



UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET



**CLUSTER REŠENJA KOD POSTGRESQL BAZE PODATAKA**  
**-Seminarski rad-**

Student:  
Sara Savić, br.ind. 1758

Mentor:  
Prof. dr Aleksandar Stanimirović

Niš, jun 2024.

## SAŽETAK

U seminarskom radu “Cluster rešenja kod PostgreSQL baze podataka” opisan je i predstavljen pojam klastera baza podataka, sama arhitektura klastera, kao i značaj uvođenja i korišćenja klastera u bazama podataka.

U okviru rada, takođe je opisano i na koji način se klasteri koriste u PostgreSQL bazi podataka, najpre kao skup baza podataka organizovanih unutar jednog PostgreSQL sistema, a zatim i kao grupa servera koji rade zajedno kako bi obezbedili visoku dostupnost i skalabilnost sistema. Pored detaljnog opisa, svaki od ovih elemenata predstavljen je i praktičnim primerima.

Klasteri kod svih baza podataka, pa tako i kod PostgreSQL baze podataka predstavljaju važan faktor jer omogućavaju poboljšanje performansi i otpornost na greške, raspoređujući podatke i radno opterećenje preko više servera za efikasniju i pouzdaniju obradu podataka.

## Sadržaj

1. UVOD .....	4
2. CLUSTER BAZE PODATAKA (DATABASE CLUSTER) .....	5
2.1 Definicija pojma cluster-a baze podataka .....	5
2.2 Arhitektura cluster-a .....	5
2.3 Značaj korišćenja cluster-a baze podataka .....	6
3. CLUSTER POSTGRESQL BAZE PODATAKA .....	7
4. KORIŠĆENJE CLUSTER-A ZA POSTIZANJE VISOKE DOSTUPNOSTI (ENG.HIGH AVAILABILITY) .....	11
4.1 Replikacija HA klastera .....	12
4.2 Cluster rešenja koja se koriste za postizanje visoke dostupnosti .....	20
5.KORIŠĆENJE CLUSTER-A ZA POSTIZANJE SKALABILNOSTI.....	26
5.1 Korišćenje sharding tehnike za postizanje horizontalnog skaliranja .....	30
6.ZAKLJUČAK .....	34
7. SPISAK KORIŠĆENE LITERATURE .....	35

## 1. UVOD

Klasteri predstavljaju skupove međusobno povezanih računarskih sistema koji rade zajedno kao jedna jedinica. U kontekstu baza podataka, klasteri omogućavaju distribuiranje podataka i radnog opterećenja kako bi se postigla bolja skalabilnost, otpornost na greške i visoka dostupnost.

Klasteri igraju ključnu ulogu u modernim sistemima za upravljanje bazama podataka, a PostgreSQL koristi klastere pre svega kao skupove baza podataka u okviru jednog sistema, ali i kao veći broj servera koji rade zajedno. Kroz različite tehnike klasterovanja, PostgreSQL obezbeđuje prethodno pomenute sposobnosti koje su prevashodne za pravilno funkcionisanje baze podataka.

U narednim poglavljima seminarskog rada, biće dat pregled definicije klastera, kao i opis arhitekture samih klastera i njihovog značaja u sistemima za upravljanje bazama podataka. Pored toga, biće objašnjeno kako se koriste klasteri kod PostgreSQL baze podataka kao skup baza podataka unutar jedinstvenog sistema i za fizičku reorganizaciju podataka u tabelama. Važan deo seminarskog posvećen je i korišćenju klastera u PostgreSQL-u, kao grupe servera, za postizanje visoke dostupnosti, najpre opisujući pojam replikacije koja omogućava kopiranje podataka sa primarnog servera na sekundarni zarad održavanja visoke dostupnosti. Takođe, su opisani razni alati koje PostgreSQL koristi za kreiranje klastera i za postizanje visoke dostupnosti. Pored dostupnosti, prikazano je i opisano kako PostgreSQL omogućava skalabilnost, konkretno se fokusirajući na tehniku shard-ovanja.

## 2. CLUSTER BAZE PODATAKA (DATABASE CLUSTER)

Prilikom rada sa bazama podataka, neophodno je obezbediti visoku dostupnost, skalabilnost i pouzdanost podataka koji se nalaze unutar same baze. Međutim, količina sačuvanih podataka može biti jako velika u nekom trenutku, te korišćenje samo jednog servera za upravljanje tim podacima ne predstavlja dobro rešenje. Tu na scenu stupaju cluster rešenja kod baza podataka, koja omogućavaju korišćenje većeg broja servera za upravljanje podacima unutar baze podataka.

U okviru ovog poglavlja biće data definicija pojma cluster-a baze podataka, arhitektura cluster-a kao i zašto je značajno koristiti cluster rešenja.

### 2.1 Definicija pojma cluster-a baze podataka

“Cluster baze podataka (*eng.Database cluster*) predstavlja skup konfiguracija u kojima više servera baze podataka (koji se još nazivaju i čvorovi baze podataka) rade zajedno kako bi poboljšali ili unapredili performanse sistema.” [1]

Svaki server unutar cluster-a čuva kopiju istih podataka, osiguravajući redundantnost i toleranciju na greške. Iako postoje različite tehnike klasterovanja baze podataka, glavni cilj korišćenja klastera jeste obezbeđivanje visoke dostupnosti, koja se odnosi na sposobnost baze podataka da ostane dostupna čak i u slučaju kvara hardvera ili pada sistema.

Klasteri baza podataka često koriste balansiranje opterećenja (*eng.load balancing*) kako bi ravnomerno raspodelili dolazne zahteve među serverima, sprečavajući da bilo koji server postane “usko grlo” (*eng.bottleneck*)<sup>1</sup>. Ovo osigurava efikasnu upotrebu resursa i optimalno vreme odgovora. Replikacija podataka je još jedna ključna karakteristika klastera baze podataka, gde se promene napravljene na jednom serveru prenose na druge, održavajući konzistentnost podataka širom cluster-a.

Klasteri su od vitalnog značaja za aplikacije koje zahtevaju neprekidan rad, integritet podataka i sposobnost da se skaliraju bez problema kako bi se prilagodile rastućim opterećenjima. Oni pružaju čvrstu osnovu za sisteme koji su od vitalnog značaja u različitim industrijama.

### 2.2 Arhitektura cluster-a

Kako bi bilo moguće razumeti način rada klastera baze podataka, neophodno je shvatiti samu arhitekturu klastera. Postoje dva arhitekturna tipa klastera: Shared-Nothing i Shared-Disk.

#### Shared-Nothing arhitektura

Kod Shared-Nothing arhitekture svaki server mora biti nezavistan u odnosu na sve ostale servere u sistemu. Ovo znači, da u ovakvoj vrsti arhitekture ni jedan server nije glavni, odnosno nijedan server nije centralizovan kako bi nadgledao i kontrolisao pristup podacima u sistemu. U ovakvoj arhitekturi, vrši se distribuiranje podataka između servera kako bi se postigla visoka dostupnost. Ovakva arhitektura nudi visok stepen skalabilnosti, ali upravljanje podacima može biti složenije u odnosu na shared-disk arhitekturu [2].

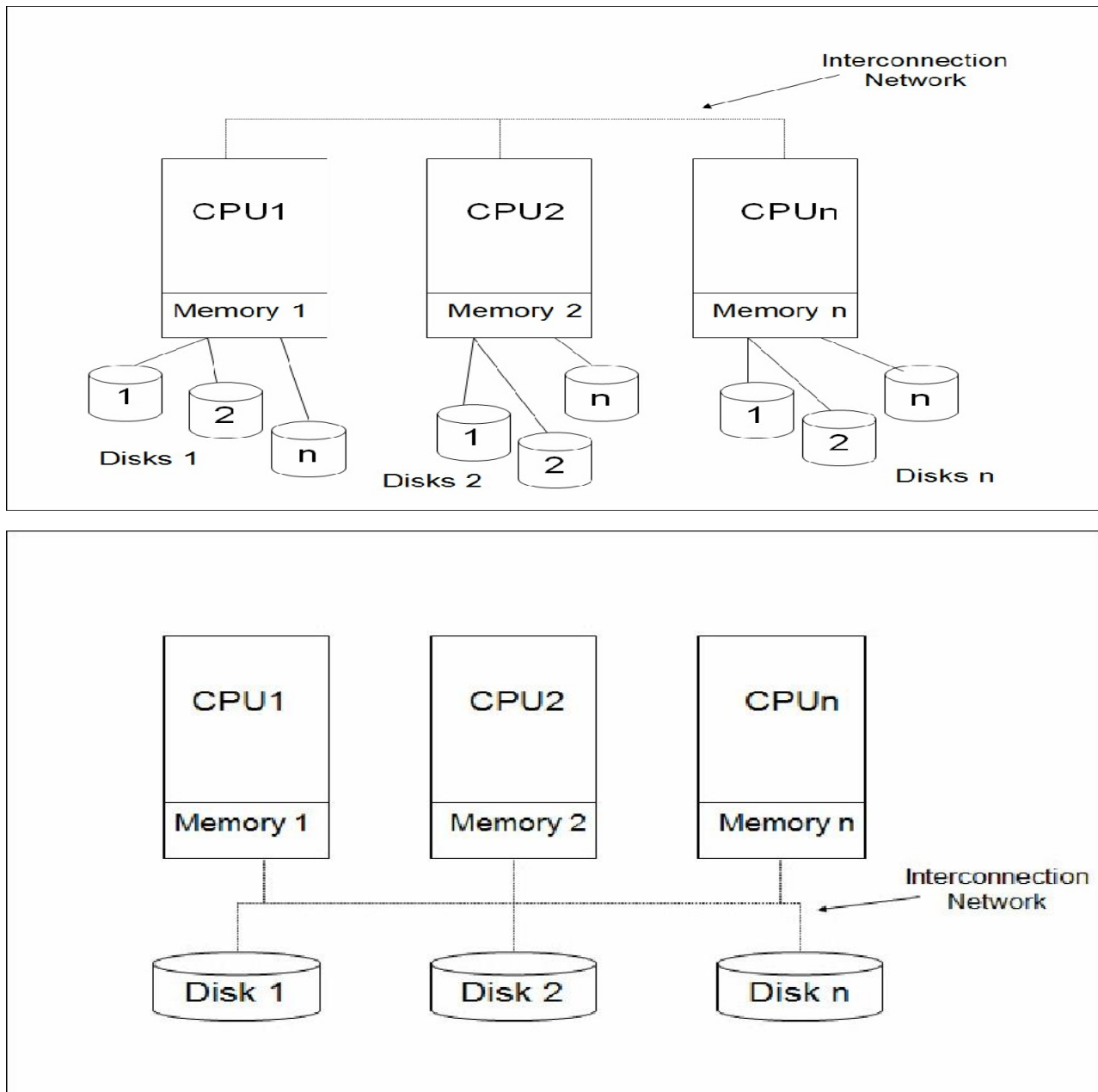
#### Shared-Disk arhitektura

Kod ovakve vrste arhitekture, svi čvorovi (CPU) baze podataka imaju pristup svim podacima sistema, zato što se mrežni sloj interkonekcije nalazi između CPU-a i servera baze podataka. Podaci se čuvaju na centralizovanom deljenom disku, koji je dostupan svim čvorovima, pri

---

<sup>1</sup> Usko grlo-deo sistema koji ograničava protok podataka kroz sistem.

čemu svaki čvor ima pristup podacima i ima mogućnost njihovog čitanja i upisa. Iako ova arhitektura ne nudi toliko visok stepen skalabilnosti u poređenju sa prethodnom, lakša je za konfiguraciju i u slučaju kvara jednog čvora, ostali mogu nastaviti sa radom neometano koristeći zajednički disk [2].



Slika 1-Prikaz Shared-Nothing (slika gore) i Shared-Disk arhitekture(slika dole)<sup>2</sup>

### 2.3 Značaj korišćenja cluster-a baze podataka

Korišćenje klastera pruža nekoliko značajnih prednosti kao što su:

1. Balansiranje opterećenja sistema (*eng.load balancing*)-Klasterovanje omogućava balansiranje opterećenja koje predstavlja proces raspodele određenog broja zadataka na

<sup>2</sup> Izvor slike- <https://www.harperdb.io/post/what-is-database-clustering>

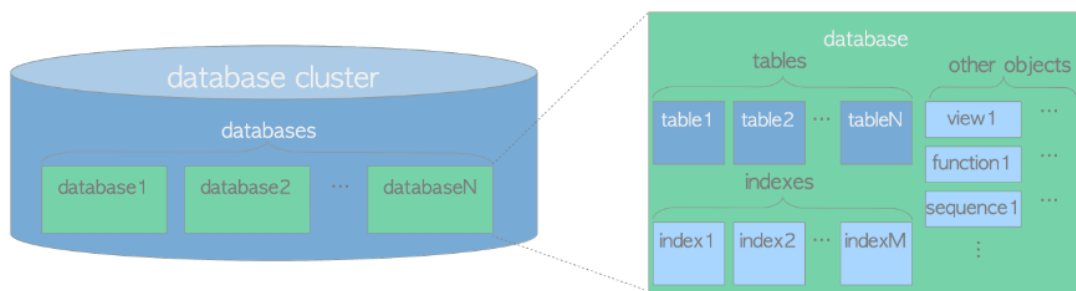
više različitih resursa kako bi se sprečila mogućnost preopterećenja sistema koje bi izazvalo iznenadni kvar sistema.

2. Skalabilnost i povećanje broja korisnika- Dodavanjem većeg broja servera, kompanije mogu da obrade mnogo veći broj korisnika iz različitih delova sveta.
3. Redundantnost podataka- Redundantnost podataka predstavlja proces čuvanja podataka na dva ili više različitih mesta za skladištenje. U slučaju klasterovanja baza podataka, svi serveri moraju imati iste i tačne podatke. Ovo je posebno važno u situacijama kada dođe do problema u bazi podataka, jer redundantnost podataka osigurava da podaci ostanu pristupačni.[2]

### 3. CLUSTER POSTGRESQL BAZE PODATAKA

U prethodnom poglavlju data je definicija pojma cluster-a baze podataka, opis arhitekture cluster-a, kao i značaj korišćenja cluster-a baza podataka. U okviru ovog poglavlja biće opisano šta su cluster-i kod PostgreSQL baze podataka i kako se kreiraju.

