UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

High availability rešenja kod PostgreSQL baze podataka

-Seminarski rad-

Mentor:
Prof. dr. Aleksandar Stanimirović

Student:
Sena Savić, br. ind. 1570

Niš, 2023.

SAŽETAK

U seminarskom radu na temu „High Availability rešenja kod PostgreSQL baze podataka“, predstavljen je pojam i koncept visoke dostupnosti (eng. high availability) i njen generalni značaj za sisteme baza podataka, ali i važnost uvođenja ovog koncepta kod PostgreSQL baze podataka.

Pored toga, detaljno su opisani neki od mehanizama kojima se visoka dostupnost postiže i garantuje kod PostgreSQL baze podataka, poput replikacije, kreiranja klastera, load balancing-a, ali i failover i hot-standby mehanizama. Svaki od ovih elemenata je opisan i ispraćen primerima podešavanja, kako bi se čitaocima dao uvid i dobra osnova za dalje istraživanje i proučavanje ove tematike, kako bi se resursi i moć PostgreSQL baze podataka iskoristila u svom punom potencijalu.

Visoka dostupnost je izuzetno bitan faktor projektovanja sistema, pa tako i PostgreSQL sistema baze podataka, posebno kada se koristi u poslovnim i korporativnim okruženjima od velikog značaja, jer obezbeđuje poverenje klijenata i korisnika sistema.

SADRŽAJ

1. UVOD

Visoka dostupnost (eng. High availability) je jedan od ključnih aspekata modernih sistema baza podataka, koji obezbeđuje da sistem bude funkcionalan i dostupan korisnicima u svakom trenutku, čak i u slučaju hardverskih kvarova, prirodnih katastrofa ili planiranih održavanja. Koncept visoke dostupnosti je od vitalnog značaja za sve organizacije, koje zahtevaju neprekidnu prisutnost svojih podataka, kao i minimalno vreme prekida usluga.

PostgreSQL, kao veoma popularna open-source relaciona baza podataka, koristi se kao sastavni deo aplikacija namenjenih širokom spektru delatnosti. Iz tog razloga, implementacija i održavanje visoke dostupnosti kod PostgreSQL baze podataka, donosi niz prednosti i benefita, pritom garantujući otpornost sistema na kvarove i njegovo kontinuirano funkcionisanje.

U narednim poglavljima seminarskog rada, biće dat pregled definicije pojma visoke dostupnosti, kao i njenog značaja i osnovnih koncepata, uključujući i razloge za uvođenjem ovog mehanizma kod PostgreSQL baze podataka. Takođe, u radu će biti reči i o arhitekturnom sklopu neophodnom za implementaciju visoke dostupnosti kod PostgreSQL baze podataka, ali i o najčešćim aspektima visoke dostupnosti kod ove baze podataka, poput replikacionih mehanizama i klastera, load balancing metoda, failovera i oporavka, ali i hot standby-a.

2. HIGH AVAILABILITY KONCEPT

Visoka dostupnost (eng. High availability) kod baza podataka se odnosi na dizajn i implementaciju sistema koji obezbeđuje da baza podataka bude neprekidno raspoloživa i pristupačna, sa što je moguće minimalnijim vremenom prekida pružanja usluga i gubitkom podataka. Cilj visoke dostupnosti je omogućiti aplikacijama i korisnicima da kontinuirano pristupaju bazi podataka i obavljaju željene operacije i aktivnosti, čak i u slučaju neočekivanih problema, poput hardverskih i softverskih kvarova ili greški, prirodnih katastrofi ili planiranih održavanja.

U ovom delu seminarskog rada, data je definicija pojma visoke dostupnosti, ali i pregled njene važnosti i značaja za sam sistem baza podataka, sa posebnim osvrtom na PostgreSQL bazu podataka i arhitekturu visoke dostupnosti koju ova baza zahteva.

2.1 Definicija i značaj pojma high availability u kontekstu baza podataka

„ High availability ili visoka dostupnost, je sposobnost sistema da radi neprekidno u određenom vremenskom periodu, bez otkazivanja“. High availability, se takođe temelji i na tome da treba da osigura da sistem ispuni dogovoreni nivo operativnih performansi, što se iz ugla IT sektora, predstavlja kao standard „pet devetki“ i to znači da sistem treba da bude dostupan 99,999% vremena[1].

Međutim, princip skoro totalne kontinualnosti, je teško održiv kod izuzetno kompleksnih sistema, pa se iz tog razloga teži standardu „petdevetki“, korišćenjem raznih tehnika i metoda, pogotovu ukoliko se govori o dostupnosti sistema baza podataka, o čemu će biti više reči kasnije u radu.

Posmatrano iz ugla sistema baza podataka, pojam visoke dostupnosti oslikava bazu podataka, koja je dizajnirana da radi čak i ukoliko dođe do neke vrste problema, vrlo često premašujući i ono što je inicijalno predviđeno ugovorom o nivou usluge određene baze [2].

Iz tog razloga, značaj high availability koncepta u kontekstu baza podataka je veliki, jer obezbeđuje pouzdan prelaz između redundantnih sistema i pravovemeno otkrivanje kvarova, koristeći tipične funkcionalnosti baza podataka visoke dostupnosti, poput replikacionih mehanizama, balansiranja opterećenja, ali i korišćenja distribuirane mikroservisne arhitekture [2]. Pored svega navedenog, važno je naglasiti da se neophodnost high availability koncepta baza podataka, danas ogleda i u smanjenju rizika od gubitaka prihoda ili klijenata, s obzirom da se sistemi baza podataka sve češće koriste od strane organizacija koje koriste velike količine poverljivih podataka [2].

2.2 Važnost uvođenja high availability koncepta kod PostgreSQL baze podataka

Kako je već ranije ustanovljeno, ne može se očekivati da ni jedan sistem, uključujući i sistem baze podataka, bude dostupan apsolutno uvek i u svakom trenutku, pa se iz tog razloga uvode neka osnovna načela koja se posmatraju prilikom dizajna visoke dostupnosti baze podataka, uključujući i obezbeđenje ovog postulata kod PostgreSQL baze podataka.

Prilikom formiranja mehanizma visoke dostupnosti kod sistema baze podataka, vodi se računa o sledećim elementima [1]:

- Eliminaciji single point of failure-a - Vrlo je važno da prilikom garancije visoke dostupnosti sistema, pažnja bude usmerena i na prevazilaženje ovog problema. Tačnije, ukoliko je sistem projektovan tako da postoji samo jedna serverska instanca, i ukoliko ona otkaže (eng. single point of failure), ceo sistem postaje nedostupan, što nije u skladu sa principom visoke dostupnosti, tako da treba izbeći formiranje ovakve serverske arhitekture;

- Formiranju pouzdanog ukrštanja – Ovaj metod, zapravo predstavlja postizanje redundantnosti sistema, odnosno omogućavanje rezervnoj kopiji komponente sistema da preuzme funkcionalnosti neispravne komponente. U ovom procesu je važno obezbediti pouzdano ukrštanje (eng. reliable crossover), tačnije prelazak sa jedne na drugu komponentu bez gubitaka ili uticaja na performanse sistema.
- Detekciji kvara – Sve greške koje se dogode na sistemu, moraju biti vidljive, kako bi ih bilo moguće otkloniti. To bi značilo, da u idealnom slučaju sistemi imaju ugrađene mehanizme za automatsko razrešenje kvarova ili izbegavanje istih, međutim kako nije uvek moguće predvideti ponašanje sistema, korišćenjem odgovarajućih aspekata visoke dostupnosti, mora se obezbediti neometano funkcionisanje sistema sa kvarom, ali i pravovremena detekcija istog.

Pomenuti elementi koji učestvuju u osiguranju visoke bezbednosti (eng. high availability), razmatraju se i pri postizanju ovog koncepta kod PostgreSQL baze podataka i obezbeđuju se raznim tehnikama poput balansiranja opterećenja, replikacije i kreiranja klastera i mnogih drugih, trudeći se da opravdaju važnost uvođenja visoke dostupnosti kod PostgreSQL baze podataka.

Iz tog razloga, pojam visoke dostupnosti, kod PostgreSQL baze podataka, bitan je iz nekoliko osnovnih razloga:

- Neprekidnosti u poslovanju – visoka dostupnost obezbeđuje neprekidno funkcionisanje PostgreSQL baze podataka, čak i u slučaju neočekivanih prekida ili kvarova, što znači da aplikacije povezane sa ovom bazom mogu da nastave svoj rad, bez da korisnici osete postojanje bilo kakvog problema. Održivost u poslovanju je ključna za organizacije koje direktno zavise od baza podataka za izvršenje operacija, posebno u sektorima finansija ili e-trgovine, komunikacija, itd...
- Minimiziranja gubitaka podataka – korišćenjem replikacipnih mehanizama, kojima se postiže kreiranje kopija podataka na više čvorova, gubitak podataka se znatno smanjuje ukoliko otkaže neki od čvorova, jer kopija podataka svakako postoji na drugom. Samim tim, integritet podataka je garantovan, čak i u nepredvidivim situacijama;
- Otpornost na kvarove – PostgreSQL baza podataka, može biti izložena raznim vrstama kvarova, a visokom dostupnošću obezbeđuje se otpornost i vrši automatsko preusmerenje rada ka ostalim čvorovima sistema;
- Skalabilnost i performanse – kada postoji visoko opterećenje sistema, ili veliki broj korisnika, visoka dostupnost kroz replikaciju i balansiranje opterećenja, distribuira zahteve korisnika ka većem broju čvorova, postižući na taj način bolju propustnost i performanse sistema;
- Formiranje korisničkog iskustva – visoka dostupnost ima jako veliki uticaj na formiranje korisničkog iskustva, jer ukoliko korisnici mogu neometano da pristupaju informacijama, da vrše transakcije i ostvaruju svoje poslovne ciljeve, zadovoljstvo samim sistemom je veće, a lojalnost korisnika neupitna.

2.3 Arhitektura visoke dostupnosti kod PostgreSQL baze podataka

Postoji nekoliko različitih tipova arhitekture za implementaciju visoke dostupnosti kod PostgreSQL baze podataka, ali najosnovniji su Primary-Standby i Primary-Primary tip arhitekture.

Primary-Standby (Master-Slave) predstavlja najosnovniji tip arhitekture visoke dostupnosti kod PostgreSQL baze podataka, pa je zbog toga i najlakši za implementaciju i održavanje.

Zasniva se na postojanju, jedne primarne baze sa jednim ili više standby servera, koji se nalaze u stanju pripravnosti i koji su u većoj ili manjoj meri u sinhronizaciji sa glavnim čvorom (ukoliko se radi o sinhronoj ili asinhronoj replikaciji). Ukoliko primarni server prekine sa radom, standby server preuzeće ulogu primarnog servera, s obzirom da svakako sadrži većinu podataka originalnog primarnog servera [3]. Kod ovog arhitekturnog tipa, Primary čvor je odgovoran za upis podataka, dok se upiti za čitanjem mogu obavljati i na njemu, ali i na standby čvorovima.
U zavisnosti od vrste replikacije, standby serveri mogu biti:

- Logički standby serveri – replikacija između primara i standby-a se vrši SQL naredbama;
- Fizički standby serveri – replikacija između primara i standby-a se vrši izmenom interne strukture podataka.

Kod PostgreSQL baze podataka, sinhronizacija standby-a, postiže se korišćenjem write-ahead log-a (eng. WAL) i može biti sinhrona ili asinhrona. Od PostgreSQL verzije 10, mehanizam logiče replikacije i write-ahead protokol je podrazumevana i ugrađena opcija, pa ovaj tip arhitekture omogućava replikaciju pijedinačnih tabela, bez da se odredi primarni server.
Međutim, korišćenjem ove arhitekture, visoka dostupnost se ne koristi na najefikasniji način, jer Primary-Standby ne uključuje automatski način razrešenja nastalih kvarova, zato što PostgreSQL inicijalno ne pruža softverski sistem za identifikaciju pada primary čvora i notifikaciju standby servera o tome. Ukoliko dođe do kvara ili pada standby čvora,koji se ne može restartovati i oporaviti, neophodno je izvršiti reorganizaciju primarne arhitekture, uvođenjem još jednog standby čvora, kako bi se održao high availability princip. Ako je čvor koji je otkazao, primary (master čvor), zbog ranije opisanog načina funkcionisanja ovog tipa arhitekture, standby čvor će preuzeti ulogu glavnog čvora. Ali u tom slučaju se nameće pitanje, šta ako se primary čvor oporavi?Tada je neophodno postojanje nekog eksternog mehanizma da obavesti stari glavni čvor da više ne poseduje tu ulogu, u suprotnom može doći do zabune u sistemu i ultimativnog gubitka podataka.
Iz tog razloga, potrebno je iskoristiti drugi arhitekturni tip, Primary-Primary arhitekturu.

