Proof of concept – Reservation software

Nikolina Batinić, SW-01/2016,

Nikola Zubić, SW-03/2016,

Mihajlo Kušljić, SW-53/2016

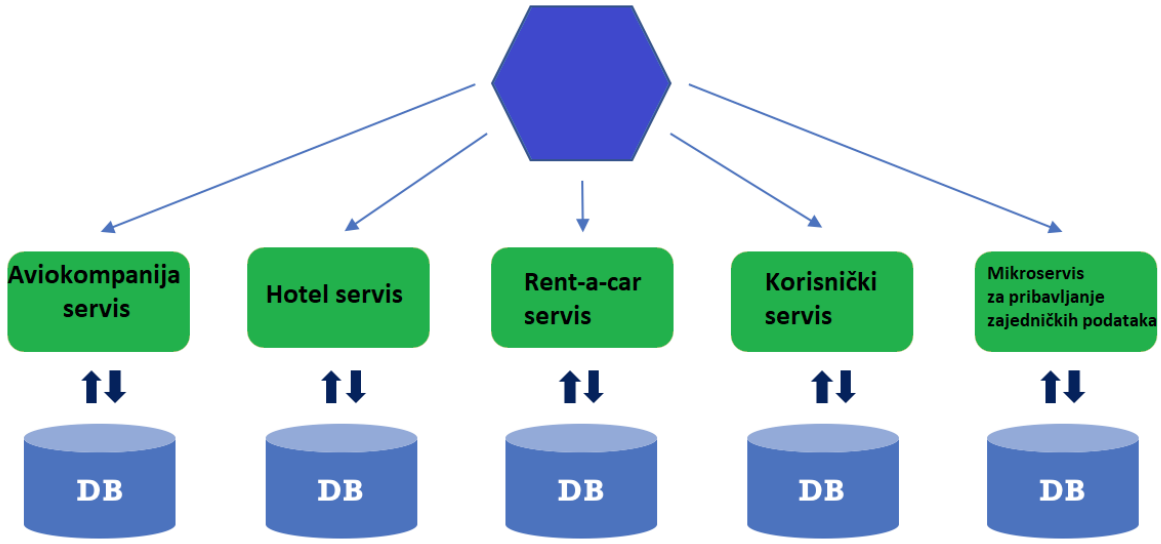
Fakultet tehničkih nauka, Novi Sad

- ## Klijentski zahtjevi i konkurentnost

  Cilj ove aplikacije jeste omogućavanje korisnicima da rezervišu avionske karte, i na to, opciono, hotelski smještaj i vozilo. U sistemu gdje bi teoretski imali milion ili više korisnika, proces rezervacije i brze rezervacije svih gore pomenutih tipova rezervacija razrješava se koristeći transakcije. Svaka rezervacija i brza rezervacija izvršava se u transakcionom režimu tako da, ako dođe do konflikta, u bilo kojem koraku, čitava rezervacija se poništava. Prilikom rezervacije i brze rezervacije, izvršava se pesimističko zaključavanje objekta koji se želi rezervisati (sjedište, soba i vozilo). Ostali korisnici ne mogu pristupati zaključanom resursu dok se ne izvrši njegova rezervacija. Optimističko zaključavanje korišćeno je kod izmjena poslovnica (aviokompanije, hoteli i rent-a-car servisi). Ukoliko više administratora pokuša da izmjeni istu poslovnicu, samo onaj koji je prvi potvrdio izmjene će uspjeti da je izmjeni. Naravno, mi smo za sad obradili uz rezervacije i brze rezervacije još po jednu za svakoga konfliktnu situaciju. Kako bi sistem rastao potrudili bismo se da obradimo sve konfliktne situacije koje mogu nastati, a koje smo mi i korisnici, uočili, kao što je npr: kada više korisnika pokušava da ocjeni isti resurs u isto vrijeme. Konkurentni pristup resursima omogućuje razrješavanje konfliktnih situacija, i olakšava rad sistema sa velikim brojem resursa. Naravno, glavni klijentski zahtjevi se tiču upravo rezervacija i njihovo pravilno funkcionisanje je najbitnije za ovaj sistem.

- ## Prelazak na mikroservise

  Tradicionalne softverske arhitekture se fokusiraju na jednu platformu. Takva arhitektura ima probleme sa skalabilnošću i međuzavisnostima u kodu, što je i logično zbog velike aplikacije. Mikroservisi smanjuju kompleksnost razdvajajući izvršavanje funkcionalnosti na više platformi, što povećava skalabilnost. Servisno orijentisana arhitektura kakvu mi sada imamo je bolja, ali i ona ima probleme. Najbolje bi bilo kada bi svi servisi funkcionisali za sebe tj. korisnički interfejs, aplikativna logika, biznis logika, sloj za komunikaciju sa bazom podataka i sama baza podataka za jedan servis je zaseban dio koji se zove mikroservis.



Kako sistem postaje složeniji, mogu se dodati i mnogi drugi mikroservisi, sve prema potrebama korisnika. Želimo da imamo u potpunosti nezavisne servise sa nezavisnim bazama podataka. Svaki servis se deploy-uje na više servera. Sa porastom popularnosti i upotrebe aplikacije, raste vertikalna i horizontalna skalabilnost. Sa porastom vertikalne skalabilnosti potrebno je povećati broj uređaja, a naravno poželjno i kvalitet hardvera, jer što je kvalitetniji hardver, taj broj će manje da raste. Dalja podjela unutar mikroservisa povećava horizontalnu

