**SEMINARSKI RAD:**

**Sistemi za upravljanje bazama podataka**

**Tema:** **Klaster rešenja kod Neo4j baze podataka**

Student:

Aleksandar Cenić 1062

S A D R Ž A J

**Strana**

[1. U](#_Toc191445171)vod [3](#_Toc191445171)

[2.](#_Toc191445172) Distribuirane baze podataka [4](#_Toc191445172)

[2.1.](#_Toc191445173)  Particionisanje podataka  [4](#_Toc191445173)

[2.2.](#_Toc191445174) Distributivnost Neo4j baze podataka [7](#_Toc191445174)

[3.Klaster rešenja kod Neo4j baze podataka](#_Toc191445175) 9

[3.1. Neo4j High Availability klaster](#_Toc191445176) 9

[3.2.](#_Toc191445177) Kauzalni klaster [11](#_Toc191445177)

[3.3.](#_Toc191445177) Kauzalni vs HA klaster [12](#_Toc191445177)

[4.Praktičan](#_Toc191445175) primer HA i Kauzalnog klastera ………………………………………………...13

[4.1.](#_Toc191445177) Docker [13](#_Toc191445177)

[4.2.](#_Toc191445177) Podizanje klastera [15](#_Toc191445177)

[5. Zaključak 23](#_Toc191445180)

[Literatura 2](#_Toc191445183)4

# uvod

Šta bi uopšteno mogle da budu danas jako popularne informacione tehnologije? Odgovor leži u samom nazivu, informacione tehnologije su tehnologije koje se koriste za obradu informacija. Sada se postavlja pitanje šta je to informacija? Informacija je zapravo podatak. Kako bi smo mogli prikupljati i obrađivati informacije potrebna nam je komponenta koja se naziva baza podataka. Baza podataka, kao što znamo, je jedna softverska komponenta u jednom računarskom sistemu, isto tako znamo da je brzina i sam rad softvera zavistan od hardvera na kome se izvršava. Iz same te zavisnosti softvera o hardveru, možemo reći da količina podataka koju baza može pohraniti direktno zavisi od kapaciteta masovne memorije raćunara na kome se nalazi.

Ovakva hardverska ograničenja stvaraju dosta poteškoća u samom radu. Jedan od mogućih scenarija je, preopterećenje servera na kome se baza nalazi i zastoji aplikacije i web adrese koja interaguje sa bazom. Kada se radi o aplikacijama koje moraju imati visoki nivo stabilnosti, pad servera bi se smatrao katastrofalnim u poslovnom pogledu. Postavlja se pitanje: Kako prevazići hardverska ograničenja servera i izbeći najgori scenario? Odgovor leži u distribuiranim bazama podataka.

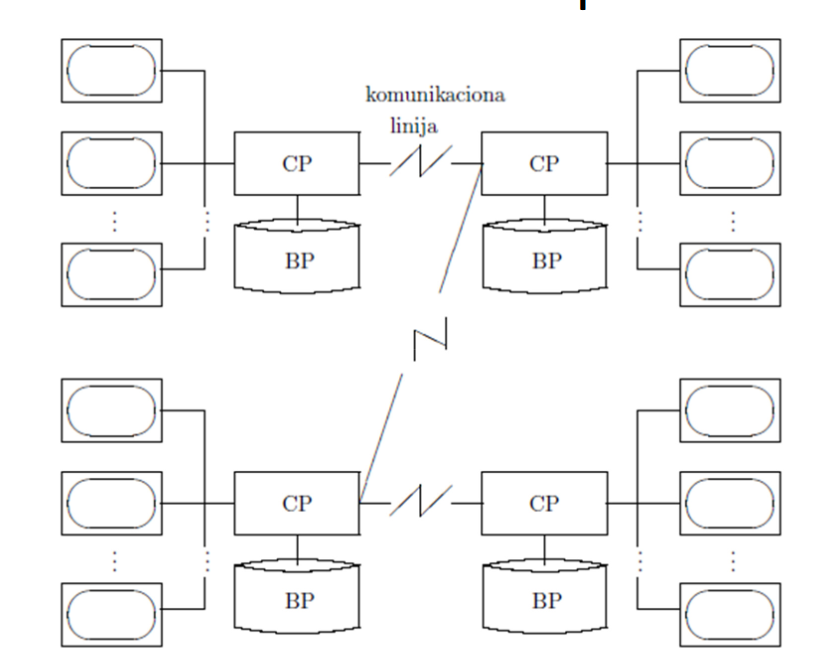
Distribuirane baze podataka su, umesto na jednom centralizovanom serveru, pohranjene na više servera, kako bi paraleno mogli da se izvršavaju upiti i kako bi se mogla, na jednom ili više servera čuvati replika. Skup dva ili više servera na kojima se nalazi ista baza podataka nazivamo klasterom. U ovom radu biće obrađena i upoređena klaster rešenja kod Neo4j baze podataka.

# Distribirane baze podataka

## Distribuirana baza podataka, kako smo naveli u uvodu, je baza podataka pohranjena na više lokacija koje su povezane komunikacionom mrežom. Svaka lokacija u konunikacionoj mreži predstavlja server (čvor). U svakom čvoru postoji lokalni sistem za upravljanje bazom podataka, koji zajedno pružaju takav servis da korisnik sa bilo kog čvora može da pristupi podacima u bilo kojoj lokalnoj bazi.

## U svakom čvoru lokalni SUBP omogućava korišćenje lokalne baze (obrada upita, transakcije, oporavak i slično) kao da ta baza nije deo virtuelne distribuirane baze podataka. Pored toga, lokalni SUBP su prošireni tako da je moguće ostvariti njihovu kolaboraciju za realizaciju virtuelne distribuirane baze podataka. Ovako prošireni lokalni SUBP čine distribuirani sistem za upravljanje bazom podataka (DSUBP).

## Sistem za upravljanje distribuiranim bazama podataka je softver koji upravlja mehanizmom pristupa distribuiranim bazama podataka i čini ga transparentnim za sve korisnike. To je više ili manje spregnut multiprocesorski računarski sistem sa sinhronizovanom vremenskom podelom zadataka.



## Slika:1

**Prednosti distribuiranih baza** **podataka** su:

* **Povećana pouzdanost i raspoloživost**: pouzdanost se najopštije definiše kao verovatnoća da sistem radi u nekom trenutku vremena. Raspoloživost se definiše kao verovatnoća da će sistem biti stalno operativan u nekom intervalu vremena. U distribuiranom sistemu postoji više SUBP. Ako jedan otkaze, ostali još uvek rade, pa je pouzdanost i raspoloživost ovakvog sistema veća, pogotovo ako se u više čvorova čuvaju replikacije cele ili delova distribuirane baze podataka.
* **Poboljšane performanse sistema**: fragmentacija podataka grupiše podatke na lokacije u kojima se najviše koriste. Lokalni upiti i transakcije pristupaju podacima u lokalnoj bazi koja je manja nego odgovarajuća centralizovana. Operacije nad lokalnim bazama se odvijaju paralelno. Svaki lokalni SUBP obrađuje konkurentno manji broj transakcija nego odgovarajući centralizovani SUBP.
* **jednostavniji rast sistema**: očigledno je da se u distribuiranoj bazi podataka može jednostavnije ostvariti rast sistema bilo proširivanjem lokalnih baza podataka, bilo dodavanjem novog čvora sa novim SUBP.

Očigledno je da je u distribuiranim bazama mnogo teže ostvariti osnovne funkcije SUBP: vođenje kataloga baze podataka, optimizaciju upita, upravljanje transakcijama i drugo.

DSUBP treba da zadovolji sledeće osnovne principe distribucije:

* + **lokalna autonomija svakog čvora** - U svakom čvoru lokalna baza podataka može da se koristi kao da nije deo distribuirane baze,
  + **nepostojanja centralnog čvora distribuirane baze** podataka - Svi čvorovi u distribuiranoj bazi podataka su ravnopravni. Postojanje centralnog čvora u velikoj meri bi degradiralo pouzdanost i raspoloživost distribuirane baze podataka. Otkaz centralnog čvora doveo bi do otkaza celokupnog sistema,
  + **transparentnost distribucije**, odnosno sakrivanje detalja distribucije od korisnika - Korisnik treba da jednostavno koristi distribuiranu BP na isti način na koji koristi i centralizovanu, bez znanja o tome kako je distribucija podataka fizički izvedena.

## 2.1. Particionisanje podataka

Podaci u distribuiranom sistemu mogu biti particionisani (eng. partitioned) ili ponovljeni (eng. replicated), a mogu biti istovremeno i particionisani i ponovljeni. U slučaju particionisanje podataka, baza je razdeljena na više disjunktnih delova od kojih je svaki na drugom čvoru komunikacione mreže. Skup podataka sa logičkog nivoa treba na neki način podeliti, a zatim te delove – fragmente raspodeliti po raznim lokacijama.

Najčešće primenjivana tehnika za očuvanje informacija pri fragmentaciji podataka je uvodjenje sistemskih identifikatora n-torki (nametnuti ključevi) kao primarnih ključeva, koji se pamte uz svaki deo pojedine n-torke i omogućuju njenu rekonstrukciju.

Postoje dve vrste particionisanja:

* Horizontalno,
* Vertikalno.

