

Elektronski fakultet, Niš

Smer: Računarstvo i informatika, Softversko inženjerstvo

**Predmet: Sistemi za upravljanje bazama podatka**

Tema:

**RavenDB – cluster rešenja**

Student: Dušan Dimitrov 1428

Mentor: Doc. dr Aleksandar Stanimirović

Niš, januar 2023.

Sadržaj

[Uvod 3](#_Toc126520351)

[Uvod u RavenDB cluster 4](#_Toc126520352)

[CAP teorema kod RavenDB baze podataka 5](#_Toc126520353)

[Rachis 7](#_Toc126520354)

[Raft protokol 7](#_Toc126520355)

[Rachis protokol 8](#_Toc126520356)

[Replikacija 10](#_Toc126520357)

[Kako radi replikacija? 11](#_Toc126520358)

[Replikacioni konflikti 12](#_Toc126520359)

[Change vector 13](#_Toc126520360)

[Praktična demonstracija 15](#_Toc126520361)

[Zaključak 22](#_Toc126520362)

[Literatura 23](#_Toc126520363)

# Uvod

Cluster je po definiciji skup od 3 ili više instance neke baze podataka. On uvodi mnoge napredne koncepte kod baza podataka i načina skladištenja podataka. Sa druge strane uvodi dodatnu složenost i postavljaju se neka pitanja o kojima se ne mora voditi računa kada je prisutna samo jedna instanca baze podataka. Takođe jako je teže održavati čitav cluster u odnosu na samo jednu instancu baze podataka.

RavenDB baza podataka uvek radi sa cluster-om, samo ostavlja mogućnost korišćenja jednog ili više čvora. Jako lako se upravlja RavenDB cluster-om preko Raven Studio-a, koji predstavlja klijentsku aplikaciju za upravljanje RavenDB bazom podataka. Ovim su mnogi koncepti cluster-a pojednostavljeni, ali opet postoji uvid u sve koncepte cluster-a.

Ovaj rad je posvećen cluster rešenjima kod RavenDB baze podataka. Nakon uvodnog poglavlja se kreće sa uvodom u svet RavenDB cluster-a i objašnjavaju se najosnovniji teorijski koncepti. Ovo drugo poglavlje objašnjava i CAP teoremu kod RavenDB baze podataka i pokazuje kako RavenDB može da zadovolji sve zahteve konzistentnosti, dostupnosti i otpornosti na particionisanje (naravno, na različitim nivoima). Treće poglavlje opisuje Rachis protokol – RavenDB implementaciju poznatog Raft protokola za postizanje konzistentnosti u distribuiranom sistemu. Četvrto poglavlje je posvećeno replikaciji, dok peto poglavlje objašnjava kako se rešavaju konflikti koji mogu doći usled replikacije. U šestom poglavlju su praktično demonstrirani osnovni koncepti rada sa RavenDB cluster-om kroz Raven Studio. Sedmo poglavlje je zaključak rada, a poslednje, osmo poglavlje predstavlja spisak korišćene literature.

# Uvod u RavenDB cluster

RavenDB cluster predstavlja 3 ili više instanci RavenDB server-a. Instance cluster-a se zovu **cluster nodes (cluster čvorovi)**. Kao članovi cluster-a, cluster čvorovi obavljaju razne operacije koje su pod kontrolom cluster-a.

Kreiranje cluster-a kod RavenDB baze podataka ima puno smisla zato što nakon kreiranja i konfiguracije baze podataka na RavenDB cluster-u, cluster sam **automatski upravlja** bazom podataka. Jedna konkretna baza podataka se može nalaziti na jednom čvoru, nekoliko čvorova, ili čak na svim čvorovima cluster-a. U tom slučaju će svaki čvor posedovati kompletnu kopiju baze podataka, pa će moći da opslužuje sve klijente baze podataka sa zahtevima, upitima, operacijama, upisima, čitanjima,…

Cluster takođe omogućava distribuciju posla različitim čvorovima, oporavak od otkaza čvorova i skladno funkcionisanje čvorova.

Sve operacije u RavenDB bazi podaka se mogu podeliti na:

1. **cluster-wide operacije (operacije na nivou cluster-a)**
2. **interne operacije**.

Cluster-wide operacije su operacije koje se dešavaju na nivou čitavog cluster-a. Na primer, kreiranje nove baze podataka je cluster-wide operacije. Sa druge strane, interne operacije su operacije koje utiču samo na jednu konkretnu bazu podataka. Na primer, kreiranje dokumenta u RavenDB bazi podataka predstavlja internu operaciju.

Jako je bitno postojanje 2 tipa različitih operacija kod RavenDB baze podataka, zato što ona sama funkcioniše na 2 različita nivoa u svom distribuiranom sistemu. Na nivou cluster-a čvorovi rade zajedno kako bi postigli isti cilj. Taj cilj je glasanje i izbor **lidera** između njih. Ovo se postiže putem konkretne implementacije **Raft konsenzus protokola**. Lider je zadužen za:

* posmatranje cluster-a,
* očuvanje njegovog “zdravlja“,
* izbor konkretnog čvora kog će klijenti koristiti kako bi pristupili bazi podataka, konfiguraciju baze podataka,
* obezbeđivanje konzistentnosti,...

Sa druge strane, na nivou baze podataka, odnosno tamo gde se obavljaju interne operacije, svi su čvorovi jednaki, rade zajedno i imaju ista zaduženja.

Postavlja se pitanje zašto je RavenDB baza podataka dizajnirana tako da ima 2 različita tipa/moda operacija. Na prvi pogled deluje da bilo lakše i jednostavnije da samo postoji jedan mod operacija. To i jeste tačno, tako bi bilo lakše i jednostavnije, međutim, ne bi bilo nužno bolje!

Naime, Raft konsenzus algoritam pruža jaku konzistentnost kod RavenDB baze podataka. Izabrani lider vodi računa o tome da čvorovi međusobno mogu nesmetano da rade i obavljaju operacije. Jaka konzistentnost kod RavenDB baze podataka pruža sigurnost pri radu. Ukoliko cluster donese neku odluku (na primer da doda bazu podataka na nekom čvoru), jaka konzistentnost nam pruža sigurnost da će ili čitav cluster prihvatiti tu odluku, ili neće nijedan čvor u cluster-u. Sa druge strane, mod baza podataka je tretiran na potpuno drugi način. Svaki čvor cluster-a ima kompletnu kopiju topologije i sve neophodne informacije i zbog toga može da funkcioniše nezavisno od ostalih čvorova. Dakle operacijama na nivou cluster-a svi rade na isti način, a operacijama na nivou baze podata svi mogu nezavisno da rade. Zato su i podeljene ove operacije kod RavenDB baze podataka.

Ovakva podela operacija i modova rada dovodi do toga da RavenDB cluster postaje jedan jako dobar, robustan, stabilan i “snažan” sistem. Najpre se definiše cluster, distribuiraju se baze podataka na različitim čvorovima, dodele se zaduženja i nakon toga svaka instanca baze podataka radi samostalno i nezavisno. Na primer, to znači da svaka instanca baze podataka može da prihvati operaciju upisa bez konsultacije sa ostalim instancama iste baze podataka.

RavenDB preuzima mnoge koncepte o cluster-u iz DynamoDB baze podataka, razrađuje neke mehanizme Cassandra baze podataka i na taj način stavlja do znanja da su **operacije upisa jako bitne**. Ukoliko neko želi da obavi operaciju upisa nad čvorom u RavenDB cluster-u, operacija upisa se prihvata bez konsultacije sa ostalim čvorovima. [1]

## CAP teorema kod RavenDB baze podataka

Kada govorimo o distribuiranim bazama podataka, postoje određeni zahtevi koje moraju da ispune distribuirane baze podataka.

**Konzistentnost (consistency)** je jedan jako bitan koncept distribuirane baze podataka i odnosi se na to da se sam sistem baze podataka mora nalaziti u konzistentnom stanju posle svake operacije.

**Dostupnost (availability)** znači da je sistem uvek dostupan i da nema takozvanog downtime-a. Takođe, sistem mora biti tolerantan na otkaz određenih čvorova, odnosno mora da postoji neka kopija/replika podatka kome klijent uvek može da pristupi.

**Otpornost/tolerantnost na particionisanje (paritition tolerance)** znači da sistem funkcioniše čak i ako ne postoji konekciju između čvorova. Ovo može nastati usled pada mreže.