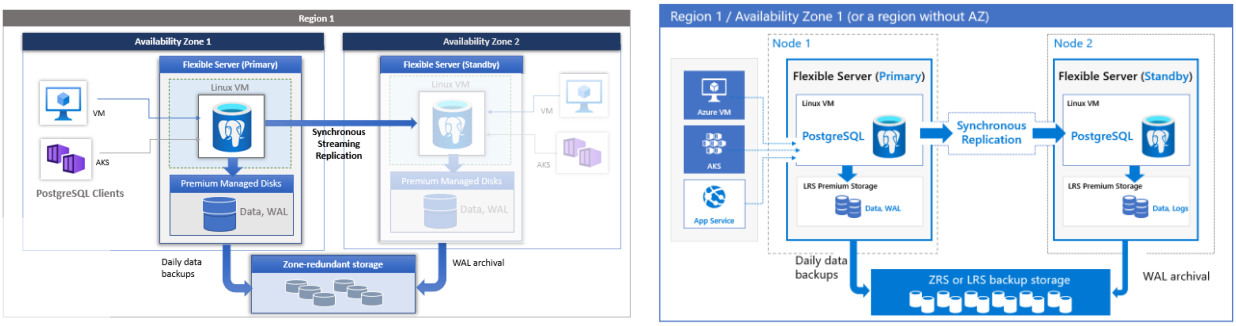
Primary-Primary (Multi-Master) arhitektura pruža načine smanjenja uticaja nastale greške na jednom čvoru, jer su ostali čvorovi u stanju da preuzmu kompletan saobraćaj zato što postoji više primary čvorova, što možda potencijalno može uticati na slabljenje performansi, ali nikako na gubljenje funkcionalnosti sistema.
Karakteristika ovog arhitekturnog tipa je upravo činjenica da postoji više glavnih čvorova, koji imaju svoje setove podataka i samostalno obavljaju upis i čitanje istih, aki međusobno replikuju podatke kako bi ostali u sinhronizitetu. Pored toga, upiti za čitanjem se mogu izvršavati na bilo kom glavnom čvoru,a u slučaju otkaza jednog od glavnih čvorova, preostali primary čvorovi nastavljaju sa prihvatanjem upisa podataka.
Ova vrsta arhitekture visoke dostupnosti, koristi se sa horizontalnim skaliranjem, ali kako PostgreSQL baza podataka još uvek nema „native“ opcije za podržavanje ove arhitekture, neophodno je korišćenje tzv. „third-party“ alata i implementacionih rešenja, kako bi se postigla ovakva vrtsa arhitekture visoke dostupnosti [3].

Glavne razlike između ova dva arhitekturna tipa visoke dostupnosti su asimetričnost Primary-Standby arhitekture, zbog postojanja samo jednog glavnog čvora, zatim način razrešenja pada glavnog čvora, ali i veća propusnost Primary-Primary sistema, zbog pristnosti više ravnopravnih čvorova.

Pregledom ova dva osnovna arhitekturna modela, jasno se može videti da je ključni faktor njihovog razlikovanja, mogućnost automatskog prevazilaženja greške. Iako PostgreSQL ne poseduje softverski mehanizam koji će upravljati razrešenjem ovakve pojave, to ne znači da ovaj

proces nema automatski način identifikacije kvara. Azure Database za PostgreSQL bazu podataka, poseduje tzv. „Fleksibilni server“ koji nudi konfiguraciju visoke dostupnosti sa mogućnošću automatskog razrešavanja greški. Rešenje visoke dostupnosti kod ovog sistema, dizajnirano je tako da obezbedi da se komitovani podaci ne izgube nikada, bez obzira na kvar ili pad sistema, zahvaljujući automatskom principu pripravnosti kojim se write-ahead log strimuje replici u sihnronom režimu, korišćenjem klasičnog PostgreSQL streaming replikacionog postupka (više detalja o replikacionim mehanizmima, dato je u narednom poglavlju). Zahvaljujući ovakvoj strukturi fleksibilnog servera, postoje još dva arhitekturna modela [4]:

- Tip redundantnih zona HA – obezbeđuje potpunu izolaciju i redundandnost infrastrukture u više zona dostupnosti unutar regiona. Pruža najviši nivo dostupnosti, ali zahteva konfiguraciju redundantnosti aplikacija u svim postojećim zonama. Visoka dostupnost sa redundantnim zonama je poželjna kada želite žaštitu od grešaka unutar same zone dostupnosti. Zone se mogu izabrati prema vašim specifičnim potrebama i u okviru njih primarni i standby serveri, ali ono što je važno jeste da se primarni i sekundarni server moraju naći u okviru istog regiona. Datoteke podataka i evidencije transkacija, čuvaju se u lokalnom, redundantnom skladištu, unutar svake zone, koja automatski čuva i po tri kopije podataka, čime je obezbeđena i fizička izolacija primarnog i sekundarnog servera.
- Tip istih zona HA – primarni i sekundarni server se nalaze u okviru istog regiona, ali i iste zone visoke dostupnosti, čime se obezbeđuje redundantnost sistema sa manjim mrežnim kašnjenjem. Visoka dostupnost je obezbeđena i ne zahteva eksplicitno definisanje redundatnosti aplikacija u različitim zonama. Ovaj arhitekturni model visoke dostupnosti, omogućava fleksibilnom serveru da kontinuirano održava high availability koncept, iako se nalazi u istoj zoni kao i standby server. Za razliku od modela redundantnih zona, kod ove vrste se fizičko odvajanje primarnog i sekundarnog servera baze podataka, zasniva na fizičkom odvajanju steka između ovih vrsti servera, zahvaljujući postojanju lokalnog skladišta koje automatskiskladišti tri sinhrono kreirane kopije unutar iste zone u kojoj su primarni i sekundarni server. Datoteke sa podacima i evidencijom transakcija se takođe nalaze u okviru iste zone dostupnosti, unutar lokalno-redundantnog skladišta. Formiranje kopija sa primarnog servera se vrši automatski i periodično i kontinuirano se pamte u skladištu rezervnih kopija, odakle ih sekundarni server može preuzeti u slučaju kvara primara.



Slika 1- Arhitekturni tipovi visoke dostupnosti Azure Database Fleksibilnog servera za PostgreSQL bazu podataka[1]

[1] Izvor slike 1- https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/concepts-high-availability

3. REPLIKACIJA KAO MEHANIZAM OBEZBEĐENJA HIGH AVAILABILITY KOD POSTGRESQL BAZE PODATAKA

Visoka dostupnost (eng. High Availability), kod baza podataka predstavlja postulat dizajna i implementacije sistema koji omogućava neprekidnu prisutnost baze podataka, sa minimalnim vremenom prekida rada ili gubitaka podataka. Jedan od ključnih aspekata, kojima se ovaj princip postiže, jeste mehanizam replikacije podataka.

Replikacija podataka kod PostgreSQL baze podataka podrazumeva postupak kopiranja podataka sa jednog PostgreSQL servera (ovakav server se naziva primarni ili master server) na drugi PostgreSQL server (ovaj server je poznat kao replika ili standby)[5].

Prilikom formiranja visoke dostupnosti PostgreSQL baze podataka, rezervni serveri se konfigurišu tako da budu u stanju da brzo preuzmu kontrolu kada glavni server otkaže. Samim tim, da bi se postigla dostupnost sistema, on mora da ispuni neke ključne zahteve, kao što su redundantnost, mehanizmi bezbednog prelaska sa jednog čvora na drugi i aktivno nadgledanje sistema radi identifikacije mogućih kvarova. Korišćenje replikacije i njeno podešavanje za prevazilaženje greški, utiče na obezbeđenje potrebne redundantnosti i pripravnosti replika, ukoliko primarni server nekada otkaže, modelujući na taj način princip visoke dostupnosti PostgreSQL sistema. Pored toga, replikacija može imati još dosta razloga za upotrebu, jer utiče na [5]:

- Uklanjanje opterećenja prilikom izvršenja upita, čime se ubrzava izvršenje istih i poboljšavaju performanse obrade transakcija;
- Toleranciju grešaka, zato što iako jedan server otkaže, drugi će preuzeti njegovu ulogu i iz ugla korisnika ovaj proces neće biti vidljiv, zato što replika već sadrži podatke primarnog servera;
- Migraciju podataka i paralelno testiranje sistema koje podrazumeva upoređivanje prenešenih podataka kako bi se procenilo i osiguralo da sistem radi onako kako se od njega to očekuje.

Modeli replikacije kod PostgreSQL baze podataka, odgovaraju osnovnim arhitekturnim tipovima za postizanje visoke dostupnosti, odnosno odgovaraju Primary-Standby i Primary-Primary arhitekturi i poznati su pod imenima Single-Master i Multi-Master replikacioni modeli[5]. Kod Single-Master replikacionog modela, promene učinjene nad tabelom master čvora, prenose se jednom ili većem broju replika servera, pri čemu replike promene mogu da prihvate samo od master čvora. Ukoliko i dođe do promene unutar replika, one nisu vidljive masteru, tj. ne repliciraju se nazad. Multi-Master replikacioni model podrazumeva postojanje većeg broja čvorova koji imaju jednaku ulogu glavnog čvora. Podaci se repliciraju između čvorova, ali operacije ažuriranja i dodavanja se izvršavaju samo na glavnim čvorovima. Ovaj tip replikacije podrazumeva sposobnost sistema da uspešno razreši potencijalne sukobe čvorova, prilikom istovremenih izmena podataka na njima.

Slika 2- Modeli PostgreSQL replikacije, Single-Master tip (levo) i Multi-Master tip (desno)[2]

[2] Izvor slike 2- PostgreSQL Replication and Automatic Failover Tutorial | EDB (enterprisedb.com)

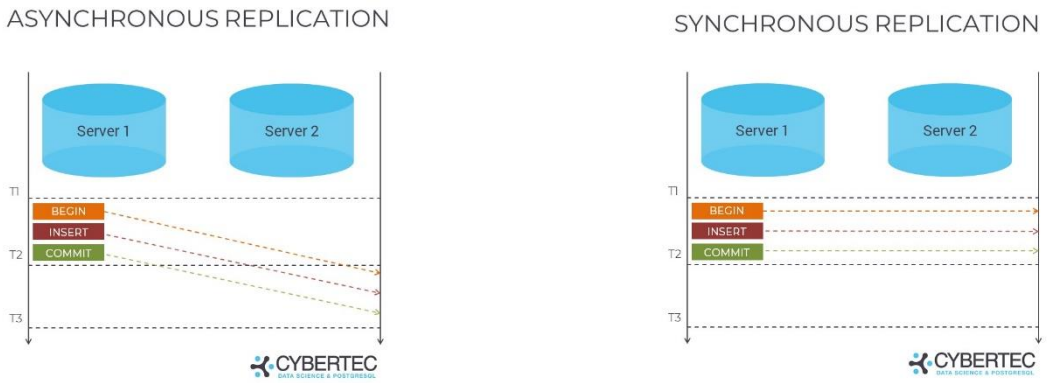
3.1 Sinhrona i asinhrona replikacija kod PostgreSQL baze podataka

Replikacija je jedan od načina postizanja visoke dostupnosti sistema baze podataka i kako bi se na pravilan način razumeo i usvojio koncept replikacionih mehanizama i vrsti, neophodno je shvatiti na koji način funkcionišu replikacioni režimi rada.

PostgreSQL baza podataka podržava dva replikaciona režima- sinhroni i asinhroni režim. Kod sinhronog režima replikacije, transakcije koje se obavljaju na glavnom čvoru se proglašavaju završenima, samo kada sve promene budu replicirane na sve sekundarne čvorove, koji moraju biti dostupni za sve vreme trajanja transakcije. U asinhronom režimu, transakcija se proglašava završenom, onda kada se uspešno izvrši na masteru,a promene se kasnije repliciraju sekundarnim serverima. Zbog ovakvog pristupa, serveri replike neko vreme mogu ostati nesinhronizovani i ova pojava je poznata kao tzv. „ kašnjenje replikacije“ [5].

Asinhroni režim replikacije je standardni način replikacije kod PostgreSQL baze podataka, jer nudi lak način distribucije podataka, što povećava dostupnost sistema. Pored toga, troškovi formiranja ovakvog replikacionog moda rada su niski, pa je asinhrona replikacija idealno rešenje za primenu kod automatskog failover-a. Ukoliko se koristi asinhrona replikacija, podaci se mogu naći u stanju prirpavnosti tek nakon što je transakcija zapamćena na primarnom serveru. Pored toga, pomenuto je da kod ovog replikacionog režima postoji i malo kašnjenje, pa u slučaju pada servera, može doći do gubitka nekog dela podataka [6].

Sinhrona replikacija je režim koji se koristi onda kada postoji veliki rizik gubitka makar i jednog commit-a, pri čemu se takav ishod može negativno odraziti na dostupnost celog sistema. PostgreSQL pruža mogućnost sinhrone replikacije podataka onolikom broju standby-a, koliko je potrebno da bi se potvrdila validnost commit-a (commit je u ovom slučaju validan jedino ako ga potvrdi određeni broj servera). Iz ovog razloga, sihnronom replikacijom se osigurava najveća moguća bezbednost transakcija, jer ukoliko jedan server i otkaže, na ostalima će biti prisutni svi validni podaci,pa do njihovog nestanka neće doći [6].



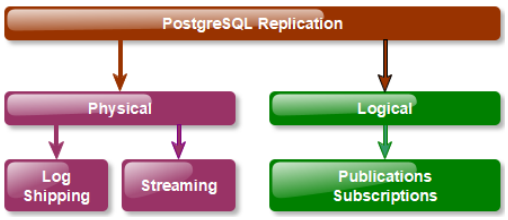
Slika 3- Sinhrona i asinhrona replikacija kod PostgreSQL baze podataka [3]

[3] Izvor slike 3- Synchronous and asynchronous replication in PostgreSQL | CYBERTEC (cybertec-postgresql.com)

3.2 Vrste replikacije i kreiranje klastera

U prethodnim delovima seminarskog rada, rečeno je da replikacija ima veoma važnu ulogu u održanju visoke dostupnosti, jer predstavlja proces kreiranja i održavanja kopija između primarnog čvora i sekundarnih čvorova, čime obezbeđuje pomenuti koncept pouzdanosti sistema. Ukoliko primarni čvor ne uspe da obavi neku od funkcija ili zakaže prilikom rada, sekundarni čvor preuzima ulogu primarnog i nastavlja sa pružanjem usluge bez prekida. Replikacija se vrlo često koristi i u kontekstu stvaranja redundantnih kopija podataka i obezbeđivanja otpornosti na kvar. Iz tog razloga važno je razumeti na koji način funkcioniše sam mehanizam replikacije, odnosno koje vrste replikacije postoje i na koji način se primenjuju kako bi se obezbedio princip visoke dostupnosti kod PostgreSQL baze podataka.