**Horizontalno particionisanje** je fragmentacija slogova i ona se najčešće koristi kod jednostavnih upita. Iz tog razloga horizontalno particionisanje zovemo i intuitivnim. Kod njega svaka lokacija bi trebalo da sadrži sve informacije koje se koriste prilikom upita, dok bi informacije na lokaciji trebale biti fragmentisane kako bi odgovori na upite bili brži. Horizontalno particionisanje je definisano kao selekcija operacija, σp(R), što znači da su rezultati kompleksniji, odnosno da, kao za jednostavan upit po svojoj prirodi ono nije u stanju da izvrši visoku selekciju potrebnih podataka, već vraća veliku količinu podataka od kojih verovatno jedan deo nije od interesa za korisnika. Prednost ovog načina selekcije je njegova brzina izvršenja a negativna strana je cena izvršenja što se manifestuje velikim dotokom podataka na klijentsku stranu. Horizontalna fregmentacija može biti:

• Primarna horizontana fregmentacija o Kriterijum fragmentacije je predikat definisan upravo na fregmentisanoj relaciji.

• Izvedena horizontalna fregmentacija13

o Kriterijum fregmentacije je predikat definisan na drugim relacijama.

**Vertikalna fragmentacija** je usmerena na particionisanje relacije u mnoštvo manjih relacija tako da više aplikacija može biti pokrenuto na samo jednom fragmentu odnosno definiše se na neki način paralelizam izvršavanja.Vertikalna fragmentacija relacije R proizvodi fragmente R1, R2,..., od kojih svaki sadrži podskup osobina relacije R. Definisana je korišćenjem operacije projekcije relacione algebre : ΠA1,A2,...An (R) Ono što se odmah vidi je da je vertikalna fragmentacija neizostavno komplikovanija od horizontalne fragmentacije.

Kao prilog ovoj tvrdnji stoji matematički model koji kaže da je:

• u horizontalnom particionisanju: za n prostih predikata, broj mogućih minterma 2n (neki od njih mogu biti nezadovoljavajući za upit ali je broj neznatno manji)

• u vertikalnom particionisanju: za m ne – primarnih ključnih atributa, broj mogućih fragmenata je B(m) tj. broj particija sa m stanja B(m) ~ mm. Važno je reći da optimalno rešenje ne postoji, tako da se primenjuju heuristička rešenja. Ovde razlikujemo dva tipa:

• Grupisanje: gde se svakom fragmentu pridružuje po jedna osobina, i na svakom koraku, pridružuju se neki fragmenti dok se ne zadovolje postavljeni kriterijumi (prilaz bottom-up).

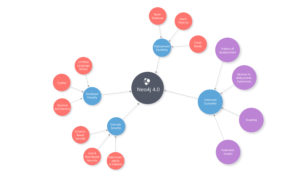
• Deljenje: gde se počinje sa relacijom i onda se donose odluke za povoljno particionisanje, zasnovane na ponašanju pristupne aplikacije prema osobinama (prilaz top-down).



Slika:2

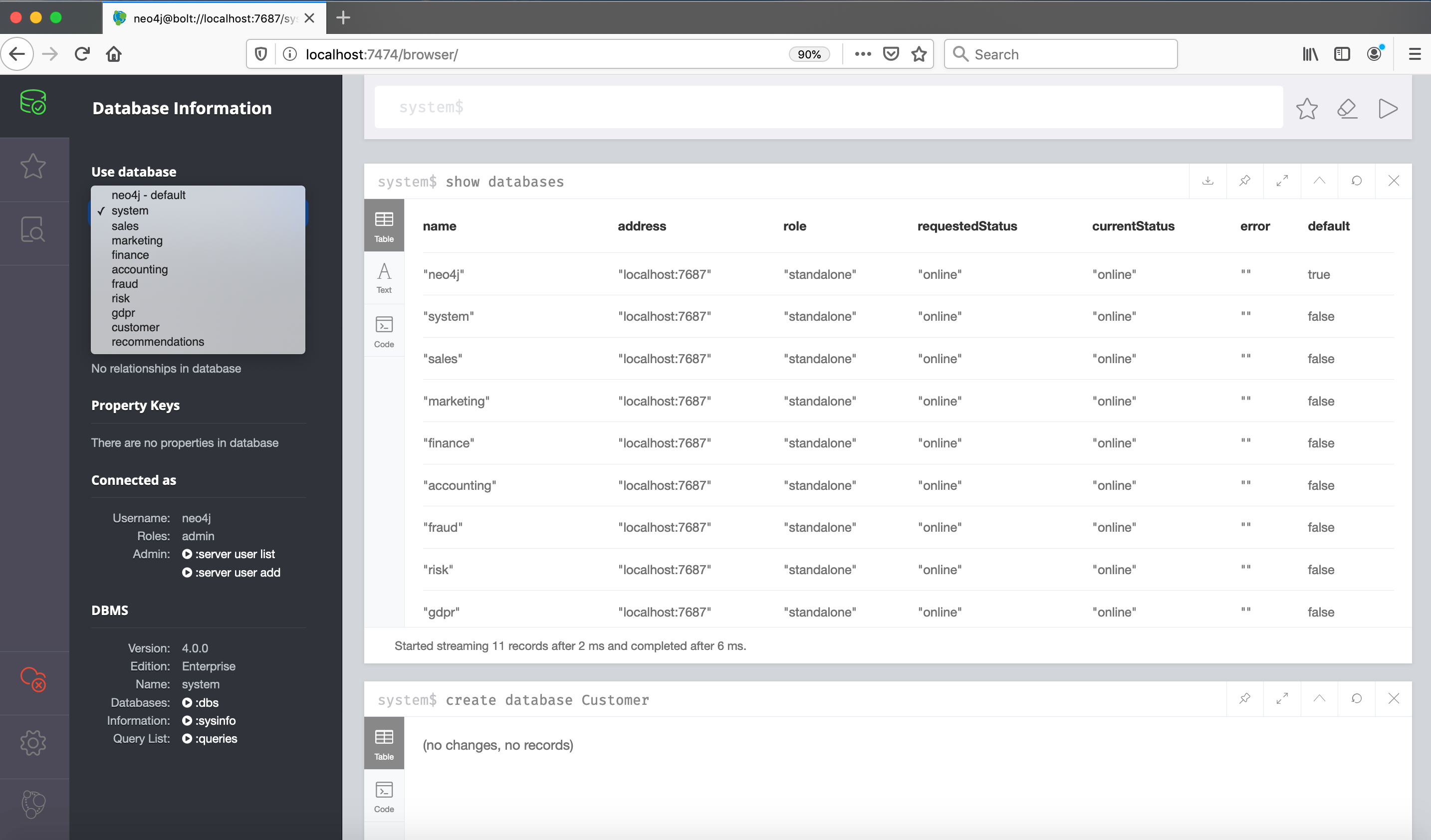
* 1. **Distributivnost Neo4j baze podataka**

Neo4j je vodeća baza podataka na tržištu kada su u pitanju graf baze podataka, sa 17 godina development-a i mnogobrojnim open-source korisnicima. Neo4j baza podataka imala jedno veliko ograničenje u pogledu prilagodljivosti, esencijalno je bila ograničena na izvršenje na samo jednom serveru. Uz nekoliko rešenja, kao što su, high availability, i causal klaster rešenja, Neo4j baza podataka nije u potpunosti postala distributivna.



Slika:3

Sa Neo4j 4.0 verzijom, korisnici su odlučili na koji će način podeliti podatke u više baza podataka, ili u pod grafove koje mogu pokrenuti na odvojenim serverima, ali da se upiti mogu izvršavati kao jedan entitet kroz novi fabrički server. S druge strane, možemo ih tretirati kao podgrafove u većem grafu i moći ćemo da pokrećemo upite i izračunamo agregacije preko svih njih paralelno.

