Skaliranje Reservify platforme – aplikacije za rezervisanje letova, hotela i automobila

Aplikacija je pisana u Spring Boot-u, pa kao takva ima dostupno mnogo alata i servisa za skaliranje, ali je potrebno o tome razmišljati na vreme. Motivacija za skaliranje je uvećani broj korisnika, odnosno uvećani broj zahteva u jedinici vremena zbog kojih je potrebno napraviti distribuirani sistem koji će moći na pravi način da odgovori na potrebe korisnika. Problem je što nije jednostavno usaglasiti rad između zasebnih instanci.

Baza podataka

Trenutna implementacija aplikacije koristi MySQL. Radi skaliranja, često se preporučuje izbegavanje MySQL-a. Relacione baze podataka su dizajnirane da se nalaze na jednom serveru da bi se održao integritet šeme. Postoje određeni modeli, npr. master-slave, koji rade tako što su "slaves" dodatni serveri na koje su raspoređeni podeljeni podaci. Često da bi ovakvi modeli mogli parirati nerelacionim bazama žrtvuje se fleksibilnost radi performansi. Isto tako, troškovi su znatno veći.
Sa druge strane, NoSQL baze su dizajnirane za masovno horizontalno skaliranje. Za male aplikacije često nisu pogodne jer se isti delovi podataka čuvaju na više mesta - dovodi do redundantnosti. Ipak, u slučaju skaliranja ove platforme, preporučuje se migriranje na neku NoSQL bazu, recimo na MongoDB (zbog lakoće migriranja).

Load Balancing

Load balancing je odličan način za skaliranje aplikacije i poboljšanje performansi. Odnosi se na efikasno distribuiranje prometa na grupu backend servera (tzv *server pool*). Load balancer prosleđuje zahteve serverima na način koji maksimizira brzinu i iskorišćenost resursa. Ako neki server padne, load balancer može da uposli ostale da obavljaju njegov posao. Za ovo možemo koristiti *nginx*. Nginx se ponaša kao jedina tačka ulaza u distribuiranu web aplikaciju koja se izvršava na više zasebnih servera.

Postoji više algoritama na osnovu kojih load balancer bira server koji će obraditi zahtev, kao što su:

- Round robin - zahtevi se distribuiraju sekvencijalno
- Najmanje konekcija - zahtev se šalje serveru koji ima najmanje posla trenutno
- IP Hash - na osnovu IP adrese klijenta se šalje zahtev serveru

Svaki od navedenih metoda ima svoje prednosti ili mane, a da bi odredili koji je pogodan za našu platformu bilo bi potrebno testirati svaki. Morali bi koristiti neki od alata za praćenje performansi aplikacije, npr. *New Relic,* koji nam može dati uvid u to kolika je procesorska potrošnja, potrošnja memorije, vreme koje je potrebno serveru da odgovori, greške i neuspeli zahtevi itd.

Infrastruktura i kontejneri

Zbog tipa aplikacije, gde imamo frontend koji pruža mogućnost korisnicima da prave česte upite i zahteve ka backend-u, zbog rezervacija, pretraga, filtriranja, itd. ideja je da razdvojimo frontend i backend aplikacije sa redom (eng. *queue*). Frontend šalje poruke na red a backend ih obrađuje jednu po jednu. Prednosti ovoga su:

- Ako backend postane nedostupan, red se ponaša kao bafer

- Ako sa frontend-a stiže više zahteva nego što backend može da obradi, te poruke su baferovane u redu i čekaju
- Moguće je skalirati backend nezavisno od frontend-a, odnosno, možemo imati hiljadu instanci frontend-a a samo jednu backend-a (iako bi to bila loša ideja)

Za implementaciju reda možemo koristiti JMS (Java Message Service), mehanizam za slanje i primanje poruka korišćenjem standardnih protokola. Pored toga, možemo koristiti i ActiveMQ (open source server). JMS nam pruža jedinstven interfejs za primanje i slanje poruka na red (ActiveMQ server) koje backend obrađuje.

Za sam deploy možemo koristiti Kubernetes koji na osnovu naših parametara pruža odgovarajuću infrastrukturu i time olakšava setup. Prednost je i što možemo koristiti *minikube,* lokalni Kubernetes kluster za lokalno testiranje infrastrukture.

Kubernetes pruža set alata za upravljanje kontejnerima unutar aplikacije. Preporučuje se zbog:
- Skalabilnosti - povećanjem resursa koji su alocirani aplikacije se mogu sa lakoćom skalirati i odgovoriti na uvećane potrebe. To skaliranje može biti manuelno, gde navodimo tačno koliko će se instanci koristiti. To je korisno kada znamo tačan momenat kada će postojati uvećana potreba za servisima koje naša aplikacija nudi. Možemo koristiti i autoskaliranje, koje automatski uvećava broj instanci po pravilima koje mi zadamo.
- Verzionisanje - nova verzija aplikacije pravi novu repliku kada je deploy-ovana. U slučaju regresije, sa lakoćom se možemo vratiti na prethodnu verziju.
- Load-balancing - kubernetes daje ip svakom od kontejnera što olakšava load-balancing.

Potom jednostavnim komandama uvećavamo replike određenih kontejnera, npr:
```
kubectl scale --replicas=5 deployment/backend
```
Ili možemo koristiti autoskaliranje koje samo uvećava odnosno smanjuje broj replika po potrebi.

Kubernetes daje mogućnost automatskog skaliranja kluster-a takođe. Pored toga što možemo uvećavati broj instanci po potrebi, to je moguće i sa brojem klastera koji sadrže te instance.

Aplikacije koje su deploy-ovane sa Kubernetes moraju biti upakovane u kontejnere. Kontejneri sadrže dependencies (zavisnosti) koje naša aplikacija koristi. Popularna tehnologija za ovo je Docker. Docker ima brojne prednosti kao što su:
- Konzistentnost - aplikacija se razvija u istom okruženju i kao takva stiže do produkcije
- Izolacija - kontejneri su izolovani od mreže, fajl sistema i ostalih procesa
- Brzina

Preporučena, redom bazirana arhitektura je pogodna jer je relativno jednostavna za implementaciju a odlično razdvaja servise koji posle mogu da budu skalirani, analizirani i deploy-ovani nezavisno.