**CAP** (**C**onsistency, **A**vailabiliy, **P**artition tolerance) teorema tvrdi da je moguće zadovoljiti 2 od 3 zahteva. Shodno tome baze podataka mogu biti:

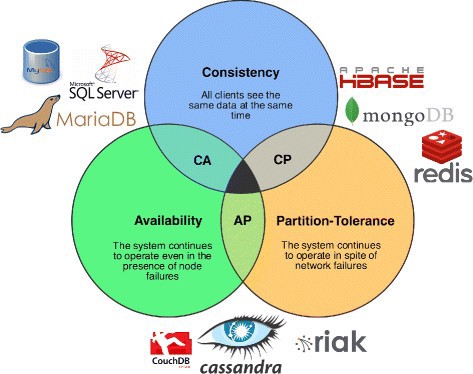
1. CA
2. CP
3. AP.

CA (Consistency & Availability) - zadovoljeni su zahtevi konzistentnosti i dostupnosti. U ovom slučaju se pravi kompromis oko partition tolerance-a. CA sistemi su karakteristični za bankarske sisteme gde je konzistentnost jako bitna.

CP (Consistency & Partition tolerance) – zadovoljeni su zahtevi konzistentnosti i otpornosti na particionisanje. Pravi se kompromis za dostupnost podataka, što znači da pristup pojedinim podacima može biti privremeno onemogućen.

AP (Availability & Partition tolerance) – zadovoljeni su zahtevi dostupnosti i otpornosti na particionisanje, a kompromis se pravi oko konzistentnosti podataka. Ovakvi sistemi zahtevaju određeni algoritam koji će služiti za razrešavanje konflikta pošto podaci nisu uvek konzistentni. [2]

Na narednoj slici se može videti takozvani **CAP dijagram** na kom su prikazane koje baze podataka zadovoljavaju koje zahteve CAP teoreme:



Na slici možemo videti par baza podataka, ali ne možemo videti gde spada RavenDB. To je zato što je teško naći RavenDB bazi podataka samo jedno mesto na CAP dijagramu, odnosno, **RavenDB je i CP i AP**. Na prvi pogled to deluje nerealno i nemoguće, ali zapravo jeste. RavenDB baza podataka je i CP i AP, ali ne na istim slojevima. Na nivou cluster-a je RavenDB baza podataka CP, odnosno konzistentnost je uvek prisutna, ali može doći do toga da određeni podaci nisu dostupni u slučaju particionisanja. Sistem će nastaviti da funkcioniše, otporan je na particionisanje, ali neki podaci mogu da ne budu dostupni. Sa druge strane, na nivou baze podataka, RavenDB je AP. Otporna je RavenDB baza podataka na particionisanje, podaci su dostupni, međutim, oni su eventualno konzistentni. Može da se desi da nisu najnoviji podaci stigli do te instance baze podataka, ali, postoje algoritmi koji sprečavaju dobavljanje starih i nevalidnih podataka. [1]

# Rachis

Rachis predstavlja RavenDB **implementaciju Raft protokola**.

Raft je jedan jako jednostavan protokol za postizanje konsenzusa u distribuiranom sistemu. On omogućava izvršavanje niza operacija nad čitavim cluster-om u tačno definisanom redosledu. Ovo znači da je moguće izvršiti niz operacija za promenu stanja svakog čvora i da će nakon njihovog izvršenja stanje svakog čvora u cluster-u biti identično. Raft komanda je komanda/operacija koja će se izvršiti u istom redosledu na svakom čvoru. Lider čvor prihvata sve Raft komande i izvršava ih na nivou cluster-a. Raft komande se izvršavaju jedino ako je većina čvorova spremna da ih uradi, dakle, neophodan je konsenzus. Ukoliko većina čvorova nije spremna da obavi Raft komandu, komanda se neće ni izvršiti. [3]

## Raft protokol

Posmatrajmo jedan prost primer cluster-a sa samo jednim čvorom. Kada klijent želi da pošalje neku vrednost takvom cluster-u, dosta je lako zato što nema potrebe ni za kakvim dogovorom, samo je jedan čvor i vrednost će biti konzistentna u čitavom sistemu. Međutim, ukoliko postoje više čvorova u cluster-u, dogovor oko konsenzusa postaje malo složeniji. Ovo je problem distribuiranog konsenzusa koji Raft protokol nastoji da reši.

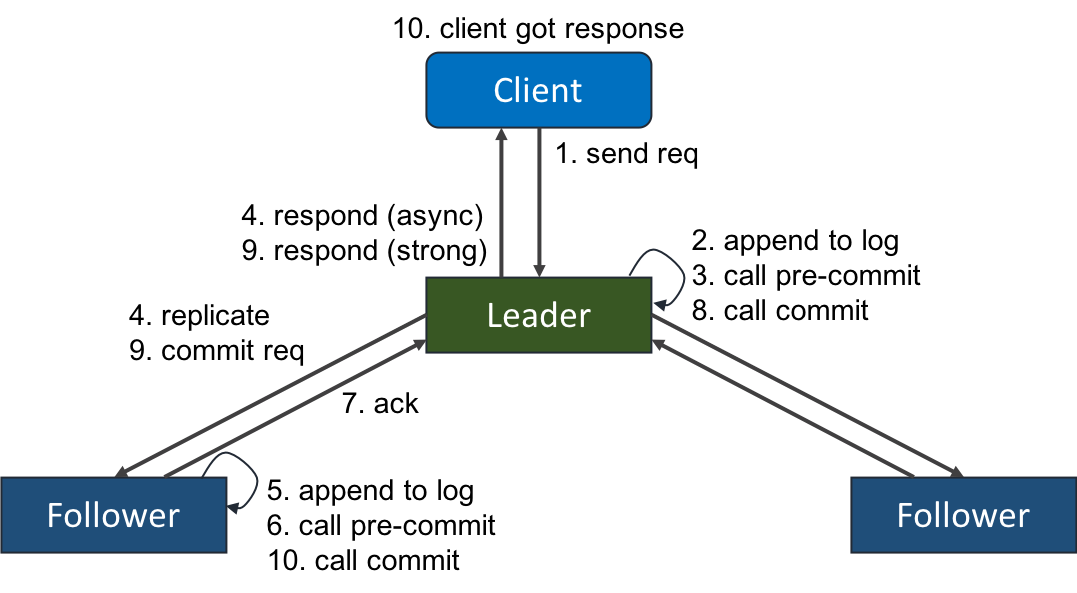
Kod Raft protokola, svaki čvor može biti u jednom od tri stanja:

1. **Follower (sledbenik)**,
2. **Candidate (kandidat)**,
3. **Leader (lider)**.

Svi čvorovi u cluster-u kreći sa stanjem sledbenika. Ukoliko sledbenik ne dobije odgovor od lidera, onda on postaje kandidat. Nakon toga, kandidat zahteva glasove (vote) od ostalih čvorova. Ostali čvorovi odgovaraju svojim glasom. Kandidat postaje lider, ukoliko je dobio većinu glasova. Ovaj proces se zove **izbor lidera (leader election)**.

Sve promene u sistemu idu preko lidera. Najpre lider dobija zahtev o promeni nekog podatka/vrednosti/informacije od klijenta. Svaka promena se najpre upisuje u log čvora. Ovo je nekomitovana vrednost, tako da se zapravo neće izvršiti ažuriranje vrednosti čvora. Nakon toga, lider obavlja takozvani pre-commit i šalje podatke svim svojim sledbenicima. Lider čeka da većina čvorova upiše novi podatak u svoj log, obavi pre-commit i obaveštava lidera o tome. Tek nakon toga lider obavlja komit podatka i obaveštava svoje sledbenike o tome, pa i oni obavljaju komit, odnosno zvanično upisuju podatak. Ovako je postignut konsenzus o stanju sistema. Ovaj proces se zove **replikacija logova (log replication)**. [4]

Na sledećoj slici je moguće videti objašnjeni protokol:



## Rachis protokol

Rachis predstavlja RavenDB implementaciju Raft protokola i originalnom protokolu dodaje nove funkcionalnosti, a životni ciklus čvora je prikazan na sledećoj slici:



1. Novi čvor započinje u stanju **passive**, što znači da on zapravo nije član nijednog cluster-a. Ukoliko se čvor izbaci iz cluster-a, on opet postaje passive.
2. Kada se čvor doda u cluster, on odmah postaje **leader** **(lider)** ukoliko je jedini čvor u cluster-u.
3. Kada se čvor doda u cluster, ali već postoji lider, onda čvor postaje **promotable (koji napreduje)**. Nakon toga može postati **member follower (član sledbenik)** ili **watcher follower (posmatrač sledbenik)** zavisno od razloga kreiranja čvora.
4. Čvor postaje member follower ukoliko nije drugačije definisano.
5. Čvor postaje watcher follower ukoliko je tako definisano prilikom dodavanja čvora.
6. Member follower može da bude degradiran u watcher-a.
7. Watcher follower može biti unapređen u member foolower-a.
8. Member follower postaje **candidate (kandidat)** kada dođe vreme za izbor novog lidera.
9. Kada izbor lidera prođe, jedan kandidat postaje lider, a ostali postaju follower-i.