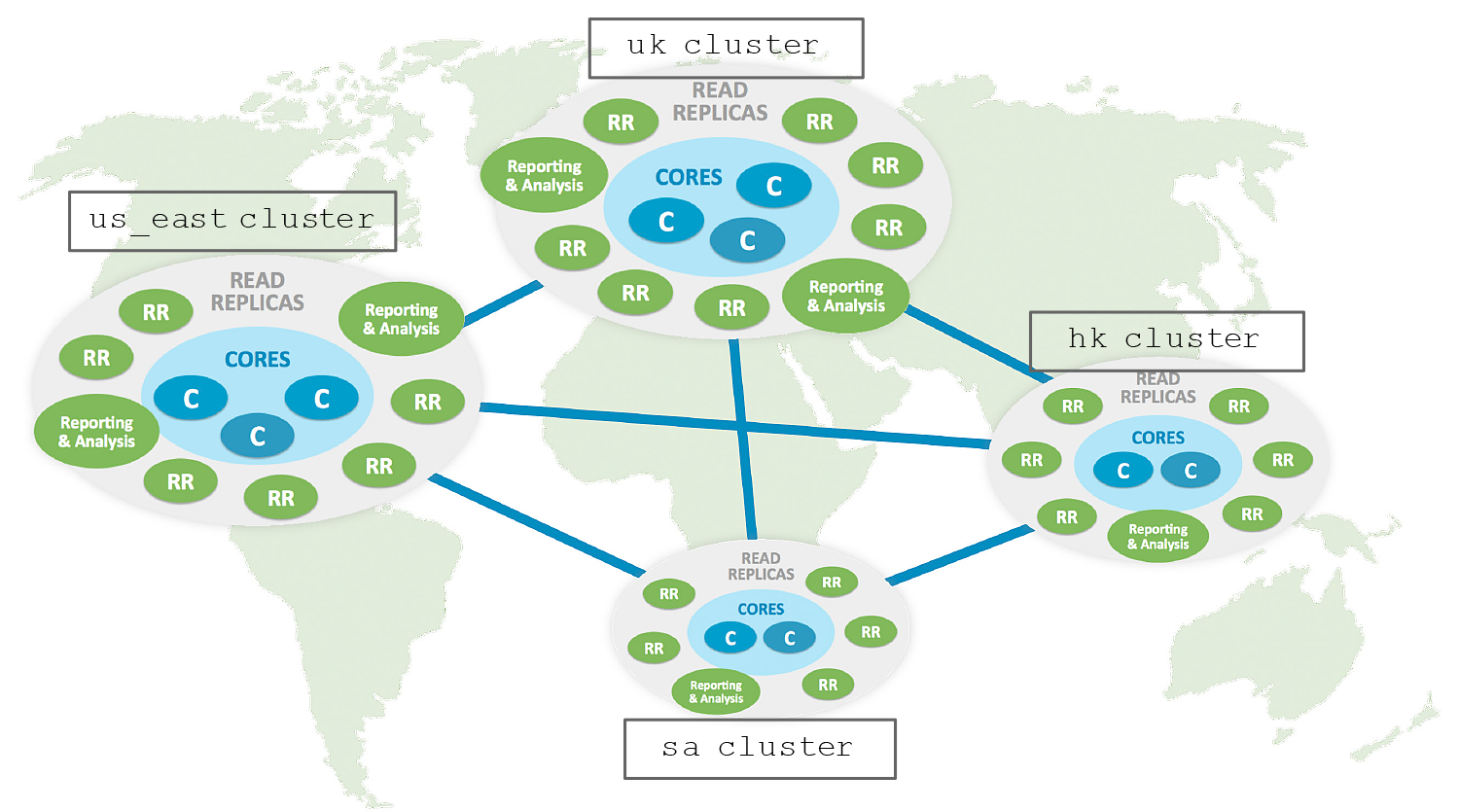


Slika:4

Klaster projekat baze podataka trajao je tri godine u izradi i zahteva opsežne inženjerske resurse za prevazilaženje teških dizajnerskih izazova. Neo4J se sve više koristi kao osnova za proizvodnju analitičkih i mašina za upotrebu u učenju u problemima poput prepoznavanja prevara i preporuka u realnom vremenu. U narednom poglavlju detaljno će biti objašnjena sva klaster rešenja kod Neo4j baze podataka.

# Klaster rešenja kod neo4j baze podataka

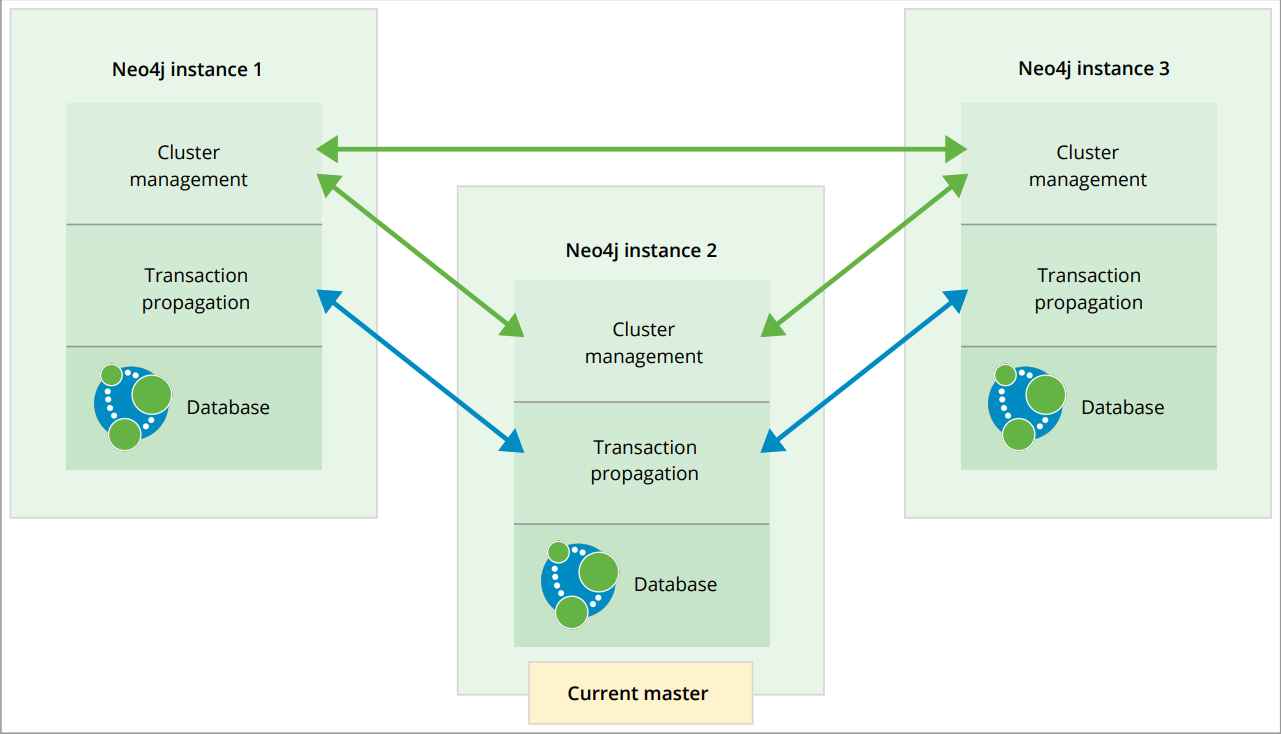
U predhodnom podpoglavlju smo rekli da Neo4j baza podataka ima dva rešenja vezana za klasterizaciju, a to su high availability klaster i klausani klaster. Sada ćemo detaljno, pojedinačno i uporedno objasniti ova rešenja.



Slika:5

## 3.1. Neo4j High Availability klaster

High Availability klister kod Neo4j-a organizovan je kao master-slave arhitektura. Ova arhitektura se sastoji od jedne master instance i nula ili više slave instanci. Svaka od instanci u klasteru sadrži celukupnu kopiju podataka u njihovoj lokalnoj bazi podataka. Jedna od osnovnih kofiguracija HA klastera sastoji se od tri instance, jednom master i dve slave instance.



Slika:6

Svaka od instanci u klasteru sadrži logiku potrebnu za kordinisanje sa ostalim članovima klastera za replikaciju podataka i management. Svaka slave instanca, koja nije arbitrarna, komunicira sa master instancom kako bi održala podatke ažuriranim u svojoj lokalnoj bazi podataka.

**Arbitrarna instanca**

Arbitrarna instance sadrži pun Neo4j softver pokrenut u arbritrar modu, kao i ostali učesnici u klister komunikaciji, ali ne replicira kopiju baze podataka.

**Propagiranje transakcije** Kada se transakcije za upis vrše direktno na master instanci, izvršiće se kao da instanca nije u kalster modu. Ako se uspešno izvrši, biće prosleđena konfigurisanom broju slave instanci. To se sve radi optimistično, što znači da ako prosleđivanje ne uspe, transakcija će i dalje biti uspešna. Kada se trasakcije upisa vrše na slave instanci, svaka operacija će biti sinhronizovana sa master instancom. Lock će biti na obe strane i kod slave i kod master instance.

Kada se transakcija potvrdi, prvo će biti potvrđena na master, pa ako je uspešno potvrđena, potvrdiće se i na slave instanci. Da bi se osigurala koenzistentnost, slave instaca mora biti sinhronizovana sa masterom pre nego što započne operaciju upisa. Automacko ažuriranje slave instanci ugrađeno je u komunikacioni protokol između master i slave instanci.

**Kvorum** Klaster mora imati kvorum kako bi mogao izabrati novu master instancu.Kvorum je postignut kada više od 50% članova klastera aktivno. Kada se dizajnira klaster bitno je poštovati pravilo palca koje glasi: Klaster mora imati mogućnost tolerancije n otkaza master instanci, tako da zahteva 2n +1 instanci kako bi zadovoljio kvorum i dozvolio i izbor novog mastera. Osnovna HA klaster konfiguracija sadrži tri instance jednu master i dve slave, što podržava otkaz jedne master instance.

**Okazivanje** Ako u nekom lučaju master instanca otkaže, drugi član biće izabran i preći iz slave-a u master, nakon dostignutog kvoruma. Kada se instanca preobrati u master, obavestiće sve članove klastera.

**Pravila izbora** Ako master instanca otkaže, ili je cold-start klastera, slave instanca sa najvišim ID-jem potvrđene transakcije biće izabran za novu master instancu. Ovo pravilo osigurava da će slave instanca sa najažurnijom bazom podataka biti izabran za master instancu.

Ako master instanca otkaže, a dve ili vše slave instance budu vezane tj.imaju isti ID najviše potvrđene transakcije, onda će slave instanca sa najmanjom ha.server\_id vrednošću biti izabrana za master instancu. Ovo je dobar tiebreaker jer je ha.server\_id jedinstven u klasteru i omogućava konfigurisanje koje instance mogu postati glavne pre drugih.

**Granjanje** Razgranavanje podataka može biti uzrokovano na dva različita načina:

• svale zaostaje previše iza master-a, a zatim napušta ili se ponovo pridružuje klasteru. Ova vrsta grananja je bezopasna.