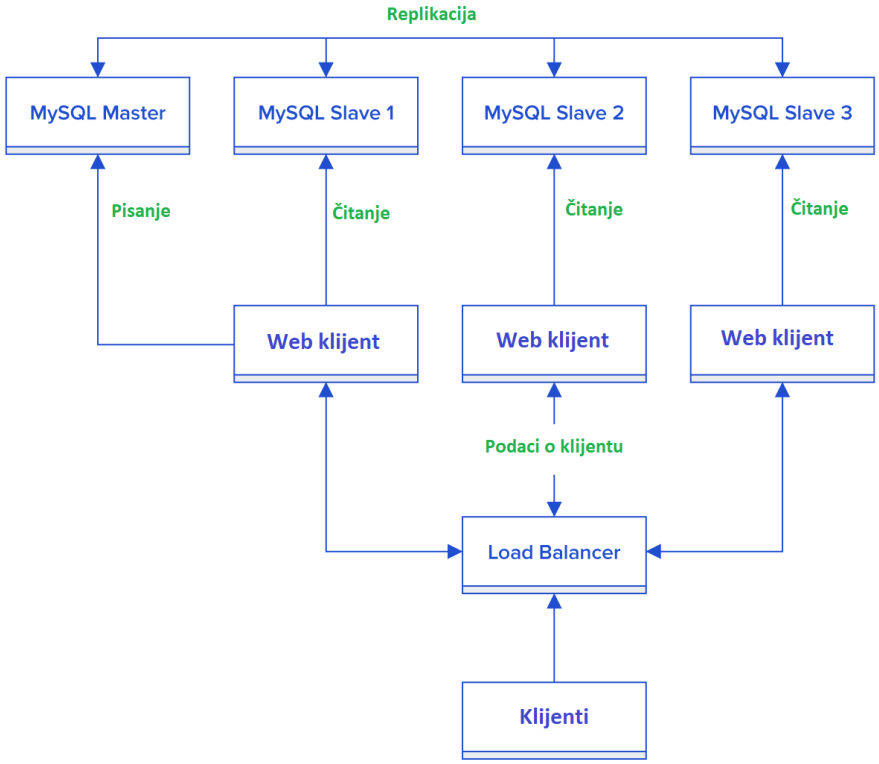
skalabilnost sistema, tj. dolazi do sve veće podjele sistema na što više komponenti koje rade na sve više računara i servera. Ono što je ovdje dobro jeste činjenica da se održavanje i rad sa svakim mikroservisom radi nezavisno od drugih. Svaki od njih može imati svoju arhitekturu, koristiti različite tehnologije, programske jezike i baze podataka. Ne postoji ograničenje u pogledu dodavanja novih mikroservisa i integrisanja istih u sistem, osim novca.

- ## Keširanje

  Keširanje podataka se uvodi kako bi se smanjila komunikacija sa bazom, tj. izvršavanje upita, koje, u velikim sistemima, vrši mnogo bespotrebnog posla. Npr: ukoliko korisnik često ulazi na stranicu neke aviokompanije jer najčešće njene usluge koristi, bilo bi logično to keširati kako se prilikom svakog njegovog ulaska na profil te aviokompanije (ako joj podaci nisu mijenjani) ne bi uvijek iznova učitavali podaci o toj aviokompaniji. Podrazumjevano Spring keširanje omogućuje da unutar jedne sesije svaki zahtjev za objektom iz baze uvijek vraća istu instancu objekta. To je keširanje prvog nivoa. Keširanje drugog nivoa je nešto nama interesantno, kada govorimo o velikom sistemu, a ono omogućuje da se privremeno čuvaju objekti dobijeni iz baze na serveru, tako da pri svakom novom upitu upućenom ka bazi podataka, prvo provjerimo da li se traženi objekti već nalaze u kešu, što se može i parametrizovati dužinom trajanja keša.

- ## Replikacija baze podataka

  Želimo da održimo kopiju baze podataka na više servera, ukoliko bi došlo do kvara nekih računara. To održavanje ne podrazumjeva striktno kopiranje svih podataka sa jednog servera na druge, a najviše zavisi od toga šta želimo kopirati. Može se koristiti Master-Slave pristup.



Master baza podataka je glavna baza u koju se vrši upis (write), i iz koje se može vršiti, po potrebi, čitanje (read). Slave baze podataka postaju master baze podataka u slučaju da master baza podataka prestane sa radom zbog nekog kvara na serveru. Master i Slave baze bi trebale biti sinhronizovane. Što više dodajemo Slave baza možemo da još poboljšamo performanse sistema, zato što njih koristimo za čitanje, a Master za pisanje, i tako rasterećujemo Master bazu.

- ## Load balancer-i

  Na slici vezanoj za bazu podataka nalazi se i Load Balancer koji ima ulogu da usmjeri klijentske zahtjeve ka najbližem serveru, tj. onom koji je najpogodniji za njega u pogledu brzine odziva i rješenja zahtjeva koji on upućuje. Kod njih se može iskoristiti Round Robin algoritam kod kojih se zahtjevi serverima distribuiraju sekvencijalno, Least Connections algoritam koji se šalje serveru koji trenutno ima najmanji broj konekcija ka klijentima, koji je nama najinteresantniji algoritam za potrebe ove aplikacije.

- ## CDN

  Content distribution network (CDN) je veliki distribuirani sistem servera raspoređenih u više centara podataka širom interneta, i njegov cilj je da što prije krajnjim korisnicima dostavi, uz visoke performanse i maksimalnu dostupnost, sav potreban sadržaj. U našem softveru ga ima smisla koristiti naročito zato što bi softver za rezervacije trebalo da koriste ljudi širom svijeta, te da neko iz Afrike ne bi morao da dugo čeka na odgovor na neki zahtjev od servera koji je u Americi, ne bi bilo loše širom svijeta distribuirati servere, kako bi on onda uz Load Balancer mogao da iskoristi, recimo, njemu najbliži server, koji će odraditi posao koji je njemu potreban.