Watcher follower je potpuno nov koncept kod Rachis protokola i on ne postoji kod originalnog Raft protokola. Naime, dodavanje novog čvora donosi mnoge benefite samom cluster-u, međutim, ukoliko se razmisli malo, uvideće se činjenica, da što je više čvorova follower-a, Raft protokol će biti sporiji. Baš zbog toga Rachis protokol definiše watcher follower-e koji ne učesvuju u donošenju odluka, ali su totalno ravnopravni za obavljanje operacija klijenata. Tako se odluke na nivou cluster-a svode na mali broj čvorova, a sam cluster je dosta veći. [5]

# Replikacija

Replikacija kod RavenDB baze podataka predstavlja proces prenosa podataka sa jedne instance baze podataka na drugu.

Kod RavenDB baze podataka podaci su visoko dostupni i sve operacije upisa i čitanja se mogu obaviti na bilo kom čvoru koji sadrži instancu baze podataka. Operacija upisa je uvek prihvaćena i automatski se obavlja replikacija na ostalim čvorovima u okviru grupe **baze podataka (database group)**. [6]

Grupa baze podataka, odnosno database group se odnosi na sve instance jedne baze podataka u cluster-u. Svaki čvor u grupi baze podataka ima kompletnu kopiju svih podataka koju čine svi dokumenti, indeksi i tombstone-ovi. [7]

Takođe, replikacija se može obaviti i samo na čvorovima definisanim replikacionim taskom.

Postoje 3 replikaciona tipa:

1. **interni (internal)**
2. **eksterni (eksternal)**
3. **hub/sink.**

**Interna replikacija** predstavlja replikaciju između svih čvorova jedne grupe baze podataka (database group). Ova replikacija se obavlja automatski i obavljaju je sami čvorovi grupe. Ne postoji nikakvo filtriranje podataka prilikom replikacije, odnosno sve što postoji na izvoru će biti replicirano i na odredištu. Konflikte rešava database resolution policy koji je isti za sve baze u grupi. Generalno kod interne replikacije ne postoji nikakvo kašnjenje, već se sve replicira trenutno. Interna replikacija služi kako bi baze podataka održavala sinhronizovanom. Operacija upisa se može obaviti nad bilo kojim čvorom u grupi baza podataka. Ta izmena će internom replikacijom biti dostupna trenutno kod svih čvorova u grupi i moćiće drugi klijenti da pristupaju novom podatku.

**Eksterna replikacija** je proces kopiranja/replikacije podataka između 2 čvora baze podataka koja se nalaze u 2 različita cluster-a. Ovaj proces replikacije se obavlja pomoću **eksternog replikacionog taska** koga definiše korisnik baze podataka. Ovo je samo jednosmerna replikacija, a filtriranje ne postoji, odredište sadrži onakve podatke kakve ima i izvor replikacije. Konfilikte rešava database resolution policy kod odredišne baze podataka. On ovde može biti drugačiji nego kod izvorne baze podataka, što nije bio slučaj kod interne replikacije. Takođe, za razliku od interne replikacije, kod eksterne replikacije može postojati kašnjenje, a definiše se u okviru taska za eksternu replikaciju. Sama primena eksterne replikacije je ta da omogućava postojanje replike baze podataka u drugom klasteru koja se može koristiti u slučaju otkaza inicijalnog cluster-a.

Poslednji tip replikacije je **hub/sink replikacija**. Hub/sink replikacija predstavlja replikaciju između više sink-ova koji se mogu povezati na jedan hub u različitom cluster-u. Da bi se obavila ova replikacija, potrebno je definisati takozvani **hub/sink replikacioni task**. Smer replikacije kod hub/sinka može biti različit, od huba do sinka samo, od sinka do huba samo, ali i u oba smera, a sve se to definiše u okviru hub/sink replikacionog taska. Ovaj tip replikacije, za razliku od internog i eksternog, omogućava filtriranje dokumenata. Odlazni i dolazni dokumenti koriste wildcard-ove i documend id-eve kako bi selektovali dokumente koje žele da repliciraju. Kašnjenje kod hub/sink replikacije može da postoji i kao kod eksterne replikacije, moguće ga je definisati prilikom kreiranja taska.

## Kako radi replikacija?

Svaki čvor sa RavenDB instancom baze podataka drži TCP konekciju ka svim ostalim bazama podataka u grupi, ukoliko je interna replikacija. Ukoliko nije reč o internoj replikaciji, onda svaki čvor sadrži TCP konekciju ka svim čvorovima sa kojima treba da obavlja replikaciju.

Kada se desi upis na nekom čvoru, podatak će se automatski poslati ostalim čvorovima zaduženim za replikaciju. Slanje podataka se obavlja **asinhrono**. Ukoliko instanca baze podataka nije u mogućnosti da trenutno obavi replikaciju podatka, svejedno će prihvatiti akciju upisa, ali će je obaviti kasnije.

Svaka RavenDB instanca baze podataka ima svoj lokalni **ETag baze podataka**. On predstavlja zapravo jedan brojač, čija se vrednost povećava na svaki upis. Svaki podatak koji je potrebno upisati dobija sledeći broj. Redosled replikacije podataka zavisi od njihovom Etag broja. Manji brojevi predstavljaju ranije replikacije, dok veći predstavljaju novije.

Podaci se šalju za replikaciju u **batch**-evima. Kada se obavi replikacija jednog batch-a, pamti se Etag poslednjeg podatka koji je obrađen u batch-u i to predstavlja **last-accepted-ETag**. Odredište za replikaciju šalje odgovor izvoru i u odgovoru se nalazi last-accepted-ETag. Tako izvor zna gde da nastavi sa slanjem podataka. Ukoliko dođe do otkaza replikacije prilikom slanja nekog batch-a, replikacija će nakon oporavka nastaviti da se obavlja od last-accepted-ETag-a.

# Replikacioni konflikti

Konflikt nastaje kada se jedan dokument konkurentno ažurira na 2 različita čvora. Do ovoga može doći ukoliko klijenti ažuriraju brže nego što je potrebno da bi se podatak replicirao između čvorova.

Generalno u svim distribuiranim sistemima postoje 2 načina razmišljanja kod konflikta. Jedan je da se izabere korišćenje nekog konsenzus algoritma koji zahteva donošenje odluke o upisu prilikom svake operacije. Sa jedne strane ovo je bezbednije rešenje, međutim, sa druge strane, može da bude sporije. Drugi način razmišljanja kod konflikta je da se prihvate potencijalni konflikti i da se nauči funkcionisanje sa njima, odnosno da postoji mehanizam za detekciju konflikta.

Naime, na nivou dokumenta, RavenDB bira prihvatanje konflikta i tako dozvoljava upis u svakom trenutku na svakom čvoru.

RavenDB kao mehanizam za detekciju konflikta koristi **vektore promene** (**change vector**-e). Svaki dokument ima svoj vektor promene, tako da kada čvor dobije batch za replikaciju, on najpre uporedi vektore promene novog dokumenta sa svojim lokalnim.

Radi lakšeg objašnjavanja, neka ***remove\_cv***predstavlja vektor promene nekog remote dokumenta koji treba da se upiše/promeni, a neka ***local\_cv***bude vektor promene lokalnog dokumenta. Postoje 3 situacije:

1. ***remote\_cv*** <= ***local\_cv*** – nije potrebno ništa raditi. Lokalni dokument je noviji od remote dokumenta koji je potencijalno trebao da se upiše.
2. ***remote\_cv*** > ***local\_cv*** - remote dokument je noviji od lokalnog, pa je potrebno zameniti lokalni remote dokumentom.
3. ***remote\_cv*** nije ni <= ni > ***local\_cv*** - pokušati da se reši konflikt.