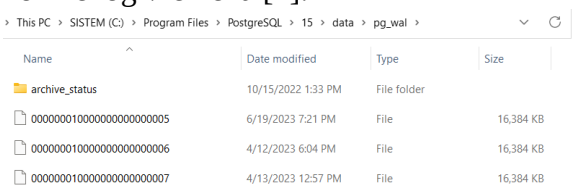
Dve osnovne vrste replikacije kod PostgreSQL baze podataka su fizička i logička replikacija. Fizička replikacija se bavi datotekama i direktorijumima, odnosno vrši se na nivou sistema datoteka ili diska i može biti omogućena kroz log shipping replikaciju ili streaming replikaciju. Sa druge strane, logička replikacija se bavi bazom podataka, odnosno tabelama i ostalim elementima koji postoje u bazi, kao i operacijama koje se nad njima izvršavaju. Iz tog razloga, kod logičke replikacije je moguće replicirati samo određeni skup tabela, pa se ova vrsta replikacije vrši na nivou klastera baze podataka [5]. Logička replikacija, uglavnom koristi publisher/subscriber model za ostvarenje replikacionog mehanizma.



Slika 4- Vrste replikacije kod PostgreSQL baze podataka[4]

3.2.1 Fizička replikacija

Metode fizičke replikacije se korste kako bi se postigla potpuna dostupnost PostgreSQL sistema, jer formiraju detaljne kopije svih podataka posmatranog sistema.
U toku rada, PostgreSQL server generiše uređenu seriju WAL (eng. Write Ahead Log) zapisa, koji je u osnovi dnevnik svih promena, vrli sličan Redisovom AOF-u ili MySQL-ovom binlogu. Suštinski, fizička replikacija je trasportovanje ovih zapisa sa jedne mašine na drugu i navođenje druge mašine da privati i primeni ove zapise u svoju lokalnu bazu podataka. WAL zapisi su podeljeni u datoteke jednake veličine, obično od po 16MB i one se drugačije nazivaju WAL segmentima ili WAL datotekama. Kreiraju se u okviru direktorijuma pg_wal u okviru direktorijuma sa podacima PostgreSQL baze podataka i verzije ovih datoteka koje više nisu neophodne se odbacuju nakon nekog vremena [7].



Slika 5- Prikaz pg_wal direktorijuma i WAL datoteka

[4] Izvor slike 4- Streaming replication with PostgreSQL 9.6, 10 & 11 - PostgreSQL Standby Databases (sqlpac.com)

WAL datoteke su neophodne prilikom obezbeđenja visoke dostupnosti putem fizičke replikacije i neizostavni su deo replikacionog procesa, bez obzira na vrstu fizičke replikacije. Kod PostgreSQL baze podataka, fizička replikacija može se ostvariti kroz log shipping ili korišćenjem streaming replikacije [7].

Ukoliko postoji triger koji će se pozivati na primarnom serveru uvek kada se kreira nova WAL datoteka, koji potom sihnronom replikacijom može kopirati WAL datoteku na drugi server i staviti je da radi u režimu oporavka, postavlja se pitanje da li je moguće izvršiti konfiguraciju standby servera, koji će moći da privati ovako kreiranu datoteku. Odgovor je potvrdan i ovo je bila praksa korišćenja fizičke replikacije za obezbeđenje visoke dostupnosti pre PostgreSQL verzije 9. Ovaj pristup se naziva log shipping i podešava se korišćenjem shell skripti i archive_command fajla. Log shipping je zapravo direktno premeštanje WAL zapisa sa jednog servera baze podataka na drugi i PostgreSQL implementira ovaj mehanizam tako što prenosi jednu po jednu datoteku. WAL datoteke se vrlo lako šalju na bilo koju udaljenost, bez obzira da li se radi o sistemu na istoj lokaciji ili sistemu na drugom kraju sveta, dok propusni opseg potreban za ovu tehniku varira u zavisnosti od brzine transakcije primarnog servera.

Kako govorimo o postizanju visoke dostupnosti kod PostgreSQL baze podataka korišćenjem log shipping fizičke replikacije, obično je pametno kreirati primarni server i više standby servera i to tako da budu što sličniji , barem iz perspektive servera baze podataka. U ovom slučaju hardver ne mora biti potpuno isti, ali je praksa pokazala da je ipak bolje održavati dva ista ili prilično slična sistema, tokom životnog veka aplikacije koja ih koristi. Međutim, arhitektura hardvera mora biti identična, jer nije moguće vršiti slanje log datoteka sa 64-bitnog na 32-bitni sistem. Takođe, slanje WAL datoteka nije moguće na različitim verzijama PostgreSQL servera, jer PostgreSQL politika ne prihvata izmene u formatima diskova tokom update-ova verzija i ne nudi formalnu podršku za pokretanje različitih izdanja (u realnosti je moguće da viša verzija prihvata rad nižih verzija), tako da je generalni savet da ipak i master server i sve kopije pripadaju istoj verziji PostgreSQL baze.

Kako bi replikacija log shippingom bila moguća i time obezbeđena visoka dostupnost, potrebno je podesiti i primarni server, ali i standby servere. Podešavanje primarnog servera podrazumeva da je uključeno kontinuirano arhiviranje log datoteka, od lokacije primarnog servera pa do lokacije na kojoj se nalazi direktorijum arhive, dostupan i standby serverima i u trenutku pripravnosti celokopnog sistema. Pristup direktorijuma arhive, treba da bude omogućen čak i kada primarni server ne radi, kako bi princip high availability-ja bio zadovoljen. Iz tog razloga, prilikom konfiguracija standby servera, zahteva preuzimanje rezervnih kopija podataka sa primara i kreiranje standby.signal datoteke u okviru direktorijuma podataka u stanju pripravnosti. Parametar restore_command treba da bude u ovom slučaju postavljen tako da bude sinhronizovan sa jednostavnom komandom za kopiranje datoteka iz WAL arhive, a kako se vodi računa o visokoj dostupnosti, neophodno je parametar recovery_target_timeline podesiti na vrednost „latest“, kako bi server bio u mogućnosti da prati promenu vremenske linije i da detektuje prelazak na drugi standby server. Takođe, s obzirom da se u radu govori o visokoj dostupnosti, neophodno je pomenuti da svi rezervni serveri ostale parametre WAL arhive, veze i autentifikaciju, moraju da imaju podešenu na isti način kao primarni server, da bi standby ukoliko primarni server otkaže, uspešno nastavio da radi kao prethodni primarni server. Kod ove vrste fizičke replikacije, često se nepotrebne WAL datoteke uklanjaju, ali iz ugla visoke dostupnosti, poželjno je zadržati ih (makar datoteke nastale kopiranjem najnovije verzije baze), kako bi proces oporavka bio efikasno omogućen,a princip dostupnosti nenarušen [8].

Na narednim slikama prikazan je prethodno opisani postupak podešavanja primara i standby-a.

```
# - Archiving -

archive_mode = on
archive_library = ''
archive_command = 'copy "C:\Program Files\PostgreSQL\15\data" "C:\Program Files\PostgreSQL\15\server\arhnew" '
```

Slika 6- Podešavanje kontinualnog arhiviranja na primarnom serveru za log shipping u postgresql.conf fajlu

Na slici je prikazano kontinualno arhiviranje, gde su kao parametri navedeni lokacija servera, ali i lokacija fajla za arhiviranje. Komanda archive_libraray sadrži prazan string, kako bi se ukazalo sistemu da treba da iskoristi archive_command naredbu.

```
primary_conninfo = 'host= 192.168.1.100 port= 5432 user= sena password= studentic options='' -c wal_sender_timeout=5000'''
restore_command='cp "C:\Program Files\PostgreSQL\15\server\arhnew" "C:\Program Files\PostgreSQL\15\data"'
```

Slika 7- Podešavanje standby servera za log shipping

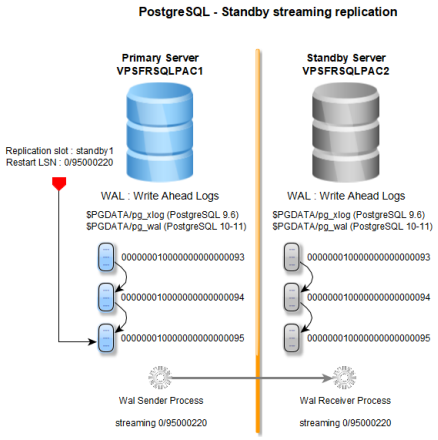
Važna napomena za korišćenje datih komandi je da se prva komanda ignoriše od strane primatnog servera, a drugu komandu je moguće postaviti i kod primara i kod sekundarnih servera kako bi oporavak sistema bio moguć.

Nakon obrade arhiviranih WAL datoteka i datoteka pristutnih u pg_wal direktorijumu, PostgreSQL može da se poveže sa primarnim serverom preko mreže i više puta preuzima novokreirane WAL datoteke. Ova funkcija je dodata u PostgreSQl tek nakon verzije 9 i naziva se streaming replikacija. Primarni server na koji se vrši povezivanje može se navesti u konfiguracionom fajlu.

```
standby_mode = on;
primary_conninfo = 'host= 192.168.1.100 user= sena password= studentic';
```

Slika 8 – Podešavanje konfiguracionog fajla za streaming replikaciju

Streaming replikacija je u današnje vreme jedna od najčešće korišćenih vrsti fizičke replikacije od strane PostgreSQL baze podataka, pogotovu kada se govori o high availability principu. Ova vrsta replikacije u stanju pripravnosti sistema, serverima omogućava da ostanu ažurniji u odnosu na log shipping princip. Stanje održanja pripravnosti se povezaju se primarnim serverom, koji odmah prenosi WAL zapise dalje ka svojim standby čvorovima, bez čekanja da se WAL datoteka popuni [8]. Streaming replikacija je asinhrona, što znači da postoji malo kašnjenje između urezivanja transakcija u primaru i vidljivosti promena na standby čvorovima. Ipak, ovo kašnjenje je daleko manje nego kašnjenje kod log shipping-a, pa streaming replikacija omogućava brzo i pouzdano kopiranje podataka i pruža sekundarnim čvorovima sinhronizovanost sa primarnim čvorom, što je jedan od elementarnih načela visoke dostupnosti. Ova vrsta replikacije je karatkeristična po tome što za kontinualno slanje WAL logova koristi TCP/IP protokol. Streaming replikacija replicira promene na nivou bajt po bajt, stvrajući identičnu kopiju baze podataka na drugom serveru. WAL zapisi se time direktno premeštaju sa jednog servera na drugi, pri čemu se mogu prenositi ili bajt po bajt ili kao kod log shippinga, korišćenjem kontinuiranog arhiviranja, ali se ne čeka da datoteke budu u potpunosti popunjene da bi replike dobile zapise. Ukoliko se koristi kontinualno arhiviranje, obavezno je omogućiti WAL arhiviranje, pogotovu ako se radi o robusnim sistemima i aplikacijama, koje koriste veliki broj čvorova. U praksi, proces prijemnik, koji se nalazi na standby serveru, ostavaruje konekciju sa primarnim serverom putem TCP/IP konekcije (zato je na slici 8 navedena adresa hosta, tj. primarnog servera). Proces koji se nalazi na primaru, je proces pošiljaoc i on je zadužen za slanje WAL logova kada se detektuje stanje pripravnosti, kako bi se održala dostupnost sistema i korisnici ostali u mogućnosti da isti koriste, bez obzira na nastale promene [9].



Slika 9- Primer rada streaming replikacije kod PostgreSQL baze podataka

U nastavku je dat primer konfigurisanja streaming replikacije i smatra se da trenutno u sistemu postoje dva čvora- primarni i sekundarni. Na primarnom čvoru je neophodno da sve sistemske skripte budu obezbeđene i baza podataka inicjalizovana, kao i da se izvrši onesposobljavanje firewall-a. Firewall pruža dodatnu zaštitu od eksternih konekcija ka serveru baze podataka, ali ukoliko se koristi inicjalni port PostgreSQL baze podataka, 5432, ukoliko je firewall aktivan replika neće moći da pristupi ovom portu, jer firewall blokira navedeni port i replika ne može da uspostavi konekciju sa masterom. Iz tog razloga se firewall isključuje, ali u realnim okruženjima, pored visoke dostupnosti treba voditi računa i o bezbednosti, pa se umesto totalnog onesposobljavanja firewall-a preporučuje striktnija konfiguracija, odnosno dozvoljavanje određenog saobraćaja između mastera i standby severa. Na sekundarnom čvoru je takođe neophodno obezbediti odgovarajuće sistemske skripte.

