Proof of Concept

Mikroservisna arhitektura

Kako bi trenutni sistem učinili dostupnim za veliki broj korisnika, kao prvo je potrebno promeniti arhitekturu čitavog sistema. Ne bi više imali jednu monolitnu aplikaciju već bi nju izdelili na više, što nezavisnijih, mikroservisa. Svaki mikroservis bi bio zadužen za odredjene funkcionalnosti sistema, ukoliko se nadje velika potražnja za odredjene funkcionalnosti mikroservis zadužen za njih se može pokrenuti u više instanci.

Load Balancer

Ukoliko dodje previše zahteva do odredjenog servera može doći do njegovog zagušenja. Rešenje ovog problema nam pruža Load Balancer. Load Balancer je softver koji prihvata zahteve od korisnika i te zahteve dalje prosledjuje odgovarajućim serverima koji nisu previše opterećeni. Još jedna prednost Load Balancer-a je što korisnik preko njega komunicira sa serverima pa samim time i sakriva internu strukturu servera.

Database

Upiti ka bazi su skupi i zbog toga ih treba maksimalno optimizovati. Upite treba konstruisati tako da vraćaju samo podatke koji su nam od interesa, koristiti paginaciju, koristiti index-e itd. Takodje je potrebno koristiti i optimističan način zaključavanja tabela koji podarzumeva da se tebela neće zaključati pri pristupanju, već se svakoj trojki dodaje novi atribut koji ukazuje na verziju trojke. Verzija se povećava svaki put kada dodje do izmene trojke, pa se na taj način detektuje da li je došlo do izmena za vreme obavljanja nekog upita. Ovakav način rada je pogodan za povećavanje performansi, ali pod predpostavkom da će do kolizija retko dolaziti.

Možemo dodati i više baza ukoliko se javi potreba za tim, time bi sprečili problem uskog grla gde veliki broj mikroservisa pristupa jednoj bazi, što usporava rad sistema. Postojale bi baze zadužene za write zahteve i baze za read zahteve. Baze za write zahteve bi medjusobno bile sinhronizovane i za njih vezane baze za read operacije kojima bi se slale promene kada dodje do njih.

Cache mehanizam

Dobra utilizacija keširanja omogućava poboljšanje performansi sistema. Možemo keširati statičke fajlove frontend aplikacije kao što su css ili js, što se tiče backenda mogu se čuvati proračuni metoda.

Hardverske komponente

Kada je sve maksimalno optimizovano sa strane aplikacije i sistema, povećanje dostupnosti sistema vršimo poboljšanjem hardvera. Kako raste broj korisnika moraju se kupovati bolje komponente i uredjaji, nabavljaju se novi serveri, resursi se reorganizuju itd. Odredjene mikroservise, kod kojih nismo sigurni koliko korisnika će njihove funkcionalnosti koristiti, možemo u početku postaviti na cloud i kasnije kupiti fizičke komponente ili samo zakupiti veći prostor na cloud-u.

Otpornost na otkaze

Kako bi sistem bio što dostupniji i poverljiviji korisnicima i kako bi im omogućio da svoje poslove obavljaju što efikasnije, sistem mora biti otporan na otkaze. Otpornost na otkaze podrazumeva da će sistem reagovati na najčešće otkaze i najopasnije otkaze i otkloniti ih ukoliko je to moguće bez narušavanja funkcionalnosti sistema. Za komponente koje su krucijalne za rad sistema uvek treba imati duplikat. Prethodno spomenuto korišćenje više baza nam nudi očuvanje podataka u slučaju kvara neke od baza. Aplikaciju možemo pokretati na više servera, ako jedan zakaže uvek ćemo imati duplikat. Uvek bi trebalo da imamo backup, kopiju, sistema. Backup se može primeniti na samu aplikaciju, stanje u bazi pa čak i na sam hardver koji je neophodan za rad sistema. Performanse i dostupnost sistema takodje povećava i računarski cluster. Cluster se sastoji iz medjusobno povezanih računara, gde svaki računar ima isti zadatak. Zadaci koje sistem obavlja se dele izmedju clustera što omogućava nastavak rada sistema ukoliko neki njegov deo otkaže. Za upravljanje clasterima bi mogli koristiti Docker ili Kubernetes.