IMPLEMENTACIJA MIKROSERVISNE ARHITEKTURE KORIŠĆENJEM HETEROGENIH ALATA I BIBLIOTEKA

Irfan Fetahović, [1] Aleksandar Milanović[2], Aldina Avdić[3], Lejlija Memić[4],

Rezime: Mikroservisna arhitektura je nova paradigma za kreiranje savremenih softverskih sistema. Ona strukturira aplikaciju kao kolekciju labavo povezanih, sitno granuliranih servisa koji komuniciraju preko jednostavnih protokola. Ovaj pristup omogućava paralelizaciju razvoja tako što dozvoljava manjim, autonomnim timovima ili pojedincima da nezavisno razvijaju, isporučuju i skaliraju servise. Zbog svojih prednosti, postao je široko prihvaćen i korišćen od strane mnogih vodećih kompanija u softverskoj industriji. U ovom radu opisana je mikroservisna arhitektura i predstavljena je na primeru aplikacije *Netflix Clone* koja je razvijena korišćenjem ovog arhitektonskog stila. Aplikacija pokazuje mnoge prednosti mikroservisa, kao što su povećana modularnost, lakše razumevanje, razvoj i testiranje. Štaviše, mogućnost korišćenja različitih programskih jezika, alata i biblioteka, aplikacija za razvoj jasno pokazuje osobinu fleksibilnosti, što je jedan od glavnih razloga popularnosti arhitektonskog pristupa mikroservisima.

Ključne reči: mikroservisi, arhitektura softvera, granulirani servisi

AN IMPLEMENTATION OF MICROSERVICES ARCHITECTURE USING HETEROGENEOUS TOOLS AND LIBRARIES

Abstract: Microservices architecture is a new paradigm for creating modern software systems. It structures an application as a collection of loosely coupled, fine-grained services which communicate through lightweight protocols. This approach facilitates development parallelization by allowing small, autonomous teams or individuals to independently develop, deploy and expand services. Due to its advantages, it became widely adopted and used by many leading companies in the software industry. In this paper, we discuss microservices architecture and present the *Netflix Clone* application which was developed using this architectural style. The application demonstrates many benefits of microservices, such as increased modularity, easier understanding, development and testing. Moreover, by using different programming languages, tools and libraries, the application clearly shows the feature of flexibility, which is one of the main reasons of popularity of microservices architectural approach.

Key words: microservices, software architecture, fine-grained services

1. UVOD

U tradicionalnim monolitnim aplikacijama, sve funkcije su enkapsulirane u jednoj aplikaciji. Ovaj pristup može imati višestruke prednosti, kao što su brži razvoj i testiranje, i laka primena, ali samo kada je cela aplikacija relativno mala. Međutim, savremeni softverski sistemi postaju sve složeniji i ovaj arhitekturni stil dovodi do slabosti kao što su loša pouzdanost i ograničena skalabilnost [1]. Poslednjih godina, mikroservisi su postali široko prihvaćena paradigma u razvoju softverskih sistema, a mnoge kompanije, uključujući *Amazon, Twitter, PayPal* i druge počele su da implementiraju mikroservise u oblaku ili su izvršile migraciju sa zastarele monolitne na mikroservisnu arhitekturu [2]. Savremeni softverski sistemi zahtevaju visoku konkurentnost, visoku dostupnost, visoku skalabilnost, visoku koheziju i nisku povezanost. Mikroservisi predstavljaju arhitekturni stil u razvoju savremenih softverskih aplikacija. Stil je zasnovan na razgradnji monolitne aplikacije na posebne, labavo povezane mikroservise, koji se mogu nezavisno razvijati, proširivati, raspoređivati i testirati. Mikroservisi imaju sledeće karakteristike: nezavisan razvoj, nezavisna primena, nezavisna realizacija, visoka konkuren-tnost, visoka dostupnost, visoka skalabilnost, visoka koheziju i niska povezanost [3].

[1] Docent, Državni univerzitet u Novom Pazaru, Vuka Karadžića bb, 36300 Novi Pazar, ifetahovic@np.ac.rs

[2] Student, Državni univerzitet u Novom Pazaru, Vuka Karadžića bb, 36300 Novi Pazar, milanovicalex77@gmail.com

[3] Asistent sa doktoratom, Državni univerzitet u Novom Pazaru, Vuka Karadžića bb, 36300 Novi Pazar, apljaskovic@np.ac.rs

[4] Asistent, Državni univerzitet u Novom Pazaru, Vuka Karadžića bb, 36300 Novi Pazar, lmemic@np.ac.rs

Oni komuniciraju i razmenjuju podatke preko jednostavnog *HTTP* protokola i mehanizama kao što su *RESTAPI* ili magistrale za poruke.