Kako je firewall isključen u postgresql.conf fajlu treba postaviti parameter listen_addresses tako da prihvata i udaljene konekcije, što se postiže navođenjem '*' elementa. Nakon toga je potrebno kreirati korisnika na serveru i generalna ideja sa ovim je da se izbegne strimovanje logova od strane podrazumevanih PostgreSQL superusera. Kada je primarni server konfigurisan, potrebno je sve podatke preneti na repliku. To se postiže pg_basebackup komandom, koja se povezuje na primarni server i kopira sve podatke sa njega. Tada se navode kreirani user, odredišni direktorijum parametrom –D, i odmah se sekundarni server može pripremiti za streaming replikaciju navođenjem –R parametra. Na ovaj način je formiran replikacioni slot, koji obezbeđuje da i replika ima validne verzije WAL logova i onda kada primarni server odluči da ih reciklira. Na kraju, potrebno je proveriti da li je streaming replikacija uspela, jednostavnim navođenjem komande za prikaz stanja wal pošiljaoca i primaoca. Opisani postupak streaming replikacije, prikazan je na narednim slikama.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (15.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \conninfo
You are connected to database "postgres" as user "postgres" on host "localhost" (address "::1") at port "5432".
postgres=# CREATE USER senica REPLICATION;
CREATE ROLE
```

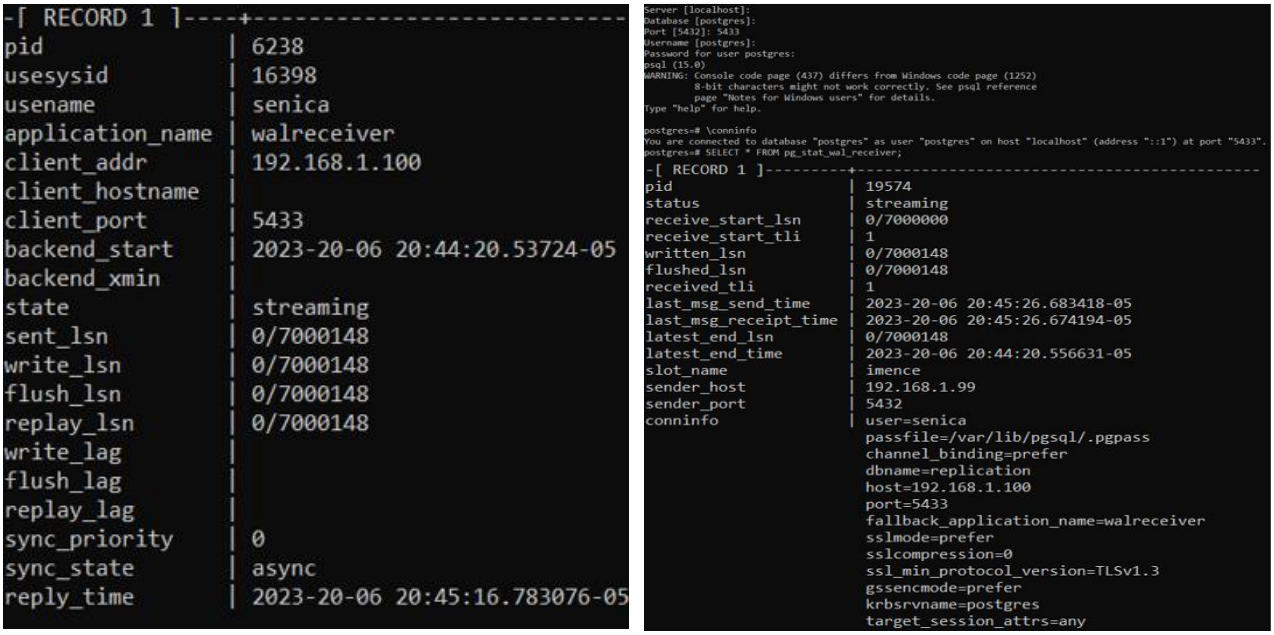
Slika 10 – Kreiranje usera za kontrolu streaming replikacije

```
# Put your actual configuration here
# ----------------------------------
host replication senica 192.168.1.100/30 trust
```

Slika 11- Podešavanje pg_hba.conf fajla primarnog servera da prihvati TCP/IP konekciju sa zadate adrese

```
C:\Program Files\PostgreSQL\15 - 1\bin>pg_basebackup -h 192.168.1.100 -U senica -- checkpoint=fast\ -D "C:\Program Files\PostgreSQL\15 - 1\data" -R --slot=  imence -C
```

Slika 12- Konfigurisanje sekundarnog servera koji predstavlja repliku



Slika 13- Provera streaming replikacije na primarnom i sekunadrnom čvoru PostgreSQL baze podataka

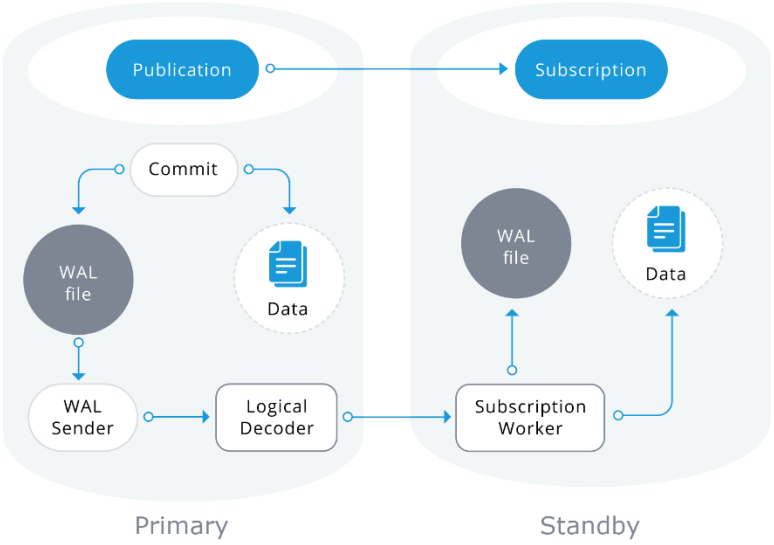
3.2.2 Logička replikacija

PostgreSQL baza podataka je uvela fizičku replikaciju kao jedan od načina postizanja visoke dostupnosti sistema, još od verzije 9.0. Kod fizičke replikacije, promene nastale na glavnom serveru se kroz WAL datoteke prenose i primenjuju na sekundarnim serverima. Međutim, kod fizičke replikacije nije moguće izvršiti selektivnu replikaciju, ili replikaciju samo dela podataka. Takođe, kao što je već opisano u delu rada o fizičkoj replikaciji, ne može se vršiti replikacija između dve različite verzije ili različitih platformi, ali ni sam server ne može dozvoliti upis u stanju pripravnosti.

Iz navedenih razloga, od PostgreSQL verzije 10.0, uvodi se novi pojam replikacionog mehanizma- logička replikacija, koja vodi računa o svim navedenim ograničenjima fizičke replikacije pružajući dosta mesta za razvoj novih tehnika, kako bi visoka dostupnost sistema bila zagarantovana. Suština funkcionisanja logičke replikacije, koja je drugačije poznata i kao transakciona replikacija, je da server koji je standby (ili kako je kod ove vrste nazvan pretplatnik) incijalno prima kopiju repliciranog objekta baze podataka od glavnog servera (kod ove vrste je nazvan izdavač) i potom povlači sve naknadne promene na istom objektu, onako

kako se promene dešavaju u realnom vremenu na čvoru izdavaču, odnsono sve što se promeni na izdavaču, šalje se odmah i pretplatniku [10].

PostgreSQL logička replikacija je vezana za objekte baze podataka, odnosno ona predstavlja tip replikacije koji formira replike objekata i prati njihove promene, uglavnom na osnovu praćenja njihovih jedinstvenih identifikatora (npr. primarni ključ tabele) [9]. Zasniva se na publisher-subscriber modelu, što znači da se u sekundarnom čvoru- subscriberu, kreira pretplata koja se može povezati na više publikacija kreiranih na primaru- publisheru. Replikacija počinje formiranjem kopija publikacija, koje će biti objavljenje čvorovima subsrciberima. Ovaj proces je poznat i kao „faza sinhronizacije“. U ovom trenutku, moguće je pokrenuti više mehanizama za sinhronizaciju objekata koji se repliciraju, kako bi se smanjilo vreme neophodno za ovu operaciju, ali je važno napomenuti da za svaku tabelu može da postoji isključivo jedan takav mehanizam (vrlo često se kod PostgreSQL on naziva sihronizacionim radnikom). Nakon kopiranja, sve promene koje naknadno nastaju na primarnom čvoru, šalju se ostalim čvorovima u realnom vremenu i te promene se urezuju na sekundarnim čvorovima po redosledu kreiranja, što dodatno utiče na poboljšanje konzistentnosti transakcija.



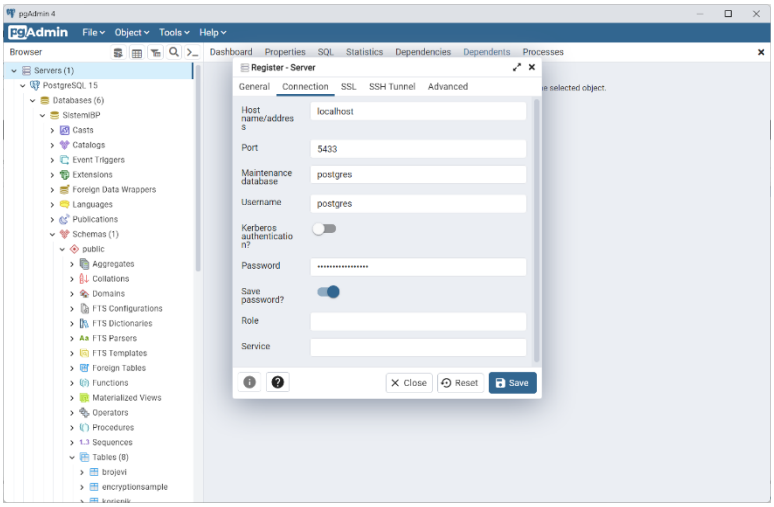
Slika 14- Primer rada ligičke replikacije kod PostgreSQL baze podataka[5]

Primetimo da se logička replikacija oslanja na publisher-subscriber model, ali i na razmenu WAL fajlova, kao što je prikazano na slici. Publikacija je zapravo skup promena generisanih iz tabela baze podataka, ili grupe tabela i čvor u kome je definisan ovakav skup je izdavač. Pretplata je nizvodna strana logičke replikacije i ona je odgovorna za stvranje veze sa čvorom na kome se nalazi publikovan materijal. Logička replikacija po arhitekturi, donekle odgovara fizičkoj streaming replikaciji, što znači da i ona podržava prenošenje wal fajlova. Proces koji sa primarnog servera na sekundarni prenosi logove je wal_sender i on započinje logičko dekodiranje. Logičko dekodiranje podrazumeva transformaciju promena izvršenih nad podacima u format koji odgovara protokolu logičke replikacije, pri čemu se vrši i filtriranje podataka prema specifikaciji kreirane publikacije na primarnom čvoru. Podaci se zatim kontinuirano prenose korišćenjem replikacionog protokola i na standby čvoru se vrši mapiranje podataka u lokalne tabele, pri čemu se vodi računa i o pojedinačnim promenama učinjenim nad tim

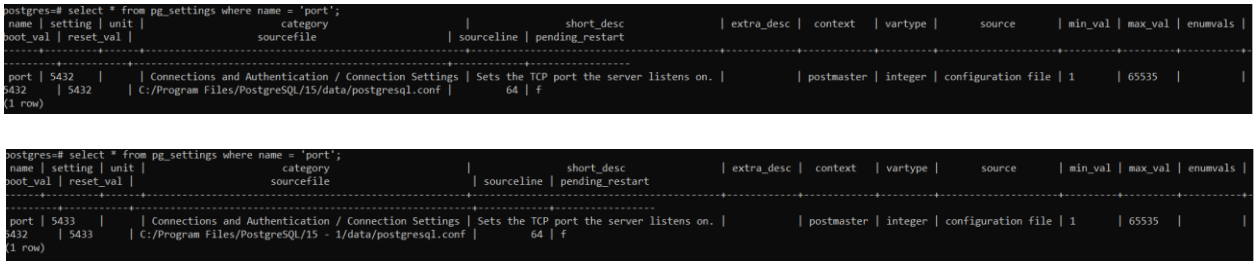
[5] Izvor slike 14 - PostgreSQL Replication Best Practices - Part 1 | Severalnines

podacima, odnosno upisuju se u redosledu u kome su izvršene i to tako da odgovaraju redosledu transakcija [10]. Opisani deo se odnosio na prenos log podataka, ali kako se u radu govori u visokoj dostupnosti, važno je razumeti na koji način logička replikacija omogućava ovaj princip. Proces logičke replikacije započinje pravljenjem snapshot-a podataka u bazi čvora izdavača i kopiranjem istih čvoru pretplatniku. Postojeći podaci na čvorovima pretplatnicima se takođe paralelno kopiraju sa kopiranjem novih podataka sa master čvora, čime se stvara privremeni slot za logičku replikaciju. Kada se postojeći podaci kopiraju, započinje se sa procesom sinhronizacije kako bi se osiguralo da i primar i sekundari imaju ažurne kopije, pri čemu se za strimovanje koristi standardna logička replikacija. Po završetku sinhronizacije, kontrola se vraća primaru i replikacija se nastavlja normalnim tokom, odnosno sve promene učinjene na primaru šalju se i sekundarnim čvorovima i to u realnom vremenu [9].

U nastavku je dat primer logičke replikacije korišćenjem publisher-subscriber metoda. Za potrebe primera, kreirane su dve instance servera na jednoj mašini, pri čemu je jedna instanca izabrana da predstavlja primarni čvor i u slučaju logičke replikacije izdavača, odnosno publisher-a. Druga instanca je konfigurisana tako da predstavlja sekundarni čvor i pretplatnika, tj. subscriber-a. Podrazumeva se da su obavljena sva podešavanja konfiguracionih fajlova kod obe instance, uključujući i podešavanje portova na kojima instance rade (primar radi na podrazumevanom portu 5432, a sekundar na portu 5433), ali i podešavanja neophodna kako bi i server i standby čvor prihvatili logičku replikaciju.



Slika 15- Registracija standby servera u pgAdmin alatu



Slika 16- Primer uspešno formiranih instanci servera PostgreSQL baze podataka

Nakon registracije servera, i ranije pomenutih podešavanja konfiguracionog fajla (podrazumeva se da za svaki čvor treba odraditi podešavanja u zavisnosti da li je čvor master ili slave), može se izvršiti logička replikacija. Za potrebe primera, na primarnom čvoru kreirana je tabela „broj“ i

popunja je odgovarajućim podacima, a zatim je kreirana publikacija na koju će standby server svojom subskripcijom moći da se pretplati.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (15.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# create table broj ( prvi int primary key, drugi int);
CREATE TABLE
postgres=# insert into broj  values (1, 10);
INSERT 0 1
postgres=# create publication prvaPub for table broj;
CREATE PUBLICATION
```

Slika 17- Kreiranje tabele i publikacije na primarnom serveru

Na čvoru koji je određen za kreiranje subskripcije, neophodno je formirati tabelu sa istim imenom kao što ima tabela na primarnom čvoru, da bi replikacija mogla da se uspešno izvrši.

```
Server [localhost]:
Database [postgres]:
Port [5432]: 5433
Username [postgres]:
Password for user postgres:
psql (15.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# create table broj ( prvi int primary key, drugi int);
CREATE TABLE
```