• Događa se ponovni izbor master-a i stari master ima jednu ili više izvršenih transakcija koje

slave nisu primili pre nego što je umro. Ova vrsta grananja je štetna i zahteva akciju.

Baza podataka najbolje koristi situaciju stvaranjem direktorijuma sa sadržajem baze podataka

datoteke od pre grananja kako bi se mogla pregledati i rešiti situacija. Podaci

pri normalnim operacijama ne dolaze do grananja.

## 3.2. Kauzalni klaster

Neo4j arhitektura kauzalnog klastera sastoji se od dve komponente:

* Core servera
* Replika za čitanje

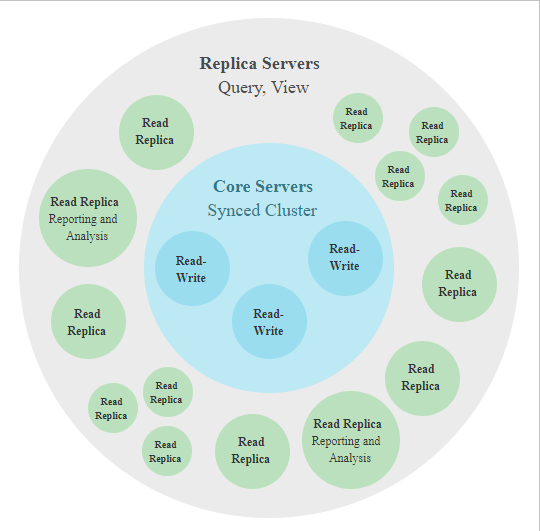
Neo4j kauzalni klaster obezbeđuje tri glavne karakteristike, a to su:

**Sigurnost**: Core serveri obezbeđuju platformu tolerantnu na greške za procesiranje transakcija koja je dostupna sve dok većina Core servera funkcioniše.

**Skalabilnost**: Replike za čitanje pružaju masivno skalabilnu platformu za graf upite što omogućava izvršavanje veoma velika opterećenja na širokim topologijama.

**Uzročna koenzistencija**: Kada se pozove, klijentska aplikacija garantuje da pročita barem sopstveni zapis.

Ovo sve zajedno, omogućava krajnjem korisniku sistema da bude potpuno funkcionalan, da upisuje i čita iz baze u slučaku višestrukih kvarova hardvera i mreže.



Slika: 7

**Core server** Glavna odgovornost core servera je zaštita podataka. Core server to postižu repliciranjem svih transakcija koristeći Raft protokol. Raft osigurava da su podaci bezbedni i trajni pre nego što potvrde predaju transakcije aplikaciji krajnjeg korisnika. U praksi to znači da kada je većina Core servera u klasteru (N / 2 + 1) prihvatila transakciju, sigurno je potvrditi predati aplikaciji krajnjeg korisnika.

**Replika za čitanje** Glavna odgovornost Replika za čitanje je da skaliraju opterećenje grafa. Replike za čitanje se ponašaju kao cache za graf podatke na kojima su Core serveri u potpunosti sposobni da izvrše proizvoljne (samo za čitanje) upite i postupke. Replika za čitanje je asinhrono replicirana iz Core servera putem transakcije dnevnika. Povremeno će povući sa Core servera nove transakcije. Mnoge replike za čitanje mogu se hraniti podacima iz relativno malog broja Core servera, omogućavajući veliki ventilator iz radnog opterećenja upita za skali.

Replika za čitanje obično se treba pokretati u relativno velikom broju i tretira se kao jednokratna. Gubitak pročitane replika ne utiče na dostupnost klastera, osim gubitka njegovog frakcije grafičkog upita. To ne utiče na mogućnosti tolerancije na grešku klastera.

## 3.3. Kauzalni vs HA klaster

Za uporedni pogled na visok nivo HA klastera protiv kauzalnih klastera, tabela ispod pruža rezime razlika:

|  |  |
| --- | --- |
| Kauzalni | HA |
| Raft Protocol. | |  |  | | --- | --- | |  | Paxos Protocol. | |
| Minimalno 3 instance. | Minimalno 1 instanca + opcionalni Arbiter(i). |
| skup Core servera (3, 5 or 7) + Replika za čitanje. | |  |  | | --- | --- | |  | Master(1) + Slave(0 ili više). | |
| Transakcije se izvršavaju nakon što je većina Core servera u klasteru (2N + 1) prihvatila transakciju. | Transakcije se prvo izvršavaju na Master-u i optimistično se guraju u Slaves (ne garantuje obaveze u slave-u). |
| Nema grananja. | Može dovesti do grananja. |
| Ugrađeno balansiranje opterećenja i usmeravanje preko Bolt Driver-a (Driver održava tabelu usmeravanja za sve vođe jezgra, sledbenike i replike za čitanje). | Potreban je dodatni sloj za balansiranje terete. |
| Enkripcija unutar klastera. | Nema. |
| Višestruko klasterovanje | Nema. |
| Višestruki DC | Nema. |

Jedna od najvećih razlika je u vezi sa dodatnim slojem zaštite koji uzročno grupisanje pruža u obezbeđivanju da će biti nula šansi za grananje i / ili oštećenje podataka, na šta je HA ranjiva zbog svoje osnovne arhitekture.

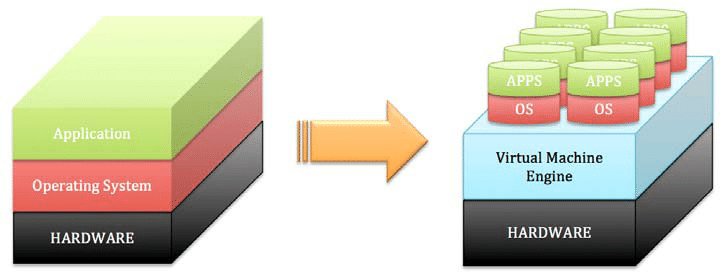
Još jedna zanimljiva tačka je karakteristika "vijak + usmeravanje" u uzročnim klasterima koja omogućava ugrađeno automatsko uravnoteženje opterećenja za aplikacije koje se povezuju sa klasterom, eliminišući tako potrebu za dodatnim slojem uravnoteženja opterećenja koji je obično potreban kod HA klastera, u jednostavnijoj i robusnijoj celokupnoj arhitekturi.

# 4. Praktičan primer HA i Kauzalnog klastera

U ovom poglavlju ćemo podići HA i kauzalni klister i uporediti ih. Za tu svrhu iskoristićemo Docker slike.

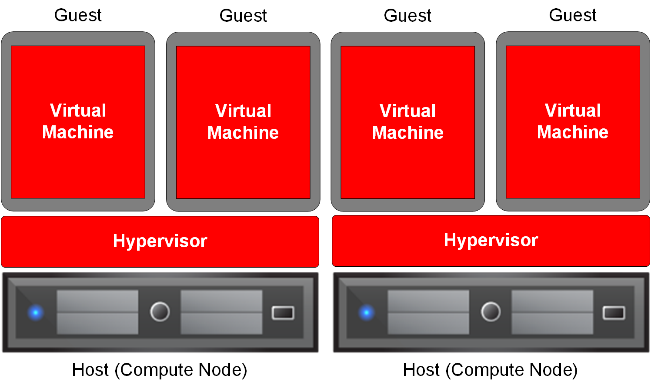
**4. 1. Docker**

Danas, kada je sve više web aplikacija, potrebni su i web serveri koji će držati web servise za te aplikacije. Kako sve više korisnika koristi web aplikacije, opterećenje servera koji sadrži servise drastično raste. Tako da se zbog svega toga, došlo do rešenja da se servisi i aplikacije distribuiraju na više servera. Ovakva vrsta rešenja, dovodi nas do još jednog problema, a to je visoka cena samih servera. Rešenje ovog problema se naziva virtualizacija.



Slika:8

**Virtualizacija** Kako će nam oko problema troškova pomoći virtualizacija? Kako smo rekli, zbog opterećenja jednog servera, servise i aplikacije ćemo postaviti i izolovati na više njih. To zahteva kupovinu dodatnog hardvera. Postavlja se pitanje, da li bi smo mogli jedan jak server sa dosta resursa podeliti na više manjih? Odgovor na ovo pitanje je da i taj odgovor nam daje **VIRTUALIZACIJA**. Virtualizacija nam omogućava da na našem matičnom računaru, delimo hardverske resurse i kreiramo zasebne virtualne mašine.



Slika:9

Softver koji nam omogućava virtualizaciju naziva se hypervisor. Hypervisor direktno komunicira sa hardverskim resursima matičnog računara i dodeljuje resurse virtualnim mašinama. Međutim, opet nam se javlja problem. Svaka virtualna mašina se može posmatrati kao poseban računar, što znači da ako želimo da na njemu pokrenemo aplikaciju moramo na toj mašini instalirati sve neophodne zavisnosti kao što je baza podataka, biblioteke i sve ostalo. Opet i sama virtualna mašina uzima deo operativne memorije, jezgra procesora i prostor na masovnoj memoriji i sve što joj je potrbno kako bi funkcionisala. Kako broj virtuelnih mašina raste, opterećenje matičnog računara raste. Tokom vremena pokušavalo se korišćenje resursa smanji. Jedno od rešenja jeste docker kontejner.



