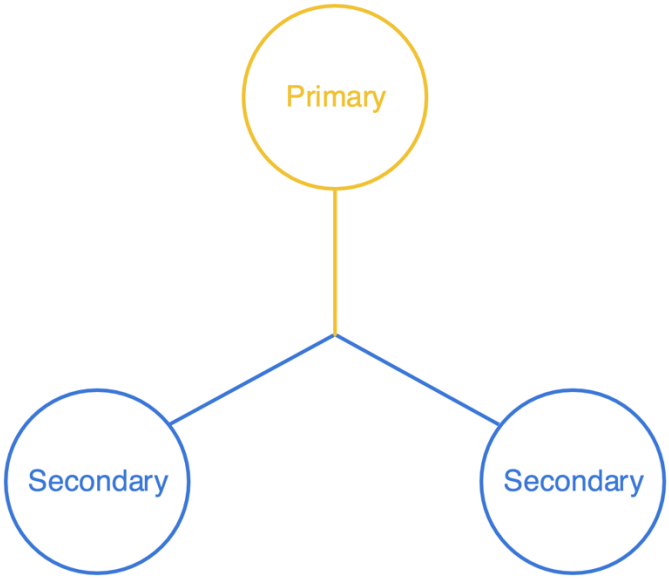

Figure 3 A *Config* szerverek *Replica Set-e*

Sharded cluster

A cluster alapját avagy összetételét a *sharding* fogja adni. A rendszert úgy kell elképzelni, hogy az adatok elosztva legyenek tárolva különböző szervereken úgy, hogy a koruk szerint csoportosítva vannak elhelyezve.

A művelethez a már említett *router* szerverre, a *Config Replica set*-re és a *shard* szerverekre lesz szükség. A router-hez kapcsolódik az összes kliens, aki a kéréseket vezérli, a *Config Replica set* feladat pedig a meta fájlok biztosítása, hogy a router általuk a megfelelő *shard* szerverekhez csatlakozzon, s rajtuk hajtsa végre az adatbázis műveleteket. A *shard* szerverek az adatokat tárolják. Egy collection dokumentumai adott

indexelt kulcs értékei szerint bonthatóak tartományokra, s ezeket a kialakított tartományokat *chunk*-nak nevezik. A hallgatók feladata lesz a *chunk*-ok kialakítása, hogy a megfelelő rendben legyenek a felvett dokumentumok eltárolva az egyes *shard* szervereken. A *shard* szerverek az alábbi ábrán egyszerűsítve vannak *standalone* szerverként, azonban a beadandó megoldásában egy-egy *Replica set* kell, hogy legyenek:

