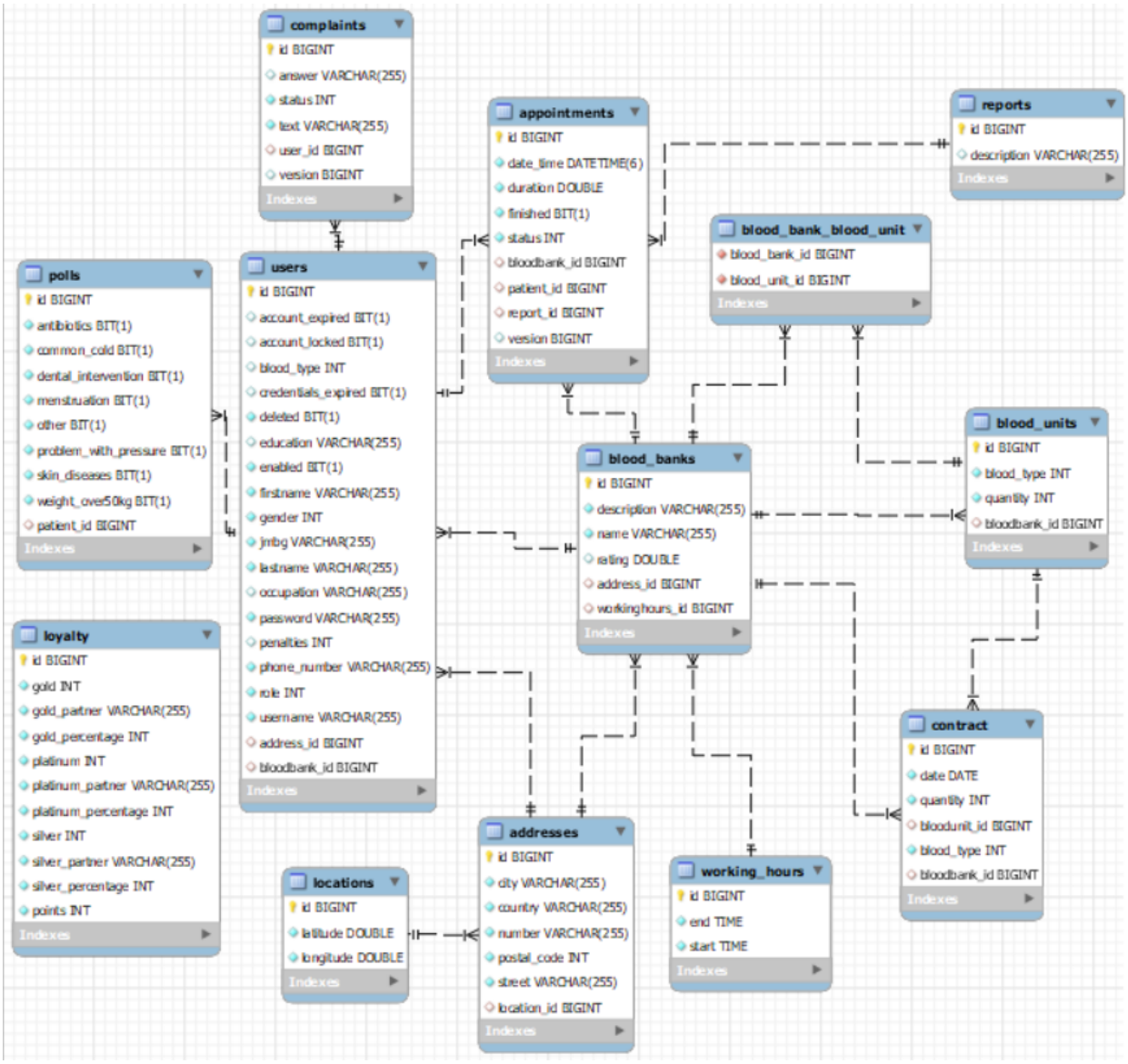
Proof of concept arhitektura

-idealno resenje-

1. Dizajn šeme baze podataka



2. Predlog strategije za parcionisanje podataka

Svaki deo baze podataka pohranjen podacima kojima se ne pristupa toliko cesto, pogodan je za particionisanje. Takođe, logično je I da se onaj deo koda koji je anotiran sa “@LazyLoading” odvoji u zasebnu tabelu, tako da jedna sadrzi podatke za koje se ocekuju da cirkulisu mnogo vise od podataka koji ce se nalaziti u drugoj tabeli, dakle od podataka koji su malo staticniji.