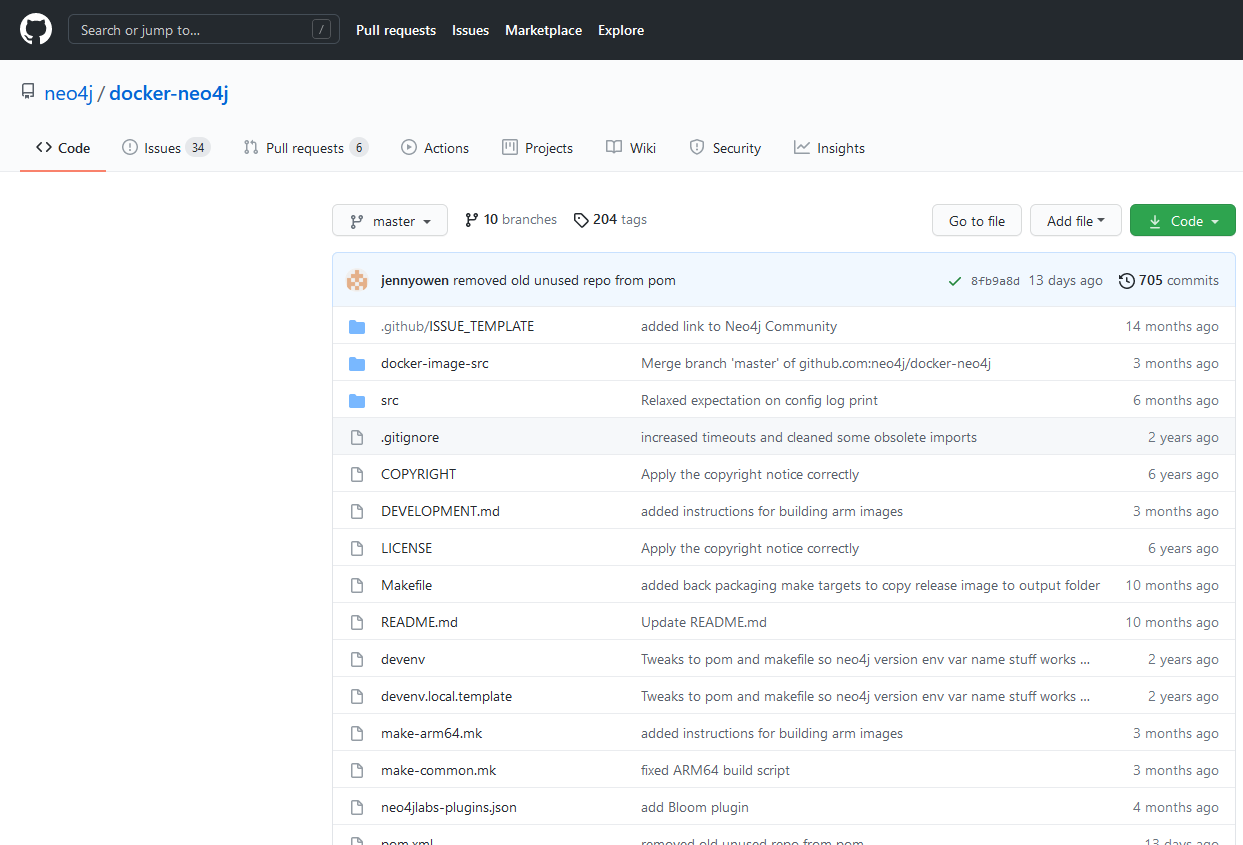
Slika:10

**Docker slika i kontejner** Šta je docker? Docker je jedna od novih tehnologija koja nam omogućava da različite aplikacije pokrećemo izolovano. Kod docker-a, svaka aplikacija je predstavljena uz pomoć docker slike. Pokretanje svake aplikacije preko docker slike zahteva kreiranje docker kontejnera. Docker slika je zapravo kolekcija fajlova i meta podataka. Tehnički fajlovi predstavljaju koren fajil sistema kontejnera.

Docker slike se sastoje od magacina slojeva. Svaki od slojeva može dodavati neke izmene i uklanjati fajlove. Docker slike mogu deliti međusobno slojeve kako bi optimizovali vreme transfera, korišćenje masovne i ram memorije. Kakva je razlika između docker slike i kontejnera? Doker slika predstavlja read-only fajl sistem, dok docker kontejner predstavlja enkapsulirani skup procesa pokrenutih u read-write kpiji tog fajil sistema.

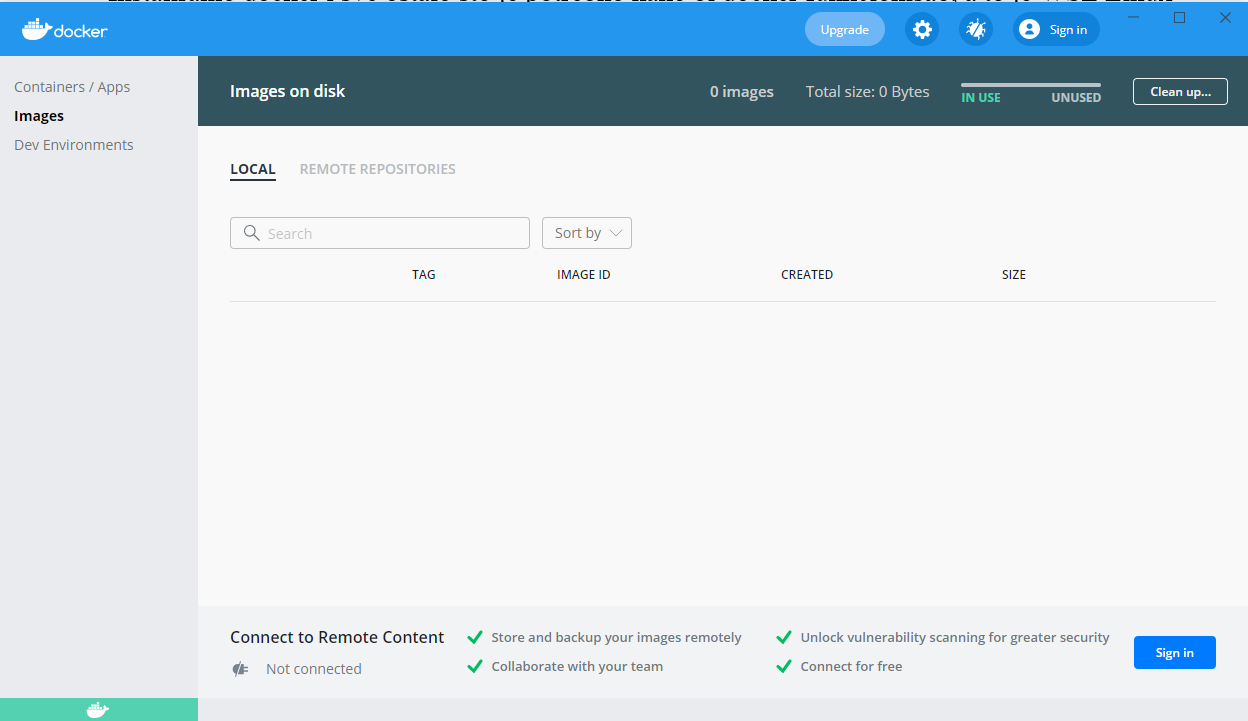
## 4.2. Podizanje klastera

Sada ćemo uz pomoć docker-a podignuti prvo HA klaster, a potom i kauzalni. Docker slike i yml fajlove preuzećemo sa zvaničnog Neo4j github repozitorijuma: <https://github.com/neo4j/docker-neo4j>.



Slika:11

Instaliramo docker i sve ostalo što je potrebno kako bi docker funkcionisao, a to je WSL Linux Kernel i njegov update. Kada je docker uspešno instaliran možemo da povlačimo već gotove kontejnere i slike ili pravimo svoje. Docker ima i jako zgodan GUI koji nam prikazuje koje sve doker slike imamo na računaru i mnoge ostale stvari.



