Internet softverske arhitekture

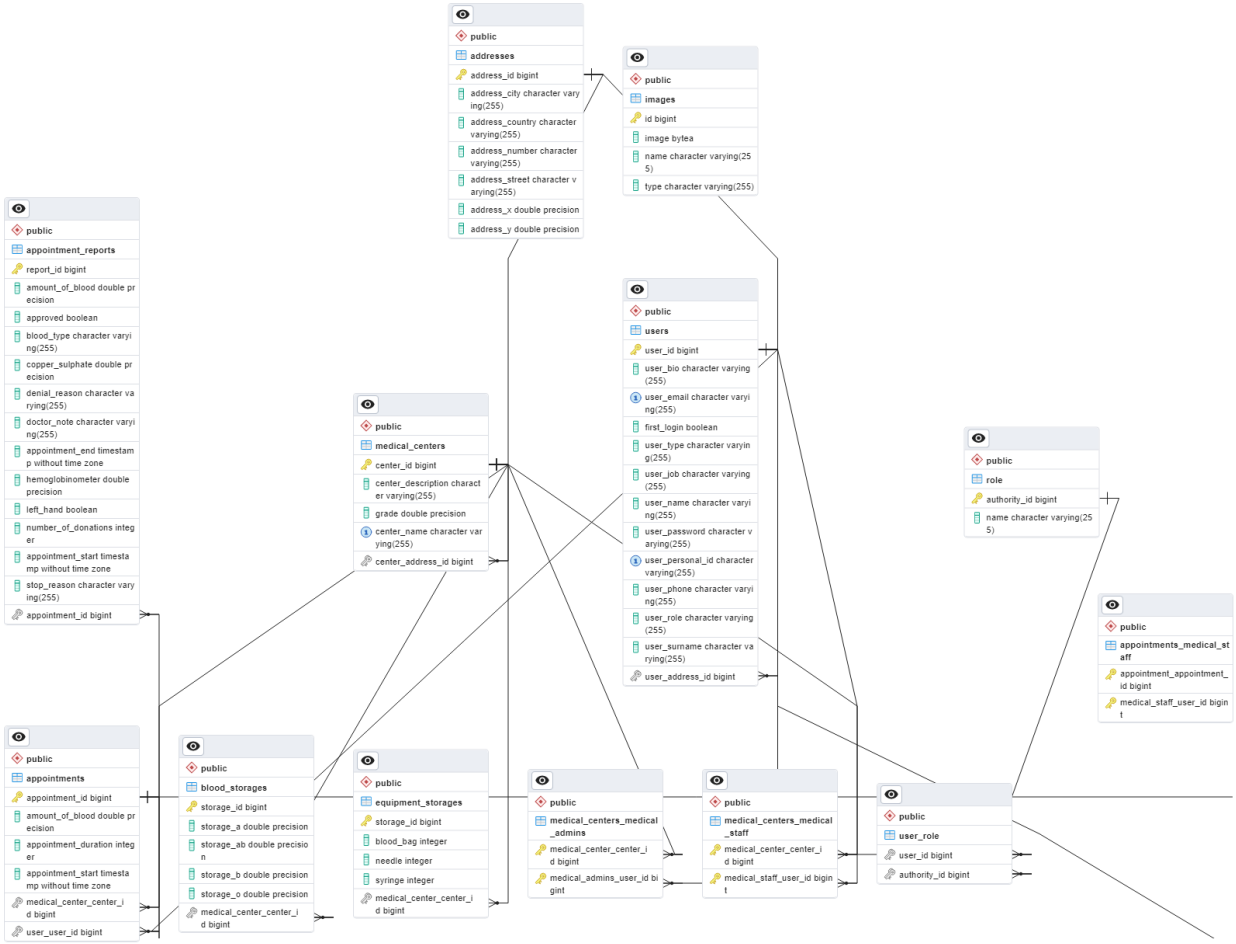
Tim 9 - Dušan Marković

Šema baze podataka

Rad na projektu počeo je timskom analizom specifikacije i kreiranjem slabo razrađenje konceptualne šeme baze podataka.

Daljim radom na projektu, ova šema je dorađena: pojedini entiteti i kardinaliteti su izmenjeni, mnogi entiteti dodati.