Slika 18- Kreiranje tabele sa istim imenon na standby(sekundarnom) čvoru

Važno je primetiti da sekundarni čvor radi na drugom portu, pa se pri formiranju konekcije kod subskripcije, navodi port čvora na kome se nalazi publikacija.

```
Server [localhost]:
Database [postgres]:
Port [5432]: 5433
Username [postgres]:
Password for user postgres:
psql (15.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# create subscription prvaSub connection 'host=localhost port=5432 dbname=postgres' publication prvaPub;
CREATE SUBSCRIPTION
```

Slika 19- Kreiranje subskripcije za postojeću pretplatu

Provera da li je logička replikacija uspešno izvršena, obavlja se na čvoru pretplatniku. Ukoliko se u tabeli sa istim nazvom kao tabela na izdavaču, nalaze i identični podaci, onda je proces replikacije izvršen pravilno. Iz ugla visoke dostupnosti, ukoliko bi nakon ovakve akcije, primarni čvor prekinuo sa radom, na čvoru pretplatniku podaci bi i dalje postojali i ne bi bili nepovratno uništeni. Ukoliko obe instance i dalje rade, sve što bude dodato na čvoru izdavaču, biće prenešeno na čvor pretplatnik.

```
postgres=# select * from broj;
 prvi | drugi
------+-------
    1 |    10
(1 row)
```

Slika 20- Provera uspešnosti logičke replikacije na sekundarnom serveru

```
postgres=# insert into broj values (2,10);
INSERT 0 1
postgres=# select * from broj;
 prvi | drugi
------+-------
    1 |    10
    2 |    10
(2 rows)
```

```
postgres=# select * from broj;
 prvi | drugi
------+-------
    1 |    10
    2 |    10
(2 rows)
```

Slika 21- Dodavanje novog podatka na publisher-u i provera njegove replikacije na subscriber-u

3.2.3 Kreiranje klastera

Vrlo često kada se govori o praćenju performansi sistema i obezbeđenju visoke dostupnosti istog, većina ljudi razmišlja o inspekciji i održavanju jednog po jednog servera PostgreSQL baze podataka, gde se svaki od njih posmatra kao individualna celina. Međutim, u praksi se vrlo često korsti status klastera baze podataka, koji predstavlja skup povezanih servera (dosad su u radu pominjani i kao čvorovi) koji rade zajedno kako bi obezbedili toleranciju na kvarove i pouzdanost sistema [11]. Iz ugla korisnika, klaster baze podataka izgleda kao jedna instanca, kao jedna celina, koja je sposobna da odgovori na sve zahteve korisnika. Posmatrano sa stanovišta visoke dostupnosti, klasterizacija se vrlo često koristi u kombinaciji sa opisanim vrstama replikacije, kako bi svi čvorovi u klasteru imali ažurirane podatke, kojima korisnici mogu pristupiti sa bilo kog čvora. Ukoliko neki od čvorova otkaže, zahvaljujući prethodno obavčjenoj konfiguraciji, naredni čvor preuzima i njegove funkcionalnosti i sistem nastavlja da radi dalje, onako kako se od njega i očekuje.

Pored navedenih aspekata, kreiranje klastera pruža još nekoliko značajnih aspekata za održavanje visoke dostupnosti (eng. high availability) kod PostgreSQL baze podataka:

- Pružanje kontinuirane prisutnosti baze podataka, čak i u slučaju planiranih ili neplaniranih održavanja, nadogradnji softvera ili fizičkih kvarova na hardveru;
- Garantovanje redundantnosti, koja je rezultat efikasne primene replikacije u klasteru;
- Automatsko preuzimanje je omogućeno kod klastera, jer kako se sam klaster sastoji od nekoliko čvorova između kojih je omogućena replikacija (opisana je ranije u radu), kada neki čvor otkaže, drugi nastavlja dalje. Time se smanjuje potreba za ručnom intervencijom i osigurava se minimalno vreme nedostupnosti baze podataka;
- Skalabilnost je prisutna i to kao horizontalna skalabilnost, čime se postiže bolje upravljanje opterećenjem i omogućava da baza raste zajedno sa porastom potrebe za resursima.

Termin klastera se kod PostgreSQL baze podataka, radije vezuje za klaster baze podataka u smislu skupa više baza na jednom serveru, nego klasičnom skupu više servera ili virtuelnih mašina koje rade zajedno. Klaster se kod PostgreSQL baze podataka sastoji od dva elementa – direktorijuma podataka i postgres procesa. Direktorijum podataka sadrži kolekciju datoteka u kojima se fizički čuvaju svi objekti poput baza podataka, njihovih tabela, indeksa itd. i njima se upravlja preko različitih procesa i alata. Postgres proces je glavni proces koji kontroliše datoteke u direktorijumima i glavni je kada se manipuliše njihovim sadržajem [12].Međutim, iako je

asocijacija na klastere u PostgreSQL vezana za predstavljene elemente, to ne znači da ova baza ne koristi klaster kako bi formirala skup servera koji sinhronizovano rade i za spoljašnji svet deluju kao jedan, naprotiv, klasterizacija se vrlo često koristi i upravo iz razloga postizanja visoke dostupnosti PostgreSQL sistema koji se projektuje.
Klasterizacijom se skup podataka razdvaja u više grupa i to tako da slični podaci čine jednu grupu i potom se dodeljuju serverima koji se nalaze u klasteru. Jednostavan primer koji slikovito objašnjava ovaj proces je prodavnica u kojoj se nalaze različiti proizvodi. U takvoj prodavnici, nemoguće je formirati plan na osnovu želja svakog od kupaca, jer ne možemo unapred znati šta će ko zahtevati. Na isti način, ne može za svakog korisnika da postoji posebna instanca servera ili baza podataka. Međutim, u zavisnosti od proizvoda koje prodavnica poseduje, mogu se formirati grupe i shodno onome šta kupci uzimaju, oni se mogu podeliti u grupe. Na isti način se vrši i raspodela podatka među instancama servera u klasteru i taj proces je poznat kao klasterizacija i može se obaviti na dva načina[12]:

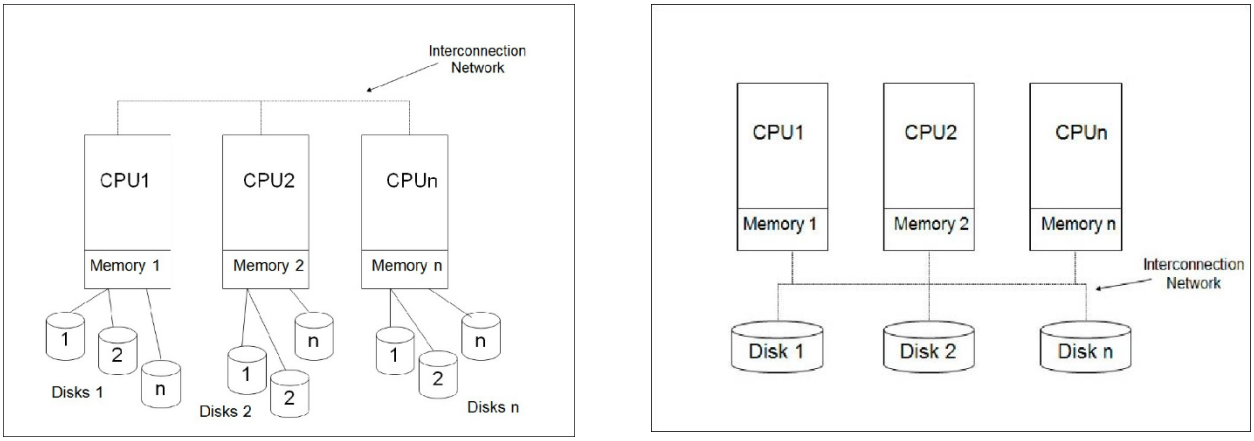
- Čvrsto klasterovanje, gde je svaki skup podataka ili porpuno ili delimično povezan sa instancom servera u klasteru;
- Meko klasterovanje, grupa podataka se može naći na većem broju međusobno povezanih servera.

Najvažniji princip visoke dostupnosti je da sistem nastavi da radi u kontinuitetu i da bude na raspolaganju korisnicima, čak i ukoliko dođe do nekog prekida u radu sistema. Klaster je kolekcija servera koji rade zajedno i predstavljaju se kao jedan entitet, ali zahvaljujući fleksibilnosti moguće je u takvo okruženje dodati i novi server, čime se postiže horizontalna skalabilnost. Ukoliko u ovakvom okruženju, neki od servera zakaže ili dođe do kvara, sistem je sposoban da izdrži takvo opterećenje i kombinujući metod replikacije sa klasterizacijom, ostali čvorovi nastavljaju da rade transparentno iz ugla korisnika. Na taj način je ispoštovan pomenuti princip visoke dostupnosti u potpunosti[12].

Kada se govori o klasterima, da bi se razumeo način njihovog rada, pa samim tim i ostvarivanje visoke dostupnosti korišćenjem istih, potrebno je razmotriti i shvatiti arhitekturne tipove klastera. Postoje dva tipa, Shared-Nothing arhitektura i Shared-Disk.

Kod Shared-Nothing arhitekture klastera svaki server mora biti nezavistan u odnosu na sve ostale čvorove (servere) u sistemu, što znači da u ovakvoj vrsti arhitekture ni jedan čvor nije glavni, odnosno ni jedan server nije centralizovan kako bi kontrolisao i nadgledao pristup podacima u sistemu[13].Podaci se u ovakvoj arhitekturi distribuiraju između čvorova kako bi se postigla visoka dostupnost.Ovakva arhitektura nudi visok stepen skalabilnosti, ali upravljanje podacima i održanje konzistentnosti može biti složenije u odnosu na shared-disk arhitekturu.

Sa druge strane, kod Shared-Disk arhitekture, svi čvorovi imaju pristup svim podacima sistema, zahvaljujući mrežnom sloju interkonekcije između CPU-a i servera baze podataka, koji omogućava serverima višestruki pristup podacima. Podaci se čuvaju na centralizovanom deljenom disku i on je dostupan svim čvorovima, pri čemu svaki čvor ima pristup podacima i može ih i čitati i pisati.Ova arhitektura, ne nudi toliko visok stepen skalabilnosti u poređenju sa Shared-Nothing arhitekturom[13], ali je lakša za konfiguraciju i u slučaju kvara jednog čvora, ostali mogu nastaviti sa radom koristeći zajednički disk.

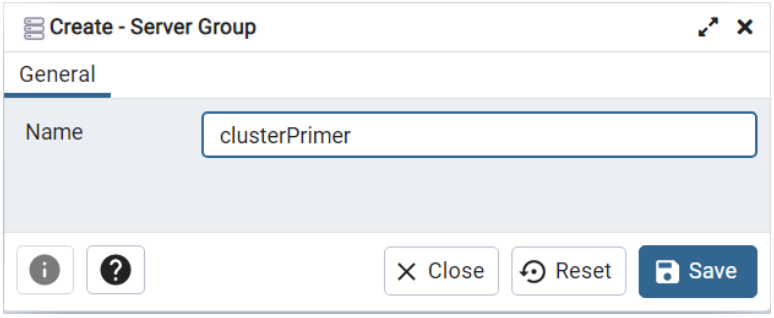


Slika 22 – Prikaz Shared-Nothing i Shared-Disk arhitekture klastera[6]

Kreiranje klastera može imati izuzetno velike prednosti za formiranje visoke dostupnosti sistema PostgreSQL baze podataka, a danas postoji veliki broj alata koji nude podršku za kreiranja i razvoj klastera PostgreSQL baze podataka. Među njima neki od najpoznatijih su PgCluster, Pgpoll-Iio, RubyRep. Bucardo, Postgres-Xc, ali i alati koji se mogu koristiti online i na Cloud platformama, poput Patroni alata i ClusterControl alata.

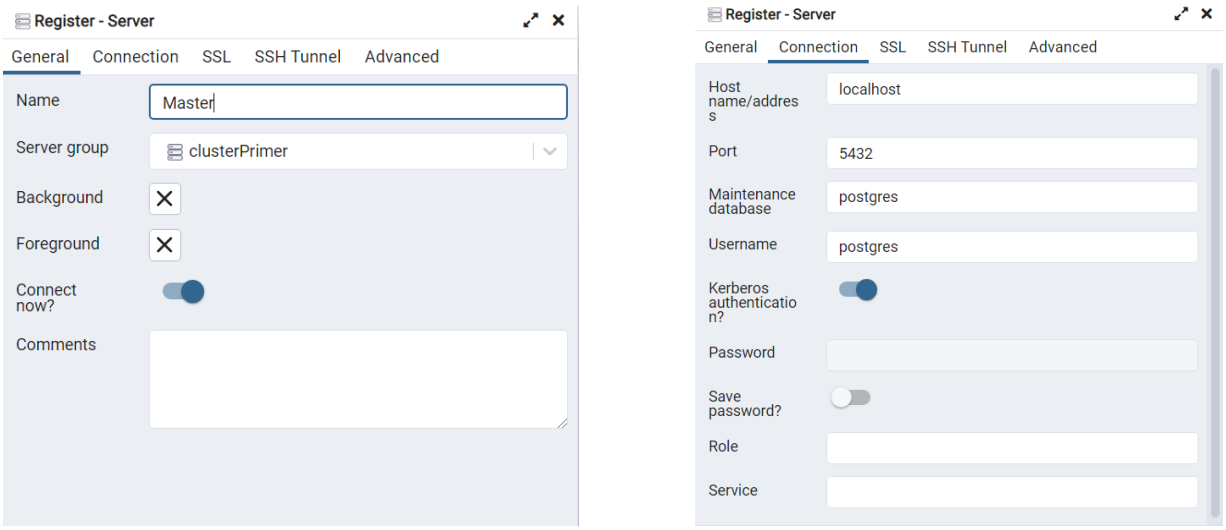
Na lokalnoj mašini, klaster se može kreirati manuelno, korošćenjem pgAdmin4 alata i podešavanjem konfiguracionih fajlova pojedinačnih servera, na način koji je prikazan ranije u seminarskom radu (misli se na replikacione mehanizme i podešavanje parametara čvorova u zavisnosti od uloge koju imaju).

