



Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Proof of Concept (PoC) arhitektura za skalabilnu aplikaciju

Jun 2022.

Autori:

Nikola Damjanović, SW71/2019

Vladan Mikić, SW76/2019

Ilija Kalinić, SW65/2019

Predmet:

Internet Softverske Arhitekture

Tema:

Proof of concept arhitektura

Sadržaj

1	Uvod	1
1.1	Rad sa podacima	1
1.2	SUBP	1
2	Dizajn šeme baze podataka	2
3	Predlog strategije za particionisanje podatka	3
4	Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške	3
5	Predlog strategije za keširanje podataka	3
6	Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina	4
7	Predlog strategije za postavljanje load balansera	4
8	Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema	5
9	Dizajn arhitekture	6

1 Uvod

Prepostavke proof of koncepta su sledeće:

- ukupan broj korisnika aplikacije je 100 miliona,
- broj rezervacija svih entiteta na mesečnom nivou je milion,
- sistem mora biti skalabilan i visoko dostupan za sve korisnike

U skladu sa prepostavkama zaključujemo da je najbojni izbor arhitekture za ovakav sistem distribuirani sistem na bazi mikroservisa. Za aplikaciju ovolikog razmara, neophodno je podržati komunikaciju između distribuirane mreže mikroservisa na sinhron i asinhroni način, podržati oporavak od greške na svim nivoima (od nivoa operativnog sistema do nivoa cele aplikacije) i load balancing. Takođe su od izuzetnog značaja alatke koje mogu automatizovati određene procese u distribuiranom sistemu kao što je ažuriranje određenih delova aplikacije. Kubernetes bi nam u ovom slučaju pružao većinu alatki koje bi zadovoljile naše potrebe za orkestracijom distribuiranog sistema ovakve razmara.

1.1 Rad sa podacima

Sistem skladištenja, ažuriranja i dobavljanja podataka bi takođe morao biti vrlo efikasan da bi mogao doličiti distribuiranom sistemu mikroservisa. Zbog potrebe za međusobnom nezavisnošću mikroservisa, mikroservis na neki način mora da "poseduje" podatke sa kojima rukuje. U praksi se pokazalo da je najefikasnija implementacija "database per service" šablon skladištenja, dobavljanja i ažuriranja podataka, ako se radi sa mikroservisima.

1.2 SUBP

Za ispunjavanje potreba distribuiranosti kao i konkurentnih pristupa resursima, potrebno je podatke skladištiti u SUBP koji podržava paralelan rad u više instanci. Distribuirane transakcije, replikacije i skaliranje su sve neophodne potrebe koje naš SUBP treba da podržava. RavenDB je odličan izbor zbog toga što je ovo distribuirana NoSQL baza podataka koja podržava mehanizme optimističkog i pesimističkog zaključavanja resursa i ACID transakcije. RavenDB baze se mogu smeštati u klastere u kojem svaka instance sadrži sve podake. Postoje "main" i "replica" instance u svakom klasteru od kojih svaka ima mogućnost upisa i čitanja, nezavisno od uloge. RavenDB podržava redove događaja za potrebe asinhronne sinhronizacije, ali pored toga je pogodno koristiti i "message broker" kao što su Redis i RabbitMQ.

2 Dizajn šeme baze podataka

U realnoj primeni, ova šema bi se verovatno dodatno proširila kako zbog potreba sistema, tako i zbog potreba broja korisnika u redu od stotina miliona.