Finalna šema baze podataka koja se nalazi na grani feature-3.19-backup predstavljena je sledećim dijagramom:



Predlog strategije za particionisanje podataka

Početna pretpostavka :
Broj korisnika aplikacije je 10 miliona. Broj rezervacija na mesečnom nivou je 500 hiljada. Korišćeni DBMS je PostgreSQL.

Particionisanje ne bi bilo neophodno izvršiti nad svim tabelama, već samo nad onim koje poseduju veću količinu podataka i nad kojima se učestano vrše upiti.

Shodno tome, tabele koje ne bi bilo potrebe particionisati jesu : role, user_role, medical_centers-medical_staff, medical_centers_medical_admin, equipment_storages, blood_storages i medical_centers.

Slično kreiranju indeksa, za kreiranje strategije particionisanja, neophodno je odrediti atribute tabela koji se nalaze u većini upita pod WHERE klauzulom, kao i u spojevima pod JOIN ON. Ovi atributi biće particioni ključevi.

PostgreSQL počevši od verzije 10 nudi mogućnost **deklarativnog particionisanja** (Declarative Partitioning). Deklarativno particionisanje pojednostavljeno je u odnosu na nasledno particionisanje (Inheritance Partitioning) koje je do tada korišćeno. Prilikom particionisanja podataka, neophodno je odraditi migraciju podataka na novu bazu.

Primer particionisanja tabele appointments korišćenjem metode **Range partitioning**:

1. Kreiranje nove baze podataka
2. Kreiranje nove tabele appointments sa istim atributima. Prilikom kreiranja tabele navesti da će se ista particionisati po koloni appointment_start.
3. Kreiranje tabele appointments_old koja će sadržati podatke o zakazanim pregledima nad kojima pretpostavljamo da se neće vršiti veliki broj upita u narednom periodu.
4. Kreiranje nove tabele appointments_recent koja će sadržati podatke pregledima zakazanim za relativno kratak prethodni period, nad kojima pretpostavljamo da će se vršiti upiti (analiza i kreiranje izveštaja za ovaj period).
5. Kreiranje nove tabele appointments_new koja će sadržazi podatke o rezervacijama koje se odnose na relativno trenutni period. Očekujemo da će se u skoroj budućnosti nad ovim podacima vršiti veliki broj upita (kreiranje izveštaja o pregledu).
6. Kreiranje nove tabele koja će sadržati rezervacije u ne tako bliskoj budućnosti.
7. Migracija podataka

Napomena: tabele kreirane u koracima 3-6. neophodno je kreirati kao particije tabele kreirane u koraku 2.

Deklarativnim particionisanjem, većina posla obuhvaćenog koracima 1-7. svela bi se na kucanje nekoliko linija u pg prompt-u.

Deklarativno particionisanje svaku particiju pretvara u sopstvenu tabelu, što omogućava lako kasnije particionisanje i eventualno brisanje particija.

Daljom analizom potrebno je odrediti adekvatne metode particionisanja ostalih tabela, kao i njihove particione ključeve.

Primer particionisanja tabele addresses bio bi sličan koracima za particionisanje tabele appointments, osim što bi se u ovom sličaju koristio metod **List partitioning**, te bi se particije kreirala na nivou lsite vrednosti (npr. gradova). Ovako bismo mogli napraviti particije adresa po regionima države.