Primer za ovo je upravo tabela Users, naime posto se radi o zdravstvenoj ustanovi logicno je da ce se podaci kao sto su infomacije o samom korisniku ili istorija doniranja nalaziti u prvoj, a infomarcije kao sto su njegove ulozene zalbe ili pohvale u drugoj tabeli.

Postoji I geografsko particioniranje koje je primenjivo na svaki vid arhitekture, ali veoma zavisi od konteksta koriscenja. U nasem slucaju ovo bi imalo smisla, ali sa razumnim brojem servera koje cemo koristiti u odrzavanju nase aplikacije za koju se ocekuje da ima oko 10 miliona korisnika.

Na kraju vredno je I spomenuti horizontalno particioniranje koje bi za nasu aplikaciju bilo I najkorisnije I najprirodnije. Kao sto se I radi u realnoj praksi svaki pregled, donacija krvi I slicno cuva po kvartalima, tako bi I mi mogli da unapredimo nas sistem particioniranjem nase baze u vremenske od interval od mesec I godinu dana. Samim tim bi dosta olaksali pretragu termina, kao I omogucili sebi kreiranje raznih analiza koje bi potencijalno unapredile nas biznis.

3. Predlog strategije za replikaciju baze I obezbeđenje otpornosti na greske

Jedna jako opasna odluka, kada se projektuje sistem za ovaj broj korisnika, je imati samo jednu bazu, tu smo u riziku od single point failure-a, odnosno ako se nesto desi s tom bazom, ne postoji backup. Taj problem se resava uvođenjem replikacije baze, odnosno implementiranjem više verzija iste baze koje međusobno komuniciraju, gde neke služe isključivo za čitanje (slave) a druge za pisanje I koordinaciju sa slave bazama (master). Ovaj standardni master-slave princip je podržan od strane MySql baze za koju smo se mi odlučili. Još jedna opcija koju nude MySql baze je multi-master princip u kojem se određene instance baze mogu ponašati kao I master I slave u slučaju otkaza jedne od njih, a kako je projektovani broj korisnika nase aplikacije preko 10miliona ne bi bilo loše razmisliti o ovoj opciji.

Što se tiče strategija za repliakciju baze, full table replikaciju bi trebalo primeniti na neke osetljivije podatke kao što su podaci o zakazanim pregledima. A za manje bitne podatke je pogodniji logical replication da ne bismo opterećivali sistem sa kopiranjem kompletnih tabela.

Očuvanje konzistentnosti bi bilo osigurano jednostavnim propagiranjem operacija koje dovode do promena u bazi, dok bi se od slave instanci zahtevao feedback o uspešnosti izmene podatka nakon čega bi se ponavaljala propagacija.

4. Predlog strategije za kesiranje podataka

Kako JPA nudi automatsko L1 kesiranje o ovom aspektu kesiranja nece biti puno reci u ovom segmentu.

Za L2 kesiranje smo odabrali EhCache zato sto se cini kao optimalno resenje za sistem koji opsluzuje veliki broj korisnika zbog proste cinjenice da nudi opcije kao sto su

- Distribuirano kesiranje
- Vec ugradjena replikacija
- Mehanizam za perzistenciju
- Mogucnost definisanja razlicitih strategija kesiranja
- Mogucnost definisanja razlicitih strategija izbacivanja objekata iz kesa

Jos jedan veliki bonus je to sto je EhCache *Open Source,* sto automatski znaci da ce imati Community Support, da je besplatan (odnosno da ne donosi skrivene troskove u vidu licenci) I da je moguce videti kod, sto olaksava resavanje neocekivanih ponasanja I bagova.