Prvo se kreira grupa servera kojoj se dodeli ime, potom se u okviru nje registruju serveri i podešavaju im se konfiguracioni fajlovi. Na taj način je manuaelno kreiran klaster PostgreSQL baze podataka.

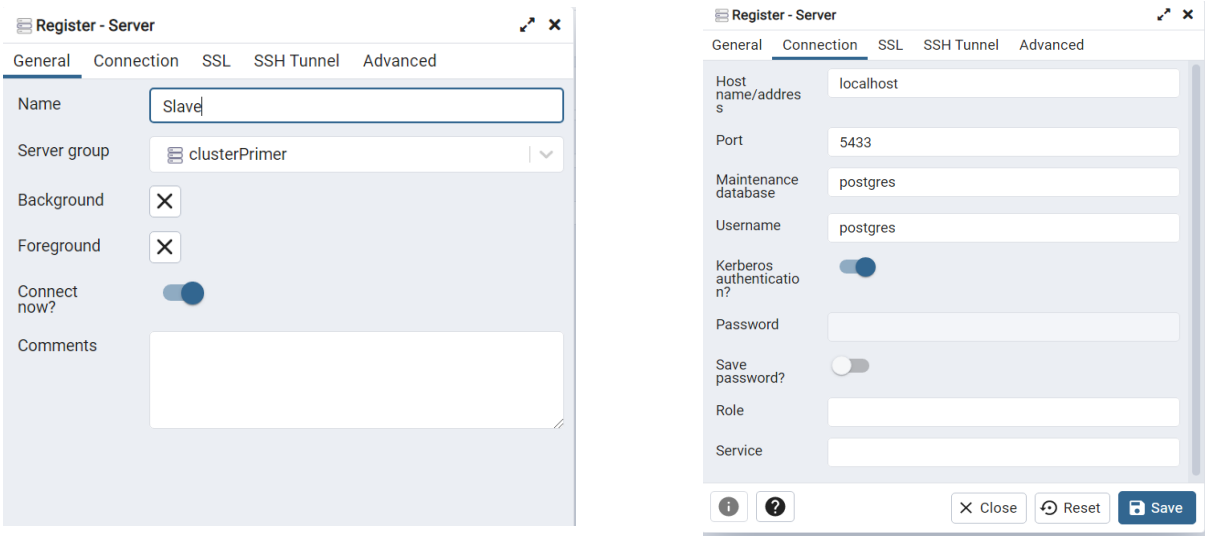


Slika 23- Kreiranje grupe servera u pgAdmin alatu

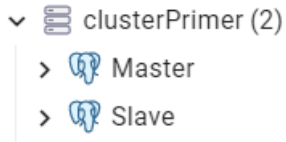
[6] Izvor slike 22-What is Database Clustering? (harperdb.io)



Slika 24- Kreiranje i registrovanje master servera



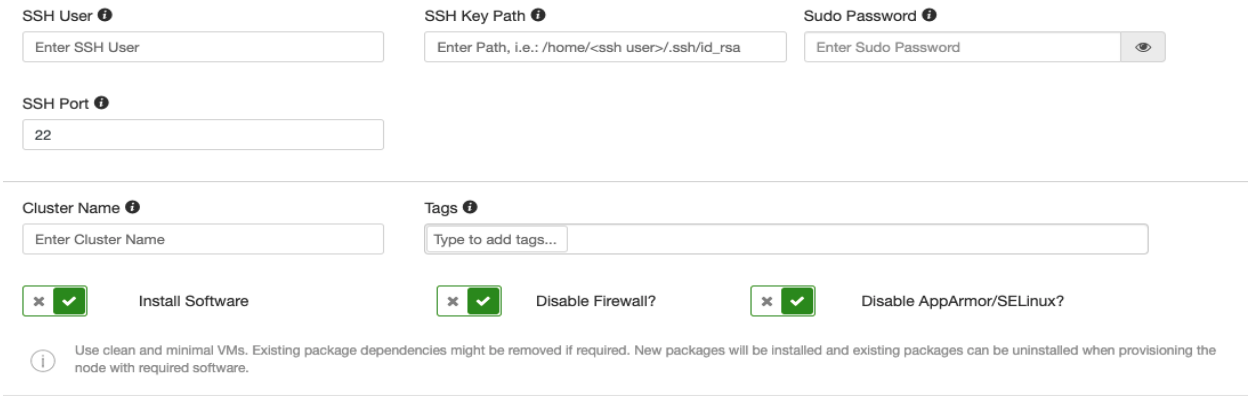
Slika 25- Kreiranje i registrovanje slave servera



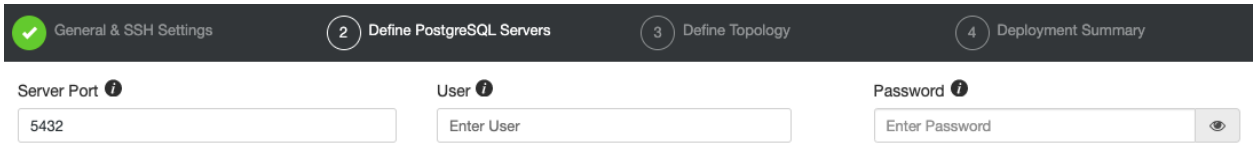
Slika 26- Prikaz kreiranog klastera

I master i server čvor podešeni su na kerberos autentifikaciju, što znači da prilikom konektovanja kreiranih korisnika na bazu, za njihovu autentifikaciju biće iskorišćena Kerberos infrastruktura, koja je opisana u seminarskom radu na temu „Sugurnost PostgreSQL baze podataka“.
Klasterizacija ima veliku primenu u održanju visoke dostupnosti, posebno kada se kombinuje sa replikacionim mehanizmima, a jedna od prednosti koju ovakav pristup organizovanju servera

pruža je garantovanje balansiranja opterećenja sistema. Proces balansiranja opterećenja (eng. load balancing) i vrste load balansera, biće naknadno opisani u narenom poglavlju. Ovde će biti dat prikaz kreiranja klastera koji će se koristiti za uvođenje HAProxy load balancera u narednom poglavlju. Klaster je kreiran korišćenjem ClusterControl alata i sadrži tri čvora- jedan koji je proglašen masterom i dva standby čvora.
Kreiranje klastera, podrazumeva izbor baze (u našem slučaju to je PostgreSQL), navođenje SSH porta (korišćen je podrazumevani PostgreSQL), imena klastera, ali i navođenje porta koji će se povezati na zadati SSH port (iskorišćen je podrazumevani PostgreSQL port 5432). Nakon ovih podešavanja potrebno je kreirati servere koji će se dodati u klaster. Ovde je moguće odmah izvršiti kreiranje master-slave topologije, a serveri se kreiraju tako što se prvo formira master čvor i dodele mu slave čvorovi, a tom prilikom se svima istovremeno dodeljuju IP adrese. Poslednji korak konfiguracije servera u klasteru je izbor tipa replikacije – da li će biti sinhrona ili asinhrona. Po završetku ove operacije, klaster je kreiran i nadalje se može koristiti u različite svrhe- demonstracija i prevazilaženje failovera, load balancing-a itd.

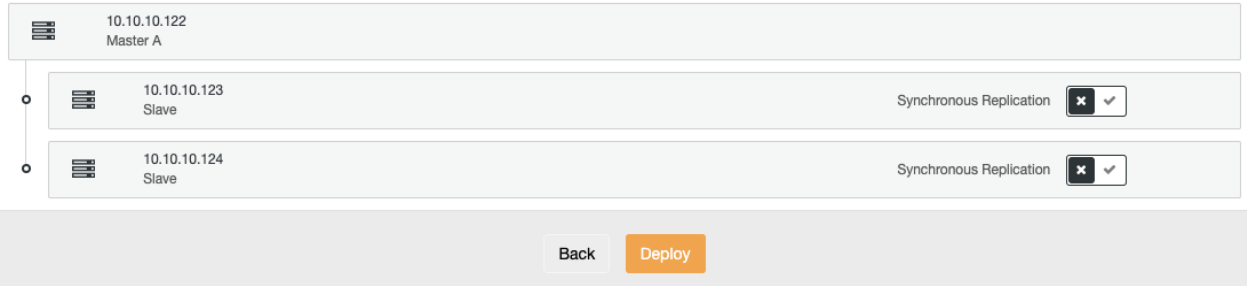


Slika 27 – Kreiranje klastera i navođenje SSH porta



Slika 28- Navođenje porta koji se povezuje sa SSH portom

Slika 29 – Kreiranja master-slave čvorova u klasteru



Slika 30 – Izbor tipa replikacije za kreirani klaster

4. LOAD BALANCING

Deo seminarskog rada koji je posvećen kreiranju klastera ističe važnost primene klasterizacije za garantovanje visoke dostupnosti PostgreSQL baze podataka, kao i značaj koji ima kod tehnike load balancinga. Ova tehnika se takođe koristi za obezbeđenje visoke dostupnosti kod postgreSQL baze podataka i vrlo često je kako je ranije u radu rečeno, povezana upravo se klasterima baze podataka.

4.1 Koncept Load Balancing-a i uloga u obezbeđenju High Availability-a

Load Balancing je tehnika distribucije opterećenja između više servera kako bi se postigla ravnoteža opterećenja i poboljšale performanse sistema. Ova tehnika se zasniva na raspodeli određenog broja zadataka među resursima sistema, a glavni razlog je sprečavanje mogućnosti da bilo kakvo preopterećenje izazove iznenadni kvar i time naruši dostupnost sistema, onemogućujući korisnicima da nastave da isti koriste.

Iako realno gledano, neka jednostavna i mala aplikacija, neće zahtevati rad većeg broja servera, kako ona bude rasla i razvijala se, javiće se potreba i za uvođenjem većeg broja servera. Vrlo često, pojedine kompanije samo izvrše zamenu svojih postojećih servera efikasnijim serverima, ali se javlja problem da jedan server može da obradi određenu količinu zahteva. Iz tog razloga, kako bi se održao high availability princip, uvode se opisani pojmovi replikacije, klasterizacije, ali i load balancinga čiji je cilj i uloga da podjednako distribuiraju korisničke zahteve između čvorova i da pravilno izvrše opterećenje svih servera baze podataka [13].

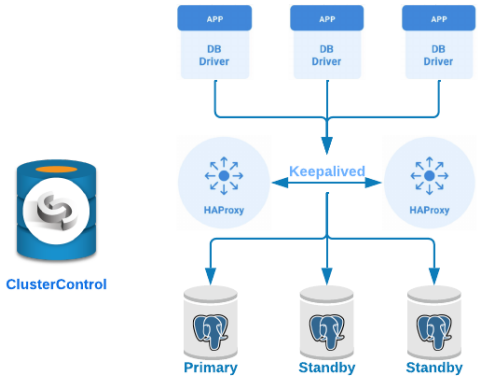
Kod PostgreSQL baze podataka, load balancing se često postiže korišćenjem HAProxy ili pgpoll alata, koji preusmeravaju zahteve sa više klijenata na odgovarajuće čvorove u replikacionom klasteru.

4.2 Metode i alati Load Balancing-a za obezbeđenje High Availability-a kod PostgreSQL baze podataka

Balanseri opterećenja su alati koji se koriste za upravljanje saobraćajem aplikacija, kako bi se izvukao maksimum iz arhitekture projektovane baze podataka. Pored toga, ovi alati ne samo da su od pomoći u balansiranju opterećenja, već pomažu i u preusmerenju zahteva ka dostupnim čvorovima, specificirajući i portove sa različitim ulogama, a sve u cilju održanja visoke dostupnosti [14].

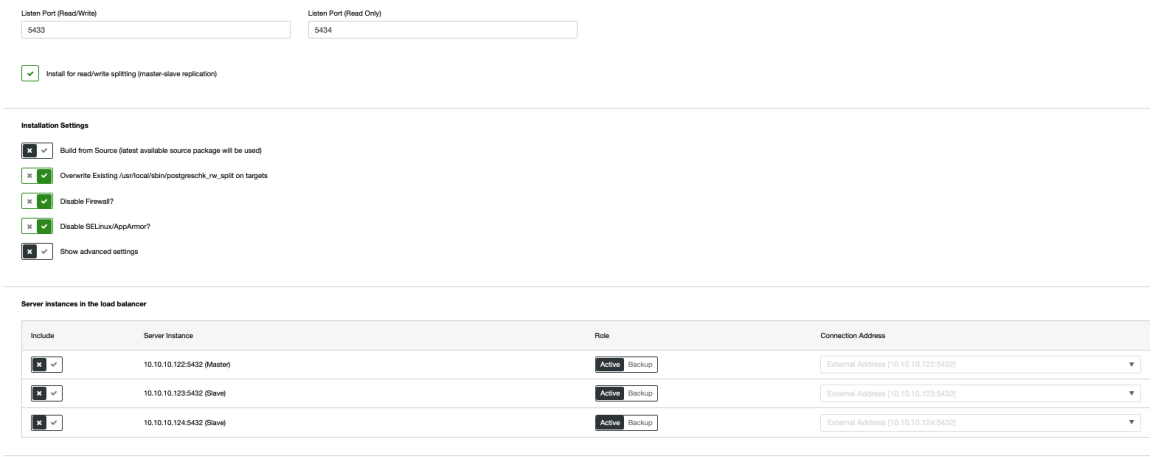
HAProxy je load balanser koji distribuira saobraćaj od jednog izvorišnog čvora do jednog ili više odredišta, pri čemu može da specificira i određena pravila, usluge ili protokole. Ukoliko neki od odredišnih čvorova prestane da odgovara, biće automatski označen kao nevažeći i sav saobraćaj se šalje na ostale dostupne destinacije. Jedna od specifičnih usluga koju ovaj load balanser pruža, je keepalived. Ona omogućava konfiguraciju virtuelne IP adrese u okviru klastera, i korišćenjem keepalived usluge ta virtuelna adresa se dodeljuje aktivnom serveru. Ukoliko taj server ne uspe ili otkaže, adresa „migrira“ na sekundarni pasivni server, omogućavajući mu da radi sa istom IP adresom na transparentan način za korisnike sistema [14].