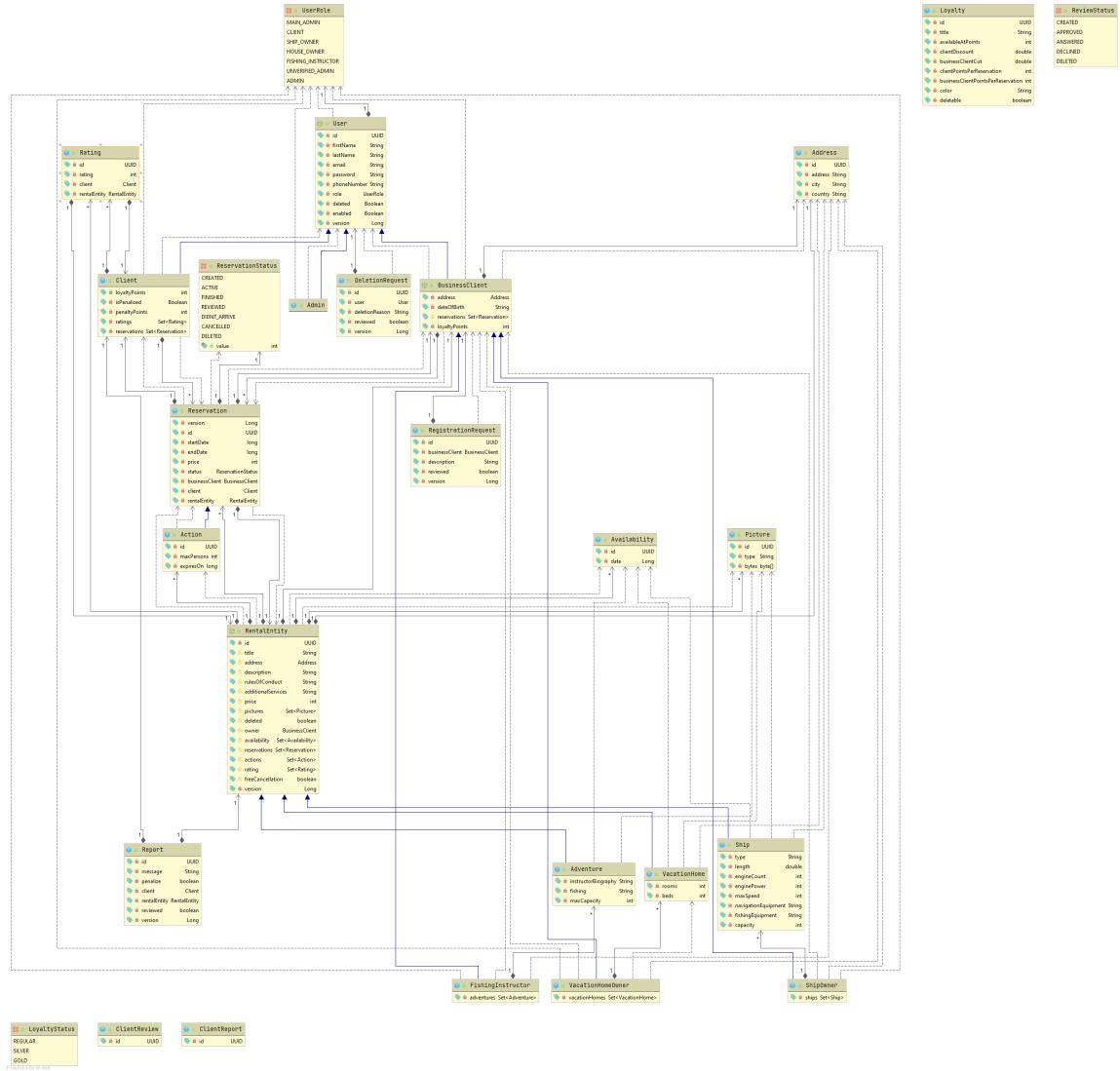


Figure 1: Šema baze podataka

3 Predlog strategije za particonisanje podatka

Zbog veoma velike količine podataka koji bi mnogobrojni korisnici na dnevnom nivou kreirali, pogodno bi bilo da se podaci podeli u sopstvene podsisteme, a takođe i da se izvrši particonisanje. Particionisanje se vrši horizontalno i vertikalno. U većini slučajeva, postoje podaci kojima se češće pristupa, i oni kojima se ređe pristupa. Iz nekih entiteta sistema možemo izvući podatke za koje smatramo da su ređe potrebni i odvojimo ih u posebne datoteke u RavenDB bazi.

4 Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Jedna od najbitnijih stvari kod aplikacija ovakvih razmara je da postoji više kopija podataka smešteni na fizički odvojenim mašinama. Na taj način se postiže kreiranje "backupa" podataka, i time i redundantnost i sposobnost oporavka sistema od vecih gresaka. Ako jedna baza prestane da radi, tu će uvek biti bar jedna da je nadoknadi. Strategija replikacije bi bila takva da postoji hijerarhija baza podataka po tome kakav autoritet imaju i koliko će se često pristupati i ažurirati podaci. Master-slave arhitektura koja je sprovedena u klasterima baze bi bilo dobro rešenje za ovakav problem, gde je u svakom klasteru baza master baza "glavna" od koje se prave kopije.

5 Predlog strategije za keširanje podataka

Keširanje bi se u sistemu radilo uz pomoć Redisa. Kako bi dodatno rasteretili mikroservise od čekanja na odgovor, koristili bismo i asinhronu sinhronizaciju za potrebe replikacije uz pomoć RabbitMQ.

Podaci koje je neophodno keširati zbog toga što korisnici najčešće njima pristupaju su:

- Opšti podaci o entitetima koji se rezervišu, sa akcentom na slikama jer će to biti resurs najveće veličine koji je potrebno brzo dobaviti do korisnika
- Informacije o rezervacijama i akcijama
- Loyalty programi uz koje se računaju popusti za klijente i bonusi za poslovne klijente.

Replikacija bi se vršila nad sledećim podacima:

- Zahtevi za registracije klijenata i menadžera sistema
- Ocene entiteta i poslovnih klijenata
- Žalbe od poslovnih i običnih klijenata

6 Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina

Prosečno jedinično zauzeće korisnika na disku je oko 1,5 KB. To bi značilo da je za 100 miliona korisnika potrebno oko 200GB za sigurno čuvanje podataka svih korisnika u sistemu sa redundancijom od 25%.