Neke od najkoriscenijih biblioteka sa slicnom primenom su Memcached I Reddis. Memcached, iako ga koriste mnogi jednostavno ne ispunjava zahteve koji dolaze sa ovakvom aplikacijom iz prostog razloga sto predstavlja jednostavan key value pair storage, sto znaci da ne podrzava kompleksne tipove podataka I takodje ima dosta nedostataka u domenu bezbednosti. Sto se tice Reddis-a, glavni problem je u tome sto cuva ceo kes u memoriji I single threaded je, ako se ne konfigurise da koristi clustere, sto moze biti ozbiljno ogranicenje kada je rec o skaliranju aplikacije na neki ozbiljniji broj korisnika.

I pored svega ovoga izbor se svodi na preferencu, jer uz dobru konfiguraciju vecina ovih biblioteka nudi zadovoljavajuce resenje.

U cilju poboljsanja performanski naseg servisa, implementirano je I Front End kesiranje, koje staticke assete kao sto su na primer slike koji se ne menjaju, posle prvog dobavljanja sa servera, cuva u memoriji browsera sa klijentske strane.

5. Okvirna procena za hardverske resurse neophodne za skladištenje svih podatak u narednih 5 godina

Address	
AddressId[Long]	**8 bytes**

City[nvarchar] prosek 10char	10 bytes
Street[nvarchar] prosek 20char	20 bytes
Country[nvarchar] prosek 10char	10 bytes
PostalCode[int]	4 bytes
Number[int]	4 bytes
LocationId[Long]	8 bytes
Total:	64 bytes

Location	
LocationId[Long]	8 bytes
Longitude[double]	8 bytes
Latitude[double]	8 bytes
Total:	24 bytes

Appointment	
AppointmentId[Long]	8 bytes
Version[Long]	8 bytes
DateTime[Date]	8 bytes
Duration[Double]	8 bytes
Finished[Bool]	1 bit
ReportId[Long]	8 bytes
Status[int]	4 bytes
BloodBankId[Long]	8 bytes
Total:	52 bytes + 1bit

Bloodbank	
BloodbankId[Long]	8 bytes
Name[nvarchar] prosek 10	10 bytes
AdressId[Long]	8 bytes
Rating[Float]	4 bytes
Description[nvarchar] prosek 50	50 bytes

Equipment[int]	**4 bytes**
WorkingHoursId[Long]	**8 bytes**
Total:	**92 bytes**

BloodUnit	
BloodUnitId[Long]	**8 bytes**
BloodType[int]	**4 bytes**
Quantity[int]	**4 bytes**
BloodbankId[long]	**8 bytes**
Total:	**24 bytes**

Complaint	
ComplaintId[Long]	**8 bytes**
Version[Long]	**8 bytes**
Text[nvarchar] prosek 30	**30 bytes**
Answer[nvarchar] prosek 50	**50 bytes**
ComplaintStatus[int]	**4 bytes**
UserId[Long]	**8 bytes**
Total:	**108 bytes**

Contract	
ContractId[Long]	**8 bytes**
BloodType[int]	**4 bytes**
Date[Date]	**8 bytes**
BloodBankId[Long]	**8 bytes**
Quantity[int]	**4 bytes**
Total:	**30 bytes**

Loyality	
LoyalityId[Long]	**8 bytes**
Platinum[int]	**4 bytes**
Gold[int]	**4 bytes**
Silver[int]	**4 bytes**
SilverPercentage[int]	**4 bytes**
GoldPercentage[int]	**4 bytes**
PlatinumPercentage[int]	**4 bytes**
SilverPartnet[int]	**4 bytes**
GoldPartner[int]	**4 bytes**
PlatinumPartner[int]	**4 bytes**
Points[int]	**4 bytes**
Total:	**48 bytes**