Predlog strategije za replikaciju podataka

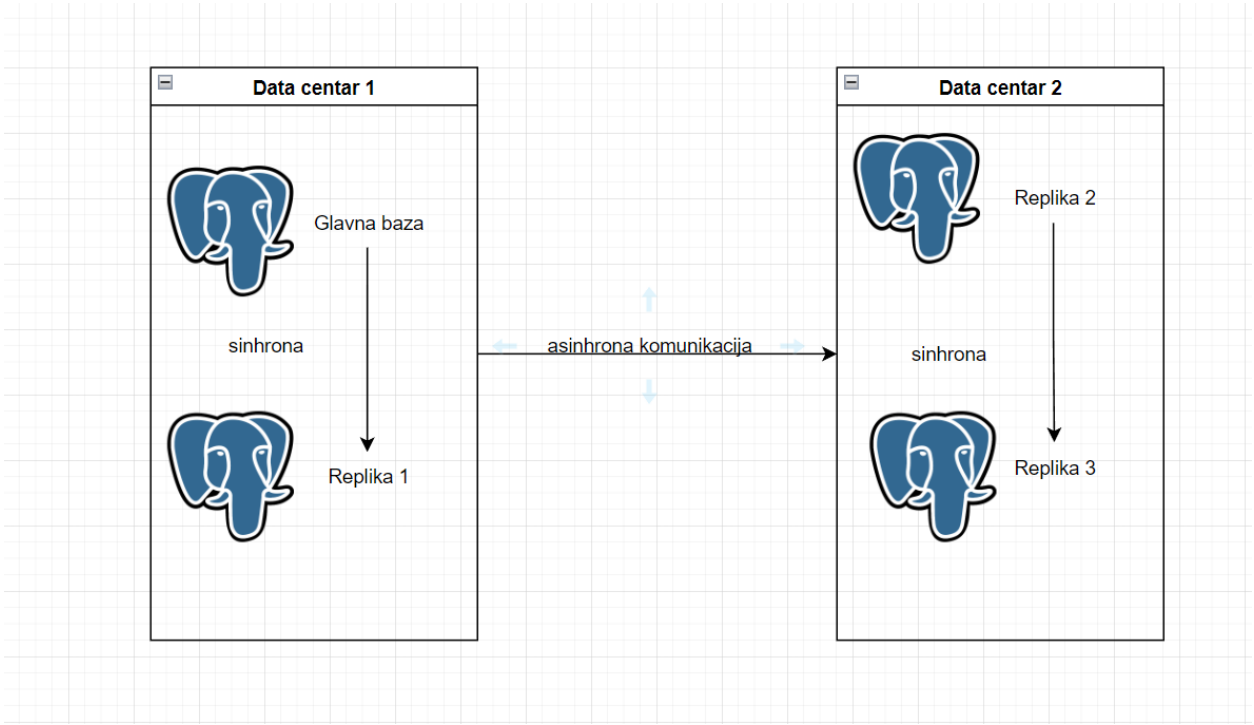
U slučaju otkaza baze, neophodno je imati bar jednu repliku. PosgreSQL nudi više mogućnosti za replikaciju podataka.

Od svih mogućnosti, **Streaming Replication** metoda poznata je kao najsigurnija i najbrža metoda replikacije. Primarna baza podataka šalje stream WAL fajlova replici. WAL je log fajl koji

se ažurira svakom izmenom u bazi. Za razliku od Log-Shipping-a koji je zahtevao da se WAL fajlovi popune, Streaming Replication je mnogo bolja opcija jer se logovi šalju u stream-u. Takođe, Streaming Replication nudi lako prebacivanje sa asinhrone na sinhronu replikaciju, što znači da će aplikativni sloj dobiti uspešan response tek kada se podaci ažuriraju i na replici.

U slučaju otkaza glavne baze podataka, neophodno je obezbediti brz prelazak na repliku (Fail-over). OpenSource alat koji nam ovo omogućava i automatizuje je **Patroni** za PostgreSQL.

Podešavanje replika baze podataka u idealnom slučaju prikazano je na sledećoj slici.



Za ovaj slučaj bilo bi potrebno staviti nekakav proxy između aplikativnog i fizičkog sloja, koji bi mogao da preusmeri saobraćaj na Data centar 2 u slučaju otkaza prvog klastera. Između svake baze i replike u jednom data centru potrebno je podesiti Patroni za automatski Fail-over.

Postoji više mogućnosti konkretizacije navedenog dijagrama.

Jedno od optimalnijih rešenja bilo bi uvođenje kaskadne replike. Ovo znači da bi Glavna baza kreirala jednu sinhronu repliku (Replika 1) i jednu asinhronu repliku (Replika 2), dok bi Replika 3 nastala kao sinhrona replika Replike 2. Ovakva arhitektura Patroni alatu okalšava posao, pošto u slučaju otkaza Glavne baze, Replika 1 postaje glavna, zatim Replika 2 i zatim Replika 3.

Za još bolje rezultate, mogu se koristiti bi-direkcione replike ili multi-master arhitektura.

Napomena: Replika 1 može se sve vreme koristiti za read operacije dok je Glavna baza aktivna.

