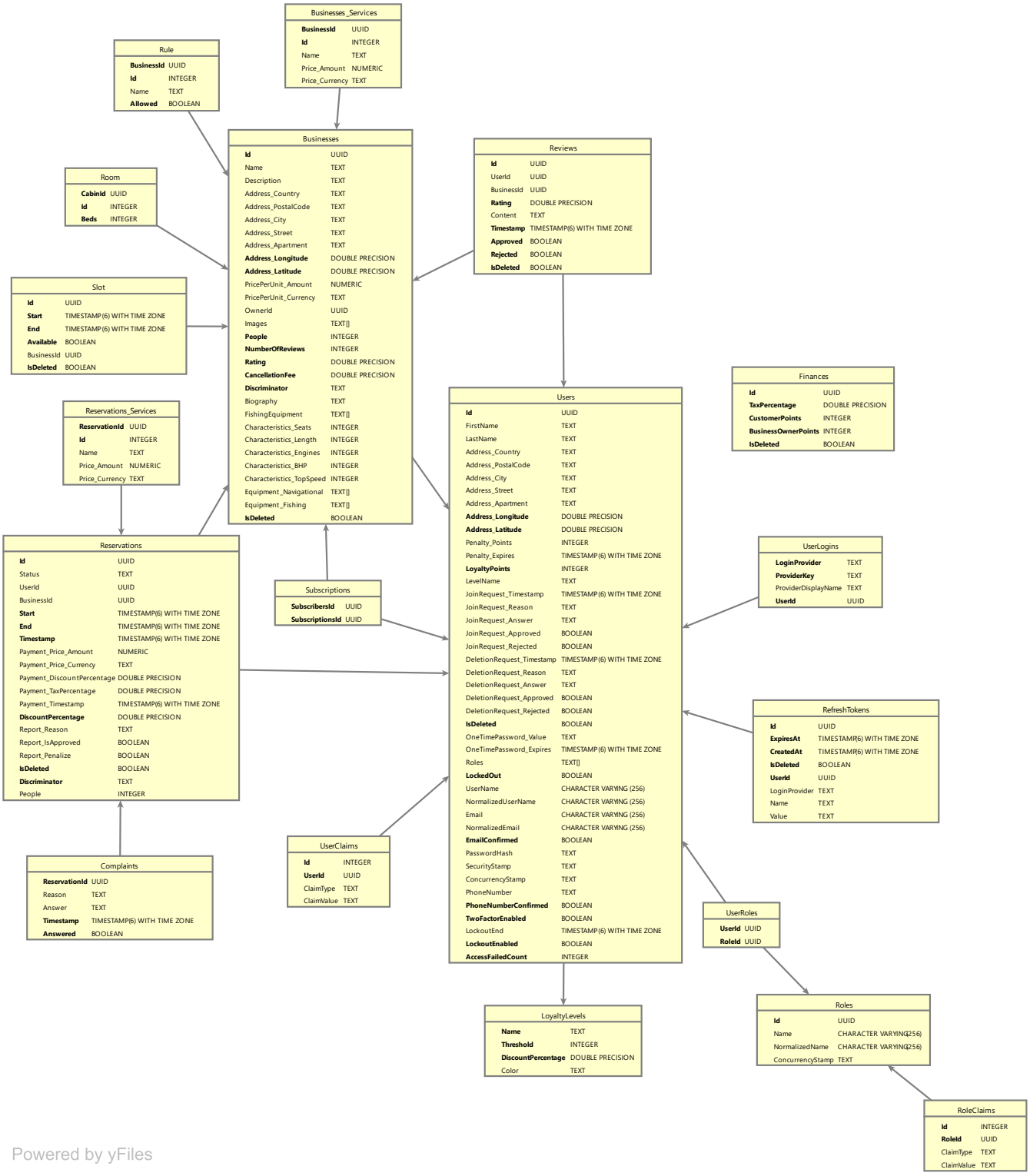
PROOF OF CONCEPT

ISA/MRS Tim 14

1. Dizajn šeme baze podataka

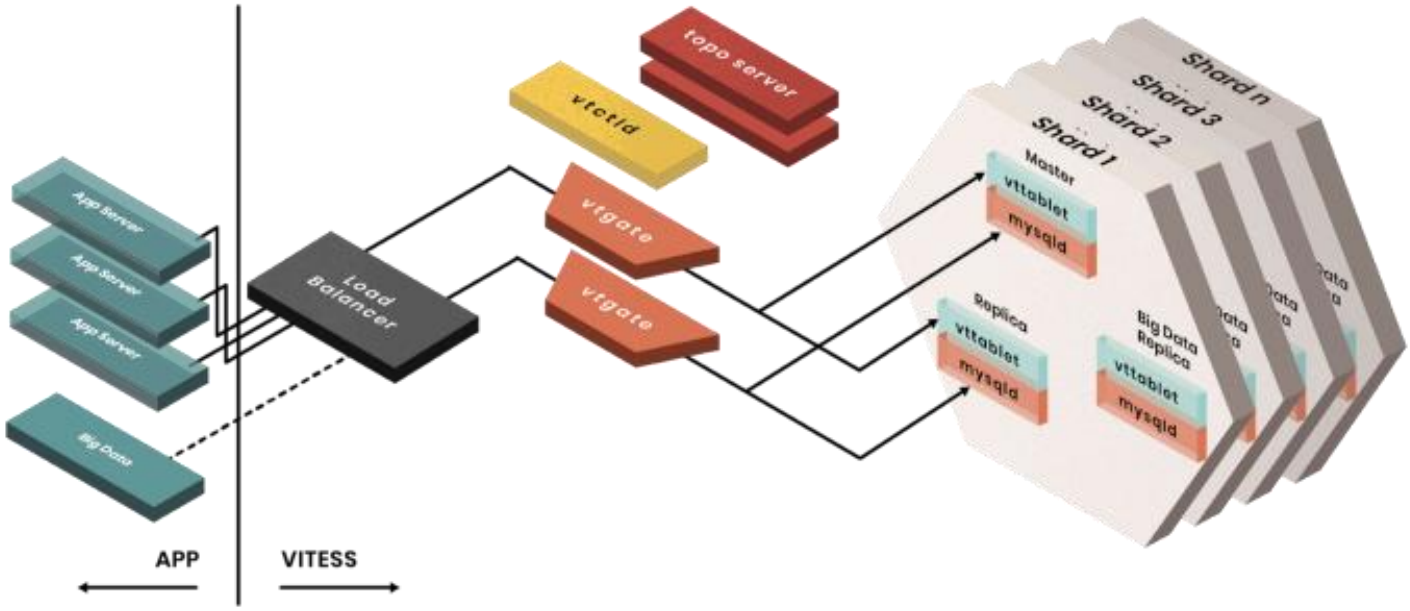


2. Particionisanje podataka

Biramo horizontalno sharodvanje (particionisanje) podataka (Iako se termini shard i particija ne odnose na isti koncept, oni su postali sinonimni). Vertikalno shardovanje nije potrebno jer redovi u našoj bazi podataka nisu toliko široki, u suprotnom bi koristili column-oriented baze podataka. Biznise bi bilo dobro particionisati po lokaciji (grad, država) iz razloga što bi veliki deo upita korisnika bile pretrage svih dostupnih biznisa na nekoj lokaciji. Rezervacije bi particionisali prema vremenima početka i kraja, iz razloga što je to najčešći način pristupa upitima koji računaju dostupnost nekog biznisa. Nakon odabira primarnog VIndexa prema kojem će se tabela particionisati, Vitess izračunava ključ ili ključeve za shardovanje za svaki upit i zatim usmerava taj upit na odgovarajući shard. Na primer, upit koji ažurira informacije o određenom korisniku može biti usmeren na jedan shard u "Users" keyspace-u. S druge strane, upit koji potražuje podatke o nekoliko biznisa na različitim lokacijama (npr. lista vlasnikovih vikendica) može biti usmeren na jedan ili više shardova u "Businesses" keyspace-u (Pogledati sliku 1 ispod).

3. Replikacija podataka

Vitess je rešenje za automatsku master-slave replikaciju koje koriste NewSQL baze podataka kao što je PlanetScale. On je apstrakcija nad MySql klasterom i Implementira semi-sync replikaciju, koja u slučaju particije na mreži ili otkazivanja mastera garantuje da bar jedan slave ima kompletnu repliku svih podataka mastera. Ovo omogućava visoku konzistentnost tako što se slave koji je najažurniji, pomoću Vitess-ovog internog orkestratora unapređuje u novog mastera. Vitess ne podržava active-active replikaciju, poznatiju kao master-master, iz razloga visoke konzistentnosti. Skalabilnost se rešava shardovanjem, svaki shard će imati svoj master node. Ova tehnologija je implementirana 2010. Godine u Youtube-u i sem njega je koriste Github, Slack i Pinterest.