Poll	
PollId[Long]	**8 bytes**
PatientId[Long]	**8 bytes**
AdressId[Long]	**8 bytes**
WeightOver50kg[Bool]	**1 bit**
CommonCold[Bool]	**1 bit**
SkinDisases[Bool]	**1 bit**
ProblemWithPressure[Bool]	**1 bit**
Antibiotics[Bool]	**1 bit**
Menstuation[Bool]	**1 bit**
DentalIntervention[Bool]	**1 bit**
Other[Bool]	**1bit**
Version[Long]	**8 bytes**
Total:	**32 bytes + 8 bit**

Report	
ReportId[Long]	8 bytes
Description[nvarchar] prosek 40	40 bytes
Total:	48 bytes

WorkingHours	
WorkingHoursId[Long]	8 bytes
Start[Date]	8 bytes
End[Date]	8 bytes
Total:	24 bytes

User	
UserId[Long]	8 bytes
Deleted[Bool]	1 bit
FirstName[nvarchar] prosek 8	8 bytes
Lastname[nvarchar] prosek 12	12 bytes
Username[nvarchar] prosek 20	20 bytes
Password [nvarchar] prosek 60	60 bytes
PhoneNumber[nvarchar] prosek 15	15 bytes
Occupation[nvarchar] prosek 15	15 bytes
Education[nvarchar] prosek 20	20 bytes
Jmbg[nvarchar] prosek 13	13 bytes
Gender[int]	4 bytes
Role[int]	4 bytes
Penalties[int]	4 bytes
Enabled[Bool]	1 bit
BloodType[int]	4 bytes
AddressId[Long]	8 bytes
BloodBankId[Long]	8 bytes
Total:	203 bytes + 2 bit

Estimacija:

- Adresa vrlo koriscena, svaki korisnik prilikom registracije mora da unese I svoju adresu. (10 miliona)
- Appointment vrlo koriscena, 500 000 mesecno iz toga sledi godinjse (6 milion).
- Banka krvi retko koriscena, u proseku 50 novih parnerstava godinje. (50)
- BloodUnit ima ih 8 za svaku krvnu grupu. (8)
- Complaint od 6 miliona predpostavlja se da samo 3% ostavi sugestivnu kritiku. (180 000)
- Contract sa polovinom nasih patrnera imamo ugovor u mesecnoj isporuci krvi. (bankakrvi/2)
- Location kao I adresa vrko korisceno, nalazi se u okviru adrese. (10 miliona)
- Loyality preko 40% nasih korisnika je u nekom loyaliry programu (4 miliona)
- Poll pre svakog termina davanja krvi potrrebno je ponuniti ankretu (6miliona)
- Report nakon svakog obavljenog termina, izdaje se izvestaj (6 miliona)
- WorkingHours vrlo retko korisnjeno. (3)

Finalna racunica:

- Adresa: 64 bytes * 10 000 000 = 640 000 000 bytes = 0.64 TB
- Appointment: 52 bytes * 5 * 6 000 000 = 1 560 000 000 = 1.56 TB
- BloodBank: 50 * 5 * 92 = 23 000 bytes = 0.000023TB
- BloodUnit: 8 * 24 bytes = 192 Kb
- Complaint: 180 000 * 5 * 108 bytes = 97 200 000 bytes = 0.09720000000000001 TB
- Contract: 0.0000115TB
- Location: 10 000 000 * 24 bytes = 240 000 000 bytes = 0.24TB
- Loyality: 4 000 000 * 48 bytes = 192 000 000 bytes = 0.192TB
- Poll: 6 000 000 * 5 * 33 bytes = 990 000 000 bytes = 0.99TB
- Report: 6 000 000 * 5 * 48bytes = 1 440 000 000 bytes = 1.44TB
- WorkingHours: 3 * 24 bytes = 72 Kb

Sve ukupno => priblizno 5.2TB.

6. Predlog strategije za postavljanje Load Balancera

Vremenom svaka upotrebljiva i uspesna aplikacija raste, unapredjuje se. Sam progres ponajvise zavisi od korisnika, kojih u pozitivnom slucaju sve vise i vise ima. S toga, u samom pocetku, manje monolitne aplikacije (kao i nasa u ovom slucaju) teze da se prosire u horizontalnom smislu, kako bi se inicijalni API, odnosno server rasteretio od mnogobrojnih zahteva od strane sve veceg broja klijenata.

