Skaliranje node.js-a

Naš projekat je izrađen koristeći Node.js, koji je poznat po dobroj skalabilnost. Pošto je node.js jednonitni server, moguće je skalirati ga horizontalno na nivou jednog računara i na nivou više računara.

Horizontalno skaliranje na jednom računar

Na jednom računaru je moguće skaliranje tako što se pokrene više instanci našeg programa, koji je jednonitni. Može se pokrenuti onoliko instanci koliko ima jezgara. Load balancer u ovom slučaju je softverski. Node js ima posebne module (PM2, The cluster module) koji automatizuju proces skaliranja na jednoj mašini (i uključuju load balancer).

Horizontalno skaliranje na više računara

Za skaliranje na više mašina potrebno je koristiti mrežni load balancer. U ovom slučaju horizontalno skaliranje node.js-a se ne bi razlikovalo značajno od horizontalno skaliranja servera drugih tehnologija. Moguće je ukombinovati horizontalno skaliranje na nivou jednog računara i na nivou više računara da bi se postigla maksimalna iskorišćenost resursa.

Pored toga, naša aplikacija ne koristi sesije, već web tokene. Stoga je komunikacija sa klijentom potpuno stateless. Ovo znači da nijedan zahtev upućen od strane klijenta ka serveru ne zavisi od stanja servera ni prethodne komunikacije između klijenta i servera. Zbog ovoga je moguće horizontalno skalirati aplikaciju na više računara bez brige o praćenju korisnika.

Baze podataka

Naša aplikacija koristi 2 baze podataka: PostgresSQL i MongoDB. Mongo sadrži većinu podataka, dok Postgres sadrži samo podatke za koje je neophodno da ostanu konzistentni (rezervacije). Prednost u ovome je što je Postgres rasterećen i ne sadrži puno podataka. Mongo daleko bolje skalira od SQL baza podataka, ali ne podržava transakcije, pa je Postgres neophodan. Rezultat ovoga je da većinu podataka možemo lako skalirati na više mašina.

Još jedna stvar koja bi mogla da se uradi jeste da se podaci iz Postgres-a kopiraju u Mongo, kako bi bilo koja operacija čitanja nad podacima mogla biti obavljena nad MongoDB. Na taj način bismo izgubili konzistetnost podataka pri prikazu, što je prihvatljivo, jer se Postgres dodatno rasterećuje, a podaci u njemu ostaju konzistentni.

Kao poslednji način da povećamo skalabilnost, mogli bismo da podatke iz Postgres-a nad kojima više neće biti izmene (rezervacije koje su protekle) prebacimo u Mongo i izbrišemo is Postgres-a. Na taj način bi oslobađali prostor u Postgresu i dodatno ubrzali njegov rad. Ovo ne bismo radili svaki put kada bi jedna pojedinačna rezervacija istekla, pošto bi ih bilo mnogo i bilo bi prečesto, nego po predefinisanom rasporedu, i to u momentima kada je broj zahteva koji pristižu od klijenata minimalan.

Definisanje rasporeda i periodičnosti izvršavanja ovog postupka bismo mogli postići preko node-kron skripti.

Keširanje

Bez obzira što je Mongo veoma skalabilan, postoji granica opterećenju koje on može da pretrpi. Veoma velik broj zahteva koji nešto traži od Monga su isti, traže podatke koji se ne menjaju često (profilna strana hotela, lista svih hotela itd...), zbog ovoga bismo mogli da ove podatke keširamo na serveru, i pri novim zahtevima klijenata, proverimo da li se traženi podaci nalaze u kešu, i ukoliko je moguće, odgovorimo na zahtev bez dodatnog operećivanja Mongo baze.

Message queue

Još jedno moguće rešenje povećavanja skalabilnosti aplikacije bi bila putem message queue-ova. Message queue-ovi pružaju efikasan način da se podrži velik broj konkurentnih poruka. Problem sa ovim je, to što bi se arhitektura aplikacije morala značajno promeniti da bi se maksimalno iskoristio message queue sistem. Najbolje bi bilo aplikaciju podeliti u tri servisa, pritom da bi najlogičnija podela bila takva da se automobili, hoteli i letovi izdvoje u zasebne servise.

Message queue sistem je lako skalirati, pošto ukoliko nam je potrebna brža obrada zahteva, možemo povećati broj trenutno pokrenutih servisa. A čak i ukoliko ne povećamo broj servisa, i dalje će svi zahtevi biti obrađeni, samo što će poruke duže čekati u redu.

Zbog servisne arhitekture takođe dobijamo prednost da je lakše dodavati nove funkcionalnosti u aplikaciju, na primer, ukoliko bismo hteli da dodamo rezervisanje vozova, trebali bismo samo da napravimo dodatan servis za vozove.

Takođe bismo dobili povišenu pouzdanost i robustnost, pošto poruke perzistiraju u bazi dok se ne iskoriste, što bi značilo da u eventualnim slučajevima pada naših servisa oporavak bi i dalje bio moguć, a zahtevi se ne bi gubili.