Razlike između monolitne i mikroservisne arhitekture sumirane su u Tabeli 1 [4]:

Tabela 1. Razlike između monolitne i mikroservisne arhitekture

Monolitna arhitektura	**Arhitektura mikroservisa**
kompleksne komponente	male, sitno granulirane komponente
jedna jedina tačka neuspeha	nema jedne tačke neuspeha
holističko kreiranje i primena	svaki servis se kreira i primenjuje nezavisno
zajednička baza podataka	privatna baza podataka
koristi isti programski jezik i okvir	heterogenost u pogledu programskih jezika i okvira
nemogućnost skaliranja na zahtev	skaliranje na zahtev

Mikroservisna arhitektura deli aplikaciju servisa na više labavo povezanih servisa orijentisanih na različite i precizne zadatke i odgovornosti. Da bi se ovo postiglo, svaki servis se izvršava u različitim kontejnerima i svaki kontejner ima svoju privatnu bazu podataka kojoj drugi kontejneri ne mogu direktno pristupiti [5].

Tehnologija kontejnera postala je jednostavna alternativa virtuelizaciji zasnovanoj na hipervizoru, za aplikacije koje ne zahtevaju ekstremnu bezbednost ili strogu izolaciju. U odnosu na virtualizaciju, ova tehnologija obezbeđuje bolje performanse dok istovremeno zahteva manje računarskih resursa za primenu, pokretanje i upravljanje. Dodavanje ili uklanjanje funkcionalnosti sa postojeće aplikacije je jednostavno, što ih čini fleksibilnijim i prilagodljivijim [6]. Docker je postao najpopularnija tehnologija kontejnera i koristili smo je u našoj implementaciji [7].