Kada postoji konflikt, RavenDB ima niz aktivnosti koje će primeniti kako bi pokušao da razreši konflikt. Najpre će se pogledati jedan fleg: **ResolveToLatest**. Njegova default-na vrednost je true i u tom slučaju će razrešiti konflikt tako što će prihvatiti dokument čiji je property **LatestModified** (poslednji modifikovan). Ukoliko ResolveToLatest fleg nije setovan na true, RavenDB proverava da li je mozda kojim slučajem sadržaj oba dokumenta identičan. U tom slučaju zapravo ni ne postoji konflikt. Ukoliko nije slučaj da su dokumenti identični, RavenDB će primeniti **Conflict Resolver skriptu**. To je jedna skripta koja služi kako bi se razrešili konflikti i piše se na nivou kolekcije. Skripta ima sledeće ulazne argumente:

* docs – nesortirana lista dokumenata koji su u konfliktu
* hasTombstone – boolean vrednost koja pokazuje da li postoji tombstone u konfliktnim dokumentima
* resolveToTombstone – string koji ukoliko se vrati znači da će da se razreši konflikt brisanjem dokumenta.

Povratna vrednost skripte može biti:

* objekat koji će biti zarešenje konflikta
* null – ništa – što zapravo znači da konflikt nije razrešen
* resolveToTombstone – string, koji će razrešiti konflikt na tombstone-u.

Ukoliko ni ova skripta ne razreši konflikt, ostaje poslednji korak, a to je manuelno razrešavanje konflikta.

## Change vector

Change vector, odnosno vektor promene je RavenDB implementacija Vector clock algoritma. Vector clock algoritam pravi poredak događaja u nekom distribuiranom sistemu. [8]

Change vector je jedan vektor koji se kreira na osnovu entry-a baze podataka i to na osnovu entry-a u svakoj instanci. Na primer, change vector je:

**[A:1-0tIXNUeUckSe73dUR6rjrA, B:7-kSXfVRAkKEmffZpyfkd+Zw]**

Ovaj change vector je kreiran na osnovu 2 entry-a:

1. **A:1-0tIXNUeUckSe73dUR6rjrA**
2. **B:7-kSXfVRAkKEmffZpyfkd+Zw**

Svaki entry ima sledeću strukturu:

**Tag čvora : ETag : ID baze podataka**

Change vector **A:1-0tIXNUeUckSe73dUR6rjrA** zapravo nosi informaciju da se dokument promenio na čvoru **A**, da je lokalni ETag **1** i da je id baze podataka **0tIXNUeUckSe73dUR6rjrA**.

Da bi se za 2 change vector-a reklo da je x >= y, treba da ETag svakog entry-a iz x bude veći ili jednak odgovarajućem ETag-u u drugom change vector-u. Takođe, to podrazumeva da y nema nijedan entry koji x nema.

Slično važi i za operator <=. Da bi se reklo da je x <= y, moraju svi ETag-ovi iz x da budu manji ili jednaki odgovarajućim ETag-ovima u y i y ne sme da ima neke entry-e različite od x.

Međutim, nekada je nemoguće reći <= ili >= zato što može da se desi da x ima entry čiji je ETag veći od odgovarajućeg ETag-a u y, ali i neki drugi ETag u x manji nego u y. U tom slučaju change vector neće pomoći prilikom razrešavanja konflikta, već će morati da se primene malopre pomenute metode razrešavanja konflikta putem skripte ili manuelno.

Demonstracija poređenja 2 change vectora se može videti na par primera. Posmatrajmo change vectore samo sa oznaka čvora i ETag-a, znači bez id-a baze podataka radi lakšeg poređenja. Kod change vector-a v1: [A:8, B:10, C:34] i v2:  [A:23, B:12, C:65] možemo uočiti:

* A --> 8 (v1) < 23 (v2)
* B --> 10 (v1) < 12 (v2)
* C --> 34 (v1) < 65 (v2)

Iz svega ovoga se može zaključiti da je v2 > v1.

Za 2 vektora, v1: [A:23, B:12, C:65], v2: [A:18, B:12, C:51] možemo primetiti:

* A --> 23 (v1) > 18 (v2)
* B --> 12 (v1) = 12 (v2)
* C --> 65 (v1) > 51 (v2)

Dakle, u ovom slučaju je v1 > v2.

Za 2 vektora, v1: [A:18, B:12, C:65], v2: [A:58, B:12, C:51]:

* A --> 18 (v1) < 58 (v2)
* B --> 12 (v1) = 12 (v2)
* C --> 65 (v1) > 51 (v2)

Ovde je ETag A u v1 je manji nego u v2, a C veći u v1, što znači da se ne može primeniti operatori > ili <. [9]

# Praktična demonstracija

Prilikom rada sa RavenDB bazom podataka je praktično nemoguće raditi samo sa jednom instancom baze podataka bez cluster-a. Naime, uvek će se zapravo kreirati cluster, samo što će biti definisan samo 1 čvor.

Prednost ovoga je da zapravo ne postoje neki specijalni koraci za kreiranje cluster-a sa više čvorova.

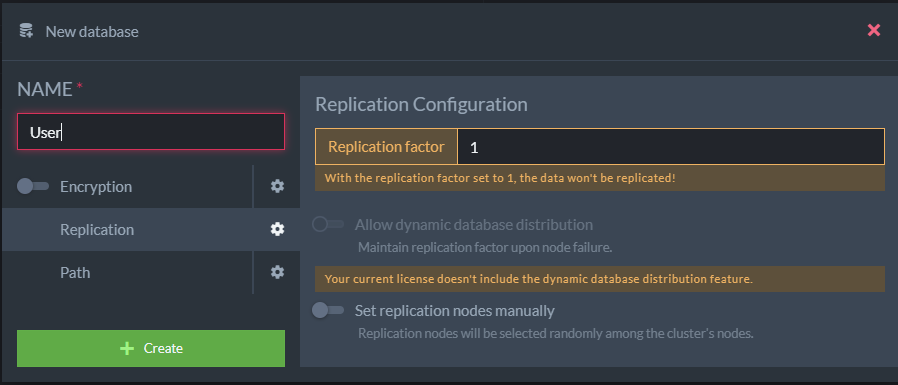
Cluster sa jednim čvorom će se kreirati lokalno, pokretanjem RavenDB servera baze podataka i slanjem odgovarajućih parametara. Parametri su za url, adresu i port, putanju za logovanje i direktorijum sa podacima:

**Raven.Server.exe --ServerUrl=http://127.0.0.1:8080 --Logs.Path=Logs/A --DataDir=Data/A**

Postoji jedan json file koji služi kao konfiguracija. To je **settings.json** fajl i u njemu je potrebno prekonfigurisati određene stvari. Neophodno je staviti vrednost **Setup.Mode** na **None** zato što ne želimo nikakvu specijalnu konfiguraciju od RavenDB baze podataka. Potrebno je prihvatiti korisćenje licence RavenDB baze podataka, pa vrednost **License.Eula.Accepted** postaviti na **true**. Potrebno je vrednosti za **ServerUrl** i **ServerUrl.Tcp** obrisati, zato što će se ServerUrl sam poslati kao parametar prilikom startovanja RavenDB baze podataka, a za tcp url ćemo pustiti interni mehanizam da dodeli neki slobodan url.

Kao što se može pretpostavili iz potrebne konfiguracije tcp i http konfiguracije, RavenDB koristi i TCP i HTTP mehanizme. TCP se koristi za međusobnu komunikaciju između čvorova. Moguće je specificirati i ServerUrl.Tcp prilikom pokretanja servera, ali, ukoliko ima više instanci, nemoguće je da ona ostane konstantna u settings.json fajlu, već se mora eksplicitno poslati prilikom startovanja servera baze podataka. Sa druge strane, kao što je ovde urađeno, može se ne konfigurisati, pa će RavenDB koristiti ServerUrl i dodeliti mu neki random port za tcp. Ovo može da funkcioniše ukoliko se radi sa lokalnom mašinom ili sa okruženjem kome se veruje!