Slika 1

4. Keširanje

U sistemu ovih veličina cache hit ratio mora biti oko 95%. Čak i najperformantnija i najskalabilnija baza bi bila beznačajna bez ovog procenta pogodaka. Za realizaciju keširanja

podataka predlog je Redis, distribuirani keš koji je raspoređen na više servera i pozitivno utiče na skalabilnost i performanse. Kombinacija tehnike tehnika lazy loading i write-through-cache. Ono što bismo keširali jesu podaci sa profila biznisa u kojima se nalaze ocene, akcije i druge informacije koje korisnike najviše zanimaju i moraji biti brzo dostupne. Prethodne dve tehnike omogućuju da korisnici vide novo stanje podataka kada se ono promeni, a da baza bude rasterećena od stalnih upita. Sa druge strane, postojeće rezervacije klijenata ne bismo keširali jer je kritično da pogled na ove podatke budu u realnom vremenu, kako ne bi došlo do neslaganja stanja rezervacija u bazi sa stanjem u kešu.

5. Potrebni resursi za skladištenje

Pretpostavke sistema:

- Ukupan broj korisnika od 100 miliona
- Milion mesečnih rezervacija
- Vlasnici usluga stavljaju u proseku 3 slike po entitetu
- U proseku se 50% rezervacija završi ocenom
- 5% korisnika sistema su vlasnici objekata i usluga, koji po proseku imaju 2 entiteta

Pod pretpostavkom da se model podataka neće drastično menjati, u proceni uzimamo u obzir da red korisnika u tabeli zauzima u proseku 3KB, red biznisa oko 2KB, svaka slika oko 600kb, ocena oko 1.5KB, i rezervacija oko 2KB. Navedene su procene tako da ne dolazi do podcenjivanja potrebnih resursa.

- Za 100 miliona korisnika potrebno je oko 280GB memorije
- Za procenjenih 40 miliona entita potrebno je oko 6 800 GB
- Za milion rezervacija mesečno, na godišnjem nivou potrebno je oko 23GB memorije
- Za procenjene ocene na godišnjem nivou potrebno je oko 8GB
- Za preostale entitete izdvojićemo još 20GB

Ukupna memorija potrebna za skladištenje podatak u narednih 5 godinu dana je 7TB.

6. Load Balancer

Predlog Kubernetes Cluster Ingressa kao ELB (External load balancer-a) koristeći API Gateway obrazac. Rutiramo eksterni saobraćaj sa klijentske aplikacije na backend servise. Kube-DNS unutar klastera automatski dobavlja ClusterIP adrese podova i radi round-robin load balancing. Ovaj pristup omogućava autoscaling pomoću Kubernetes Pod autoscalera. U zavisnosti od opterećenja klastera, koje posmatramo pomoću alata za monitoring, automatski povećavamo I smanjujemo broj backend servisa. Takođe je omogućen Cluster Rolling Update bez zastoja backend servisa u slučaju kada se nove slike izgrađuju i objavljuju na odgovarajući Container Registry pomoću CI/CD Pipeline. HAProxy Ingress je odabran zbog visoke propusnosti i male potrošnje resursa u odnosu na Ingress-e kao što su Envoy ili Jaeger. Ovakva arhitektura laka je za transformaciju u mikroservisnu arhitekturu pomoću Strangler-Fig obrasca za razbijanje monolitne aplikacije.

7. Monitoring

Ključni zadatak monitoringa je nadgledanje baze podataka i Kafka klastera. Vitess se mora nadgledati kako bi znali da li je master preopterećen, da li se replikacija prema slejvovima vrši

dovoljno brzo, koliki odziv imaju upiti… Klaster baze podataka je najosteljiviji deo cele arhitekture tako da se posebna pažnja mora obratiti na njega. Kafka se takođe mora striktno nadgledati kako bi se efektivno sprečio “stop-the-world” rebalans particija konzumera koji može da izazove veliko kašnjenje slanja obaveštenja putem mejla ili još gore gubitak podataka.

8. Dijagram arhitekture