Što se tiče rezervacija, žalbi i ocena, procenjujemo da je ukupno potrebno oko 400GB na petogodišnjem nivou ako će se na mesečnom nivou praviti milion rezervacija (svaki skup ova 3 entiteta bi jedinično zauzimao oko 10KB).

Entiteti sistema minimalno zauzimaju oko 10MB ako bismo uzeli kao primer da svaki entitet ima 5 slika od kojih je svaka 2MB, pored ostalih podataka. Ako svaki poslovni klijent ima po 2 entiteta koja se mogu rezervisati, i odnos poslovnih klijenata i svih korisnika je 20%, to bi bilo ukupno oko 400TB.

Pošto je računica u redu terabajta, smartamo da bi bilo dovoljno imati bar 600TB za skladištenje na petogodišnjem nivou.

Problem predstavljaju slike jer ih trebamo skladištiti i kompresovati na način da ne narušimo izgled slike mnogo, što predstavlja najveći problem u skladištenju podataka korisnika sistema.

7 Predlog strategije za postavljanje load balansera

Load balancing je ključna stvar našeg sistema, zbog klastera mikroservisa. Adekvatan load balancer se mora koristiti kako bi se postigla maksimalna brzina pristupa nekom od servisa ili keševa. Azure Application Gateway bi se koristio kao load balancer zbog toga što uzima u obzir fizičku i internet lokaciju servera kome će prvom pristupiti.

8 Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Nadgledanje bi u našem slučaju bilo veoma pogodno da je na nivou klastera za jedan mikroservis i na nivou klastera baze podataka. Na ovaj način imamo posebno uvid u performanse različitih mikroservisa koji bi trebalo raditi nezavisno jedni od drugih, kao i uvid u performanse baze podataka. Na ovaj način vrlo lako možemo otkriti gde je tačno usko grlo našeg sistema, i prema tome možemo unaprediti taj deo sistema nezavisno od ostatka.

9 Dizajn arhitekture

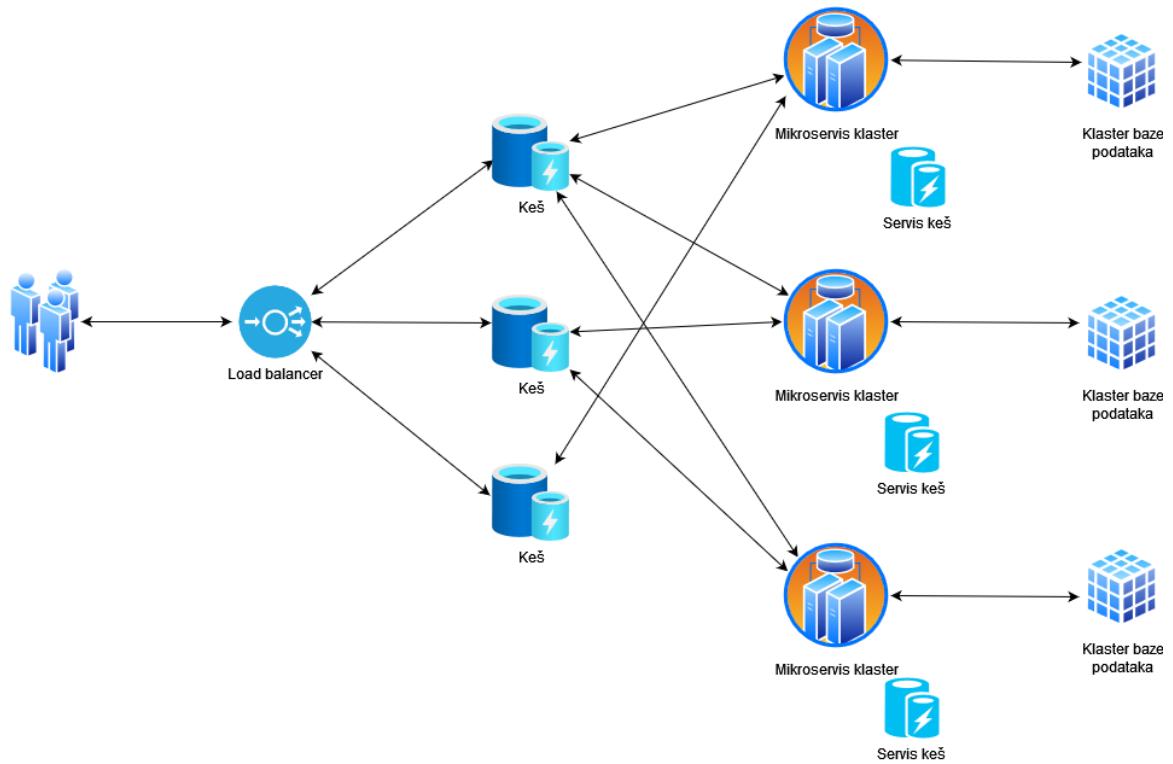


Figure 2: Šema dizajna arhitekture