Skalabilnost

Skalabilnost predstavlja mogućnost prilagođavanja kapaciteta sistema da ekonomično ispuni zahteve. Skalabilnost često podrazumeva mogućnost rukovanja sa mnogo korisnika, klijenata, podataka, transakcija, itd.

Mogućnost za skaliranje se meri u različitim dimenzijama. Najčešći problemi koji se javljaju u aplikacijama i sa kojima se aplikacija mora suočiti su:

1. Rukovanje sa puno podataka. Ovo predstavlja jedan od najčešćih izazova. Kako veb aplikacija raste i postaje popularnija, potrebno je rukovati sa puno entiteta: korisnici, podaci, delovi digitalnih sadržaja.
2. Rukovanje velikim nivoom konkurentnosti. Konkurentnost se meri time koliko klijenata sistema može da bude opslužено istovremeno, bez narušavanja iskustva korisnika aplikacije. Konkurentnost je težak izazov jer serveri imaju ograničen broj procesora i *thread*-ova.
3. Rukovanje visokim stepenom interakcije. Treća dimenzija skalabilnosti je nivo interakcije između sistema i klijenata.

Što se tiče skaliranja aplikacija napisanih u backend razvojnim okruženjima, sve se svode na horizontalno i vertikalno skaliranje. Vertikalno skaliranje predstavlja ostvarenje nadogradnjom hardvera i/ili mrežnom komunikacijom. Često je najjednostavnije rešenje za kratkoročnu skalabilnost, jer ne zahteva arhitektonske promene u aplikaciji.

Horizontalno skaliranje predstavlja povećanje kapaciteta sistema dodavanjem novih servera. Smatra se veoma korisnim jer prevazilazi problem vertikalnog skaliranja, koji ima svoju granicu po pitanju performansi, pa nema smisla posle neke granice vršiti vertikalno skaliranje. Horizontalno skaliranje se isplati u kasnijim fazama. Inicijalno teži da košta mnogo jer je komplikovano za realizaciju i samim tim je potrebno više posla.

Smatramo da bi dobro rešenje za povećanje skalabilnosti naše aplikacije bio Load balancing. Kod load balancing-a imamo više posebnih servera koji mogu da izvršavaju zahteve klijenta i jedan server koji ima svoju adresu i port i koji gađamo, dok je on zadužen da dalje raspoređuje te zahteve na neki od ostalih servera. Ovo je moguće zato što naša aplikacija koristi HTTP protokol koji je stateless, tj. podatke ne vezujemo za sesiju. Prepoznavanje klijenta omogućuje nam SpringSecurity na osnovu tokena koji se prosleđuje u zaglavlju svakog zahteva. Takođe, ovo povećava i bezbednost aplikacije, jer ako se neki od servera sruši, moguće je preusmeriti zahtev na neki od drugih servera. Isto to se može uraditi i ukoliko dođe do prekida rada samog Load balancer-a, jer možemo odrediti i takozvani slave Load balancer.

Koji će od servera load balancer izabrati zavisi od toga koji algoritam koristi. Podrazumevani je da se zahtev prosleđuje onom serveru koji je trenutno najmanje preopterećen, tj. ima najmanje otvorenih konekcija. Ali moguće je koristiti i neku od drugih metoda poput Round Robin-a, kada se zahtevi rasporedjuju kružno svakom server ili Least Response Time kada load balancer bira onaj server koji ima najmanje prosečno vreme odgovora.

Takođe, potrebno je i smanjiti broj upita ka bazi jer to oduzima najviše vremena. A što se tiče bezbednosti mogla bi takođe postojati rezervna baza, iako bi se to negativno odrazilo na performanse. Ali na slave bazu ne moramo replicirati sve podatke, već samo određene neophodne.