Postepeno uvodjenje dodatnih servera, bi nuzno bilo propraceno uvodjenjem i odredjenog load balancera izmedju web aplikacije i servera, koji bi po nekom specificnom algoritmu (od kojih je *round robin* najpoznatiji) ravnomerno rasporedjivao zahteve ka svim serverima. Ovo bi u jednu ruku znacajno rasteretilo nas inicijalni server, dok ujedno i drasticno ubrzalo rad naseg sistema. U praksi ovaj problem se naziva *overloaded servers*.

Osim prethodno navedenog problema, izbegli bi dodatno jos jedan značajan problem. U pitanju je *single point of failure*. Radi se o vrlo nepogodnoj situaciji, koja najvise moze da utiče na zadovoljstvo krajnjih korisnika, a to je otkaz glavnog servera, i nemogućnost funkcionisanja aplikacije u periodu dok se problem ne otkloni.

Generalno, load balancer je koncept koji se moze primeniti na jos nekim mestima u arhitekturi neke aplikacije. Na primer, u kontekstu nase aplikacije, mozemo ga umetnuti i na mesto izmedju cache service-a i baze podataka. U slucaju kad bi se za tako nesto odlucili, primarno bi bilo, pre svega, da povecamo broj samih instanci baza, sto bi dovelo do redundantnosti podataka, ali i do nekih prednosti koje su pomenute u prethodnom slucaju.

Za kraj, dodatna prednost ovog slucaja je odrzavanje integriteta svih transakcija koje se obavljaju u aplikaciji, gde bi load balancer vodio racuna da neka od transakcija tokom svog izvrsavanja, iznenadno “ne umre”.

7. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Još jedan od vrlo korisnih, a kasnije, i u kontekstu biznisa, profitabilnih koncepata u okviru neke aplikacije je monitoring, odnosno mogucnost nadgledanja kompletnog rada aplikacije, kao i svih korisnickih akcija na koje cemo se i osvrnuti ovog puta.