Rezultat pokretanja servera baze podata se može videti na localhost:8080 gde se pokreće nova instanca baze podataka. Kako bi se pratio rast cluster-a, kreiraće se jedna osnovna baza podataka sa user-ima. Odlaskom na stranicu Databases i izborom opcije New database se otvara dijalog sa sledeće slike gde je potrebno konfigurisati osnovne parametre:



Odlaskom na stranicu Documents, klikom na dugme New document i unosom podataka u json obliku se kreiraju prvi podaci. Za kreiranje dokumenta user-a, koristiće se samo 2 property-a: Ime i Prezime, pa je json upit:

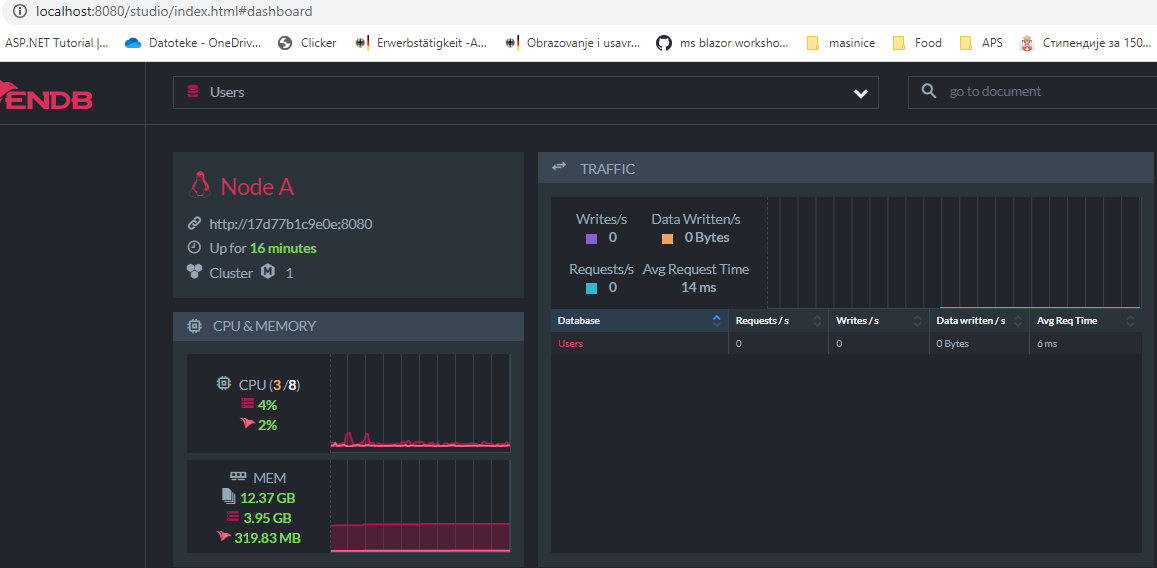
**{**

**"Ime": "Dusan",**

**"Prezime": "Dimitrov"**

**}**

Nakon toga se može videti stanje čvora A – node A na sledećoj slici:

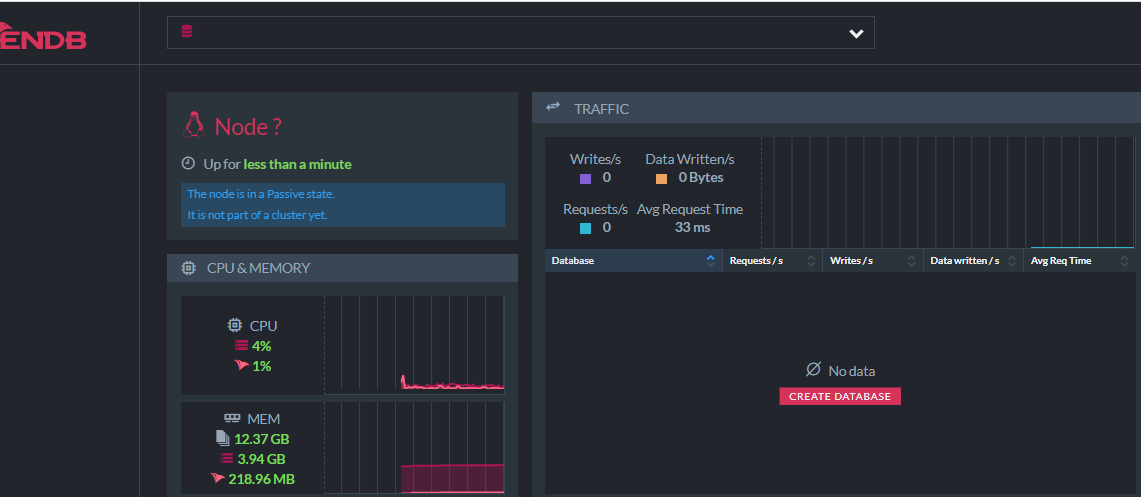


Nastavlja se sa proširivanjem cluster-a, pokretanjem nove instance, a inicijalnu komandu za startovanje je potrebno samo malo izmeniti. Potrebno je promeniti port na lokalnoj mašini, pošto će 8080 biti zauzet nakon pokretanja prvog čvora. Sledeća instanca se pokreće na portu 8081 komandom:

**Raven.Server.exe --ServerUrl=http://127.0.0.3:8081 --Logs.Path=Logs/B --DataDir=Data/B**

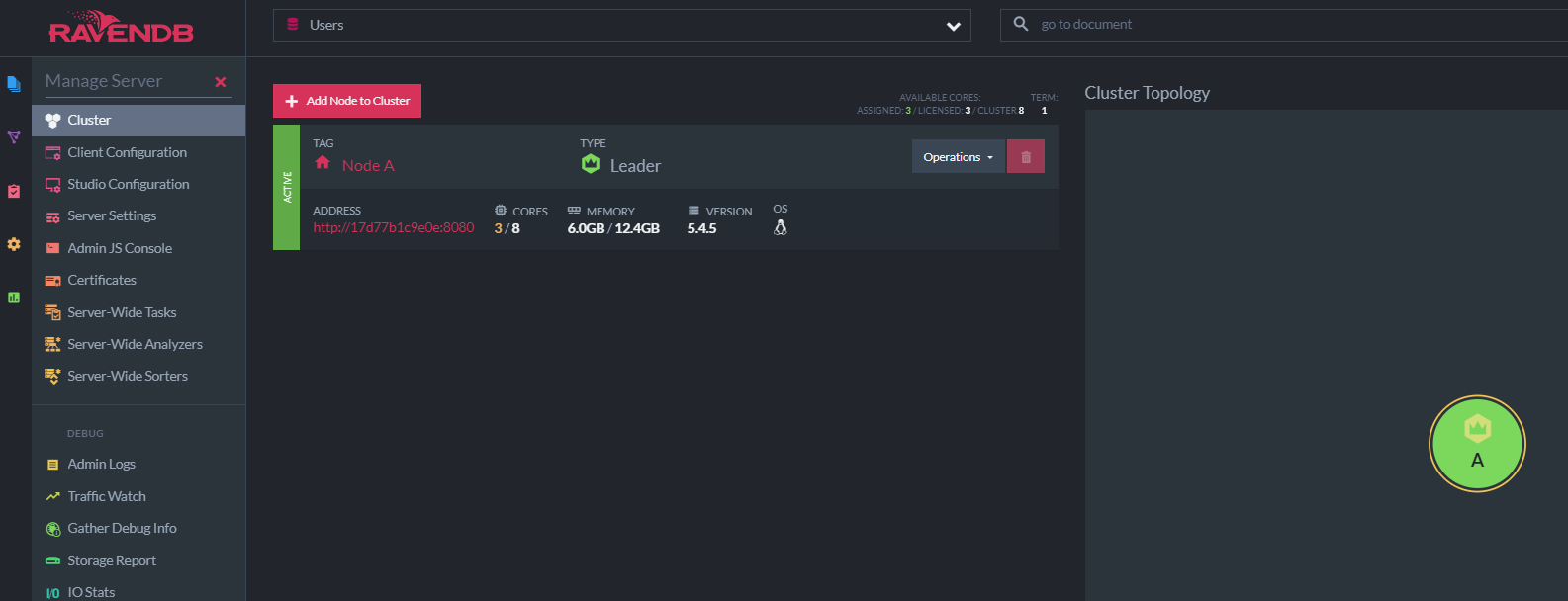
U ovom konkretnom slučaju se nova instanca RavenDB baze podataka pokreće na istoj mašini, a u praksi se to radi i na drugoj mašini.