Za potrebe seminarskog rada i demonstraciju rada HAProxy-ja, biće iskorišćen klaster kreiran korišćenjem ClusterControl interfejsa, opisan u prethodnom poglavlju rada. Klaster, kako je poznato, formiran je tako da ima tri čvora, jedan primarni i dva standby-a, a za load balancing biće formirana dva HAProxy-a.



Slika 31- Arhitektura Load Balancer-a[7]

Nakon što je kreiran klaster u prethodnom delu seminarskog rada, potrebno je izvršiti formiranje load balancer-a, izborom opcije dodavanja balansera u klaster i popunjavanjem odgovarajućih polja, koja podrazumevaju i navođenje portova koji će vršiti osluškivanje saobraćaja. Kada je osposobljen load balanser, uključuje se keepalived usluga i biraju se eksplicitno svi serveri i master i standby, ali se bira i virtualna IP adresa, kako bi moglo da se izvrši migriranje saobraćaja sa jednog na drugi čvor preko nje u skučaju otkaza nekog od čvorova, pri čemu bi sistem nastavio da funkcioniše normalno i održao high availability koncept.



Slika 32- Kreiranje load balancer-a

[7] Izvor slike- How to Deploy PostgreSQL for High Availability | Severalnines

Slika 33- Izgled klastera i load balancer-a

Prilikom formiranja load balansera, oba HAProxy-a su podešena na dva različita porta i to tako da je jednom od njih dozvoljeno i čitanje i upis, a drugom samo čitanje. Ako posmatramo port gde je dozvoljeno i čitanje i upis, primarni server je na mreži, a ostali su van, odnosno samo primar ima dozvolu upisa, a kod drugog porta gde je dozvoljeno samo čitanje svi oni se nalaze na mreži. HAProxy radi tako da ukoliko otkrije da bilo koji od čvorova, nevažno da li je primarni ili neki od standby čvorova ne reaguje, automatski ga označava kao da se nalazi van mreže. Automatsko otkrivanje greški se kod ClusterControl-a vrši pomoću skripti koje se konfigurišu za vreme primene ovako konfigurisanog sistema. Kada se identifikuje nepravilnost, označava se stanje pripravnosti i ukoliko je otkazao primar, HAProxy ga označava kao da se nalazi u offline režimu za oba porta i bira jedan od preostala dva standby-a, proglašava ga primarom i stavlja ga na mrežu gde je dozvoljeno i čitanje i upis. Ovaj proces je moguć zahvaljujući dodeli virtuelne IP adrese. Međutim, nije neretka pojava da i sam HAProxy ne uspe, zahvaljujući mehanizmu Keepalived, virtuelna IP adresa se migrira na pasivni HAProxy, koji postaje aktivan i sistem nastavlja da funkcioniše normalno bez bilo kakve ručne intervencije, što je i cilj principa visoke dostupnosti.

Pored HAProxy load balansera, PostgreSQL koristi i PGPoll-II kao proxy load balanser za balansiranje opterećenja servera i postizanje high availability koncepta.Prema definiciji, PGPoll-II je open source proxy softver koji se koristi za balansiranje opterećenja PostgreSQL servera i predstavlja sponu između serverskog sloja klijentske aplikacije i servera PostgreSQL baze podataka. Pored balansiranja opterećenja, PGPoll-II pruža i kontrolu keš upita, grupisanje veza i mnogih drugih funkcija. Najčešće se za obezbeđenje visoke dostupnosti koristi u kombinaciji sa streaming replikacijom [15].

PGPoll-II obezbeđuje visoku dostupnost procesom poliranja konekcija i automatskom podrškom za failover mehanizam. Naime, PGPoll-II je proxy, što znači da se postavlja kao posrednik između aplikacija i servera ili klastera. On prihvata konekcije aplikacija i distribuira ih između dostupnih servera u klasteru, čime direktno balansira opterećenja i poboljšava performanse. Takođe obezbeđuje i mehanizam automatskog preusmerenja saobraćaja u slučaju da jedan od PostgreSQL servera otkaže, čime pruža kontinuitet u radu. Automatsko preusmerenje saobraćaja je moguće jer PGPoll-II aktivno proverava statuse svih servera u klasteru i čim primeti da je neki od njih nedostupan, automatski ga isključuje. Podržava i mehanizam replikacije između čvorova sistema, što u slučaju kvara nekog od čvorova omogućava efikasno prebacivanje podataka sa jednog na drugi čvor.

Primer podešavanja PGPoll-II proxy-a dat je u nastavku. Podrazumeva se da je prethodno PGPoll-II preuzet i instaliran, kako bi dalja konfiguracija i povezivanje sa PostgreSQL serverima bila moguća. Za primer su iskorišćena dva prethodna servera na kojima je već podešena streaming replikacija (pogledati poglavlje seminarskog rada posvećeno replikaciji) i kreiran korisnik, ali da bi PGPPoll-II funkcionisao, potrebno je izvršiti neophodne promene u njegovom konfiguracionom fajlu, kao što je prikazano na narednoj slici.

```
## Connection Details

listen_addresses='*'
port=9999
socket_dir = 'C:\Program Files\PostgreSQL\15\data'

## Backend Server Details
# Primary

backend_hostname0 = '192.168.1.99'
backend_port0 = 5432
backend_weight0 = 1

# Secondary

backend_hostname1 = '192.168.1.100'
backend_port1 = 5433
backend_weight1 = 1

## Load Balancing

load_balance_mode = 'ON'

## Replication Responsibilty

master_slave_mode = 'ON'
master_slave_sub_mode = 'stream'

## Streaming checks

sr_check_period = 10
sr_check_user = 'sena'
sr_check_password = 'studentic'
sr_check_database = 'postgres'
delay_threshold = 10240

## Client Authentication

allow_clear_text_frontend_auth = ‘ON’
```

Slika 34- Konfiguracija PGPoll-II

Među najvažnijim parametrima konfiguracinog fajla PGPoll-II-a su svakako parametar listen_addresses, koji kako je na slici prikazano prihvata sve dolazne konekcije, zatim port na kome PGPolL-II osluškuje te dolazne konekcije. Zatim, se nevode adrese i portovi servera i omogućava se replikacija. Kada se svi parametri konfigurišu, PGPoll-II se može iskoristiti za balansiranje opterećenja PostgreSQL servera.

PGPoll-II pored nabrojanih funkcija, ima sposobnost donošenja odluke o tome koji zahtev treba da bude upućen kom čvoru u zavisnosti od vrste naredbe koja formira zahtev. Upiti koji bi trebalo da se prosleđuju samo primarnom čvoru su svi oni koji sadrže SELECT, UPDATE, INSERT, CREATE, ALTER, DROP, COPY, TRUNCATE i slične naredbe, zatim transakcione komande koje podrazumevaju setovanje, započinjanje i završetak transakcija, VACUUM naredbe, multi-statement querije i sekvencijalne funkcije. Naredbe poput SET, DISCARD i DELLOCATE ALL šalju se i primarnom i standby čvorovima, a naredbe poput EXPLAIN,

ANALYZE i SHOW, mogu se poslati na oba čvora, ali ukoliko je već omogućeno balansiranje opterećenja PGPoll-II proxy-jem i primarni server normalno funkcioniše, ovakvi upiti biće prosleđeni njemu.

Pored opisanih i prikazanih load balansera, postoji još dosta njih koji se koriste u saradnji sa PostgreSQL serverima kako bi se garantovala visoka dostupnost sistema, a izbor konkretne vrste zavisi od vrste aplikacije za koju se sistem projektuje, ličnih sklonosti i veština administratora i korisnika sistema i njihovih preferencija.

4.3 Prednosti Load Balancing-a za održavanje high availability-a

Sumiranjem prikazanog sadržaja Load Balancing-a i njegove primene kod PostgreSQL baze podataka u cilju garantovanja visoke dostupnosti sistema (eng. high availability) u okviru seminarskog rada, može se zaključiti da Load Balancing pruža niz prednosti koji omogućavaju efikasan i pouzdan rad sistema.

Kao što je već rečeno u radu, Load Balancing omogućava ravnomerno raspoređenje upita i zahteva između servera u PostgreSQL klasteru, pa je samim tim iskorišćenje resursa bolje i sprečava se preopterećenost pojedinačnih servera,a kao rezultat performanse baze su poboljšane,a vreme odziva niže. Pored toga sistem je kontinuirano dostupan, jer i u slučaju otkaza nekog od servera, load balancing detektuje otkaz servera i automatski preusmerava upite ka drugim dostupnim serverima. Na taj način obezbeđuje kontinuitet u radu sistema.

Load Balanseri pružaju i dobru osnovu za skalabilnost sistema, jer dodavanjem novih čvorova u klaster PostgreSQL baze podataka, zahvaljujući funkcijama koje poseduju, load balanseri automatski vrše raspodelu upita između novih servera, što omogućava sistemima da se prilagode rastućem broju korisnika i njihovim zahtevima.

Otpornost na kvarove sistema koji koriste load balansere je velika, jer oni podržavaju mehanizam automatske detekcije kvara i preusmerenja saobraćaja, kako sistem ne bi prestao sa radom, a još jedna od važnih karakteristika je i velika fleksibilnost. Administratori mogu lako dodavati nove servere u klaster i vršiti konfiguraciju istih, bez bojazni da će tako narušiti rad sistema, jer load balanseri automatski vrše raspodelu opterećenja prema novim konfiguracijama, što olakšava administraciju i održavanje baze podataka, pa samim tim povećava fleksibilnost sistema.

Korišćenje load balansinga pruža mnoge prednosti za održanje visoke dostupnosti kod PostgreSQL baze podataka, a implementacija odgovarajućeg balansera i load balancing strategije pruža optimalno korišćenje resursa, minimalno vreme prestanka rada sistema i bolje performanse baze podataka.

5. FAILOVER I HIGH AVAILABILITY KOD POSTGRESQL BAZE

Failover je jedan od ključnih koncepata u postizanju visoke dostupnosti kod PostgreSQL baze podataka.

Failover se odnosi na automatsko preuzimanje rada od strane rezervnog servera u slučaju kvara primarnog servera, čime se obezbeđuje neprikidno funkcionisanje sistema, čak i u prisustvu hardverskih ili softverskih problema.

Ako se pokvari primarni server, server koji se nalazi u stanju pripravnosti bi trebalo da započne proceduru failover-a. Međutim ako se pokvari neki od standby servera, a podaci su pristutni na primaru ili ostalim serverima, nema potrebe za pokretanjem postupka oporavka, jedino ukoliko ne postoji šansa da se taj rezervni server može ponovo pokrenuti.Tada se može koristiti failover, ali ukoliko je standby server trajno oštećen, potrebno je kreirati potpuno novu instancu servera [16]. U slučaju kada otkaže primarni server, a rezervni server preuzme njegovu ulogu, ukoliko se stari primar posle nekog vremena ponovo pokrene može doći do konflikta u sistemu, jer stari server nije obavešten o promeni uloga. Zato mora da postoji mehanizam obaveštenja servera o promeni uloge i ovaj proces je uglavnom poznat pod imenom „STONITH“ tj, „ Shoot The Other Node In The Head“, što je neophodno kako bi se izbegle situacije u kojima oba servera misle da su primarni, jer u tom slučaju može doći do ozbiljnog gubitaka podataka [16].

PostgreSQL baza podataka, inicijalno ne obezbeđuje sistemski softver potreban za identifikaciju kvara na čvoru i obaveštenje servera u stanju pripravnosti. Iz tog razloga neophodno je korišćenje raznih tehnika, metoda i alata, da bi se izvršila identifikacija greške i njeno razrešenje, a jedan od načina je migracija IP adrese, kao što je u pakazano u slučaju HAProxy load balansera i keepalive konekcije.

Kada je nekim od ovih mehanizama obezbeđena podrška identifikacije greške i primarni server je zamenjen, odnosno nekadašnji standby je postao primarni server, postoji mogućnost da se stari primar ponovo oporavi i mora mu se naglasiti promena uloga. Postoji pomoćni program koji PostgreSQL baza koristi kako bi ubrzala ovaj proces, ukoliko se koriste veliki klasteri i to je pg_revind. Nakon završetka ovog procesa, može se sa sigurnošću smatrati da su uloge zamenjene,a serveri obavešteni o tome. Posmatrano sa stanovišta brzine izvršenja prelaska sa jednog na drugi server prilikom otkaza, ovaj proces može relativno brzo da se izvrši, ali je potrebno neko vreme da se klaster ponovo dovede u stanje odakle može da prevaziđe neometano buduće greške u sistemu. Iz tog razloga, redovno testiranje sistema u potencijalnom stanju priravnosti je dobra praksa, jer se na taj način može stvoriti realna slika o tome kako će se sitem ponašati, ako zaista do greške ili pada nekog od servera dođe. Promovisanje ovakvog testiranja obavlja se narednim funkcijama [16]:

- Pg_ctl- trigerovanje failovera za sekundarni server;
- Pg_promote ()- kreiranje datoteke sa specifikacijom putanje do promote_triger_fajla u kome se nalaze informacije o pokretanju servera. Međutim ovaj fajl nije neophodan ukoliko se vrši prelazak na konkretnu grešku tj. na njenu obradu.