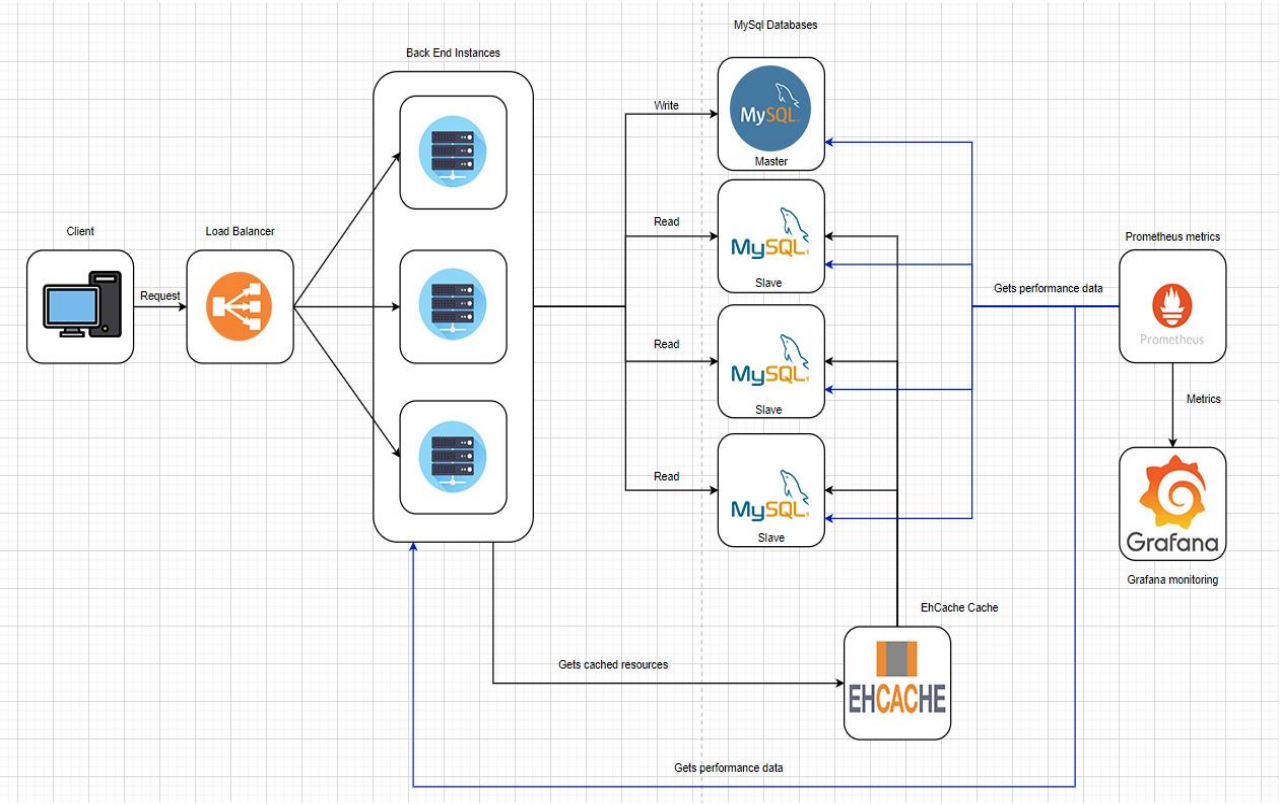
Ukoliko bi se nasa aplikacija nasla u produkciji neko odredjeno vreme, pametan korak, u cilju unapredjenja korisnickih usluga bi za pocetak, upravo bilo uvodjenje ovog koncepta, ili mozda nekog sličnog kao sto je event sourcing. U takvoj situaciji bi imali pristup podacima o samoj interakciji korisnika sa nasom aplikacijom, na osnovu kojih bi mogli da dodatno unapredimo nas softver i zadovoljstvo korisnika. Naravno, fokusirali bismo se, pre svega, na neke od najbitnijih korisnickih feature-a.

Zanimljiv predlog bi bio da se mozda implementira pametni sistem, koji nudi korisniku na prikazu, podatke o terminima koje ljudi najvise zakazuju, i to za svaki institut ponaosob, kako bi korisnik koji zeli da zakaze sebi termin izbegao eventualnu guzvu. Takodje, vrlo dobra ideja bi mogla biti da se iz statistickih podataka, izvuku najpopularniji centri za transfuziju. Pod tim, podrazumevaju se centri u kojima ljudi najvise zakazuju. Tako bi korisnici uz ocene centara i prethodno navedene podatke lakse dolazili do samog izbora.

Kako smo na samom pocetku odeljka spomenuli nadgledanje servisne aplikacije, vredno je napomenuti da nas sistem ima ugradjenu open-source aplikaciju Prometheus koja upravo prikuplja osnovne informacije o aplikaciji (iskoriscenje procesora, zauzece memorije itd.), dok za graficki prikaz istih nas sistem koristi Grafanu.