Kod PostgreSQL baze podataka termin “cluster baze podataka” se najčešće odnosi na grupu baza podataka koje su organizovane unutar jednog PostgreSQL sistema [3]. Na slici 2 prikazana je logička struktura PostgreSQL cluster-a.



Slika 2-Prikaz logičke strukture PostgreSQL cluster-a<sup>3</sup>. U teoriji relacionih baza podataka, objekat baze podataka je struktura podataka koja se koristi za skladištenje ili refenciranje podataka, i to obično predstavljaju indeksi, pogledi (views), funkcije, itd. U PostgreSQL sistemu same baze podataka su takođe objekti baze podataka i logički su odvojene jedna od druge, a svi ostali objekti (npr. tabele, indeksi itd.) pripadaju svojim odgovarajućim bazama podataka. Sve ovo čini cluster baze podataka u PostgreSQL-u.[4]

PostgreSQL cluster se sastoji od dva elementa:

1. Direktorijuma podataka koji predstavlja ključnu komponentu PostgreSQL sistema koji se koristi za upravljanje bazama podataka i odgovoran je za skladištenje i preuzimanje svih podataka vezanih za bazu. Sadrži konfiguracione fajlove, tabele baza podataka, indekse, log fajlove, SQL naredbe, informacije o korisnicima, ulogama i privilegijama, kao i fizičku strukturu baze podataka.[5]
2. Postgres procesa koji predstavlja glavni proces koji kontroliše fajlove u direktorijumu podataka i pruža interfejs za manipulaciju njihovim sadržajem. Postgres omogućava kreiranje i prilagođavanje šeme baze podataka, kao i rukovanje velikim količinama podataka i složenim upitima.[5]

<sup>3</sup> Izvor slike-<https://www.interdb.jp/pg/pgsql01/01.html>

Svi podaci cluster-a, uključujući i njegovu konfiguraciju, mogu se u potpunosti čuvati unutar direktorijuma podataka. Postgres alati kreiraju direktorijum podataka koji je samostalan, tačnije sadrži sve što je neophodno za rad cluster-a i može nezavisno da funkcioniše bez potrebe za dodatnim konfiguracijama ili fajlovima izvan ovog direktorijuma. Zaustavljanjem Postgres procesa i kopiranjem direktorijuma podataka, kreira se kompletna rezervna kopija podataka cluster-a, što znači da je ceo direktorijum podataka moguće premestiti na drugo mesto i cluster će nastaviti da funkcioniše normalno.[4]

Za kreiranje novog PostgreSQL cluster-a koristi se komanda `initdb` na način na koji je prikazan na slici 3.

```
C:\cluster>initdb -D "C:\cluster"
The files belonging to this database system will be owned by user "SARA&SENA".
This user must also own the server process.

The database cluster will be initialized with locale "English_United Kingdom.1252".
The default database encoding has accordingly been set to "WIN1252".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory C:\cluster ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... windows
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Europe/Budapest
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D ^"C:\cluster^" -l logfile start
```

Slika 3- Prikaz kreiranja novog cluster-a. Ovaj PostgreSQL cluster gradi novu bazu podataka i direktorijum podataka na lokaciji koja je navedena (u ovom slučaju "C:\cluster"), pri čemu ovaj direktorijum mora da bude prethodno kreiran od strane korisnika koji ima ovlašćenje za to. Pošto ovaj direktorijum sadrži sve podatke koji se čuvaju u bazi podataka, vrlo je važno da on bude zaštićen od neovlašćenog pristupa. Zato komanda `initdb` dozvole daje samo korisniku PostgreSQL-a ili određenoj grupi. Pristup grupi je omogućen samo za čitanje, što dozvoljava nepriviligovanim korisnicima koji su u istoj grupi kao i vlasnik klastera da naprave rezervne kopije podataka ili obave neke druge operacije koje uključuju samo čitanje. Nakon inicijalizacije moguće je povezati se na server i kreirati bazu podataka i ostale objekte. (prikazano na slikama 4 i 5).[6]

```
C:\Program Files\PostgreSQL\16\bin>pg_ctl start -D C:\cluster
waiting for server to start...2024-06-10 13:01:37.847 CEST [12756] LOG:  starting PostgreSQL 16.2, compiled by Visual C++ build 1937, 64-bit
2024-06-10 13:01:37.851 CEST [12756] LOG:  listening on IPv6 address "::1", port 5432
2024-06-10 13:01:37.852 CEST [12756] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2024-06-10 13:01:37.904 CEST [12908] LOG:  database system was shut down at 2024-06-10 12:52:35 CEST
2024-06-10 13:01:37.929 CEST [12756] LOG:  database system is ready to accept connections
done
server started
```

Slika 4-Povezivanje na server (cluster) korišćenjem `pg_ctl start` komande. Ukoliko je potrebno zaustaviti server koristi se komanda `pg_ctl stop`, a za restartovanja `pg_ctl restart` komanda.

```
postgres=# CREATE DATABASE nova_baza;
CREATE DATABASE
```

Slika 5- Prikaz kreiranja baze podataka u okviru klastera.



PostgreSQL takođe nudi mogućnost cluster-ovanja određenih tabela korišćenjem indeksa. Za to se koristi komanda CLUSTER. Ova komanda nalaže PostgreSQL-u da izvrši cluster-ovanje tabele čije se ime navodi u konstrukciji (kao što je prikazano na slici 7) na osnovu indeksa, koji prethodno mora biti definisan. Kada se klasteruje tabela ona se fizički reorganizuje na osnovu informacija iz indeksa. Klasterovanje je jednokratna operacija jer kada se tabela kasnije ažurira, te promene se ne klasteruju, odnosno ne pokušava se čuvanje novih ili ažuriranih redova prema njihovom redosledu u indeksu. PostgreSQL pamti indeks po kome je prvobitno izvršeno klasterovanje, a ukoliko se koristi samo komanda CLUSTER bez specificiranja imena tabele onda se vrši klasterovanje svih prethodno klasterovanih tabela u trenutnoj bazi podataka (prikazano na slici 10). Prilikom klasterovanja tabele, PostgreSQL stiče ACCESS EXCLUSIVE zaključavanje nad tom tabelom što znači da nijedna druga operacija, ni čitanje ni upis, ne može biti izvršena nad tabelom dok se klasterovanje na završi. Ovo garantuje da nijedna druga operacija neće uticati na proces klasterovanja ili biti pogođena njime, čime se obezbeđuje konzistentnost i integritet podataka.[7]

Query	Query History	Query	Query History
1 CREATE TABLE PROIZVODI ( 2 id SERIAL PRIMARY KEY, 3 naziv VARCHAR(100), 4 cena NUMERIC(10, 2), 5 kategorija VARCHAR(50) 6 );		1 INSERT INTO PROIZVODI (naziv, cena, kategorija) VALUES 2 ('Laptop Dell XPS 15', 1999.99, 'Laptopi'), 3 ('Smartphone Samsung Galaxy S20', 999.99, 'Smartfoni'), 4 ('Televizor LG OLED55CX', 1799.99, 'Televizori'); 5	
Data Output	Messages	Data Output	Messages
CREATE TABLE		INSERT 0 3	
Query returned successfully in 162 msec.		Query returned successfully in 67 msec.	

Query	Query History
1 CREATE INDEX proizvodi_ind ON PROIZVODI (id); 2 3	
Data Output	Messages
CREATE INDEX	
Query returned successfully in 66 msec.	

Slika 6-Prikaz kreiranja tabele “PROIZVODI” (prva slika) koja će biti klasterovana. Druga slika prikazuje popunjavanje tabele “PROIZVODI”, dok treća prikazuje kreiranje indeksa koji će se koristiti za klasterovanje.

Query	Query History
1	<b>CLUSTER</b> PROIZVODI <b>USING</b> proizvodi_ind;
2	
Data Output	Messages
CLUSTER	
	Query returned successfully in 65 msec.

Slika 7-Prikaz kreiranog klastera nad tabelom “PROIZVODI” korišćenjem indeksa proizvodi\_ind.

Query

Query History

1

2

3

4

SELECT

FROM

WHERE

tablename, indexname

pg\_indexes

tablename = 'proizvodi';

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

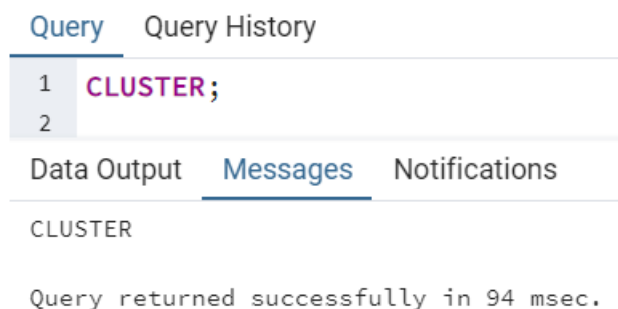
📈

	tablename name	indexname name
1	proizvodi	proizvodi_pkey
2	proizvodi	proizvodi_ind

Slika 8-Prikaz uspešno kreiranog clustera nad tabelom “PROIZVODI”. U rezultatu upita može se videti indeks “proizvod\_ind” koji je korišćenjen za kreiranje cluster-a.

Query	Query History	
1	<b>CLUSTER</b> proizvodi;	
2		
Data Output	Messages	Notifications
CLUSTER		
Query returned successfully in 105 msec.		

Slika 9- Prikaz kreiranja klastera bez eksplicitnog navođenja indeksa. U ovom slučaju PostgreSQL će klasterovati bazu podataka “PROIZVODI” korišćenjem prethodno kreiranog indeksa (proizvodi\_ind) automatski pošto je indeks već definisan i korišćenjen za klasterovanje, pa nema potrebe navoditi indeks ponovo. Ova naredba je korisna kada je potrebno ponovo klasterovati bazu podataka korišćenjem istog indeksa kao i pre.



Slika 10- Prikaz kreiranja klastera svih prethodno klasterovanih tabela.

Iako pojam cluster-a baze podataka u PostgreSQL-u prvenstveno predstavlja proces fizičke reorganizacije redova u tabeli, termin cluster se u PostgreSQL bazi podataka još može i protumačiti i kao grupa servera koji rade zajedno i predstavljaju jedan entitet [3]. U ovom kontekstu klasterizacija podrazumeva organizaciju podataka u grupe na osnovu njihovih karakteristika, pri čemu se kasnije te grupe dodeljuju odgovarajućim serverima u cluster-u. Ovakav klaster se koristi za postizanje visoke dostupnosti i skalabilnosti baze podataka, omogućavajući sistemu da se prilagodi rastućem opterećenju i očuva stabilnost u slučaju kvara pojedinačnih servera.[8]

Ovime je dat pregled osnovnih pojmova klastera u PostgreSQL bazi podataka, a u nastavku seminarskog rada biće opisano i prikazano kako se klasteri primenjuju za postizanje visoke dostupnosti i skalabilnosti, pružajući detaljan uvid u njihovu ulogu i funkcionalnost u PostgreSQL okruženju.

## 4. KORIŠĆENJE CLUSTERA ZA POSTIZANJE VISOKE DOSTUPNOSTI (ENG.HIGH AVAILABILITY)

Visoka dostupnost baze podataka (*eng.High availability*) se odnosi na sposobnost baze podataka da obrađuje zahteve bez prekida, čak i u slučaju kvara hardvera ili problema sa mrežom. Baza za koju se kaže da je visoko dostupna osigurava veće vreme dostupnosti eliminisanjem pojedinačnih kvarova, pouzdano se prebacujući sa jednog na drugi redundantni sistem i otkrivajući kvarove odmah, bilo da je reč o softverskom ili hardverskom kvaru [9]. Konfiguracija visoke dostupnosti uključuje postojanje glavnog čvora i rezervnih čvorova. Ukoliko dođe do otkaza glavnog čvora, jedan od rezervnih čvorova može automatski preuzeti njegovu ulogu. Podaci se repliciraju sa glavnog čvora na rezervne čvorove u realnom vremenu, osiguravajući da svi čvorovi imaju ažuriranu kopiju podataka. U slučaju kvara glavnog čvora, failover mehanizam automatski prebacuje opterećenje na jedan od rezervnih čvorova, minimizirajući prekid u radu sistema.[8]

Iako postoje razna rešenja za postizanje visoke dostupnosti PostgreSQL baze podataka, jedno od njih jeste korišćenje klastera. Kako je i opisano u prethodnom poglavlju, pojam klastera baze podataka može da se odnosi i na grupu servera (čvorova) koji rade zajedno kako bi postigli zajednički cilj, u ovom slučaju osiguravanje visoke dostupnosti. Klaster koji se koristi za obezbeđivanje visoke dostupnosti, kod PostgreSQL baze podataka, se naziva HA cluster (High Availability cluster) (Napomena: U PostgreSQL-u, termin visoka dostupnost podrazumeva da jedan klaster baze podataka omogućava da ceo sistem ima visoku dostupnost. Zato se koristi termin HA klaster, koji se odnosi na skup različitih instanci baza podataka koje mogu se mogu

nalaziti na većem broju servera koji sadrže iste podatke kako bi se obezbedila visoka dostupnost sistema). Kod HA cluster-a svi čvorovi koji postoje rade zajedno kao jedna virtuelna baza podataka, što znači da klijenti ne moraju znati koji je čvor glavni, a koji rezervni. Sistem za balansiranje opterećenja (*eng.load balancing*) može rasporediti upite na različite čvorove, čime se optimizuju performanse i pouzdanost. Svi ovi čvorovi imaju kopiju istih osnovnih podataka, a prilikom obrade upita bilo koji od servera može poslati odgovor, pri čemu korisnici ne znaju koji čvor je odgovorio na upit. Bilo koji čvor potencijalno može postati primarni čvor koji kod HA klastera predstavlja glavni čvor. Ovaj čvor prima sve promene koje su nastale u bazi podataka, uključujući i upise i izmene šeme, pa samim tim on uvek ima najaktuelniji skup podataka. On izvršava replikaciju ovih promena na ostalim instancama HA klastera na taj način što im šalje transakcije u listi ili tok formatu. Primarni čvor takođe može da obrađuje upite za čitanja, ali oni su obično raspoređeni između različitih čvorova u svrhu balansiranja opterećenja. Pored primarnog, postoji i sekundarni ili replikacioni čvor, koji prihvata ažuriranja od primarnog čvora i može se koristiti da obradi upite za čitanje podataka. Međutim, u zavisnosti od arhitekture, podaci koje sadrže replikacioni čvor možda neće biti u potpunosti ažurirani. Svaki HA klaster može da sadrži veći broj sekundarnih čvorova radi dodatne redundantnosti i balansiranja opterećenja. U slučaju kvara primarnog čvora, neki od sekundarnih čvorova postaje primarni čvor i on sada nadgleda sva ažuriranja baze podataka, a ovaj postupak je poznat pod nazivom preuzimanje (*eng.failover*). Administrator baze podataka može da pokrene ručno preuzimanje radi održavanja podataka, te se ova aktivnost naziva ručno prebacivanje (*eng.manual switchover*). U tom slučaju moguće je izvršiti povratak na originalni primarni čvor i taj postupak se naziva povratak (*eng.fallback*). U kontekstu HA cluster-a bitan pojam jeste i replikacija podataka koja generiše veći broj kopija originalnih podataka iz baze podataka. Drugim rečima, ona beleži sve izmene nastale u bazi podataka i prenosi ih svim čvorovima u HA klasteru. [8]

Nakon uspostavljanja osnovnih pojmova i razumevanja visoke dostupnosti u PostgreSQL-u, u nastavku će biti opisane i prikazane različite tehnike replikacije, kao i načini kreiranja i održavanja HA klastera.