Procena za hardverske resurse potrebne za skladištenje

Pretpostavka: Aplikaciju koristi 10 miliona korisnika. Rezerviše se 500 hiljada pregleda mesečno. Rezervacije i izveštaji o pregledima se ne brišu iz baze.

U narednih 5 godina ovakva baza podataka imala bi oko 80 miliona redova podataka u svim tabelama, za šta ne bi bilo potrebno više od 10GB memorije. Dodajmo na ovo još tri replike i dobićemo procenu od 40GB.

Predlog strategije za keširanje podataka

Za Level 2 keširanje koristio bi se EhCache. EhCache je open-source keš koji se može skladištiti i u memoriji i na disku. Dobar je za generalno keširanje slika i ostalih statičnih elemenata, kao i za Level 2 Hibernate keširanje.

Treba keširati podatke koji se učestano traže iz baze podataka, kako bi se smanjilo opterećenje same baze. U generalnom slučaju ove aplikacije, to bi bila lista medicinskih centara, kao i lista slobodnih termina za pojedini centar.

Neki od podataka koji se mogu keširati u cilju poboljšanja performansi jesu: medicinski centar ulogovanog medical administratora, istorija poseta ulogovanog korisnika, termin na koji se odnosi QR kod prilikom kreiranja izveštaja pregleda...

Odluka o podacima koji bi se keširali donela bi se na osnovu analize učestano korišćenih podataka za svaku rolu.

Kada govorimo o keširanju statičkih elemenata, trebalo bi keširati css, slike, logoe… Trebalo bi izbegavati keširanje celih HTML stranica, kao i privatnih podataka klijenata.

Takođe je pogodno koristiti distribuirani EhCache, što podrazumeva da će svaki web server imati svoj cache server.

Predlog strategije za postavljanje load balansera

Prethodno je napomenuto da je neophondo postaviti proxy koji će određivati nad kojmi od klastera baze podataka će se vršiti upit sa web servera.. U ove potrebe koristio bi se HAProxy.

HaProxy je TCP-HTTP load balancer pogodan za korišćenje sa PostgreSQL bazama. Pogodan je za optimalno sprečavanja overload-a jedne PostgreSQL baze upitima. Koristi svoj load balancing algoritan na osnovu kog preusmerava sve klijente koji se povežu na njegovu instancu.

Jedno od rešenja bilo bi instalacija HAProxy-ja na svakom web serveru ili aplikacionom serveru. Web server bi se zatim povezao na lokalni HAProxy i bez njega ne bi mogao da pošalje zahtev fizičkom sloju.

Prednosti HAProxy-ja:

- Klasterima se pristupa putej jedinstvene ip adrese ili hostname-a. Topologija klastera je sakrivena iza proxy-ja.
- PostgreSQL konekcije su load balansirane među slobodnim bazama.
- Moguće je isključiti celu bazu podataka iz klastera bez rizika od gubitka saobraćaja.
- Pogodan je za veliku konkurentnost, jer nakon određenog broja maksimalnih konekcija ka bazi, HAProxy stavlja sve dodatne konekcije u que. Ovime se obezbeđuje zaštita od overloada baze.

Potrebno je dodati još jedan load balancer između klijenata i web servera.

Predlog monitoringa

Praksa je pokazala da se pozivi koji zahtevaju komunikaciju sa eksternim servisima i koji obavljaju najkompleksnije upite prvi moraju podvrgnuti monitoringu.

Jedno od monitoring rešenja bilo bi sledeće:

1. Kreirati JavaScript servis koji će voditi računa o trajanju poziva i odgovorima drugih komponenti.
2. Kreirati metrike koje će biti prosleđene na AWS.
3. Za svaki poziv odrediti smislen threshold (maksimalno trajanje poziva, sve preko ovoga bilo bi nedopustivo i alarmantno).
4. Kreirati alarme koje se odnose na prethodno definisane komponente
5. Uvezati podignute alarme sa OpsGenie alatom

Na ovaj način monitoringom će biti obuhvaćeno trajanje poziva. Takođe, u slučaju otkaza neke eksterne komponente sistema, podići će se alarm. OpsGenie integracija omogućuje automatsko obaveštavanje unapred predodređenih on-call developera koji blagovremeno mogu rešiti problem u slučaju otkaza neke komponente sistema.

Predložena arhitektura