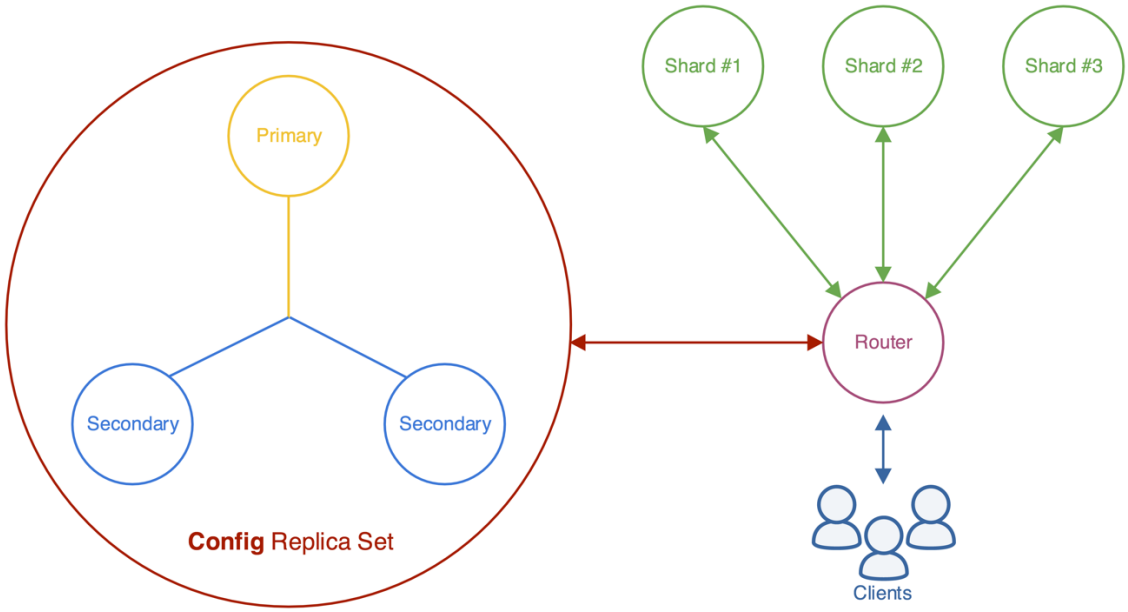

Figure 4 Az egyszerűsített *sharded cluster*

Collection feltöltése teszt adatokkal

Miután a hallgatók létrehozták az előírt cluster-t, fel kell venni a néhány adatokat ahhoz, hogy tesztelni lehessen a *Sharding* helyes működését. Ehhez először egy *cluster* nevezetű adatbázist kell létrehozni, amelyen engedélyezni kell utána a *sharding*-ot. Ezt követően ebben az adatbázisban egy *users* nevű collection-t kell létrehozni, s azon engedélyezni a *sharding*-ot az előírt *age* attribútum szerint. Ha mindez megtörtént, következhet a teszt adatok felvétele a *users* collection-be. Amint az már korábbi alfejezetben le lett írva, minden egyes dokumentumnak rendelkeznie kell a név, kor és nem kulcsokkal és értékeikkel.

Fejlesztői dokumentáció elkészítése

A cluster beüzemelése és a teszt adatok feltöltése után következik az utolsó feladata a hallgatóknak: a fejlesztői dokumentáció elkészítése. A dokumentációnak egy olyan PDF fájlnak kell lennie, amelyben szerepelnek az alábbi információk:

- a fő oldalon: a beadandó címe, a beadandót elkészítő hallgatók nevei, Neptun-kódjai, dátum
- 2. oldal: üres
- 3. oldalon: jó néhány mondatban ismertessék a csapat tagjai, hogy miért is vállalkoztak a beadandó feladat elvégzésére (pl.: engem nagyon érdekelnek az adatbázisok skálázhatóságai, érdekel a MongoDB, szeretem a kihívásokat, stb.)
- 4. oldalon: fő fejezetként ismertessék fél-1-2 oldalon, hogy mit is jelent a *Sharding* és a *Replica Set* (a célja ennek, hogy mennyire sikerült megérteniük a csapat tagjainak az egyes feature-ök működését)
- ezt követően: fő fejezetként új oldalon ismertessék, hogy milyen operációs rendszeren valósították meg a cluster-t, milyen technológiákat használtak, s miért választották azokat
- majd: fő fejezetként új oldalon lépésről lépésre írják le, hogy milyen parancsokat adtak ki a cluster összeállításának kezdetétől a végezetéig, majd röviden írják is le, hogy melyik parancs mit csinál (néhány szóval vagy mondattal). Ugyanezt szükséges megtenni a teszt adatok felvételével, tesztelésével, *chunk* migrációval és egyéb adatbázis műveletek ellenőrzésével is (pl.: hogyan ellenőrizték, hogy jelen pillanatban dolgozik-e a balancer? [melyik parancs], hogyan nézték meg, hogy melyik *chunk* melyik *shard*-on található?, melyik paranccsal állították be a *sharding*-ot az adatbázison, s melyikkel a collection-ön?, stb.)
- ezután: fő fejezetként új oldalon ismertessék, hogy milyen hibákba futottak bele a cluster egészének vagy annak egy részének beállítása, beüzemelése közben (pl.: nem tudtuk beállítani a *sharding*-ot adott collection-re azért, mert elfelejtettük indexelni az adott mezőt, stb.)

- végezetül: fő fejezetként új oldalon konklúzióként írják le, hogy hogyan élték meg a cluster felállítását, beüzemelését; milyen érzéseket váltott ki Önökben a MongoDB a feladat megoldása során; milyen benyomást keltett Önökben az adatbázis-kezelő rendszer használata, megismerése, tulajdonságai, használhatósága, egyértelműsége; mennyire sikerült megkedvelni avagy nem megkedvelni a MongoDB-t; mennyire tartják jó DBMS-nek az SQL-es társai mellett, s ezekhez hasonló dolgok...

Legvégső teendők...

A hallgatóknak legvégül a fejlesztői dokumentációt és a MongoDB szerverek számára létrehozott konfigurációs fájlokat össze kell csomagolniuk egy [.zip, .rar, .tar.gz] kiterjesztésű fájlba, s a tömörített állományt fel kell tölteniük a Moodle rendszerbe.