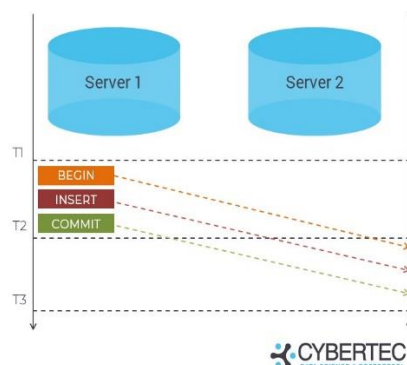
#### 4.1 Replikacija HA klastera

Replikacija predstavlja ključnu komponentu za postizanje visoke dostupnosti u klasterima. U HA klasterima, replikacija omogućava da se izvrši automatska sinhronizacija između više čvorova, obezbeđujući pritom da svi čvorovi imaju ažurirane kopije podataka.

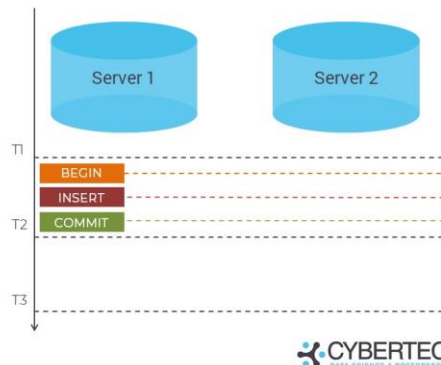
Kod PostgreSQL baze podataka postoje dva replikaciona režima i dva osnovna načina za sprovođenje replikacije. Glavni režimi replikacije su:

- Sinhrona replikacija: U ovom režimu, primarni čvor čeka potvrdu od najmanje jednog replikacionog čvora pre nego što potvrdi transakciju. Ovo garantuje da je baza konzistentna u celom HA klasteru u slučaju kvara. Konzistentnost eliminiše potencijalni gubitak podataka i ključna je za organizacije koje zahtevaju integritet transakcionih podataka, ali uvodi latenciju i može smanjiti protok [8].
- Asinhrona replikacija: Kod asinhronne replikacije, primarni čvor šalje ažuriranja replikacionim čvorovima bez čekanja na odgovor. On odmah potvrđuje uspešno izvršavanje transakcije nakon ažuriranja sopstvene baze podataka, smanjujući latenciju. Međutim ovaj pristup povećava šanse za gubitak podataka u slučaju neočekivanog failover-a. Ovaj režim replikacije je podrazumevani režim u PostgreSQL-u.[8]

## ASYNCHRONOUS REPLICATION



## SYNCHRONOUS REPLICATION



Slika 11- Prikaz asinhronog i sinhronog režima replikacije<sup>4</sup>

Algoritmi koji se koriste za implementaciju replikacije su:

- Replikacija bazirana na prenosu logova (log shipping): U ovoj metodi replikacije, primarni čvor asinhrono prenosi segmente fajlove koji sadrže WAL (Write-Ahead Log) logove, replikacionim čvorovima. Ovaj metod se ne može koristiti sinhrono jer se WAL fajlovi nagomilavaju tokom velikog broja transakcija. Primarni čvor kontinuirano beleži sve transakcije, ali replikacioni čvorovi obrađuju promene tek nakon što dobiju kopiju fajla. Ovo je dobar pristup za aplikacije koje su osetljive na latenciju i koje tolerišu gubitak podataka.[8]
- Streaming replikacija: Algoritam baziran na streaming replikaciji odmah prenosi svako ažuiranje čvorovima replikama. Primarni čvor ne mora da čeka da se transakcije nagomilaju u WAL-u pre nego što prenese ažuiranja. Ovo rezultira blagovremenijim ažuiranjima na replikama. Streaming može biti asinhron, što je podrazumevano ponašanje, ili sinhron. U oba slučaja, ažuiranja se odmah šalju replikama, s tim što kod sinhronog streaming-a primarni čvor čeka odgovor od replika pre nego što potvrdi izvršenje transakcije. [8]

Pored opisanih pojmova, još je bitno pomenuti i modele replikacije baza podataka u PostgreSQL-u, koji mogu biti: replikacija sa jednim čvorom (Single-Master Replication) i replikacija sa više glavnih čvorova (Multi-Master Replication). Kod replikacije sa jednim glavnim čvorom, promene koje su nastale u redovima tabela na glavnom serverskom čvoru se repliciraju na jedan ili više replikacijskih servera. Replikovane tabele na replikacijskim serverima ne smeju prihvatati nikakve promene (osim onih koje dolaze od glavnog čvora). Čak i ako dođe do promena, one se repliciraju nazad na glavni server. Kod replikacije sa više glavnih čvorova, promene u redovima tabela na više glavnih serverskih čvorova se repliciraju na odgovarajuće tabele u svakom drugom glavnom čvoru. U ovom modelu se često koriste šeme za rešavanje sukoba kako bi se izbegli problemi poput duplih primarnih ključeva. [10]

PostgreSQL podržava dve vrste replikacije: fizičku i logičku replikaciju. Fizička replikacija se vrši na nivou sistema datoteka ili diska, a može se omogućiti korišćenjem streaming replikacije ili log shipping-a, dok se logička replikacija bavi samom bazom podataka, tačnije tabelama i ostalim objektima unutar baze, kao i operacijama koje se izvršavaju nad njima i zasniva se na

<sup>4</sup> Izvor slike- <https://www.cybertec-postgresql.com/en/services/postgresql-replication/synchronous-asynchronous-replication/>

publisher/subscriber modelu. Logička replikacija koristi samo SQL naredbe kako bi kreirala repliku, dok se kod fizičke replikacije šalju bajtovi podataka sa glavnog servera koji su potrebni za kreiranje replike.

### Logička replikacija

PostgreSQL logička replikacija se koristi za repliciranje tabela sa glavnog servera na sekundarni (replikacioni) server. Logička replikacija se zasniva na publisher/subscriber modelu koji koristi dva važna pojma: publikaciju i pretplatu. Publikacija se definiše na glavnom serveru i naziva se publisher (izdavač). Samo jedna publikacija može postojati po bazi podataka, ali može biti više pretplatnika na tu publikaciju. Pretplata se definiše na sekundarnom serveru i naziva se subscriber (pretplatnik). Subscriber prima ažuriranja od publisher-a. Subscriber baze podataka se mogu koristiti kao izdavači za druge baze podataka definisanjem sopstvene publikacije [11]. Ova vrsta replikacije u PostgreSQL-u radi tako što prvo kreira snapshot podataka sa glavne baze podataka (publisher) i kopira ga na sekundarnu bazu podataka (subscriber). Ovaj snapshot uključuje sve postojeće podatke koji se nalaze u tabelama baze. Nakon što se početni podaci kopiraju, replikacija prelazi u režim sinhronizacije. U ovom režimu, sve promene koje se dogode na glavnoj bazi dok se podaci kopiraju, strimuju se na sekundarnu bazu kako bi se osigurala potpuna usklađenost. Kada je sinhronizacija završena, kontrola replikacije se vraća glavnom procesu primene, koji kontinuirano prenosi sve nove promene sa glavne baze na sekundarnu bazu u realnom vremenu. To znači da se svaka promena koja se dogodi na glavnoj bazi, poput promena insert, update i delete, odmah replicira na sekundarnu bazu[10]. Logička replikacija koristi streaming protokol za replikaciju na taj način što za svakog subscriber-a postoji logički slot, koji subscriber specificira prilikom povezivanja sa publisher-om. Ovaj replikacioni slot povećava fleksibilnost slanja WAL podataka. Slot predstavlja tok promena koje se mogu reprodukovati klijentu u onom redosledu u kom su napravljene na glavnom serveru. Svaki slot prenosi sekvencu promena iz jedne baze podataka i ima jedinstveni identifikator unutar PostgreSQL klastera. Slotovi su trajni, otporni na padove, i mogu postojati nezavisno od veze koja ih koristi. Više slotova može postojati za jednu bazu podataka, omogućavajući različitim korisnicima da primaju promene iz različitih tačaka u toku promena baze podataka. Svaki korisnik obično zahteva poseban slot. Logički replikacioni slot ne zna stanje primaoca, odnosno ne zna koliko podataka je svaki primaoc (receiver) već primio. Više različitih primaoca može koristiti isti slot, ali ne istovremeno. Kada jedan primaoc prestane da koristi slot, drugi primaoc može preuzeti korišćenje tog slota i nastaviće da prima promene od tačke gde je prethodni primaoc stao. Međutim, u bilo kom trenutku, samo jedan primaoc može aktivno da prima promene iz jednog slota, pa ako jedan primaoc koristi slot nijedan drugi primaoc ne može da koristi taj isti slot u tom trenutku. Logička replikacija je pogodna za visoku dostupnost u PostgreSQL klasterima jer omogućava kontinuiranu sinhronizaciju između glavnog servera i sekundarnih servera u realnom vremenu, čime obezbeđuje brz povratak u rad nakon kvara glavnog servera. Ovo je vrlo važno u kontekstu visoke dostupnosti jer minimizuje gubitak podataka i sistem čini doslednim.[11]

U nastavku je dat primer logičke replikacije korišćenjem publisher/subscriber metode. Za potrebe primera kreirane su dve instance servera, pri čemu jedan server predstavlja primarni server, odnosno publisher i on se pokreće na portu 5432, dok je drugi server sekundarni, tj. u slučaju logičke replikacije on predstavlja subscriber-a i pokreće se na portu 5433. Pre same logičke replikacije neophodno je podesiti nekoliko parametara. Najpre je potrebno proveriti da li je parametar wal\_level (značenje ovog parametra je opisano u prethodnom seminarskom radu "Backup \Restore kod PostgreSQL baze podataka") postavljen na vrednost logical, a parametri

max\_wal\_senders i max\_replication\_slots na default-nu vrednost 10 (slika 12). Ukoliko nisu potrebno je postaviti ih.

```
postgres=# show wal_level ;
wal_level
-----
logical
(1 row)

postgres=# show max_wal_senders;
max_wal_senders
-----
10
(1 row)

postgres=# show max_replication_slots;
max_replication_slots
-----
10
(1 row)
```

Slika 12 –Prikaz postavljenih parametara na odgovarajuće vrednosti

Nakon toga potrebno je kreirati korisnika za migraciju i dodeliti mu dozvolu za pristup bazama i tabelama.

```
postgres=# CREATE ROLE SaraKorisnik WITH REPLICATION LOGIN PASSWORD 'sarakorisnika';
CREATE ROLE
postgres=#
```

```
postgres=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO sarakorisnik;
GRANT
```

Slika 13- Prikaz kreiranja korisnika za migraciju na primarnom serveru

Sada je moguće koristiti logičku replikaciju. Kako bi bilo demonstrirano kako radi logička replikacija, na glavnom serveru je kreirana tabela “tabela\_osoba1” i popunjena je odgovarajućim podacima, a zatim je kreirana i publikacija na kojoj će sekundarni server moći da se pretplati korišćenjem odgovarajuće subskripcije. Na sekundarnom serveru kreirana je subskripcija, ali i tabela sa istim imenom kao i na glavnom serveru kako bi subskripcija mogla da se pretplati (prikazano na slikama 14 i 15).

```
postgres=# CREATE TABLE tabela_osoba1 (id SERIAL PRIMARY KEY, ime TEXT);
CREATE TABLE
postgres=# INSERT INTO tabela_osoba1 (ime) VALUES ('Sara');
INSERT 0 1
postgres=# CREATE PUBLICATION PrviPubOsoba1 FOR TABLE tabela_osoba1;
CREATE PUBLICATION
postgres=#
```

Slika 14- Prikaz kreiranja tabele “tabela\_osoba1” i publikacije na glavnom serveru



```
postgres=# CREATE TABLE tabela_osoba1 (id SERIAL PRIMARY KEY, ime TEXT);
CREATE TABLE
postgres=# create subscription PrviSubOsoba1 CONNECTION 'host=localhost port=5432 dbname=postgres user=sarakorisnik pass
word=sarakorisnika' PUBLICATION PrviPubOsoba1;
NOTICE: created replication slot "prvisubosoba1" on publisher
CREATE SUBSCRIPTION
```

Slika 15-Prikaz kreiranja tabele “tabela\_osoba1” i subskripcije na sekundarnom serveru

Uspešnost logičke replikacije proverava se na čvoru subscriber-u. Ukoliko se u tabeli koja je kreirana i nosi isti naziv kao i tabela na čvoru publisher-u, nalaze isti podaci, proces je uspešno izvršen. Ovo je prikazano na slici 16.

```
postgres=> select * from tabela_osoba1;
 id | ime
----+-----
  1 | Sara
(1 row)
```

Slika 16- Prikaz provere uspešne replikacije na sekundarnom serveru

```
postgres=> insert into tabela_osoba1 (ime) values ('Mila');
INSERT 0 1
postgres=> select * from tabela_osoba1;
 id | ime
----+-----
  1 | Sara
  2 | Mila
(2 rows)

postgres=> select * from tabela_osoba1;
 id | ime
----+-----
  1 | Sara
  2 | Mila
(2 rows)
```

Slika 17- Prikaz dodavanja novog podatka na primarni server i provera replikacije na sekundarnom serveru

Ovime je obezbeđeno da podaci u glavnom i sekundarnom serveru, koji predstavljaju klaster, budu sinhronizovani i konzistentni čime se postiže visoka dostupnost u tom klasteru.