Na sledećoj slici je moguće videti novu instancu RavenDB baze podataka:

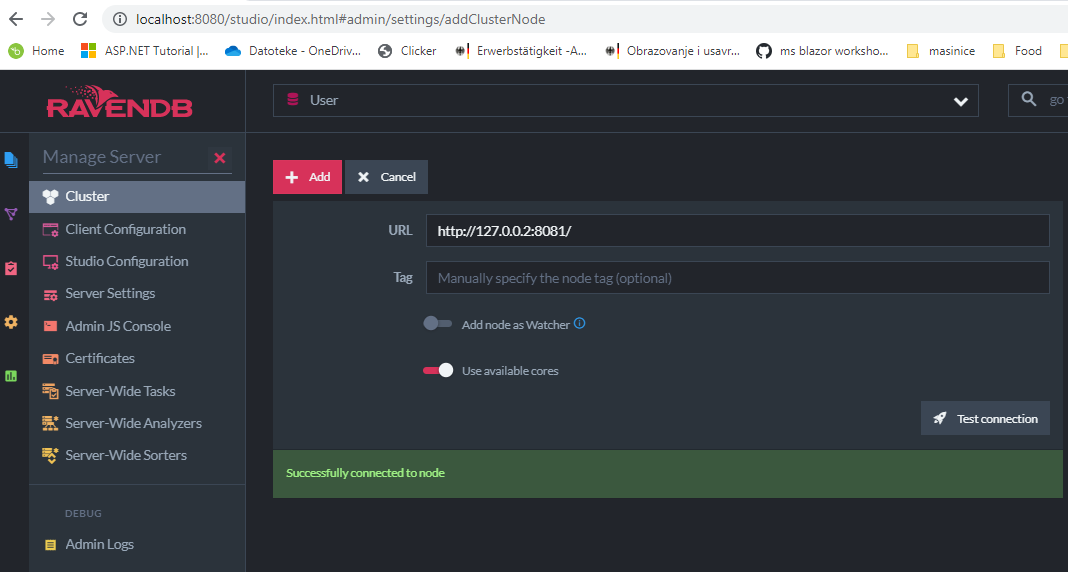


Na slici je moguće videti da je čvor obeležen sa znakom pitanja što označava nepoznati čvor. To je tako zato što je ovo novi čvor i na njemu nije obavljena nijedna operacija. Sa druge strane, na prvoj instanci smo kreirali jednu bazu podataka, pa je sam RavenDB cluster taj čvor obeležio sa A.

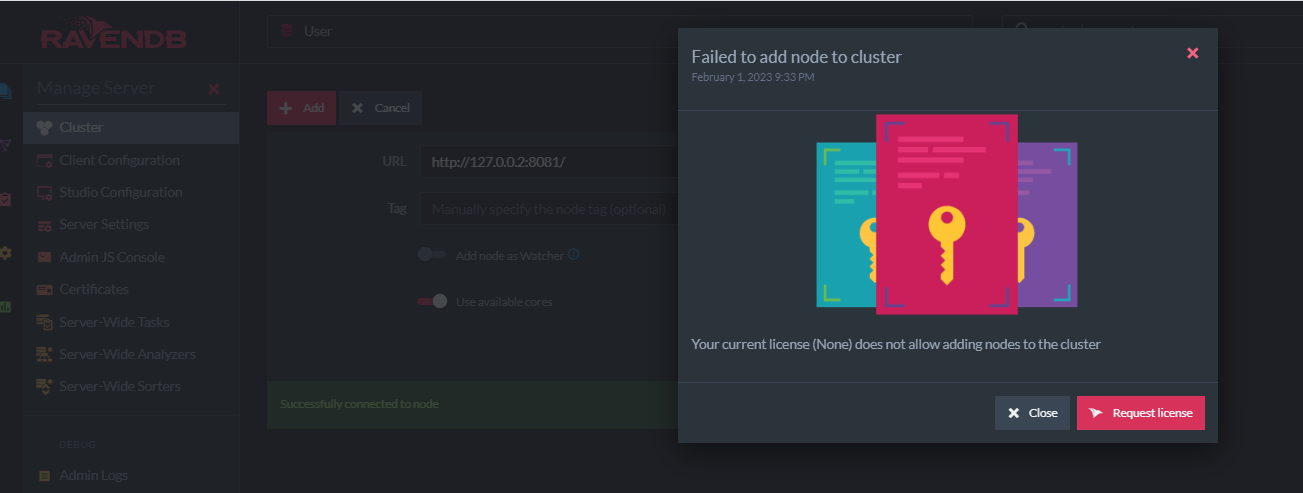
Da bi se dodao drugi čvor u cluster, neophodno je otići na čvoru A (port 8080) na **Manage Server** stranicu i nakon toga izabrati **Cluster** iz menija:



Na slici je moguće videti već pomenutu činjenicu da je samo čvor A u cluster-u i moguće je videti topologiju cluster-a sa trenutno jednim čvorom. Klikom na dugme **Add Node to Cluster** se otvara meni gde je potrebno uneti url novog čvora. Nakon unosa url-a, je potrebno testirati konekciju što se postiže klikom na dugme **Test connection**. Nakon toga se dobija poruka o uspešnosti konekcije, kao što je moguće videti na sledećoj slici:



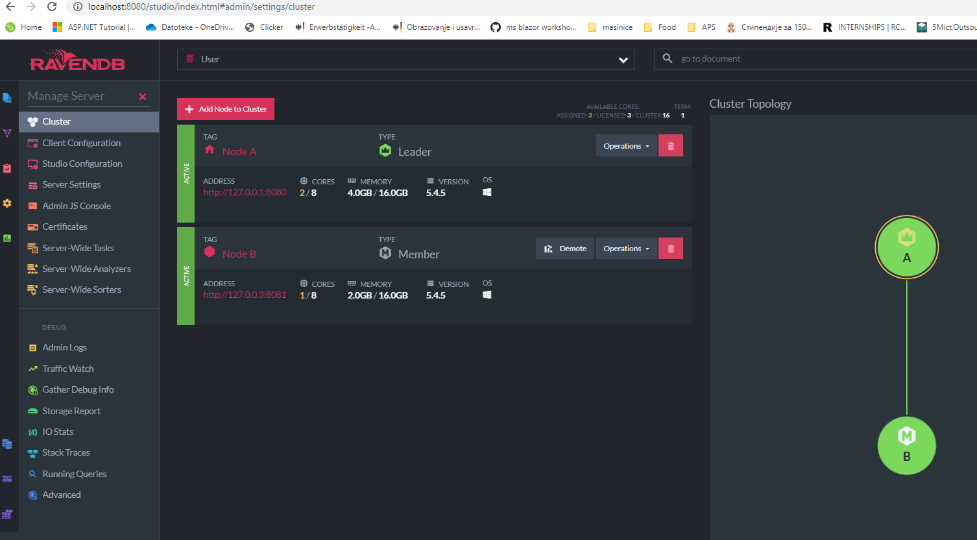
Međutim, bez dodatne konfiguracije, dakle samo sa onim opisanim do sada, klikom na dugme **Add** se **neće dodati čvor u cluster**. Informaciju o tome je moguće videti na sledećoj slici:



Na datoj slici je moguće videti poruku o licenci i svodi se na to kako je zapravo nemoguće kreirati cluster sa više od jednog čvora. Ovo je delimično tačno. Naime, kada se doda novi čvor, kao što je i ranije objašnjeno, on će biti u **Passive stanju**. Da bi server postao deo cluster-a, moguće je uraditi više različitih stvari. Moguće je bootstrap-ovati cluster, ili izvršiti neku RAFT komandu. Pod RAFT komandama spada i dodavanje licence, ali može se kreirati i baza podataka na serveru. Kako je licenca neophodna za neke druge stvari, ovde će se izabrati opcija licenciranja RavenDB-a. [10]

Odlaskom na link [https://ravendb.net/buy [11](https://ravendb.net/buy%20%5b11)] je moguće izabrati licencu. Za potrebe predmeta je dovoljna community, odnosno free licenca koja dozvoljava veličinu cluster-a od 3 instance. Nakon registracije se na ostavljeni mejl dobija **license key** koji je potrebno podesiti na serveru, a to je moguće uraditi i kroz Raven Studio odlaskom na About i izborom opcije za dodavanje licence.

