TIM 7 ISA-MRS
PROOF OF CONCEPT

Aplikacija koju imamo je Java veb aplikacija izvršava na jednom računaru. Aplikacija je razvijana pomoću Spring Boot framework-a koji koristi Tomcat kao server koji izvršava naš kod (inversion of control), aplikacija koristi i jednu relacionu bazu podataka.

Kako koristimo samo jednu bazu podataka I kako aplikaciju izvršavamo na samo jednom računaru očekivane performanse nisu najbolje. Pošto smo serversku aplikaciju razvijali prema REST principima sve veze između klijenta I servera su stateless što nam omogućava da lakše izvedemo horizontalno skaliranje.

Horizontalno skaliranje servera I baze podataka

Našu aplikaciju bi pokrenuli na više različitih računara a “ispred” njih bi se nalazio uređaj koji se naziva load balancer. Njegova uloga je da rutira pristigle HTTP zahteve nekemo logikom ka računarima – serverima koji izvršavaju našu aplikaciju.

Rutiranje paketa koje load balancer izvršava može koristiti sledeće logike:

1. random – ako se brojevi generišu na dovoljno slučajan način, svaki odn servera bi trebalo da dobije oko 1/n load-a
2. round-robin
3. load-based – kod ovog načina, load balancer ima uvid u to koji server obrađuje koliki load, pa na osnovu toga donosi odluku

Sa obzirom da aplikaciju koristi veliki broj korisnika potrebno je skalirati I bazu podataka, ovde takođe možemo uraditi horizontalno skaliranje gde bi na zasebnim računarima instalirali I pokrenuli bazu podataka. Ovim pristupom rešavamo I problem dostupnosti to jest sada baza podataka nije single-point of failure sa obzirom da ih imamo više. Problem koji se javlja je sinhronizacija stanja podataka između različitih baza podataka. Rešenje je master - slave replikacija baze podataka. Master baza se bavi upisima dok se iz slave baza čitaju podaci. Nakon upisa u master bazu postojaće lag između njega I slave baza, što i jeste problem ovog pristupa, jer narušava C u ACID. Često nam nije potrebna apsolutna konzistentnost.

Pošto se uvek čita iz slave-a kojih može biti više to doprinosi boljem raspoređivanju saobraćaja.Time što postoji barem jedna kopija master-a, lako je da se on dalje replicira, tj. da se stvore novi slave-ovi. Ako ne postoji niti jedan slave, jedina opcija za stvaranje slave-a je da se pauzira master baza podataka, ceo sajt skine sa interneta, iskopira master baza

podataka, pa da se onda sve vrati online, što je previse downtime-a (može da traje desetinama sati).
Problemi:

1. ne ubrzava write
2. replikaciono kašnjenje između mastera I slave-ova

Keširanje

Hibernate nam po default-u nudi L1 keširanje tj. keš koji traje dok nam I traje sesija sa bazom podataka, u ovom slučaju hibernate kešira rezultate upita pa se u slučaju ponavljanja istog upita vraćaju podaci iz keša. Nedostatak ovog pristupa je što se sesija ne deli između više različitih niti I što keš traje dok traje I sesija.
Korišćenjem L2 Hibernate keširanja možemo da odredimo konfigurišemo I TTL koje određuje koliko će instanca živeti u kešu pre nego što bude izbačena, takođe ovaj keš je deljen izmešu različitih niti.

Connection pooling

Po default-u Tomcat server radi connection pooling je tehnika koju serveri koriste kako bi ubrzali uspostavljanje konekcija. Tomcat server pri pokretanju kreira određeni broj konekcija pa pri svakom zahtevu on dodeljuje već kreiranu konekciju klijentu. Moguće je konfigurasati broj konekcija koji će se kreirati prilikom pokretanja. Broj konekcija u svakom trenutku je potrebno odrediti na neki pametan način sa obzirom da se broj korisnika menja u zavisnosti od doba dana.

Istu tehniku možemo primeniti I nad serverom baze podataka, korišćenjem Hikari-ja možemo podešavati konfiguraciju prema našim potrebama.

Content Delivery Network

CDN - odličan za serviranje statičkih asset-a. Njih ima na više mesta u svetu (CDN points), čime se podaci distribuiraju na razna mesta u svetu, što ubrzava učitavanje. Prvo učitavanje je sporije, jer tad CDN point mora da dobavi podatke od servera, ali nakon toga su podaci keširani. Ovo smanjuje load na serveru. CDN-ovi mogu da se koriste kao kratkoročna rešenja. Oni se implementiraju za nekoliko sati. Sa druge strane, kupovina dodatnih servera radi rukovanja saobraćajem za kratak period nema mnogo smisla, jer kupovina servera povećava vreme potrebno za pripremu, tj. lead time. Osim toga, nije jednostavno proceniti koliki je tačan broj servera potreban.

Mikro-servisna arhitektura

- Prelazak na mikroservisnu arhitekturu (= skaliranje po y-osi, tj. Y-axis scaling) - prednosti: 1. ubrzava razvoj – s obzirom na to da su mikroservisi međusobno nezavisni, mogu da se razvijaju i testiraju zasebno, po sopstvenom

rasporedu 2. olakšava skaliranje – ako neki mikroservis mora da rukuje velikom količinom saobraćaja, moguće je skalirati samo njega. Sa druge strane, u monolitnom razvoju bi morala da se skalira cela aplikacija. Mane: 1. usložnjava aplikaciju (mreža zavisnosti između delova aplikacije 2. sa organizacionog aspekta može da bude teže pratiti razvoj n aplikacija, nego pratiti razvoj jedne. 3. keširanje na svakom do n servisa može da bude složeno