## Fizička replikacija

Fizička replikacija kopira binarne podatke sa glavnog servera na replikacioni server, čime obezbeđuje visoku dostupnost u HA klasteru. Pošto funkcioniše na nivou datoteka, veoma je efikasna, ali je manje fleksibilna u poređenju sa logičkom replikacijom. Fizička replikacija osigurava da su replike tačne byte-for-byte kopije glavnog servera. Replikacioni, tj odredišni server koji sadrže kopije podataka sa glavnog servera se kod fizičke replikacije naziva standby server. U PostgreSQL-a postoje dve vrste standby servera: hot standby i cold standby. Hot standby predstavlja standby server koji se održava što je moguće ažurnijim sa primarnim serverom u realnom vremenu i omogućava klijentima izvršavanje samo za čitanje transakcija. Ovi serveri su dodati u PostgreSQL verziji 9, dok su pre njih postojali samo warm standby server, koji su slični hot standby serverima, samo što oni ne dozvoljavaju klijentima da se povežu dok rade kao replike. Cold standby server (Napomena:ovo ne predstavlja zvaničan termin) je obično standby server koji se ne pokreće sve dok ne dođe do failover-a. Pošto cold standby server nije aktivan, postoji šansa da kada se pokrene ovaj server, da će on prvo morati da primeni sve promene koje su nastale pre nego što bude mogao da prihvati klijentske konekcije. Kako PostgreSQL server, tokom normalnog rada generiše uređen niz WAL zapisa, fizička replikacija, u osnovi, predstavlja prenos ovih zapisa na drugu mašinu (standby server u ovom slučaju), i omogućavanje drugom postmaster-u koji tamo radi da prihvati i primeni ove zapise u svojoj lokalnoj bazi podataka. WAL zapisi su podeljeni na jednako velike fajlove koji se zovu WAL



segmenti ili WAL fajlovi (WAL segmenti su detaljnije opisani u drugom seminarskom radu “Backup \Restore kod PostgreSQL baze podataka”). Kako je već ranije, pomenuto fizička replikacija se ostvaruje korišćenjem streaming replikacije ili log-shipping-a [12]. U nastavku će biti opisano i prikazano kako se kreira fizička replikacija korišćenjem oba načina.

1. Log-shipping: Kod log-shipping-a primarni i standby server rade zajedno kako bi kreirali fizičku replikaciju, pri čemu primarni server radi u režimu kontinuiranog arhiviranje, dok standby server radi u režimu kontinuiranog oporavka, čitajući WAL fajlove sa primarnog servera (ovi pojmovi su opisani u drugom seminarskom radu “Backup \Restore kod PostgreSQL baze podataka”). Zato se log-shipping može opisati kao direktno premeštanje WAL zapisa sa jedne baze podataka na drugu. WAL fajlovi mogu lako i jeftino da se prenose na bilo koju udaljenost, pri čemu potrebna propusna moć za ovu tehniku varira u skladu sa brzinom transakcija na primarnom serveru. Log shipping je asinhron, odnosno WAL zapisi se prenose nakon potvrde transakcije, što znači da postoji šansa za gubitkom većeg broja podataka ukoliko dođe do otkaza primarnog servera. Preporučljivo je da primarni i standby server budu što sličniji, posebno sa aspekta baze podataka. To znači da moraju imati iste putanje za tabelarne prostore ukoliko se oni koriste. Na primer, ako se na primarnom serveru kreira novi tabelarni prostor, ista montažna tačka (lokacija u operativnom sistemu gde se fizički disk “montira”) mora biti kreirana i na svim standby serverima pre nego što se izvrši komanda. Takođe bitno je da hardveri budu kompatibilni; prenos sa 32-bitnog na 64-bitni sistem nije podržan. Standby server radi tako što kontinuirano primenjuje zapise koje dobija od primarnog servera. Server ulazi u standby režim kada detektuje prisustvo fajla standby.signal u svom direktorijumu podataka prilikom pokretanja. Server povlači WAL fajlove ili iz arhive koristeći restore\_command ili iz lokalnog pg\_wal direktorijuma. Prilikom pokretanja, server prvo obnavlja sve dostupne WAL zapise iz arhive, a zatim iz pg\_wal direktorijuma ako je potrebno. Postavljanje standby servera uključuje obnavljanje basebackup-a sa primarnog servera, kreiranje standby.signal fajla i konfigurisanje restore\_command-e za kopiranje fajlova iz WAL arhive. Za pripremu primarnog servera za standby servere, potrebno je podešavanje kontinuiranog arhiviranja ka direktorijumu arhive koji je dostupan sa standby servera. U nastavku je dat prikaz podešavanja primarnog i standby servera kako bi log shipping bio moguć [13].

Na primarnom serveru je potrebno konfigurisati kontinuirano arhiviranje u konfiguracionom fajlu, odnosno postaviti archive\_command parametar na odgovarajuću vrednost (prikazano na slici 18).

```
#archive_command = 'copy "%p" "D:\Arhivica\%f"'      # command to use to archive a WAL file
# placeholders: %p = path of file to archive
#               %f = file name only
# e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
```

Slika 18- Prikaz podešavanja parametara za log shipping na primarnom serveru

Na standby serveru neophodno je podesiti u, konfiguracionom fajlu, restore\_command komandu kako bi se WAL segmenti prebacili sa primarnog servera i primenili na standby server, i primary\_conninfo komandu koja definiše parametre za povezivanje standby servera sa primarnim serverom. Ovi koraci su prikazani na slici 19.

```
#restore_command = 'cp "D:\\Arhivica\\%f" %p'        # command to use to restore an archived WAL file
# placeholders: %p = path of file to restore
#               %f = file name only
# e.g. 'cp /mnt/server/archivedir/%f %p'

#primary_conninfo = 'host=127.0.0.1 port=5432 user=SaraKorisnik password=sarakorisnika options='-c wal_sender_timeout=5000''
#primary_slot_name = ''                             # replication slot on sending server
#hot_standby = on                                    # "off" disallows queries during recovery
# (change requires restart)
```

Slika 19- Prikaz podešavanja parametara za log shipping na standby serveru. Sa slike se može primetiti da se kao korisnik navodi korisnik koji je kreiran na slici 13.

2. Streaming replikacija: Streaming replikacija omogućava da standby server bude ažuriraniji nego što je to moguće sa log shipping metodom. Standby se povezuje sa primarnim serverom, koji strimuje WAL zapise direktno na standby dok se generišu, bez čekanja da se WAL fajl popuni. Streaming replikacija je podrazumevano asinhrona što znači da postoji mali vremenski razmak između potvrđivanja transakcije na primarnom serveru i prikazivanja tih promena na standby serveru. Ovaj razmak je manji nego kod log shipping-a, ali samo pod uslovom da je standby server dovoljno moćan da prati opterećenje. Ukoliko se streaming replikacija koristi bez kontinuiranog arhiviranja fajlova, server mode će reciklirati stare WAL segmente pre nego što ih standby primi. U tom slučaju, standby server mora da se ponovo inicijalizuje iz novog backup-a. Ovo je moguće izbeći postavljanjem parametra `wal_keep_size` na veću vrednost ili konfigurisanjem replikacionog slot-a. Ukoliko je WAL arhiviranje podešeno, ove mere nisu potrebne jer standby može koristiti arhivu za nadoknadu [13].

U nastavku je dat primer konfigurisanja streaming replikacije. U sistemu postoje dva servera: primarni i standby. Primarni server se pokreće na portu 5432, a standby na portu 5433. Konfiguracija ove replikacije započinje odgovarajućim podešavanjima na primarnom serveru. Pre svega, u okviru konfiguracionog parametra primarnog servera neophodno je postaviti određene parametre na odgovarajuće vrednosti. Ovo se, pre svega odnosi na parametar `listen_addresses` koji se postavlja na vrednost `"*"`, što znači da server sada prihvata i udaljene konekcije. Važni parametri, koji se postavljaju jesu još i parametar `archive_mode` na vrednost `"on"`, `wal_level` na vrednost `"replica"`, `max_wal_senders` i `wal_keep_segments` na odgovarajuće vrednosti. U okviru komande za arhiviranje neophodno je postaviti putanju do direktorijuma gde će se čuvati WAL segmenti. Ove postavke su prikazane na slici 20.

```
listen_addresses = '*'
wal_level = replica
max_wal_senders = 3
wal_keep_segments = 64
archive_mode = on
archive_command = 'copy "%p" "D:\Arhivica\%f"'
```

Slika 20- Podešavanje konfiguracionog fajla primarnog servera

Zatim se kreira replikacioni korisnik koji uspostavlja vezu između primarnog i standby servera i omogućava razmenu podataka između ova dva servera. Kreiranje replikacionog korisnika je prikazano na slici 21.

```
postgres=# CREATE USER sara WITH REPLICATION PASSWORD 'sara';
CREATE ROLE
postgres=#
```

Slika 21- Prikaz kreiranja replikacionog korisnika

Dalja podešavanja uključuju konfiguraciju `pg_hba.conf` fajla primarnog servera (prikazano na slikama 22).

```
host      replication      sara          127.0.0.1/32      trust
```

Slika 22- Prikaz konfiguracije `pg_hba.conf` fajla na primarnom serveru. Ova konfiguracija omogućava standby serveru da uspostavi sigurnu vezu sa primarnim serverom kako bi

preuzimao WAL segmente i primenjivao ih na svoje podatke. Samo ovlašćeni korisnik, što je u ovom slučaju korisnik “sara”, može pristupiti primarnom serveru i preuzeti potrebne podatke, čime se osigurava sigurnost i integritet podataka tokom replikacije.

Kada je primarni server konfigurisan, potrebno je sve podatke sa primarnog servera preneti na standby server. Za to se koristi pg\_basebackup alat koji pravi backup podataka sa primarnog servera na sekundarni (prikazano na slici 23).

```
C:\Program Files\PostgreSQL\16\bin>pg_basebackup -h 127.0.0.1 -D "C:\Program Files\PostgreSQL\16-3\data" -U sara -v -P --wal-method=stream
pg_basebackup: initiating base backup, waiting for checkpoint to complete
2024-06-24 13:49:42.313 CEST [7164] LOG:  checkpoint starting: force wait time
2024-06-24 13:49:42.399 CEST [7164] LOG:  checkpoint complete: wrote 0 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.001 s, sync=0.001 s, total=0.006 s; sync files=0, longest=0.000 s, average=0.000 s; distance=16383 kB, estimate=16383 kB; lsn=0/A5000000, redo lsn=0/A5000000
1 file(s) copied.
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/A5000028 on timeline 2
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot "pg_basebackup_18652"
176970/176970 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/A5000138
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
1 file(s) copied.
1 file(s) copied.
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
```

Slika 23- Prikaz kreiranja kopije podataka sa primarnog na standby server. U okviru komande se navodi putanja do direktorijuma gde će backup biti smešten i korisničko ime koje se koristi za povezivanje na primarni server. Metoda “—wal-method=stream” označava da će se WAL zapisi preuzimati u realnom vremenu dok se generišu na primarnom serveru.

Kada se kreira backup, u konfiguracionom fajli standby servera treba podesiti sledeće parametre:

```
hot_standby | = 'on'
primary_conninfo = 'host=127.0.0.1 port=5432 user=sara
password=sara'
restore_command = 'cp "D:\\Arhivica\\%f" %p'
```

Slika 24-Podešavanje konfiguracionog fajla standby servera. Parametar hot\_standby omogućava standby serveru da prihvata read-only upite dok traje proces replikacije, dok primary\_conninfo definiše informacije koje su potrebne sekundarnom serveru kako bi se povezao sa primarnim. restore\_command se koristi za povratak WAL zapisa na sekundarnom serveru.

Nakon ovoga streaming replikacija između servera je uspostavljena.

```
postgres=# select username, application_name, client_addr, state, sync_state from pg_stat_replication;
 username | application_name | client_addr | state   | sync_state 
-----+-----+-----+-----+-----
 sara      | walreceiver      | 127.0.0.1   | streaming | async
(1 rows)
```

Slika 25- Prikaz provere streaming replikacije na primarnom serveru. Nakon ovoga moguće je izvršiti verifikaciju dodavanjem podataka na primarni server i njihov prikaz na sekundarnom serveru.

```
postgres=# CREATE TABLE test_replikacija (id SERIAL PRIMARY KEY, value TEXT);
CREATE TABLE
postgres=# INSERT INTO test_replikacija (value) VALUES ('Test 1'), ('Test 2'), ('Test 3');
INSERT 0 3
```

```
postgres=# select * from test_replikacija;
 id | value 
----+-----
  1 | Test 1
  2 | Test 2
  3 | Test 3
(3 rows)
```

Slika 26- Prikaz dodavanja tabele za testiranje replikacije na primarnom serveru i prikaz podataka na sekundarnom serveru

## 4.2 Cluster rešenja koja se koriste za postizanje visoke dostupnosti

Kako je već naglašeno, klasterizacija zauzima značajno mesto u postizanju visoke dostupnosti, posebno zato što ovakva organizacija servera pruža garantovanje balansiranja opterećenja sistema, i što omogućava da prilikom kvara nekog od servera, drugi server preuzme njegovu ulogu, tj. omogućava failover mehanizam.