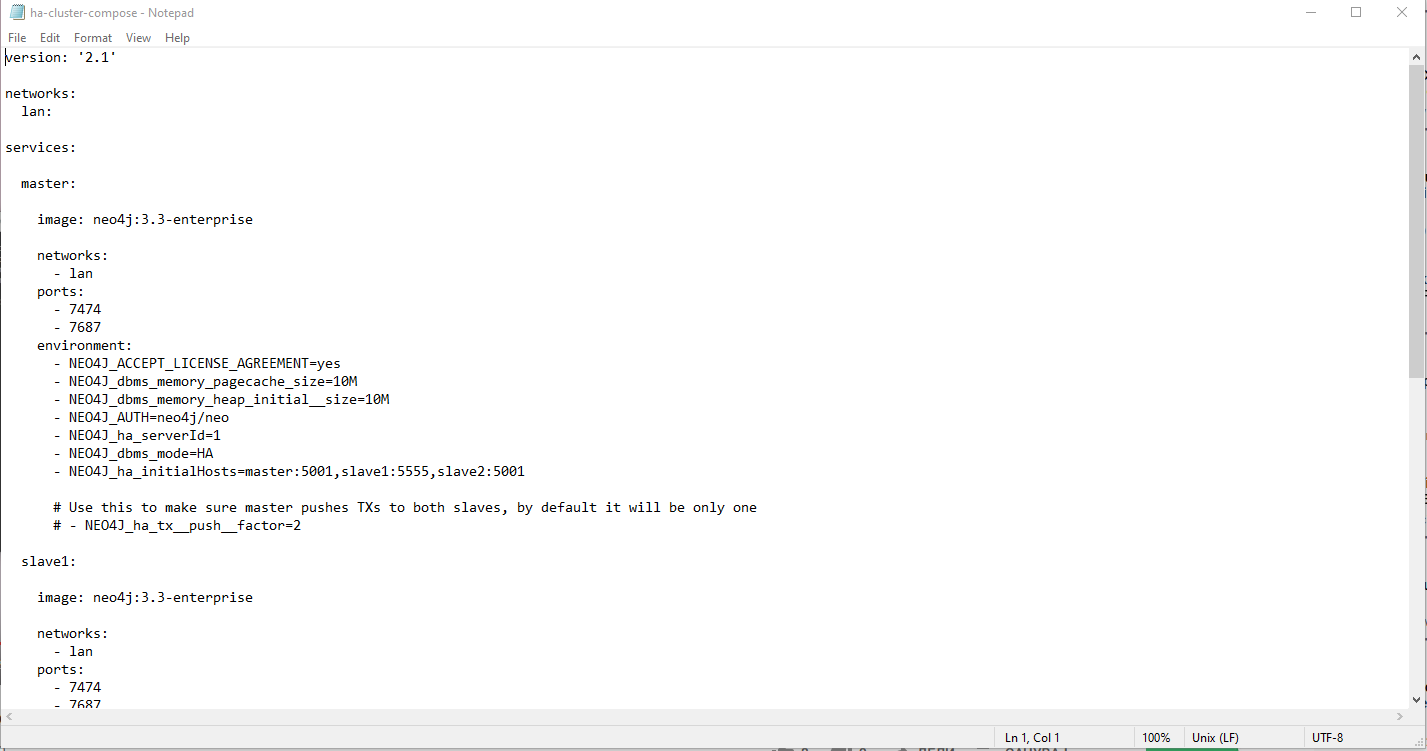
Slika:12

Sada ćemo putem docker compos-era podignuti HA Neo4j klaster. Yml fajil koji nam je potreban nalazi se u src/test/resources. To je fajil ha-cluster-compose.yml. Pre pokretanja moramo izvršiti male korekcije u ovom fajlu.



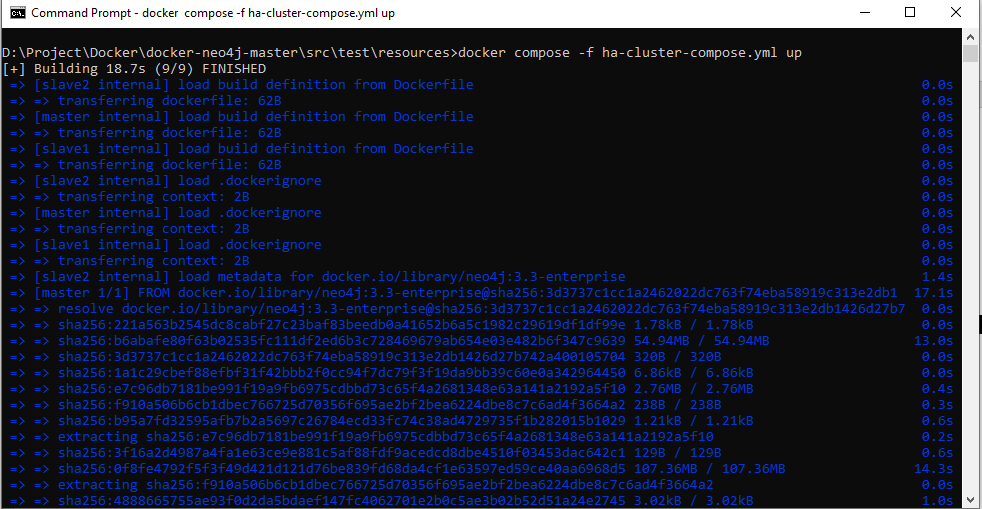
Slika:13

Kao što vidimo svaki član klastera tj. instanca je konfigurisana zasebno. Moraćemo da uklonimo celo volumes i user polje za svaku instancu i promeniti image parametar na verziju neo4j baze koju će skinuti kao sliku. Svakako ove parametre možemo i da ne brišemo nego da definišemo kao env promenjljive. Nakon korekcija fajil treba da izgleda ovako:



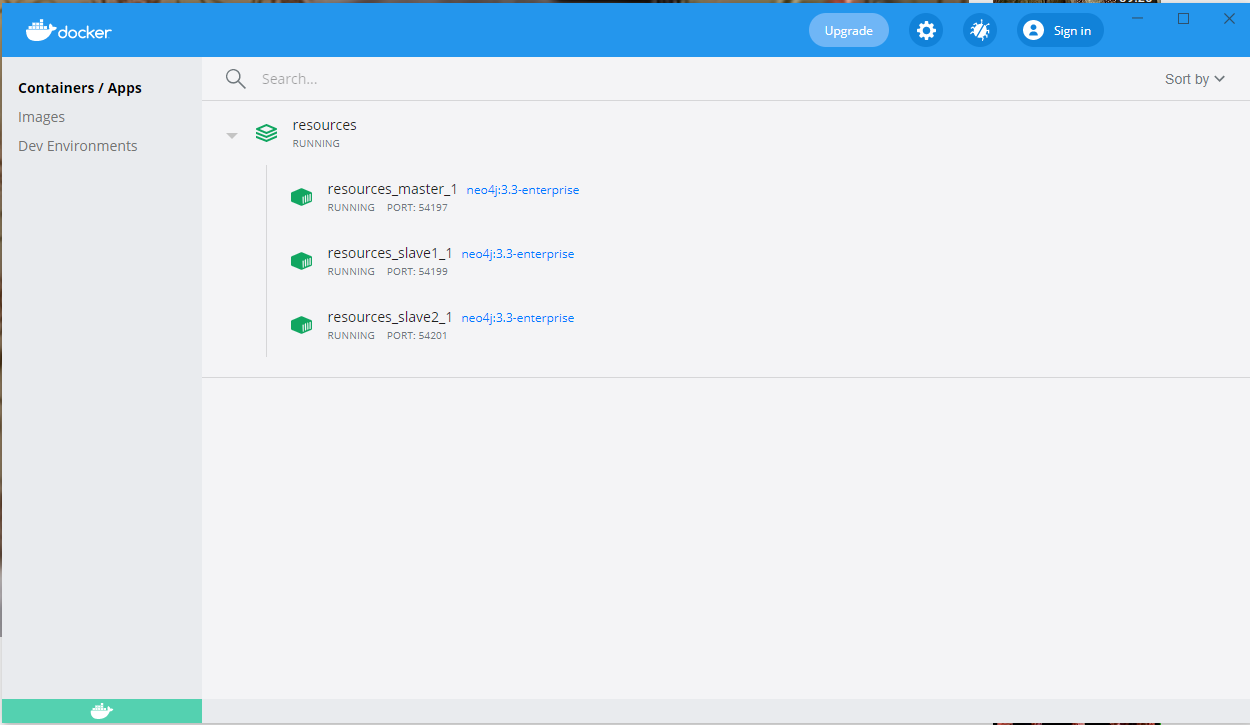
Slika:14

Koristićemo Neo4j verziju: neo4j:3.3-enterprise. U direktorijumu gde se nalazi yml fajil iz komandne linije pokrećemo komandu: **docker-compose -f ha-cluster-compose.yml up** i počinje se sa pribavljanjem i kofigurisanjem potrebnih stvari.



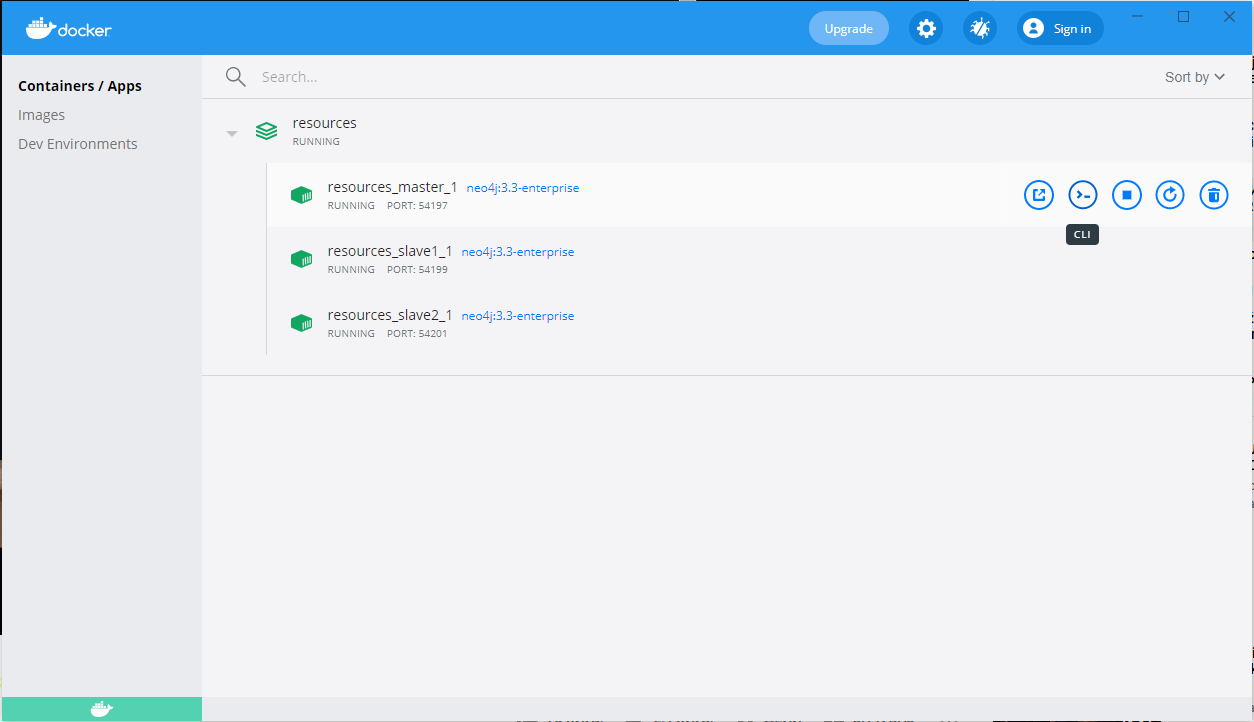
Slika:15

Ovaj HA klaster ima osnovnu strukturu sa jedno master i dve slave instance što možemo videti iz docker GUI-a:

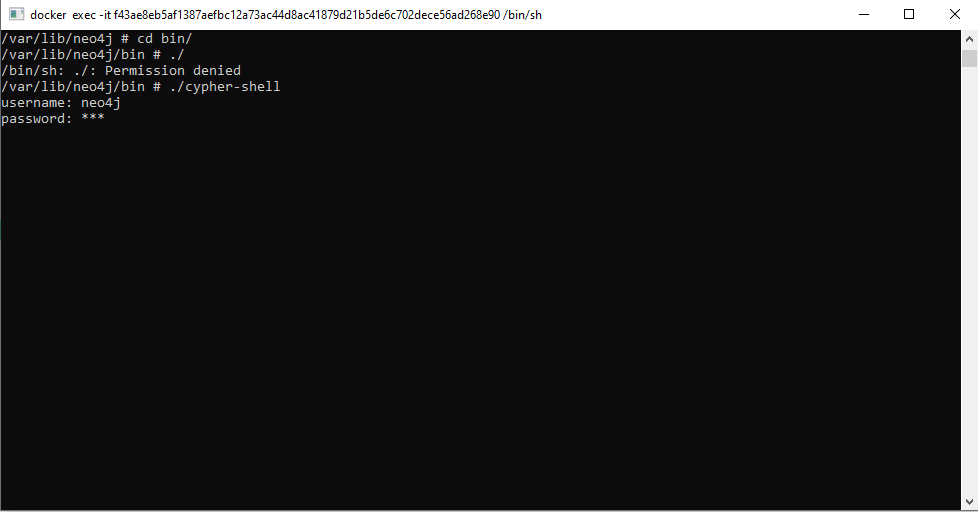


Slika:16

Sada ćemo preko konzole pristupiti master instanci klastera i ulogovati se na neo4j server.

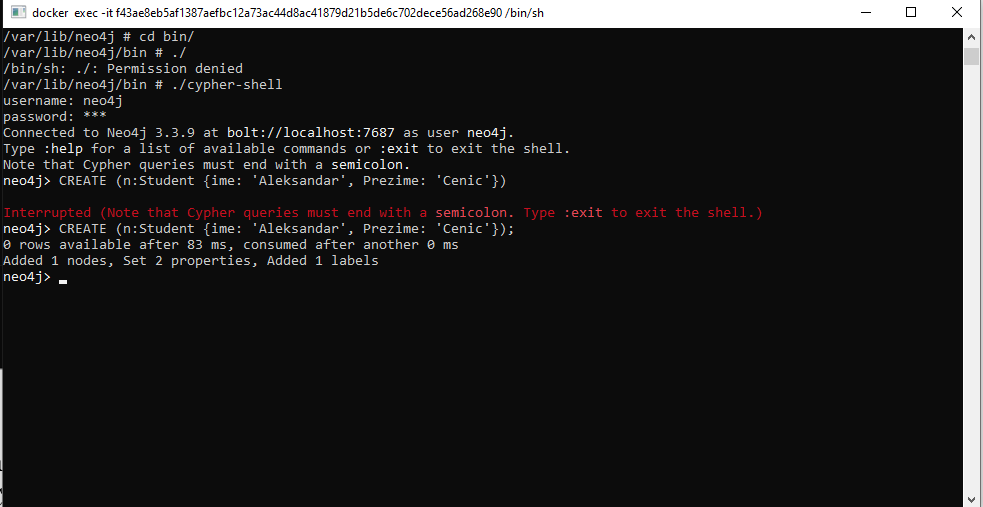


Slika:17



Slika:18

Ulogujemo se sa username: **neo4j** i passwordom: **neo**. Sada ćemo da kreiramo jedan čvor upitom: CREATE (n:Student {ime: 'Aleksandar', Prezime: 'Cenic'});.



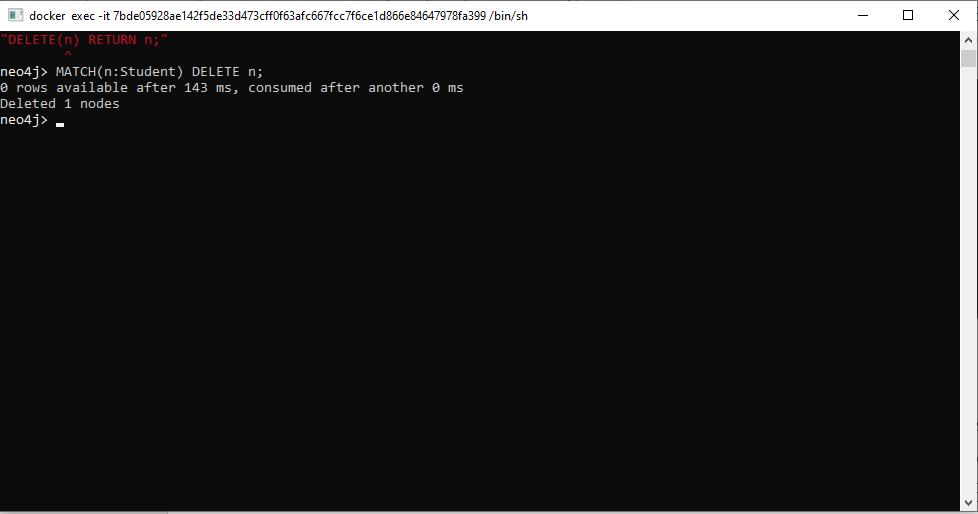
Slika:19

Sada ćemo se logovati na jednu od slave instanci i da li ona sada poseduje ovaj čvor.

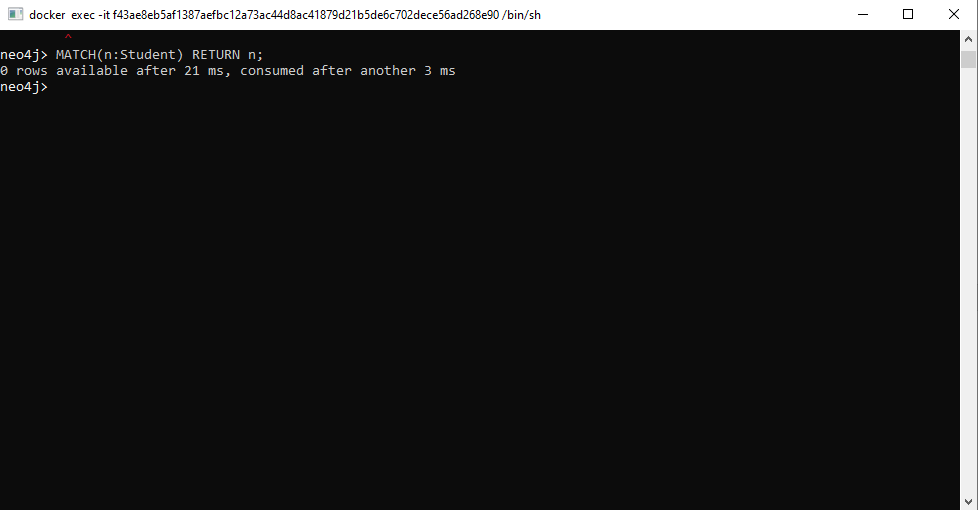


Slika:20

Čvor postoji i u bazi slave instance, znači da je transakcija uspešno potvrđena na obe instance. Hajdemo sada da na slave instanci obrišemo čvor, a potom pogledamo na master instanci šta će se dogoditi. Koristimo upit MATCH(n:Student) DELETE n;