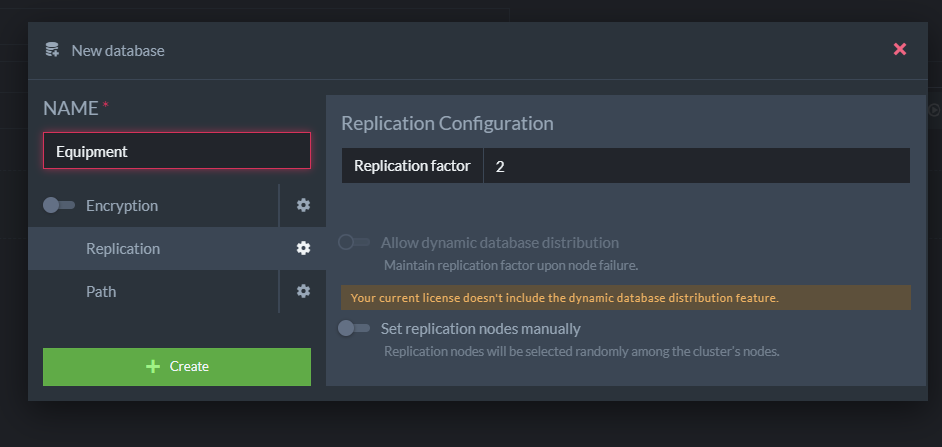
Nakon dodavanja licence je moguće dodati novi čvor u cluster, a rezultat toga je moguće videti na sledećoj slici:



Osim pomenutih informacija, na slici je moguće videti i to da je čvor A izabran za leader-a, dok je čvor B member.

Sada kada postoje 2 čvora u cluster-u, moguće je kreirati i neku novu bazu podataka sa replikacionim faktorom 2.

Na čvoru B iz menija za baze podataka (Databases) se bira opcija **New database**. Definiše se naziv druge baze koju želimo da kreiramo, ali se unosi i replikacioni faktor, koji sada može da bude i 2.



Sledeći dokument se kreira na čvoru A (na primer) i biće automatski upisan na oba čvora A i B:

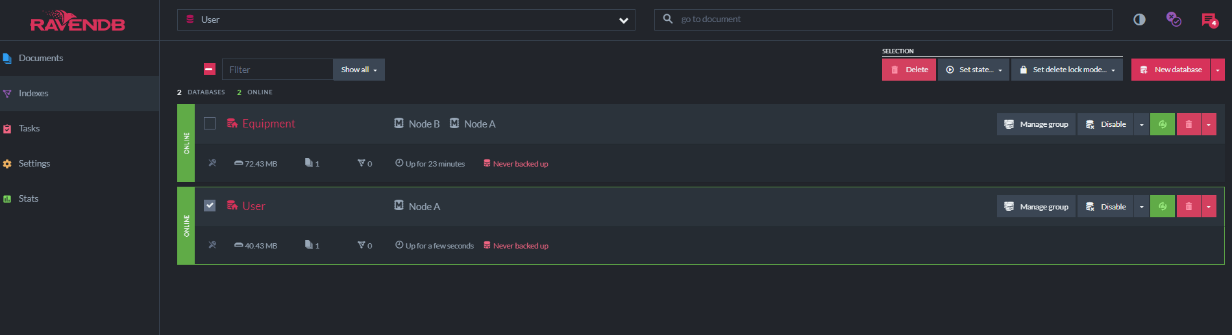
**{**

**"Naziv": "Equipment1",**

**"Vlasnik": "Dusan"**

**}**

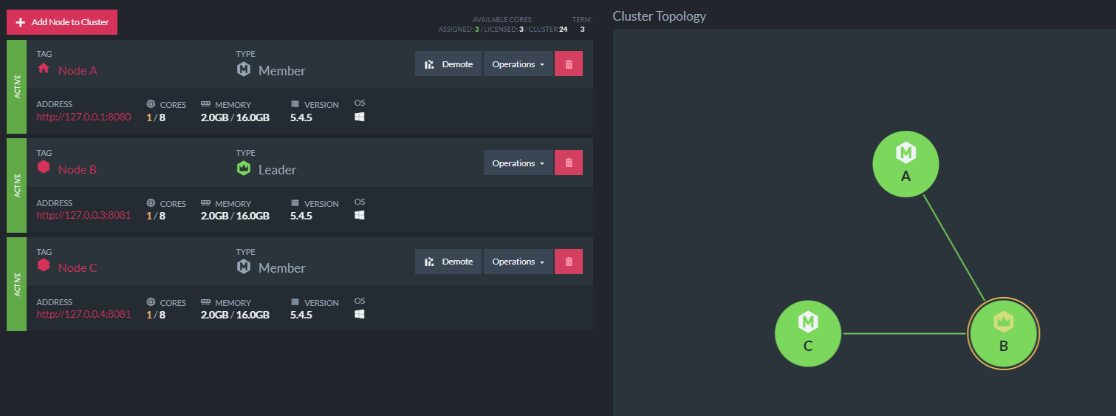
Da bi ovo funkcionisalo, neophodno je da obe baze podataka budu online, a ne offline, što se konfiguriše isto na stranici Databases. Samo je potrebno iz padajućeg menija ***Set the state*** izabrati ***Enabled*** opciju.



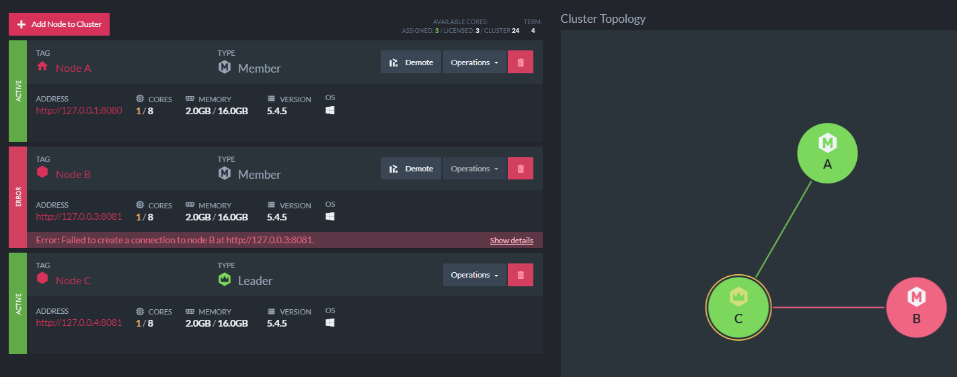
Preporučena veličina cluster-a je 3 ili 5 čvorova. Kako besplatna licenca dozvoljava kreaciju samo 3 čvora, kreiraće se još jedan čvor radi dalje demonstracije. Komanda za pokretanje novog čvora je slična kao i prethodne 2, samo će se promeni ServerURL i putanje za logove i podatke:

**Raven.Server.exe --ServerUrl=http://127.0.0.4:8081 --Logs.Path=Logs/C --DataDir=Data/C**

Tako pokrenuta instanca je još u pasivnom stanju i potrebno je dodati u cluster na već opisan način. Sada naš cluster izgleda kao na sledećoj slici:



Jako je interesantno posmatrati situaciju kada neki čvor otkaže. To je jako lako moguće simulirati “ubijanjem” RavenDB procesa, na primer čvora B. Kako je čvor B trenutno Leader, nakon otkaza čvora B, izglasaće se novi Leader, u ovom slučaju čvor C:



Iako je čvor B otkazao, mi idalje možemo pristupiti njegovim podacima zato što je prilikom kreiranja baze podataka **Equipment** na čvoru B podešen replikacioni faktor na 2, pa su svi podaci smešteni na još jednom čvoru. U ovom slučaju je to čvor A. Osim dobavljanja podataka Equipment baze podataka, moguće je i kreirati novi dokument na čvoru A:

**{**

**"Naziv": "Equipment2",**

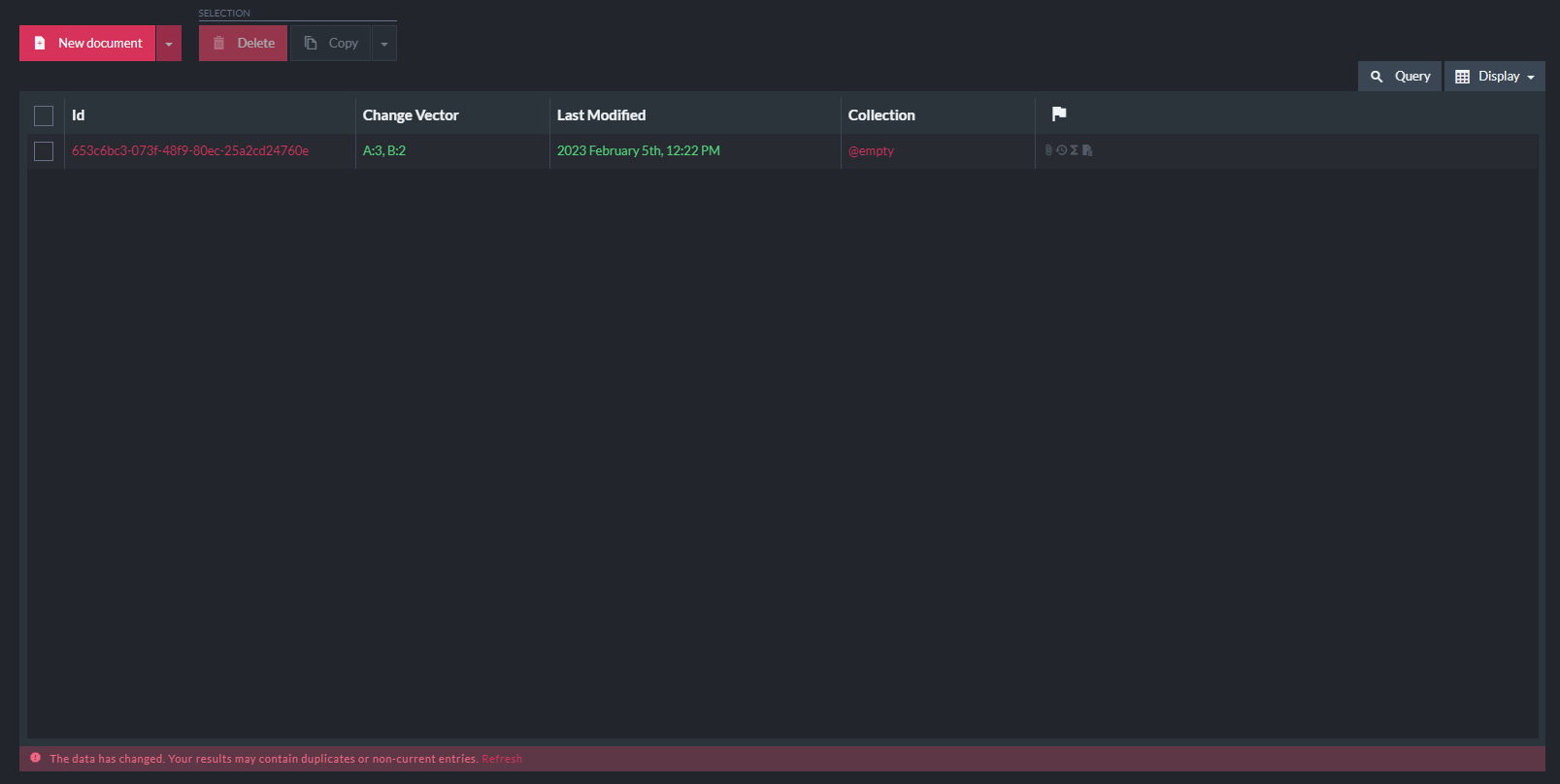
**"Vlasnik": "Dusan"**

**}**

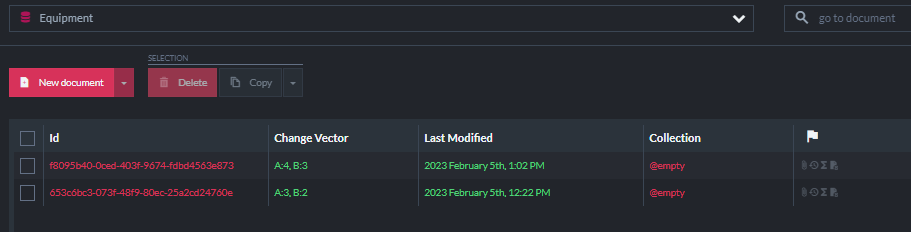
Dakle svaka instanca baze podataka sarađuje sa ostalim instancama, ali ne zavisi od njih. Kada se ovo demonstriralo, moguće je ponovo pokrenuti čvor B istom komandom kojom je i inicijalno pokrenut:

**Raven.Server.exe --ServerUrl=http://127.0.0.3:8081 --Logs.Path=Logs/B --DataDir=Data/B**

Nakon ovoga cluster može ponovo proglasiti čvor B za Leader-a, ali nije pravilo da mora. Inicijalni pregled podataka na čvoru B ne prikazuje malopre kreirani dokument Equipment2:



Međutim, nakon nekog vremena se pojavljuje push notifikacija u donjem delu ekrana kao što je moguće videti na slici iznad. Notifikacija informiše o tome da su se podaci promenili i da to može značiti da zapravo postoje neki duplikati, ili, u našem slučaju da se ne prikazuju svi entry-i baze podataka. Nakon refresh-ovanja stranice, dakle slanja upita za pribavljanje svih dokumenata baze podataka, može se videti prisutnost dokumenta na čvoru B koji je kreiran na čvoru A dok je čvor B bio ugašen. Odrađena je replikacija podataka:



Zaključak iz svega ovoga je da nam je RavenDB cluster suzbio **single point of failure** svojom replikacijom.

# Zaključak

Centralna tema ovog rada su cluster rešenja kod RavenDB baze podataka. RavenDB i klijentska aplikacija Raven Studio olakšavaju korišćenje cluster-a. Na početku ovog rada su opisani najosnovniji koncepti oko cluster rešenja kod RavenDB baze podataka. Posmatrana je CAP teorema kod RavenDB baze podataka i objašnjeno je kako ova baza podataka zadovoljava i CP i AP, ali na različitim nivoima. Na nivou cluster-a je RavenDB baza podataka CP, odnosno konzistentna i otporna na particionisanje. Na nivou baze podataka, RavenDB je AP-otporna na particionisanje i podaci su uvek dostupni, međutim, može se desiti da nisu konzistentni. Nakon objašnjenih osnovnih koncepata, prelazi se na malo složenije, a to je definitivno Rachis protokol. Ova RavenDB implementacija Raft protokola za postizanje konzistentnosti uvodi mnoge nove koncepte. Opisan je “život” jednog čvora od njegovor kreiranja, pasivnog stanja, preko ulaska u cluster, glasanja za leader-a i eventualnog izglasavanja za leader-a. Velika prednost cluster rešenja je mogućnost replikacije podataka, pa je u radu i to obrađeno, kao i problemi, odnosno konflikti koji dolaze sa replikacijom. Nakon toga se dolazi do najbitnijeg dela rada, a to je praktična demonstracija cluster-a kod RavenDB baze podataka. Kreiran je jedan cluster od 3 čvora sa više instanci baza podataka. Detaljno je opisano dodavanje svakog čvora u cluster, konfiguracija cluster-a, kao i svakog čvora. Kreirane su baze podataka sa replikacionim faktorom, a i demonstrirano je kako se RavenDB baza podataka snalazi kada dođe do otkaza nekog čvora.

Cluster poboljšava performanse rada sa bazom podataka, ali definitivno uvodi određenu složenost. Nekada nije potrebno koristiti više od jednog čvora, ali koncept cluster-a kod RavenDB-a je uvek prisutan bez obzira sa koliko čvorova se radi. Dakle, definitivno je potrebno biti upoznat sa osnovnim konceptima rada sa cluster-om kada se radi sa RavenDB bazom podataka.

# Literatura

[1] <https://ravendb.net/learn/inside-ravendb-book/reader/4.0/6-ravendb-clusters>

[2] <https://elfakrs.sharepoint.com/:p:/s/Naprednebazepodataka/EeE6BDLs_UFFn3omDZuh2BoBkH1ydGQcqIklrfPcg39Utw?e=zmOwO0>

[3] <https://ravendb.net/docs/article-page/5.3/csharp/server/clustering/rachis/what-is-rachis>

[4] <https://thesecretlivesofdata.com/raft>

[5] <https://ravendb.net/docs/article-page/5.3/csharp/server/clustering/rachis/what-is-rachis>

[6] <https://ravendb.net/docs/article-page/5.3/csharp/server/clustering/replication/replication>

[7] <https://ravendb.net/docs/article-page/5.3/csharp/studio/database/settings/manage-database-group>

[8] <https://en.wikipedia.org/wiki/Vector_clock>

[9] <https://ravendb.net/docs/article-page/5.3/csharp/server/clustering/replication/change-vector>

[10] <https://ravendb.net/docs/article-page/5.4/csharp/studio/cluster/setting-a-cluster>

[11] <https://ravendb.net/buy>