Za kreiranje klastera PostgreSQL baze podataka danas postoji veliki broj alata, kao što su PgCluster, PgPool-II, RubyRep, Bucardo, Postgres-XC, Citus, Postgres-XL. Pored ovih alata postoje i alati koji se mogu koristiti online i na Cloud platformama, poput ClusterControl alata i Patroni alata.[3]

PgCluster predstavlja rešenje za PostgreSQL klustere koje omogućava višestruku replikaciju i multi-master konfiguraciju koja je bazirana na shared-nothing arhitekturi, što znači da svaki čvor u sistemu funkcioniše nezavisno. Glavna prednost PgCluster-a je u sprečavanju gubitka podataka i prekida rada sistema. Kako su svi čvorovi u PgCluster-u sinhronizovani, bilo koji od servera može pasti bez rizika od gubitka podataka. Ovaj alat omogućava visoku dostupnost korišćenjem sinhronizovane multi-master konfiguracije, koja omogućava da se svaka promena koja nastane na jednom masteru odmah odražava i na sve ostale mastere i to se postiže bez promene načina rukovanja transakcijama. Pri obezbeđivanju visoke dostupnosti ovaj alat uključuje i servere za balansiranje opterećenja. [3]

PgPool-II predstavlja softver koji povezuje PostgreSQL server sa PostgreSQL klijentima. On komunicira sa backend i frontend delom PostgreSQL-a i uspostavlja vezu između njih. Kao rezultat toga, aplikacija baze podataka (frontend) smatra PgPool-II PostgreSQL serverom, dok server (backend) PgPool-II smatra jednim od svojih klijenata. Upravo zbog ove transparentnosti PgPool-II alata i serveru i klijentu, postojeća aplikacija baze podataka može se koristiti sa njim skoro bez modifikacije njenih izvornih kodova. Ovaj alat takođe obezbeđuje replikaciju podataka, balansiranje opterećenja, paralelno izvršavanje upita, automatski failover, a može se koristiti i u kombinaciji sa streaming replikacijom. [3]

RubyRep predstavlja više bazni, asinhroni, multi-master, višeplatformski sistem replikacije koji ne podržava DDL, korisnike ili dozvole zbog trigera. Osnovni cilj ovog alata jeste da upotrebu i upravljanje učini što jednostavnijim. Ključne karakteristike RubyRep alata jesu jednostavna konfiguracija, jednostavna instalacija i to što platforma i dizajn tabela nisu međusobno zavisni. [3]

Bucardo predstavlja asinhronu replikaciju na nivou reda sa kaskadnom master-slave konfiguracijom koristeći trigere i redove baze podataka, kao i asinhronu master-master replikaciju sa triggerima i prilagodljivim rešavanjem konflikata. Bucardo zahteva posvećenu bazu podataka i funkcioniše kao Perl daemon koji komunicira sa tom bazom i sa svim ostalim bazama u procesu

replikacije. Podržava dva režima rada: multimaster i multislave. Glavne karakteristike Bucardo alata jesu što omogućava balansiranje opterećenja, replikaciju u segmentima i replikaciju po zahtevu. Takođe, slave serveri nisu ograničeni pravilima. [3]

Postgres-XC je open-source projekat koji ima za cilj da pruži PostgreSQL klaster rešenje koje je skalabilno za upis, sinhrono, simetrično i transparentno. To je skup čvrto povezanih komponenti baze podataka koje se mogu instalirati na više mašina (fizičkih ili virtuelnih). Izraz “skalabilno za upis” se odnosi na sposobnost da se skaliira kada je reč o jednom serverskom sistemu. Postgres-XC može biti konfigurisan sa više serverskih sistema nego što je poželjno i može efikasno upravljati većim brojem upisa (SQL naredbi za ažuriranje). Postgres-XC se može podešavati da radi na više servera, dok se podaci u okviru njega čuvaju na distribuiran način, tj. particionisani ili replikovani, u zavisnosti od opcije koja je izabrana za svaku tabelu. Prilikom izvršavanja upita, Postgres-XC određuje gde se željeni podaci nalaze i šalje upite serverima koji sadrže te podatke. [3]

Citus je zamena za PostgreSQL koja uključuje mogućnosti visoke dostupnosti poput automatskog shardovanja i replikacije. Citus particioniše bazu podataka i replicira više kopija svakog shard-a među običnim mašinama u klasteru. Citus bez problema usmerava sve upise ili upite ka jednom od drugih čvorova koji sadrži repliku pogođenog shard-a, u slučaju da neki čvor u klasteru postane nedostupan. [3]

Postgre-XL je sistem za klasterovanje koji se zasniva na deljenju resursa koji podržava višemaster konfiguraciju, omogućavajući transparentnu distribuciju tabele preko kolekcije čvorova i izvršavanje upita na tim čvorovima paralelno. Takođe sadrži i Global Transaction Manager (GTM) komponentu koja pruža globalno konzistentan pogleda na klaster. [3]

ClusterControl je softver razvijen od strane kompanije Severalnines koji služi kao orkestracijski sloj za operacije raznih baza podataka poput MySQL-a, MariaDB baze podataka, PostgreSQL-a, Redis-a i drugih, kako na lokalnim serverima, tako i u cloud-u i hibridnim okruženjima. Ima tri opcije: besplatnu Community licencu, kao i dve komercijalne opcije: Advanced i Enterprise. Community licenca omogućava implementaciju, replikaciju i osnovno nadgledanje, dok ostale dve verzije dolaze sa naprednijim funkcijama za implementaciju i nadgledanje. Sve operacije mogu biti konfigurisane i upravljane putem grafičkog interfejsa, komande linije ili API-ja. ClusterControl nije vezan za osnovnu infrastrukturu, što korisnicima omogućava da implementiraju nove baze podataka i uvoze postojeće u višestrukim okruženjima- sopstvenim i/ili na njihovim cloud nalogima. [14]

Patroni je open-source Python paket koji upravlja konfiguracijom Postgres baze podataka. Može se konfigurisati da obavlja zadatke poput replikacije, backup-a i vraćanja podataka. U okviru njega, svi čvorovi Postgres sistema koriste etcd, distribuirani sistem, otporan na greške koji se koristi za čuvanje stanja Postgres klastera, kako bi održavali Postgres klaster u funkciji. [15]

Za kreiranje HA klastera, u okviru ovog seminarskog, iskorišćen je PgPool-II alat. PgPool obezbeđuje visoku dostupnost u klasteru procesom poliranja konekcija i automatskom podrškom za failover mehanizam. On se postavlja kao posrednik između aplikacija i servera ili klastera, odnosno PgPool alat je proxy. On prihvata konekcije aplikacija i distribuira ih između dostupnih servera u klasteru. U okviru PostgreSQL baze podataka, klaster može biti implementiran i svrstan u četiri konfiguracije: 1) Jedan PgPool i jedan server baze podataka koji se nalaze na istom računaru, 2) Horizontalna konfiguracija u okviru koje se svaki server nalazi na zasebnom

računaru 3) Vertikalna konfiguracija gde se svi serveri baza podataka nalaze na jednom računaru i 4) Hibridna konfiguracija koja predstavlja kombinaciju horizontalne i vertikalne konfiguracije gde postoje najmanje dva računara unutar klastera i više od jednog servera sa bazom podataka na računaru unutar klastera. Za PgPool postoje samo dve opcije: da se koristi samo jedan PgPool čvor, što predstavlja podrazumevanu implementaciju ili da se koristi horizontalna konfiguracija, odnosno da se svaki PgPool čvor nalazi na zasebnom računaru [19]. Za potrebe seminarskog, kada je reč o konfiguraciji PostgreSQL klastera, iskorišćena je vertikalna konfiguracija, odnosno kreirana su dva PostgreSQL servera na istom računaru, pri čemu se jedan pokreće na portu 5432, a drugi se pokreće na portu 5433. Za konfiguraciju PgPool alata iskorišćena je podrazumevana konfiguracija, odnosno kreiran je jedan PgPool čvor. Kako PgPool alat nije dostupan na Windowsu, sva neophodna konfiguracija (i konfiguracija PostgreSQL servera i konfiguracija PgPool čvora), obavljena je na virtualnoj mašini (Oracle Virtual Box), koja koristi Linux operativni sistem kao glavni.

Nakon što su instalirani serveri PostgreSQL baze podataka i PgPool-II alat neophodno je podesiti streaming replikaciju između servera. Čvor koji radi na portu 5432 postavljen je za primarni server, dok je čvor koji radi na portu 5433 konfigurisan da bude standby server. Konfiguracija parametara na primarnom i sekundarnom serveru se vrši na isti način kao i na Windows operativnom sistemu kako je i opisano u poglavlju 4.1. Naredne slike prikazuju konfiguraciju primarnog i standby servera za streaming replikaciju.[20]

```
postgres=# create user replicatorSarica with replication password 'saricaa';
CREATE ROLE
postgres=#
```

Slika 27- Prikaz kreiranja korisnika na primarnom serveru preko koga će se povezati standby server za strimovanje. Ovaj korisnik mora da ima uloga REPLICATION.

Nakon toga potrebno je podesiti odgovarajuće parametre u konfiguracionom fajlu (kao i na slici 20). Ova podešavanja prikazana su na slici 28.

```
postgres=# alter system set wal_level to 'replica';
ALTER SYSTEM
postgres=# alter system set archive_mode to 'ON';
ALTER SYSTEM
postgres=# alter system set max_wal_senders to '5';
ALTER SYSTEM
postgres=# alter system set wal_keep_segments to '10';
ALTER SYSTEM
postgres=# alter system set listen_addresses to '*';
ALTER SYSTEM
postgres=# alter system set hot_standby to 'ON';
ALTER SYSTEM
postgres=# alter system set archive_command to 'test ! -f /mnt/server/archivedir/%f && c
p %p /mnt/server/archivedir/%f';
ALTER SYSTEM
postgres=#
```

Slika 28–Prikaz postavljanja parametra na primarnom serveru

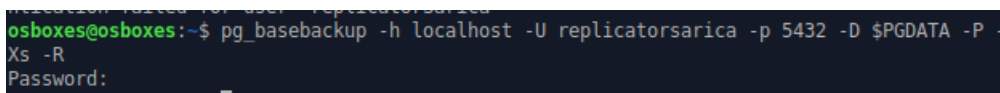
Na primarnom serveru je još neophodno dozvoliti konekcije za replikaciju sa standby servera, dodavanjem sledeće linije u okviru pg\_hba.conf fajla (prikazano na slici 29).

```
host      replication      replicatorsarica_127.0.0.1/32      md5
```

Slika 29– Prikaz konfiguracije pg\_hba.conf fajla na primarnom serveru



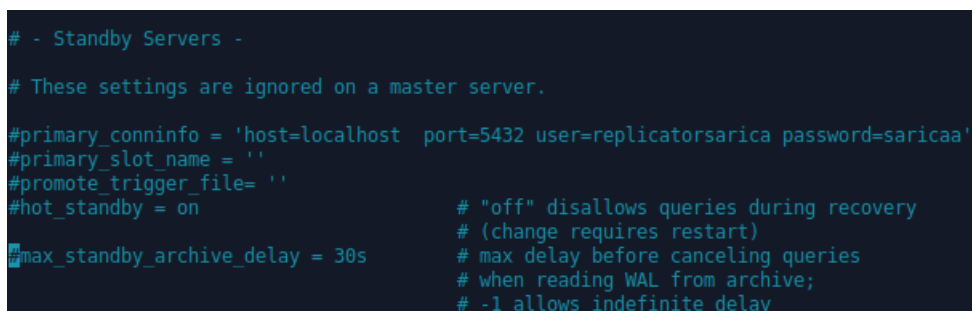
Kada se podesi primarni server, konfigurirše se standby server ali se pre toga kreira backup kopija podataka sa primarnog servera, u ovom slučaju korišćenjem pg\_basebackup alata (kao i na slici 23) , koja pomaže u strimovanju podataka putem procesa slanja WAL zapisa sa glavnog servera na standby server. Izvršenje ove komande prikazano je na slici 30.



```
osboxes@osboxes:~$ pg_basebackup -h localhost -U replicatorsarica -p 5432 -D $PGDATA -P -Xs -R
Password:
```

Slika 30–Prikaz kreiranja backup-a podataka sa primarnog servera

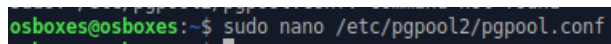
I na standby serveru je potrebno podesiti parametre konfiguracionog fajla (na isti način kao i na slici 24).



```
# - Standby Servers -
# These settings are ignored on a master server.
#primary_conninfo = 'host=localhost port=5432 user=replicatorsarica password=saricaa'
#primary_slot_name = ''
#promote_trigger_file= ''
#hot_standby = on
# \"off\" disallows queries during recovery
# (change requires restart)
#max_standby_archive_delay = 30s
# max delay before canceling queries
# when reading WAL from archive;
# -1 allows indefinite delay
```