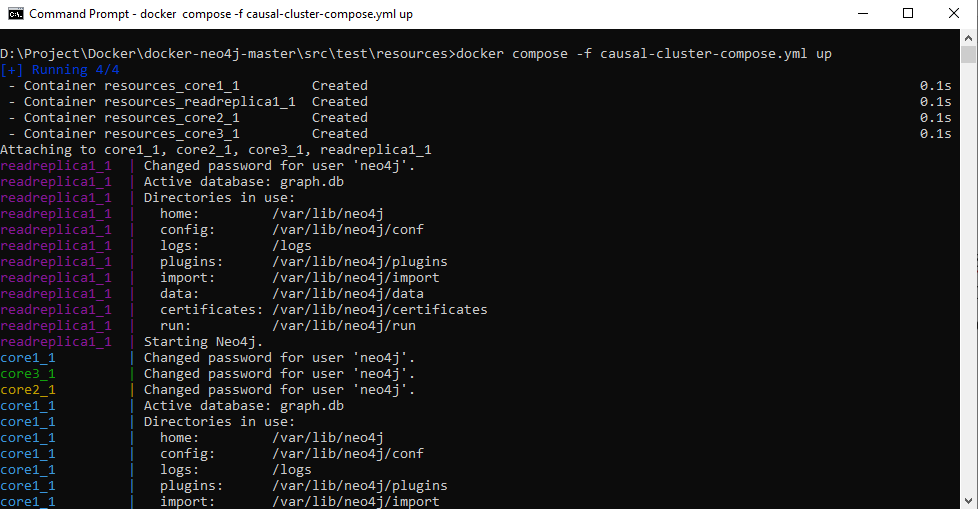


Slika:21



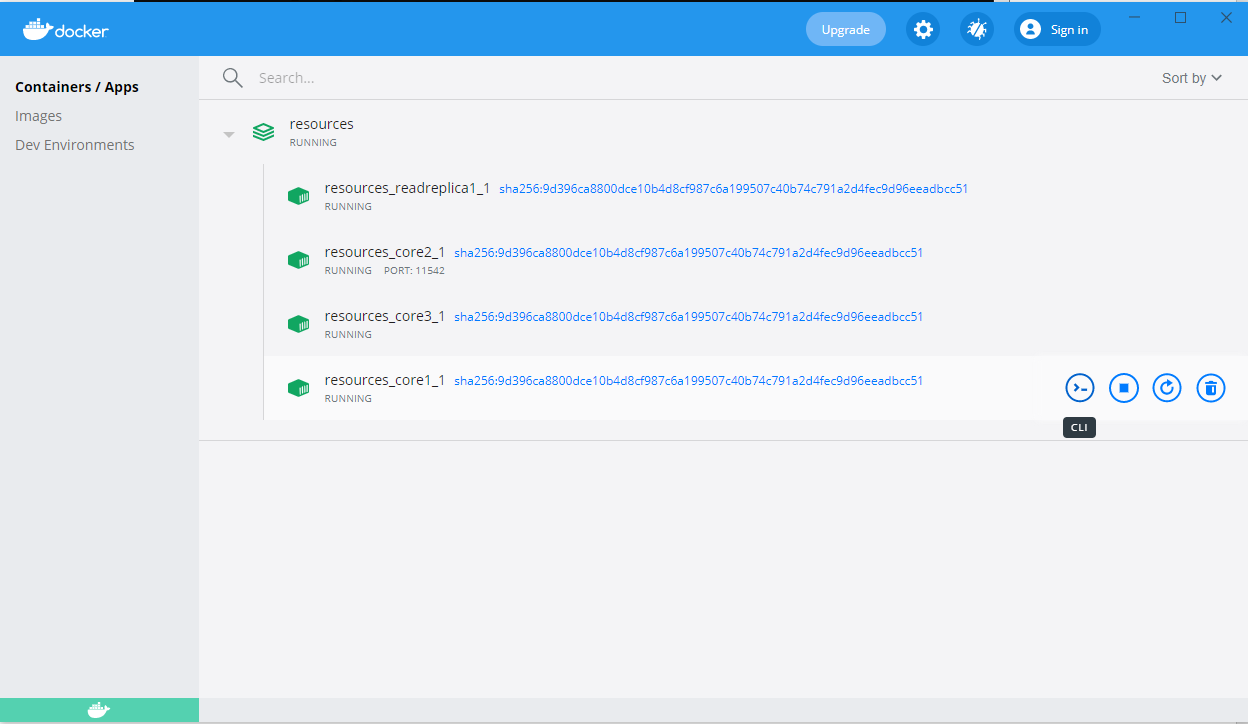
Slika:22

Vidimo da sada nema čvora ni na master instanci. Sada ćemo podići kauzalni klaster. Yml fajil koji nam je zato potreban nalazi se u istom direktorijumu: src/test/resources. To je fajil: causal-cluster-compose.yml. Takođe vršimo ispravku fajla kao i u prethodnom slučaju i koristimo sliku: neo4j:3.3-enterprise. Klaster podižemo komandom: **docker compose -f causal-cluster-compose.yml up**.



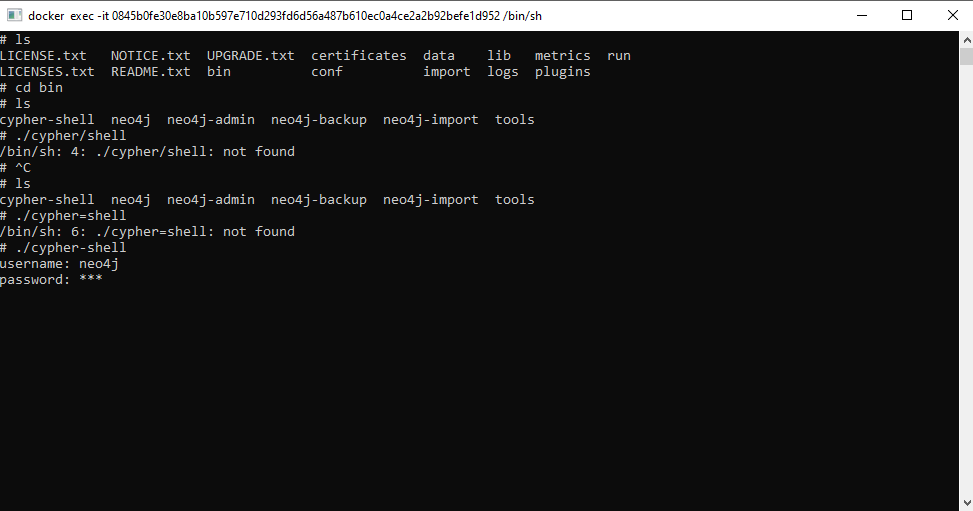
Slika:23

Pristupićemo prvoj Core instanci putem komandne linije.



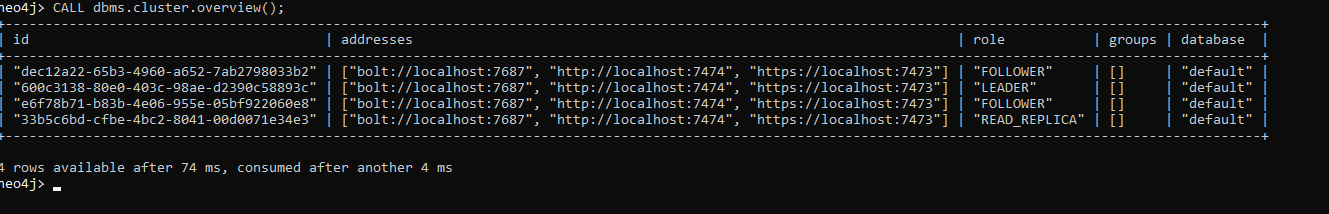
Slika:24

Logujemo se na prvu Core instancu sa istim username-om i passwordom ne4j neo.



Slika:25

Sada ćemo kao i u predhodnom slučaju kreirati čvor, pa ćemo proveriti na ostalim Core instancama i na replici za čitanje. Pre toga ćemo komandom **CALL dbms.cluster.overview()** pogledati sastav klastera.



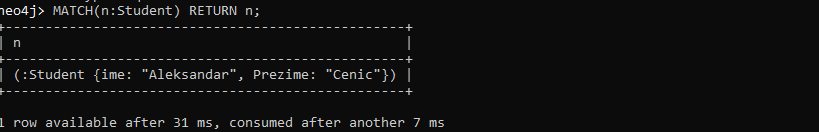
Slika:26

Vidimo da imamo 3 Core instance i jednu instancu koja je replika za čitanje. Sada ćemo kreirati čvor upitom: CREATE (n:Student {ime: 'Aleksandar', Prezime: 'Cenic'});.



Slika:27

Čvor je uspešno dodat. Pogledajmo sada kod replica instance.



Slika:28

Vidimo da se čvor nalazi i kod replike. Pokušajmo sada da sa replike obrišemo ovaj čvor. Ta operacija trebala bi biti neuspešna jer je replika read-only instanca u klasteru.



Slika:29

Kako smo i rekli operacija nije uspela! Za razliku Od HA klister arhitekture gde smo mogli dodavati i brisati sa slave instance!

# 5. zaključak

Distributivnost baze podataka je veoma bitna, pogotovo danas kada se obrađuje mnogo podataka, jer se opterećenje raspodeljuje na više servera, U ovom radu smo opisali i okvirno se upoznali sa klister rešenjima kod Neo4j baze podataka. Videli smo HA i kauzalnu arhitekturu. Neo4j je od verzije 3.5 izbacio HA klastere i posvetio se samo kausalnom rešenju. U praktičnom delu smo pokazali samo osnovnu funkiconalnost oba klister rešenja.

# literatura

[1. ] Master rad: Tehnologije distribuiranih baza podataka. Poglavlje 2 Teorijsko istraživanje strane[10,11,12,13].

[2. ] Članak: Neo4j Going Distributed with Graph Database. Link: <https://www.datanami.com/2020/02/04/neo4j-going-distributed-with-graph-database/>.

[3. ] Dokumentacija: The Neo4j Operations Manual v3.5. PoglavljeAppendix E: HA cluster strana[371].

[4. ] Članak: Introduction to the Neo4j Causal Clustering architecture. Link: <https://neo4j.com/docs/operations-manual/current/clustering/introduction/#causal-clustering-introduction>.

[5. ] Github repozitorijum docker-neo4j. Link: <https://github.com/neo4j/docker-neo4j>.