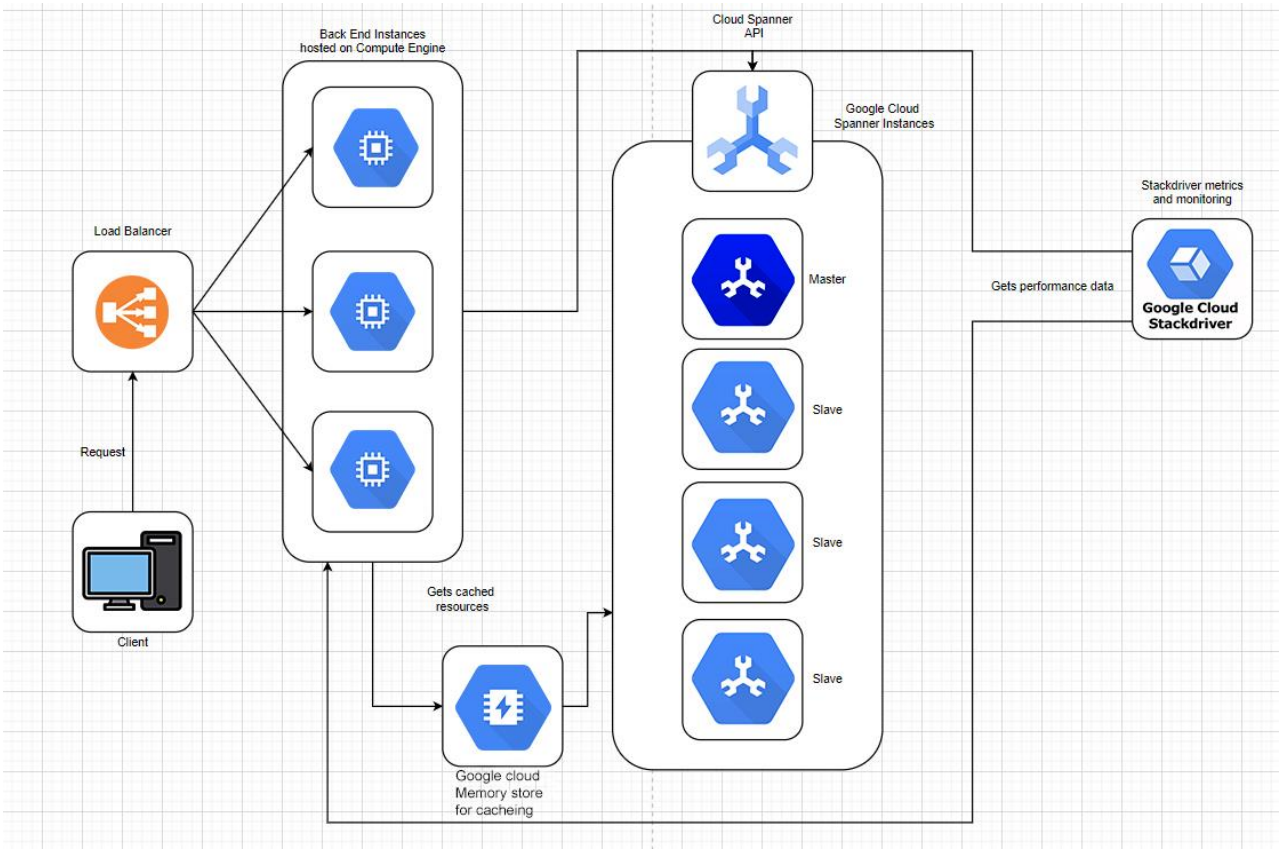
8. Kompletan crtež dizajna predložene arhitekture

Uzevši u obzir sve gorenavedene stavke I detalje implementacije iz priloženog projekta deplozment dijagram bi izgledao ovako:



Naravno ova arhitektura ima dosta očiglednih mana kao što je cinjenica da zahteva ručnu replikaciju I podelu baza kao I ručno horizontalno skaliranje samih aplikativnih servera što sa sobom nosi I konfiguraciju Load Balancera.

Kako bismo zaobišli sve ove probleme mi smo osmislili I jednu teoretsku verziju arhitekture koja se u potpunosti oslanja na Google usluge:



Korišćenje Compute Engina omogućava skoro beskonačno skaliranje aplikacije koje se automatski izvršava prateći potrebe našeg servisa. Google cloud memory store ima jednostavnu integraciju sa ostalim Google cloud elementima ove arhitekture I jednostavnu integraciju sa Google Cloud CDN-om ukoliko za to bude potrebe. Google Cloud spanner je cloud baza koja automatski radi I paprticiju I replikaciju baze opet u zavisnost od potreba servisa, I ima veoma ejdnsotavnu integraciju preko svog API-ja sa svim ostalim segmentima ove arhitekture. A kada je vec sve toliko kohezivno, dodavenje Google Cloud Stackdrivera za monitoring I grafički prikaz je bio jedini logičan korak.