Važna napomena je da ovaj proces nije potrebno izvršavati za obezbeđenje visoke dostupnosti ukoliko su serveri inicijalno podešeni kao read-only.

Jedna od najnovijih opcija, dodatih u PostgreSQL bazu koja podržava failover i zasniva se na load balansing tehnici, opisanoj u prethodnom delu rada, jeste load_balance_hosts funkcija. Ona ima samo dva parametra koja imaju isti efekat, u slučaju pada, aktivira se load balanser i failover tehnika je omogućena ili po random parametru, što znači da se saobraćaj povezuje na bilo koji od

čvorova koji su bili u pripravnosti, ili po disable parametru gde je suština da se povezivanje izvrši u ranije navedenom redosledu navođenja standby servera.
Naredni primer koristi ranije kreiran klaster, kome je pridodat još jedan standby čvor i svi serveri su pokrenuti i konfigurisani u skladu sa svojim ulogama, kao što je prikazano na slici.

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host    replication     all             192.168.1.100/32            trust
```

```
#------------------------------------------------------------------------------
# WRITE-AHEAD LOG
#------------------------------------------------------------------------------

# - Settings -

wal_level = logical                     # minimal, replica, or logical
                                        # (change requires restart)
```

Slika 35- Podešavanje master servera

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host    replication     all             192.168.1.101/32            trust
```

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host    replication     all             192.168.1.102/32            trust
```

```
#------------------------------------------------------------------------------
# WRITE-AHEAD LOG
#------------------------------------------------------------------------------

# - Settings -

wal_level = replica                     # minimal, replica, or logical
```

Slika 36- Podešavanje standby servera

Kako bi se simulirao pad sistema, potrebno je u konfiguracionom fajlu željenog čvora čiji pad prikazujemo umesto trust parametra u pg_hba.conf fajlu, navesti reject parametar. Nakon toga port servera će biti isključen i pad čvora će biti simuliran. Failover i detekcija greške, kao i oporavak se postiže korišćenjem tehnike balansiranja i ukoliko je naveden parametar random bilo koji slobodni i zdrav server preuzeće ulogu primara,a ukoliko je navedena opcija disable prvi naredni server iz connection stringa biće uzet kao novi primarni server. Ova opcija je relativno nova i još uvek nedovoljno ispitana, pa sve njene efekte treba uzeti sa rezervom.

```
postgres=# select setting as "port" from pg_settings where name='port';
 port
------
 5432
(1 row)
```

Slika 37- Provera ispravnosti master čvora klastera

```
C:\Program Files\PostgreSQL\15 - 1\data>psql 'host= 192.168.1.100, 192.168.1.101, 192.168.1.102 port= 5432,5433,5434 load_balance_hosts= disable dbname=postgres user=postgres' -c show port'
------
 5433
(1 row)
```

Slika 38- Simulacija failovera i provera uspešnosti prelaska na standby čvor korišćenjem parametra disable

```
C:\Program Files\PostgreSQL\15 - 1\data>psql 'host= 192.168.1.100, 192.168.1.101, 192.168.1.102 port= 5432,5433,5434 load_balance_hosts= random dbname=postgres user=postgres' -c show port'
------
 5433
(1 row)
```

Slika 39 – Simulacija failovera i provera uspešnosti prelaska na standby čvor korišćenjem parametra random

Kao što se može zaključiti sa priloženih slika, u oba slučaja naredni izabrani čvor radi na portu 5433, sistem nastavlja da funkcioniše i dalje i iz ugla korisnika problem nije primetan.

6. HOT-STANDBY I HIGH AVAILABILITY KOD POSTGRESQL-A

Hot-Standby je termin koji se koristi za opisivanje mogućnosti povezivanja sa serverom i pokretanja upita samo za čitanje dok je server u režimu oporavka ili u režimu pripravnosti. Ovaj pristup je koristan za potrebe replikacije i vraćanje rezervnih kopija u željeno stanje sa velikom preciznošću. Termin hot-standby se odnosi i na sposobnost samog servera da pređe sa oporavka u normalni režim rada, dok korisnici nastavljaju da izvršavaju upite i drže svoje konekcije otvorenim [17], što je pokazatelj visokog stepena dostupnosti sistema.

U okviru Hot-Standby arhitekture, postoji jedan primarni server koji prihvata upite i ažurira podatke i postoji jedan ili više standby servera, koji se mogu naći u pasivnom režimu, gde samo primaju replikovane podatke, ili u aktivnom režimu, omogućavajući čitanje podataka dok primarni server nije dostupan. Glavni cilj hot-stanby tehnike je obezbeđivanje kontinuirane dostupnosti baze podataka u slučaju pada primarnog servera. Kada primar postane nedostupan , jedan od standby servera može preuzeti njegovu ulogu i ovaj proces je poznat kao failover (opisan u prethodnom poglavlju).

Visoka dostupnost (eng. high availability) se postiže konfigurisanjem Hot-Standby arhitekture uz pomoć odgovarajućih replikacionih tehnika, kao što su streaming ili logička replikacija, o kojima je ranije bilo reči u radu. Ovim mehanizmima, obezbeđena je sinhronizacija podataka između glavnog i standby servera, ali i konzistentnost i pouzdanost same replikacije.

Podrazumevano, kada se PostgreSQL sistem nalazi u stanju oporavka ili pripravnosti, on neće prihvatati klijentske konekcije, naprotiv odbijaće ih uz izdavanje obaveštenja: „ database system is in recovery mode“. Međutim, aktiviranjem linije konfiguracionog fajla standby servera, koja kao parametar ima hot_standby na „on“ za režim rada, može se omogućiti PostgreSQL sistemu da prihvata klijentske konekcije i dozvoli read-only transakcije. Dozovoljene su samo read-only transakcije, jer se sistem nalazi u stanju pripravnosti i podešavanjem hot-standby parametra proces uspostavljanja konekcija tek kreće, kako bi se sistem našao u konzistentnom stanju. Iz tog razloga sve veze koje se formiraju u tom trenutku moraju striktno biti samo za čitanje, čak ni privremene tabele ne mogu vršiti upis. U ovakvoj situaciji, podacima sa glavnog servera je potrebno neko vreme da stignu do ostalih servera, tako da u stanju pripravnosti postoji neko kašnjenje u prenosu podataka. Samim tim, pokretanje upita istovremeno i na primarnom i na nekom od standby servera, može da da različite rezultate, pa se zato u stanju pripravnosti kaže da su podaci u skladu sa onim što prikazuje primarni server [17]. Što se tiče standby servera, onog trenutka kada je izvršen commit zapisa transakcije u stanju pripravnosti, sve promene napravljene tom transakcijom biće vidljive snapshot-ovima standby-a u stanju pripravnosti. Kada će se izvršiti snapshot upita, zavisi od trenutnog nivoa izolacije transakcije [17].

Transkacije koje su započete tokom hot standby tehnike, mogu imati uticaj na sledeće komande:

- Pristup upitu SELECT i COPY TO naredbama;
- Komande kursora: DECLARE, FETCH, CLOSE;
- Komande podešavanja: SHOW, SET, RESET;
- Komande za upravljanje transakcijama: BEGIN; END; ABORT; TRANSACTION itd;
- Komande za zaključavanjem pristupa tabeli: LOCK TABLE;
- Komande za planiranje, plugin i uključivanje ekstenzija.

Međutim i pored pomenutog kašnjenja, transakcionih pravila i mogućih uticaja na određene upite, hot- standby je jedna od tehnika koja na globalnom nivou PostgreSQL sistema baze podataka smanjuje vreme nedostupnosti sistema, jer ukoliko je uključena proces failovera može gotovo momentalno započeti i biti transparentan za korisnike. Takođe povećava i otpornost na

hardverske kvarove, jer u slučaju hardverskog kvara jednog servera, naredni nastavlja da radi i sistem ostaje dostupan, što je primarni faktor održanja visoke dostupnosti. Samim tim poboljšana je pouzdanost sistema i dodat je dodatni sloj zaštite od gubitaka informacija i podataka.
U nastavku je dat primer podešavanja standby servera kreiranog za potrebe demonstracije prethodnih metoda obezbeđenja visoke dostupnosti koje su opisane u radu, pa će jedan od standby servera biti iskorišćen i za prikaz podešavanja hot-standby-a. Podrazmeva se da je sistem podešen za prihvatanje replikacije i recovery-a, tj da je kreiran i replikacioni slot na kome će se čuvati backup-ovani podaci. Prilikom konfigurisanja standby servera ne navodi se putanja do tog direktorijuma, već simboličko ime slota.

```
# - Standby Servers -

# These settings are ignored on a primary server.

primary_conninfo = 'host=192.168.1.100 port=5432 sslmode=prefer sslcompresion=0 krbsrvname=postgres target_session_attrs=any'
primary_slot_name = 'standby1'
#promote_trigger_file = ''
hot_standby = on
```

Slika 40 – Primer podešavanja hot- standby-a kod PostgreSQL servera

7. ZAKLJUČAK

Visoka dostupnost (eng. High Availability) je ključan faktor za održanjem neprekidnog rada i pouzdanosti PostgreSQL baze podataka, pri čemu PostgreSQL pruža različite mehanizme i funkcionalnosti za postizanjem visoke dostupnosti, što korisnicima omogučava kontinuirani pristup podacima i minimalno vreme prekida rada sistema.

Implementacija visoke dostupnosti kod PostgreSQL baze podataka zahteva pažljivo planiranje i konfiguraciju sistema, uspostavljanje sigurnosnih mehanizama,ali i redovno testiranje i nadgledanje sistema. Takođe je važno i pravovremeno reagovanje na izdata upozorenja kako bi se sprečili potencijalni problemi i minimiziralo vreme prekida rada. Kroz pravilno konfigurisane mehanizme visoke dostupnosti, PostgreSQL baza podataka može obezbediti pouzdanu i neprekidnu uslugu za korisnike, što je od vitalnog značaja u poslovnim okruženjima. Visoka dostupnost pomaže u održanju integriteta podataka, sprečavanju trajnog gubitka istih i smanjenju vremena neefikasnosti sistema, čime se postiže viši nivo zadovoljstva korisnika i poverenja u sistem.

U seminarskom radu, pored definicije i pregleda svojstava visoke dostupnosti, prikazani su i obrađeni različiti mehanizmi koje PostgreSQL baza koristi kako bi postigla ovaj koncept, počevši od replikacije i njenih tipova, load balancing-a i klasterizacije, zaključno sa failover i hot-standby tehnikama. Svaki od prikazanih elemenata, propraćen je i primerima, pri čemu je skrenuta pažnja na ključne faktore koji se koriste prilikom konfigurisanja istih. Pregledom rada, može se ustanoviti da svaki element može da se kombinuje sa nekim od ostalih koji su pomenuti, pri čemu se postiže još viša efikasnost, a koji od mehanizama će biti izabran,zavisi pre svega od svrhe za koju će se PostgreSQL sistem koristiti, ali i specifičnih zahteva aplikacija i klijenata povezanih sa sistemom.

8. SPISAK KORIŠĆENE LITERATURE

[1] B. Lutkevich, A. S. Gillis, "High availability", TechTarget Data Centar
Dostupno: https://www.techtarget.com/searchdatacenter/definition/high-availability
[2] "What is a high availability database", Erospike
Dostupno: https://aerospike.com/glossary/high-availability-database/
[3] S. Insausti, "How to Deploy PostgreSQL for High Availability", 2018
Dostupno: https://severalnines.com/blog/how-deploy-postgresql-high-availability/
[4] "High availability concepts in Azure Database for PostgreSQL - Flexible Server", Microsoft Documentation, 2023.
Dostupno: https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/concepts-high-availability
[5] "PostgreSQL Replication and Automatic Failover Tutorial", EDB, 2023.
Dostupno: https://www.enterprisedb.com/postgres-tutorials/postgresql-replication-and-automatic-failover-tutorial
[6] "SYNCHRONOUS AND ASYNCHRONOUS REPLICATION", Cybertec
Dostupno: https://www.cybertec-postgresql.com/en/services/postgresql-replication/synchronous-asynchronous-replication/
[7] "Physical Replication Mechanisms in PostgreSQL", pgDash
Dostupno: https://pgdash.io/blog/postgres-physical-replication.html
[8] "Log-Shipping Standby Servers", PostgreSQL 15 Documentation
Dostupno: https://www.postgresql.org/docs/15/warm-standby.html
[9] S. Insausti, "PostgreSQL Replication Best Practices", seeveral9s,2021.
Dostupno: https://severalnines.com/blog/postgresql-replication-best-practices-part1/
[10] K. Ghosh, "Logical Replication in PostgreSQL Explained", EDB, 2023.
Dostupno: https://www.enterprisedb.com/postgres-tutorials/logical-replication-postgresql-explained
[11] H. J. Schönig ,"POSTGRESQL CLUSTERS: MONITORING PERFORMANCE", Cybertec
Dostupno: https://www.cybertec-postgresql.com/en/postgresql-clusters-monitoring-performance/
[12] S.Rithika, "Setup PostgreSQL Clusters in 4 Easy Steps [+7 Clustering Options]", Hevo, 2023.
Dostupno: https://hevodata.com/learn/postgresql-cluster/#12
[13] M. Ibrahim, "What is a database cluster", 2022.
Dostupno: https://www.harperdb.io/post/what-is-database-clustering
[14] S. Insausti, "How to Deploy PostgreSQL for High Availability", seeveral9s,2018.
Dostupno: How to Deploy PostgreSQL for High Availability | Severalnines
[15] A. Muthuramalingam,"PgPool – II Installation & Configuration", Mydbops, 2021.
Dostupno: PgPool – II Installation & Configuration – (Part-I) (wordpress.com)
[16] Failover, PostgreSQL Documentation, 2023.
Dostupno: https://www.postgresql.org/docs/15/warm-standby-failover.html
[17] Hot Standby, PostgreSQL Documentation, 2023.
Dostupno: https://www.postgresql.org/docs/15/hot-standby.html