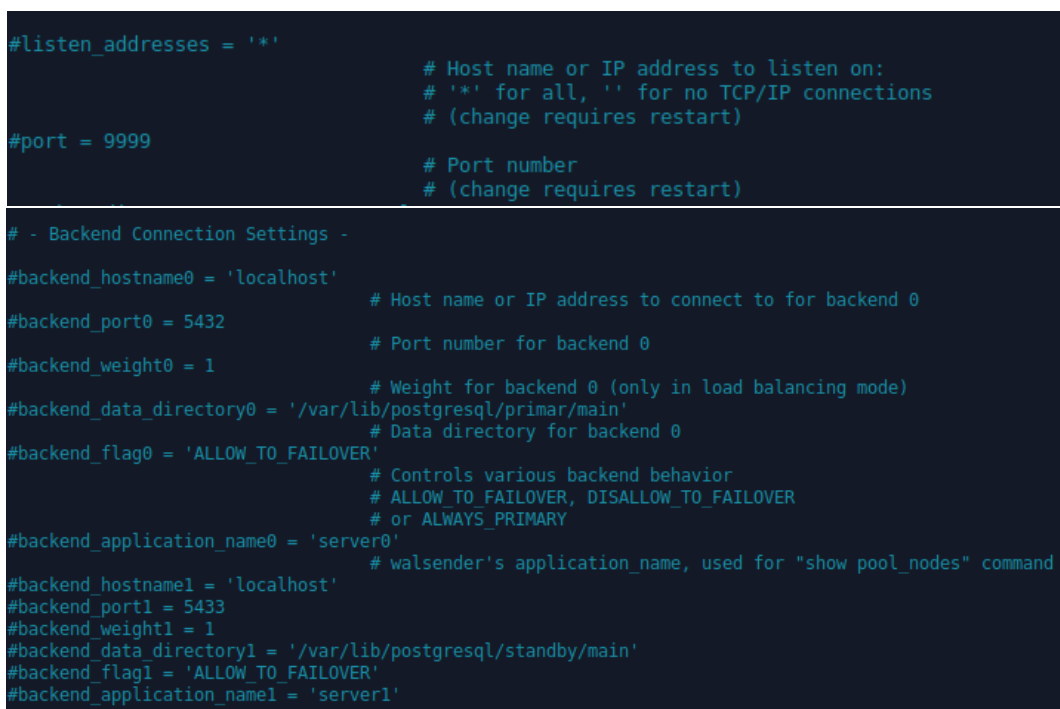
Slika 31–Prikaz podešavanja konfiguracionog fajla standby servera

Nakon ovih koraka vrši se konfiguracija PgPool-a. Ova podešavanja se vrše unutar konfiguracionog fajla PgPool-II alata kome se pristupa na način na koji je prikazano na slici 32, dok su izmene koje nastaju u konfiguracionom fajlu prikazane na slici 33.



```
osboxes@osboxes:~$ sudo nano /etc/pgpool2/pgpool.conf
```

Slika 33–Prikaz pristupanja konfiguracionom fajlu PgPool alata.



```
#listen_addresses = '*'
# Host name or IP address to listen on:
# '*' for all, '' for no TCP/IP connections
# (change requires restart)
#port = 9999
# Port number
# (change requires restart)

# - Backend Connection Settings -
#backend_hostname0 = 'localhost'
# Host name or IP address to connect to for backend 0
#backend_port0 = 5432
# Port number for backend 0
#backend_weight0 = 1
# Weight for backend 0 (only in load balancing mode)
#backend_data_directory0 = '/var/lib/postgresql/primar/main'
# Data directory for backend 0
#backend_flag0 = 'ALLOW_TO_FAILOVER'
# Controls various backend behavior
# ALLOW_TO_FAILOVER, DISALLOW_TO_FAILOVER
# or ALWAYS_PRIMARY
#backend_application_name0 = 'server0'
# walsender's application_name, used for \"show pool_nodes\" command
#backend_hostname1 = 'localhost'
#backend_port1 = 5433
#backend_weight1 = 1
#backend_data_directory1 = '/var/lib/postgresql/standby/main'
#backend_flag1 = 'ALLOW_TO_FAILOVER'
#backend_application_name1 = 'server1'
```

```

#-----
# STREAMING REPLICATION MODE
#-----

# - Streaming -

#sr_check_period = 10
# Streaming replication check period
# Disabled (0) by default

#sr_check_user = 'replicatorsarica'
# Streaming replication check user
# This is necessary even if you disable streaming
# replication delay check by sr_check_period = 0

#sr_check_password = 'saricaa'
# Password for streaming replication check user
# Leaving it empty will make Pgpool-II to first look for the
# Password in pool_passwd file before using the empty password

#sr_check_database = 'postgres'

#-----
# HEALTH CHECK GLOBAL PARAMETERS
#-----

#health_check_period = 10
# Health check period
# Disabled (0) by default

#health_check_timeout = 20
# Health check timeout
# 0 means no timeout

#health_check_user = 'postgres'

```

Slika 34 –Prikaz konfiguracije PgPool-II alata

Na slici 34 može se videti da je najpre potrebno podesiti parametar `listen_addresses` tako da prihvata sve dolazne konekcije, a zatim i port na kome PgPool-II osluškuje te dolazne konekcije. Nakon toga, navode se imena hostova za svaki server, i u ovom slučaju imena hosta oba servera, i primarnog i standby, imaju vrednost `localhost`, jer se oba servera nalaze na istoj lokalnoj mašini. Pored toga, navode se i njihovi portovi (port 5432 za primarni server, port 5433 za standby server), putanje do podataka svakog PostgreSQL servera, težina za balansiranje opterećenja između servera (vrednost 1 označava da će svaki server dobiti jednak udeo saobraćaja), imena aplikacije za svaki server. Ukoliko je potrebno obezbediti automatski failover parametar `backend_flag` se postavlja na vrednost “ALLOW TO FAILOVER”, čime se nalaže PgPool-u da u slučaju pada jednog servera on preusmeri saobraćaj automatski na drugi server.

Kada se navedu prethodno pomenuti parametri, omogućava se replikacija postavljanjem prethodno kreiranog korisnika za proveru replikacije u okviru parametra ‘`sr_check_user`’. Podešavanja replikacije zahtevaju i podešavanja šifre korisnika (‘`sr_check_password`’), kao i vremenskog intervala za proveru statusa replikacije u okviru parametra ‘`sr_check_period`’. Za postizanje visoke dostupnosti od značaja su i health check global parametri. Parametar ‘`health_check_period`’ određuje vremenski interval između health provera servera. Redovne provere omogućavaju da se problemi sa serverom brzo detektuju. Parametar ‘`health_check_timeout`’ predstavlja vreme koje PgPool-II čeka da završi health proveru, i u ovom slučaju je postavljeno na 20 sekundi. Pored ovih, za health provere, potrebno je odrediti i postaviti i korisnika koji se koriste za ove provere (parametar ‘`health_check_user`’). Za automatsko omogućavanje balansiranja opterećenja neophodno je postaviti parametar `load_balancing_mode` na vrednost ‘on’ (prikazano na slici 35).



```
#-----
# LOAD BALANCING MODE
#-----

#load_balance_mode = on
                                # Activate load balancing mode
                                # (change requires restart)
```

Slika 35 – Prikaz postavljanja parametra load\_balancing\_on kako bi se aktiviralo balansiranje opterećenja

Kada se postave svi ovi parametri moguće je pokrenuti PgPool-II alat.(Prikazano na slici 36)

```
osboxes@osboxes:~$ sudo systemctl start pgpool2
[sudo] password for osboxes:
```

Slika 36–Pokretanje PgPool-II alata

Psql se koristi za povezivanje na PgPool-II, kako bi se verificovala konfiguracija i PostgreSQL status.

```
osboxes@osboxes:~$ sudo -u postgres psql -p 9999 -h localhost -c 'SHOW pool_nodes;'
-[ RECORD 1 ]-----+-----
node_id              | 0
hostname             | localhost
port                | 5432
status              | up
pg_status            | up
lb_weight            | 0.500000
role                | primary
pg_role             | primary
select cnt          | 0
load balance node    | false
replication_delay    | 0
replication_state    | 
replication_sync_state | 
last_status_change   | 2024-06-21 17:05:48
-[ RECORD 2 ]-----+-----
node_id              | 1
hostname             | localhost
port                | 5433
status              | up
pg_status            | up
lb_weight            | 0.500000
role                | standby
pg_role             | standby
select cnt          | 0
load balance node    | true
replication_delay    | 0
replication_state    | streaming
replication_sync_state | async
last_status_change   | 2024-06-21 17:05:48
```

Slika 37–Prikaz verifikacije konfiguracije. Sa slike se može videti da je PgPool-II uspešno povezan sa serverima PostgreSQL baze podataka.

Ukoliko primarni server otkáže (kao što je prikazano na slici 38), PgPool-II odmah detektuje failover i postavlja standby server za primarni server(prikazano na slici 39).

```
osboxes@osboxes:~$ sudo systemctl stop postgresql@primar-main
```

Slika 38–Prikaz simulacije otkazivanja primarnog servera

```

osboxes@osboxes:~$ sudo -u postgres psql -p 9999 -h localhost -c 'SHOW pool_nodes;'
-[ RECORD 1 ]-----+
node_id           | 0
hostname          | localhost
port              | 5432
status            | down
pg_status         | down
lb_weight         | 0.500000
role              | standby
pg_role           | unknown
select_cnt        | 0
load_balance_node | false
replication_delay | 0
replication_state  |
replication_sync_state |
last_status_change | 2024-06-21 18:44:52
-[ RECORD 2 ]-----+
node_id           | 1
hostname          | localhost
port              | 5433
status            | up
pg_status         | up
lb_weight         | 0.500000
role              | primary
pg_role           | primary
select_cnt        | 0
load_balance_node | true
replication_delay | 0
replication_state  |
replication_sync_state |
last_status_change | 2024-06-21 18:44:52

```

Slika 39–Prikaz automatskog postavljanja standby servera od strane PgPool-II alata kao primarnog servera u slučaju failover-a. Može se primetiti da je primarni server sada ugašen, ali da sada on ima ulogu standby servera.

Pored svih ovih funkcionalnosti, PgPool-II alat ima i sposobnost donošenja odluke o tome koji zahtev treba da bude upućen kom serveru u zavisnosti od vrste naredbe koja formira zahtev. Upiti koji sadrže SELECT, UPDATE, INSERT, CREATE, ALTER, DROP, COPY i slične naredbe se prosleđuju primarnom serveru, kao i sve transakcione naredbe poput setovanja, započinjanja i završavanja transakcije, VACUUM naredbe i sekvencijalne funkcije. Naredbe kao što su SET, DISCARD, DELOCATE ALL se šalju i primarnom i standby serveru, a naredbe poput EXPLAN, ANALYZE i SHOW, mogu da se pošalju na oba servera, ali ukoliko je omogućeno balansiranje opterećenja i primarni server normalno funkcioniše, ovakvi upiti će biti poslati njemu.

Iz svega opisanog u ovom poglavlju može se primetiti, da klasteri kreirani uz pomoć odgovarajućih alata kao što je PgPool-II, obezbeđuju visoku dostupnost kombinovanjem replikacije podataka, balansiranja opterećenja i failover mehanizma. Ove metode osiguravaju da sistem ostane operativan i dostupan čak i slučaju kvara pojedinačnih komponenti, čime se minimizuju prekidi u radu i gubitak podataka.

## 5.KORIŠĆENJE CLUSTER-A ZA POSTIZANJE SKALABILNOSTI

“Skalabilnost baze podataka se odnosi na sposobnost baze da efikasno upravlja rastućim količinama podataka, brojem korisnika i zahtevima, bez gubitka performansi i dostupnosti.” Skalabilna baza podataka se suočava sa izazovima servera tako što se prilagođava rastućim zahtevima putem dodavanja resursa, kao što su hardver ili softver, optimizacijom dizajna ili konfiguracije ili primenom kombinovanih strategija. [16] Skalabilnost baze podataka je jako važna u aplikacijama koje trebaju da upravljaju velikom količinom podataka kao što su e-

trgovina, društvene mreže i finansije. Skalabilan dizajn baze podataka može pomoći kompanijama da izbegnu “usko grlo” (*eng.bottlenecks*), smanje vreme nedostupnosti i obezbede pozitivno korisničko iskustvo čak i pri povećanju zahteva [16]. Klasteri se mogu koristiti za postizanje skalabilnosti jer oni omogućavaju raspodelu podataka i opterećenja preko više servera, što bazi dozvoljava da raste dodavanjem novih servera u klaster kako se povećava potražnja i poboljšava performanse sistema.

Kod svih baza podataka, pa tako i kod PostgreSQL-a, postoje dve osnovne vrste skalabilnosti [16]:

- Vertikalna skalabilnost, poznata i kao vertikalno skaliranje (*eng.scale-up*), uključuje povećanje resursa pojedinačnog sistema dodeljenih bazi podataka na jednoj vertikalno integrisanoj mašini, kao što su CPU ili broj CPU-ova, memorija i diskovi.
- Horizontalna skalabilnost, poznata još i kao horizontalno skaliranje (*eng.scale-out*), se postiže dodavanjem više pojedinačnih mašina na neki način, stvarajući klaster za rad baze podataka. Različite baze podataka ovo rešavaju na različiti načine, a PostgreSQL kao relaciona baza podataka, koristi različite strategije poput replikacije podataka, ali i strategiju shard-ovanja samih podataka (biće opisano u nastavku poglavlja) [21]. PostgreSQL koristi i fizičku i logičku replikaciju (detaljno opisane u poglavlju 4.1). Fizička replikacija se koristi (prvestveno streaming replikacija), na isti način kako je već opisano, jer ona podrazumeva kopiranje celokupnog stanja podataka sa jednog primarnog servera na standby servere, što omogućava distribuciju opterećenja između većeg broja servera, čime se osigurava skalabilnost. S druge strane, logička replikacija se koristi za postizanje horizontalne skalabilnosti jer omogućava distribucija selektivnih podataka na veći broj servera putem replikacije SQL naredbi koje menjaju podatke. Na slikama 40–49 je prikazano kako se koristi logička replikacija za postizanje skalabilnosti. Replikacija se vrši između dva servera koja su kreirana i konfigurisana u potpoglavlju 4.2 na virtualnoj mašini.

#### Logička replikacija jedne tabele:

Kreiranje tabele za replikaciju i publikacije na primarnom serveru (opisano u potpoglavlju 4.1):

```
postgres=# create table tabelal (a int primary key);
CREATE TABLE
postgres=# insert into tabelal (a) select generate_series(1, 100);
INSERT 0 100
postgres=# create publication publ for table tabelal;
CREATE PUBLICATION
```

Slika 40–Kreiranja publikacije na primarnom serveru

Kreiranje tabele za replikaciju i pretplate na replikacionom serveru (opisano u potpoglavlju 4.1):

```
postgres=# create table tabelal (a int primary key);
CREATE TABLE
postgres=# create subscription sub1 connection 'host=localhost port=5432 dbname=postgres' publication publ;
NOTICE: created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=# select count(*) from tabelal;
 count
-----
    100
(1 row)
```

Slika 41–Kreiranje pretplate na replikacionom serveru

#### Logička replikacija određenih vrsta iz tabele:

Ova vrsta replikacija podrazumeva specificiranje filtera za vrste za replikaciju, i samo te vrste će biti dodate u replikaciju, pri čemu ti filteri ne smeju biti previse komplikovani.

```
postgres=# create table tabela2 (a int primary key);
CREATE TABLE
postgres=# insert into tabela2 (a) select generate_series(1, 100);
INSERT 0 100
postgres=# create publication pub2 for table tabela2 where (a % 2 = 0);
CREATE PUBLICATION
postgres=#
```

Slika 42–Kreiranje publikacije na primarnom serveru. U okviru ovog primera filter nalaže da u publikaciji treba da se nalaze samo one vrste iz tabele koje sadrže parnu vrednost kolone a.

```
postgres=# create table tabela2 (a int primary key);
CREATE TABLE
postgres=# create subscription sub2 connection 'host=localhost port=5432 dbname=postgres' publication pub2;
NOTICE: created replication slot "sub2" on publisher
CREATE SUBSCRIPTION
postgres=# select count(*) from tabela2;
count
-----
      50
(1 row)
```

Slika 43–Kreiranje pretplate na replikacionom serveru. Na slici se može videti da je na drugom serveru prisutna samo polovina podataka.

#### Logička replikacija određenih kolona iz tabele:

Replikaciju je moguće vršiti i samo za određene kolone iz tabele. Ovo podrazumeva, da na primarnom serveru, prilikom kreiranja publikacije, zajedno sa imenom tabele specificira i lista kolona.

```
postgres=# create table tabela3 (a int primary key, b text, c text);
CREATE TABLE
postgres=# create publication pub3 for table tabela3 (a, c);
CREATE PUBLICATION
postgres=# insert into tabela3 (a, b, c) values (1, 'Sara', 'Savic');
INSERT 0 1
```

Slika 44–Kreiranje publikacije sa primarnom serveru

Na strani pretplate, tabela koja prima replikovane podatke sadrži samo podskup kolona kako je i prikazano na slici 45.

```
postgres=# create table tabela3 (a int primary key, c text);
CREATE TABLE
postgres=# create subscription sub3 connection 'host=localhost port=5432 dbname=postgres' publication pub3;
NOTICE: created replication slot "sub3" on publisher
CREATE SUBSCRIPTION
postgres=# select * from tabela3;
 a | c
---+---
  1 | Savic
(1 row)
```

Slika 45–Kreiranje pretplate na replikacionom serveru

#### Logička replikacija određenih operacija iz tabele:

Vrste tabele mogu biti ubačene, ažurirane, obrisane i sama tabela može biti ispražnjena. Publisher tabela može biti konfigurisana da uključi samo jednu ili više od ovih 4 operacija u podatke za replikaciju. (Prikazano na slici 46)

```
postgres=# create table tabela4 (a int primary key, b text);
CREATE TABLE
postgres=# create publication pub4 for table tabela4 with (publish = 'insert, update');
CREATE PUBLICATION
postgres=# select * from tabela4;
 a | b
---+---
(0 rows)
```

Slika 46–Kreiranje publikacije sa primarnom serveru. U ovom primeru publisher je konfigurisan tako da vrši replikaciju samo ubacivanja i ažuriranja u tabelu.

```
postgres=# create table tabela4 (a int primary key, b text);
CREATE TABLE
postgres=# create subscription sub4 connection 'host=localhost port=5432 dbname=postgres' publication pub4;
NOTICE: created replication slot "sub4" on publisher
CREATE SUBSCRIPTION
postgres=# select * from tabela4;
 a | b
---+---
(0 rows)
```

Slika 47–Kreiranje pretplate na replikacionom serveru

Kada se sada ubace, ažuriraju i obrišu neke vrste na publisher-u kao na slici 48:

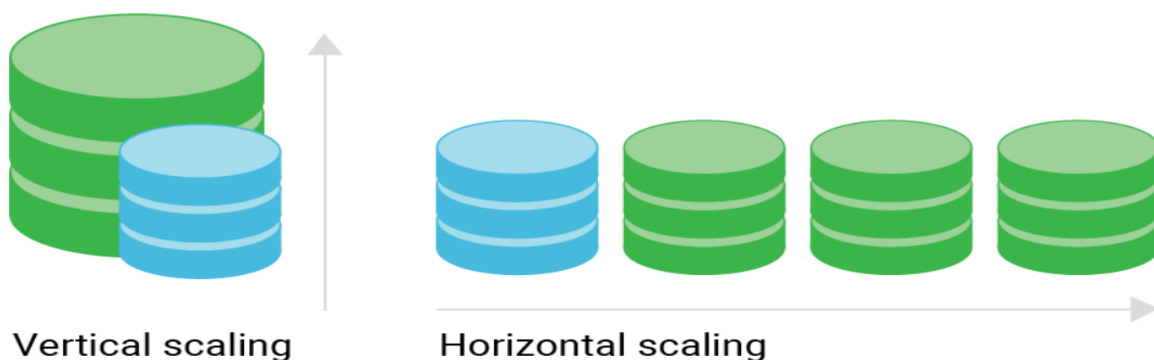
```
postgres=# insert into tabela4 (a, b) values (1, 'Sara');
INSERT 0 1
postgres=# insert into tabela4 (a, b) values (2, 'Sarica');
INSERT 0 1
postgres=# update tabela4 set b='Mila' where a=1;
UPDATE 1
postgres=# delete from tabela4 where a=2;
DELETE 1
postgres=# select * from tabela4;
 a | b
---+---
 1 | Mila
(1 row)
```

Slika 48– Dodavanje, brisanje i ažuriranje podataka na publisher-u

```
postgres=# select * from tabela4;
 a | b
---+---
 2 | Sarica
 1 | Mila
(2 rows)
```

Slika 49– Sadržaj tabele na subscriber-u nakon izmena na publisher-u

Onda tabela na subscriber-u sadrži vrstu koja je obrisana sa publisher-a, jer operacije brisanja nisu bile uključene u promene koje su poslate. Ovo takođe ilustruje kako nepravilna upotreba logičke replikacije narušava atomičnost transakcija: ukoliko je vrsta kreirana a zatim obrisana unutar jedne transakcije na strani publisher-a, subscriber će ipak završiti sa vrstom koja nije bila uključena u bilo koju drugu transakciju na strani publisher-a.



Slika 50–Prikaz horizontalnog i vertikalnog skaliranja<sup>5</sup>

<sup>5</sup> Izvor slike-<https://www.scylladb.com/glossary/database-scalability/>

Za horizontalno skaliranje, moguće je dodati više baza podataka kao slave čvorove. To može pomoći u poboljšanju performansi čitanja balansiranjem saobraćaja između čvorova. U ovom slučaju, potreban je i load balancer da distribuira saobraćaj ka odgovarajućem čvoru u skladu sa politikom i stanjem čvora. Međutim, kako ne bi došlo do kvara dodavanjem samo jednog load balancer-a, trebalo bi razmotriti dodavanje dva ili više load balancer čvorova.

Vertikalno skaliranje često zahteva novi hardver i migraciju na neku veću mašinu. Međutim, ponekad može biti dovoljno dodati više CPU-a, RAM-a ili prostora za skladištenje. Ako je potreban samo jedan element za postizanje vertikalnog skaliranja, moguće ga je dodati ili zameniti. Na primer, ako baza podataka stalno opterećuje CPU da bi obrađivala upite, tada je potrebno dodati još jedan procesor ili da se taj procesor zameni portujećim bržim procesorom koji koristi veći broj jezgara za obradu. Ako dođe do preopterećenja memorije, potrebno je samo povećati maksimalnu memoriju u sistemu. Naravno, ograničenja vertikalnog skaliranja se uvek javljaju kada postoji najbrži CPU ili ukoliko je memorija maksimalno opterećena. U takvoj situaciji, može biti potrebno razmotriti horizontalno skaliranje. Zbog postizanja vertikalnog skaliranja, u PostgreSQL bazi podataka, može biti potrebno izmeniti neke konfiguracione parametre kako bi se bazi omogućilo da koristi nove ili bolje hardverske resurse. Neki od parametara koji bi zahtevali promenu jesu: `work_mem` parametar koji određuje koliko memorije će se koristiti za interna sortiranja i hash tabele pre nego što se započne upisivanje privremenih fajlova na disk. Više sesija može istovremeno obavljati ove operacije, pa ukupna upotreba memorije može biti mnogo veća od vrednosti `work_mem` parametra. Drugi parametar jeste `maintenance_work_mem` koji određuje količinu memorije koja može biti korišćena za operacije održavanja, a veća vrednost prilikom podešavanja može poboljšati performanse povratka podatka iz backup-a. Potrebno je podesiti i parametre kao što su `autovacuum_work_mem`, koji određuje količinu memorije koju može da koristi svaki autovacuum radni proces, parametar `autovacuum_max_workers`, koji određuje maksimalni broj autovacuum procesa koji istovremeno mogu raditi. Još jedan od bitnih parametara koji može pomoći prilikom vertikalnog skaliranja jeste `effective_io_concurrency`, koji postavlja broj istovremenih operacija I/O diska koje PostgreSQL očekuje da mogu biti izvršene istovremeno. Povećanje ove vrednosti će povećati broj I/O operacija koje bilo koja pojedinačna PostgreSQL sesija pokušava pokrenuti paralelno. Pored pomenutih, postoji još značajan broj parametara koje je potrebno izmeniti kako bi se poboljšali resursi koje baza podataka koristi. [17]

## 5.1 Korišćenje sharding tehnike za postizanje horizontalnog skaliranja

Značaj postizanja skalabilnosti ogleda se, kako je već pomenuto, u sposobnosti sistema baze podataka da efikasno raste i da se prilagođava povećanju zahteva za obradom podataka bez značajnog gubitka performansi. Horizontalno skaliranje, koje predstavlja jednu od dve vrste skaliranja (skalabilnosti), postiže se kreiranjem klastera, a iako postoji mnogo načina za horizontalno skaliranje, jedan od najsitaknutijih jeste sharding tehnika. [18]

Sharding tehnika je postupak optimizacije baze podataka tako što se podaci iz neke velike baze podataka razdvajaju u više manjih tabela, koje se nazivaju shard-ovi (ili particije). Shard-ovanje podrazumeva da se podaci raspoređuju preko većeg broja računara.

I kod shard-ovanja postoje horizontalno i vertikalno shard-ovanje, kao i kod same skalabilnosti. Kod horizontalnog shard-ovanja, svaka nova tabela ima istu šemu kao i originalna velika tabela, ali ima jedinstvene redove. Ovaj vid shard-ovanja je koristan kada upiti često vraćaju podskup redova koji su često grupisani zajedno. Kroz horizontalnog shard-ovanje, podaci se distribuiraju preko više



fizičkih servera ili računara, što omogućava paralelno procesiranje upita i smanjenje opterećenja na svakom pojedničnom serveru, što dovodi do skalabilnosti sistema. [18]

S druge strane, kod vertikalnog shard-ovanja, svaka nova tabela ima šemu koja je podskup šeme originalne tabele, te je ovo korisno kada upiti često vraćaju samo podskup kolona podataka. Vertikalno shard-ovanje optimizuje performanse upita tako što segmentira podatke po specifičnim šemama ili kolonama, što omogućava efikasnije upravljanje podacima koji se često koriste zajedno. Ova strategija ne samo da oslobađa resurse već i smanjuje opterećenje na sistemima, doprinoseći ukupnoj održivoj skalabilnosti sistema baza podataka. [18]

Može se zaključiti, da shard-ovanje podrazumeva deljenje podataka na dva ili više manjih delova, koji se nazivaju logički shardovi. Logički shard-ovi se distribuiraju preko različitih baza podataka, nazvanih fizički shardovi, koji mogu sadržati veći broj logičkih shard-ova. Podaci koji se čuvaju u svim shard-ovima predstavljaju celokupan logički skup podataka. [18]

Za praktičan prikaz tehnike shard-ovanja iskorišćeni su serveri koji su kreirani u potpoglavlju 4.2, koji su uz pomoć PgPool-II alata podešeni da čine klaster, dok su za samu tehniku shard-ovanja korišćene PostgreSQL particije i Foreign Data Wrapper (FDW) ekstenzija, koja omogućava PostgreSQL-u pristupa podacima koji se čuvaju na drugim serverima. PostgreSQL server koji radi na portu 5432 predstavlja primarni server, koji se u ovom slučaju naziva shard1, a server koji radi na portu 5433 predstavlja sekundarni server i u ovom slučaju se naziva shard2.

Za pravilno funkcionisanje shard-ovanja najpre je potrebno omogućiti FDW ekstenziju na primarnom serveru (shard1). Ova ekstenzija je podrazumevano dostupna u PostgreSQL instalaciji, pa ju je zato moguće uključiti na način na koji je prikazano na slici 51.

```
postgres=# create extension postgres_fdw;  
CREATE EXTENSION
```

Slika 51–Prikaz uključivanja fdw ekstenzije na primarnom serveru (shard1)

Kada je omogućeno koristiti foreign data wrapper, na shard1 serveru se kreira glavna tabela sa svojim lokalnim particijama.

```
postgres=# create database shard_db;  
CREATE DATABASE
```

Slika 52–Prikaz kreiranja glavne tabele na serveru shard1

```
shard_db=# create table kupci(id int not null, ime varchar(30) not null, registrovan date  
not null) partition by range (registrovan);  
CREATE TABLE  
shard_db=# create table kupci_2024 partition of kupci for values from ('2024-01-01') to (  
'2025-01-01');  
CREATE TABLE  
shard_db=# create table kupci_2023 partition of kupci for values from ('2023-01-01') to (  
'2024-01-01');  
CREATE TABLE
```

Slika 53–Prikaz kreiranja lokalnih particija (kupci\_2024 i kupci\_2023) za tabelu kupci na serveru shard1

U okviru ovog primera, kreirana je pre svega posebna baza podataka koja će se koristiti za shard-ovanje, “shard\_db” baza podataka. Glavna tabela je tabela “kupci” koja sadrži tri kolone (kolonu id, kolonu ime i kolonu registracija koja pamti datum kada je kupac registrovan), a particionisanje

ove tabele je izvršeno po datumu registracije. Svaka particija sadrži podatke koji odgovaraju određenom opsegu datuma, što olakšava brzu i efikasnu manipulaciju nad podacima u bazi.

Slika 54 prikazuje dodavanje podataka u lokalna particije (kupci\_2024 i kupci\_2023) i kreiranje upita za proveru dodatih podataka u samu tabelu kupci i njene particije.

```
shard_db=# insert into kupci(id, ime, registrovan) values (1, 'Sara', '2024-06-22');
INSERT 0 1
shard_db=# INSERT 0 1

shard_db=# insert into kupci(id, ime, registrovan) values (2, 'Milan', '2023-05-24');
INSERT 0 1
shard_db=#

shard_db=# select * from kupci;
 id | ime | registrovan
-----+-----+-----
  1 | Sara | 2024-06-22
  2 | Milan | 2023-05-24
(2 rows)

shard_db=# select * from kupci_2024;
 id | ime | registrovan
-----+-----+-----
  1 | Sara | 2024-06-22
(1 row)

shard_db=# select * from kupci_2023;
 id | ime | registrovan
-----+-----+-----
  2 | Milan | 2023-05-24
(1 row)
```

Slika 54–Prikaz dodavanja podataka u tabelu kupci i proveru podataka u samoj tabeli i njenim particijama

Sada je potrebno kreirati udaljenu particiju tabele kupci na drugom serveru (shard2). Ova particija će biti fizički smeštena na drugom serveru(serveru shard2 koji radi na portu 5433), ali će definisanje i upravljanje particijom biti izvršeno na glavnom serveru(shard1). Kreiranje udaljene particije je prikazano na slici 55.

```
postgres=# create database shard_db;
CREATE DATABASE
postgres=# \c shard_db
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
You are now connected to database "shard_db" as user "postgres".
shard_db=# create table kupci_2022(id int not null, ime varchar (30) not null, registrovan date not null);
CREATE TABLE
shard_db=#
```

Slika 55–Prikaz kreiranja udaljene particije kupci\_2022 na serveru shard2. Ovaj korak zahteva da se na serveru shard2 takođe kreira posebna tabela za shard-ovanje (tabela shard\_db).

Na narednim slikama prikazano je postavljanje i konfigurisanja PostgreSQL foreign data wrapper-a na serveru shard1 kako bi se omogućio pristup podacima sa udaljenog servera i upravljanje particijama.

```
shard_db=# create server shard2 foreign data wrapper postgres_fdw options(host 'localhost', port '5433', dbname 'shard_db');
CREATE SERVER
```

Slika 56–Prikaz definisanja novog servera shard2 koji je povezan putem fdw-a sa glavnim serverom (shard1) i omogućava pristup podacima na udaljenom serveru (shard2).



```
shard_db=# create user mapping for postgres server shard2 options (user 'postgres', password 'sarica');
CREATE USER MAPPING
shard_db=#
```

Slika 57–Prikaz izvršenja create user mapping komande. Ova komanda omogućava korisniku postgres sa glavnog servera pristup udaljenom serveru shard2 koristeći određene opcije, kao što su korisničko ime i lozinka.

```
shard_db=# create foreign table kupci_2022 partition of kupci for values from ('2022-01-01') to ('2023-01-01') server shard2;
```

Slika 58–Prikaz kreiranja udaljene tabele kupci\_2022 kao particije glavne tabele kupci na glavnom serveru. Particija se mapira na server shard2 i podaci koji odgovaraju određenom opsegu (u ovom slučaju, registracija između '2022-01-01' i '2023-01-01') se smeštaju na udaljeni server.

```
shard_db=# insert into kupci(id, ime, registrovan) values (3, 'Ivan', '2022-07-09');
INSERT 0 1
shard_db=# insert into kupci(id, ime, registrovan) values (4, 'Mila', '2022-09-01');
INSERT 0 1
shard_db=#
```

Slika 59–Prikaz dodavanja podataka u tabelu kupci\_2022. Komande prikazane na slici dodaju novu vrstu tabelu kupci na glavnom serveru. Zbog definicije particije, podaci koji odgovaraju opsegu '2022-01-01' i '2023-01-01' će biti automatski usmereni i smešteni na udaljeni server shard2.

Nakon ovoga, moguće je pristupiti podacima i sa servera shard1 i sa servera shard2, kako je i prikazano na slikama 60 i 61.

```
shard_db=# select * from kupci;
 id | ime | registrovan
-----+-----
  1 | Sara | 2024-06-22
  2 | Milan | 2023-05-24
  3 | Ivan | 2022-07-09
  4 | Mila | 2022-09-01
(4 rows)

shard_db=# select * from kupci_2022;
 id | ime | registrovan
-----+-----
  3 | Ivan | 2022-07-09
  4 | Mila | 2022-09-01
(2 rows)
```

Slika 60–Prikaz svih podataka iz glavne tabele kupci na shard1 serveru. Rezultat uključuju sve podatke koji su lokalno smešteni na shard1 serveru i podatke koji su smešteni na shard2 serveru preko fdw particije kupci\_2022.

```
shard_db=# select * from kupci_2022;
 id | ime | registrovan
-----+-----
  3 | Ivan | 2022-07-09
  4 | Mila | 2022-09-01
(2 rows)
```

Slika 61–Prikaz podataka koji su smešteni u particiji kupci\_2022 na shard2 serveru. Ovime je uspostavljeno shard-ovanje u klasteru.

Iz svega prethodno opisanog može se zaključiti da sharding omogućava skalabilnost jer omogućava distribuciju podataka preko većeg broja servera u klasteru, čime se osigurava sistemima da efikasnije obrađuju velike količine podataka i zahteve i time poboljšaju svoje performanse.

## 6.ZAKLJUČAK

Klasteri u PostgreSQL-u predstavljaju ključnu arhitektonsku komponentu koja omogućava efikasno upravljanje bazama podataka u modernim poslovnim okruženjima. Njihova osnovna uloga jeste distribucija podataka i radnog opterećenja preko više servera, što značajno doprinosi boljoj skalabilnosti i visokoj dostupnosti.

Pregledom seminarskog rada, može se zaključiti da klasteri, kako su inicijalno definisani u PostgreSQL-u omogućavaju efikasnu reorganizaciju podataka unutar jednog sistema. Pored toga, ukoliko postoji više servera, korišćenjem replikacije i raznih alata za kreiranje klastera PostgreSQL osigurava kontinuiranu sinhronizaciju podataka između primarnog servera i većeg broja servera, čime se garantuje očuvanje integriteta podataka i minimizacija rizika od gubitka jer u slučaju otkaza primarnog servera neki od rezervnih će preuzeti njegovu ulogu i nastaviti sa radom, čime se dostupnost baze obezbeđuje u velikoj meri.

Pored toga, PostgreSQL koristi napredne tehnike klasterovanja poput shard-ovanja, koje omogućava horizontalno deljenje podataka preko više servera radi efikasnije upravljanja velikim količinama informacija. Ovo nije samo korisno za poboljšanje performansi aplikacija već i za prilagođavanje promenljivim potrebama korisnika i rasta poslovanja.

Generalno gledano, PostgreSQL klaster rešenja ne samo da obezbeđuju kontinuitet operacija i pouzdanost podataka u dinamičnom poslovnom okruženju, već i pružaju fleksibilnost i skalabilnost potrebnu za podršku raznovrsnim aplikacijama i analitičkim zahtevima. Ova kombinacija funkcionalnosti čini PostgreSQL klaster rešenje ključnim faktorom za organizacije koje teže visokim standardima u upravljanju podacima i informacionim tehnologijama.

## 7. SPISAK KORIŠĆENE LITERATURE

- [1]- Milad Karimyar, "How to Setup a PostgreSQL Database Cluster", Septembar 2023.  
Dostupno na: [https://www.servermania.com/kb/articles/setup-postgresql-cluster?fbclid=IwZXh0bgNhZW0CMTAAR2WW1250YNR7ZknDVnFsAMzZWPIBpY1def5WylMC8udHOTMPeIz-K1FuM\\_aem\\_AdEO5LvjMAbgSkd\\_tgeItM3cpIWNNAV0soUeItu6u0WUz\\_B4-TJA5ooHAcfHlrsNNioo-WyU-w3\\_gVgYya18mA9uU](https://www.servermania.com/kb/articles/setup-postgresql-cluster?fbclid=IwZXh0bgNhZW0CMTAAR2WW1250YNR7ZknDVnFsAMzZWPIBpY1def5WylMC8udHOTMPeIz-K1FuM_aem_AdEO5LvjMAbgSkd_tgeItM3cpIWNNAV0soUeItu6u0WUz_B4-TJA5ooHAcfHlrsNNioo-WyU-w3_gVgYya18mA9uU)
- [2]- Mostafa Ibrahim, "What is Database Clustering?", Oktobar 2022.  
Dostupno na: <https://www.harperdb.io/post/what-is-database-clustering>
- [3]- Sharon Rithika, "Postgres Cluster Setup in 4 Easy Steps [+7 Clustering Options]", Januar 2024.  
Dostupno na: [https://hevodata.com/learn/postgresql-cluster/#Steps\\_for\\_Setting\\_Up\\_a\\_Basic\\_PostgreSQL\\_Cluster](https://hevodata.com/learn/postgresql-cluster/#Steps_for_Setting_Up_a_Basic_PostgreSQL_Cluster)
- [4]- Hironobu Suzuki, "The Internals of PostgreSQL"  
Dostupno na: <https://www.interdb.jp/pg/pgsql01/01.html>
- [5]- A User's Manual for PostgreSQL Cluster: Key Features, Data Directory, Step for Setting Up & User Guide  
Dostupno na: [https://www.boltic.io/blog/postgres-cluster?fbclid=IwZXh0bgNhZW0CMTAAR3IEBjjvnUy3IBsQTa7JZMzZ0fapek\\_jo9Yo79WzZBya3izvkwm\\_4zmUno\\_aem\\_AdEAwH3QJuyYQHmi9O27R0OhGUn3nyev5j\\_RwKbluOgeHs11o4jnZSQ32vVWuBzPUuT1gcPwlqKy6zHIA4mXtG-s](https://www.boltic.io/blog/postgres-cluster?fbclid=IwZXh0bgNhZW0CMTAAR3IEBjjvnUy3IBsQTa7JZMzZ0fapek_jo9Yo79WzZBya3izvkwm_4zmUno_aem_AdEAwH3QJuyYQHmi9O27R0OhGUn3nyev5j_RwKbluOgeHs11o4jnZSQ32vVWuBzPUuT1gcPwlqKy6zHIA4mXtG-s)
- [6]- PostgreSQL 16 Documentation -Creating a Database Cluster, The PostgreSQL Global Development Group 1996–2024.  
Dostupno na: <https://www.postgresql.org/docs/current/creating-cluster.html>
- [7]- PostgreSQL 16 Documentation -CLUSTER, The PostgreSQL Global Development Group 1996–2024.  
Dostupno na: <https://www.postgresql.org/docs/current/sql-cluster.html>
- [8]- Jeff Novotny, "A Comparison of High Availability PostgreSQL Solutions", Mart 2024.  
Dostupno na: <https://www.linode.com/docs/guides/comparison-of-high-availability-postgresql-solutions/>
- [9]- "What is a high availability database?"  
Dostupno na: <https://aerospike.com/glossary/high-availability-database/>
- [10]- "PostgreSQL Replication and Automatic Failover Tutorial", EDB, 2023.  
Dostupno na: <https://www.enterprisedb.com/postgres-tutorials/postgresql-replication-and-automatic-failover-tutorial>
- [11]- Özlem Güneş, "Logical Replication in PostgreSQL", Jul 2023.  
Dostupno na: <https://medium.com/@ozlemgunes97/logical-replication-in-postgresql-a6d849ceb83e>
- [12]- Physical Replication Mechanisms in PostgreSQL  
Dostupno na: <https://pgdash.io/blog/postgres-physical-replication.html>
- [13]- PostgreSQL 16 Documentation - Log-Shipping Standby Servers, The PostgreSQL Global Development Group 1996–2024.  
Dostupno na: <https://www.postgresql.org/docs/current/warm-standby.html>
- [14]- "What is ClusterControl?"  
Dostupno na: <https://platformengineering.org/tools/clustercontrol>
- [15]- Dmitry Romanoff, "Highly Available PostgreSQL Cluster using Patroni and HAProxy", Avgust 2022.

Dostupno na: <https://jfrog.com/community/devops/highly-available-postgresql-cluster-using-patroni-and-haproxy/>

[16]- Database Scalability

Dostupno na: <https://www.scylladb.com/glossary/database-scalability/>

[17]- Sebastian Insausti, "Scaling PostgreSQL for Large Amounts of Data", Jun 2019.

Dostupno na: <https://severalnines.com/blog/scaling-postgresql-large-amounts-data/>

[18]- Sebastian Insausti, "How to Configure PostgreSQL Sharding with ClusterControl", Jul 2021.

Dostupno na: <https://severalnines.com/blog/how-configure-postgresql-sharding-clustercontrol/>

[19]- HA PostgreSQL and HA PGPool Topologies

Dostupno na:

<https://documentation.sas.com/doc/en/calcdc/3.5/dplyml0phy0lax/p0844sho9jm6e8n1mwlu0iwocf2k.htm>

[20]- Bo Peng, "Installing Pgpool-II on Debian/Ubuntu", Mart 2022.

Dostupno na: <https://b-peng.blogspot.com/2022/03/pgpool-debian.html>

[21]- Horizontally Scaling PostgreSQL

Dostupno na: <https://pgdash.io/blog/horizontally-scaling-postgresql.html>