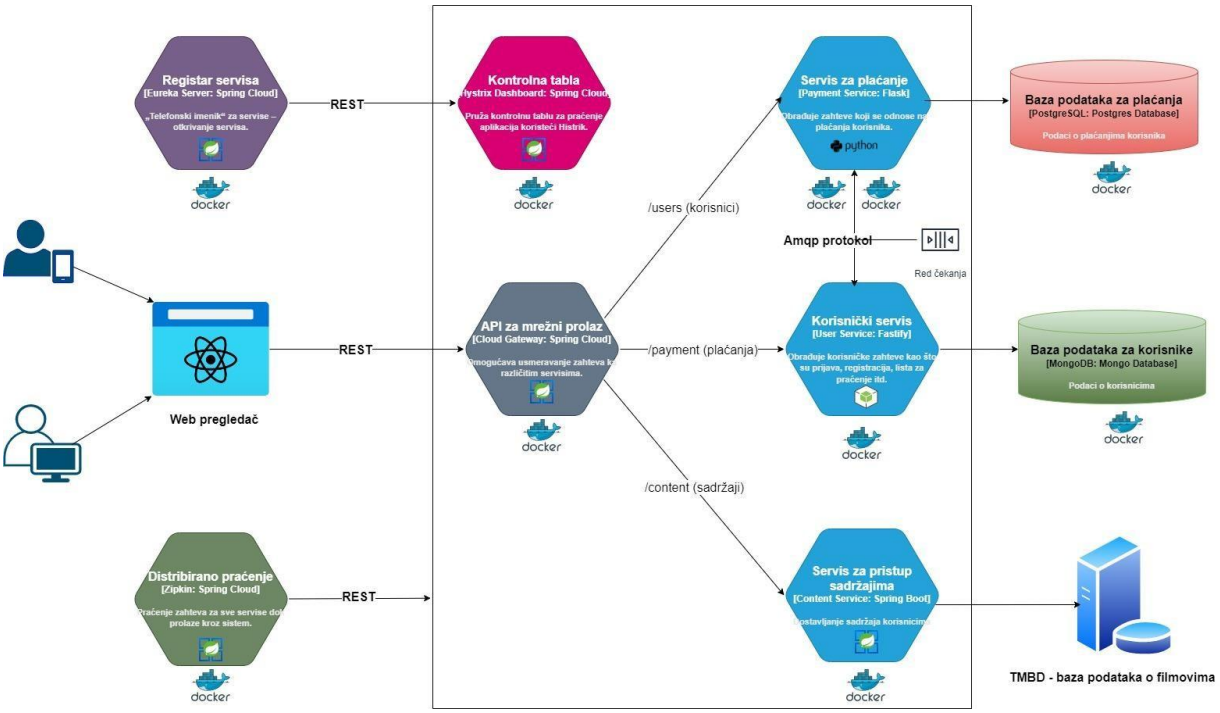
2. STANJE U OBLASTI

Većina implementacija arhitekture mikroservisa koja se nalazi u literaturi zasniva se na korišćenju jedne odabrane tehnologije, često u kombinaciji sa nekoliko platformi otvorenog koda. Autori u [8, 9] su predstavili implementaciju arhitekture mikroservisa korišćenjem tehnologija i alata zasnovanih na programskom jeziku Java. Mikroservis za pretragu razvijen korišćenjem Python-a i njegovog okvira Django predstavljen je u [10], a upotreba mikroservisa za izgradnju aplikacije za e-trgovinu sa JavaScript tehnologijama je data u [11]. U disertaciji [12] prikazana je mikroservisna arhitektura za realizaciju servisa pametnog zdravstva zasnovana na različitim tehnologijama i servisima (LMS sistemi, GIS sistemi, društvene mreže, kraudsorsing, kraudsenzing, blokčejn, IoT, obrada prirodnih jezika).

U ovom radu koristili smo mnogo različitih alata, biblioteka i jezika da pokažemo koliko se lako mogu integrisati i na taj način opširno demonstrirati fleksibilnost arhitekturnog pristupa mikroservisa.

3. IMPLEMENTACIJA MIKROSERVISNE ARHITEKTURE

Sa ciljem da pokažemo arhitekturni stil mikroservisa i njegove prednosti, razvili smo aplikaciju Netflix Clone, koja ima slične funkcionalnosti kao popularna aplikacija Netflix [13]. Arhitektura aplikacije je prikazana na slici 1. Klijentski deo aplikacije, odnosno korisnički interfejs, napisan je programskim jezikom JavaScript, dok je serverski deo aplikacije kombinacija nekoliko tehnologija, za koje su korišćena tri programska jezika - JavaScript, Java i Python.

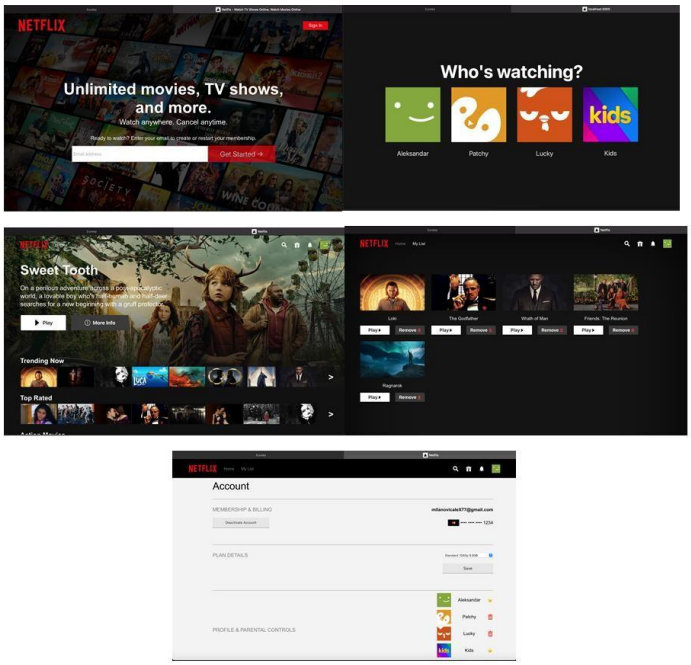


Slika 1 Arhitektura aplikacije

Mnoge biblioteke i okviri su korišćeni za kreiranje oba dela aplikacije, o čemu će biti reči u nastavku.

3.1. KLIJENTSKI DEO

Interakcija korisnika sa aplikacijom odvija se preko veb pretraživača. Korisnik pristupa aplikaciji razvijenoj pomoću Next.js okvira. Ovde korisnik može da izvrši sve radnje kao što su registracija, prijavljivanje, pregledanje sadržaja, sviđanje/nedopadanje TV emisija i filmova, dodavanje na listu za gledanje, promena plana plaćanja i još mnogo toga. Next.js je okvir na vrhu veoma popularne JavaScript biblioteke pod nazivom React.js.



Slika 2 Korisnički interfejs

React je trenutno najčešće korišćeni veb okvir za razvoj klijentskog dela dinamičkih aplikacija na jednoj stranici, a Next.js dodaje dodatni nivo jednostavnosti [11]. Neke od karakteristika koje čine Next.js prirodnim izborom za ovu vrstu aplikacije su: prikazivanje na strani servera, jednostavno rutiranje na strani klijenta, automatsko razdvajanje koda, lakoća nadogradnje itd.

Slika 2 prikazuje neke primere korisničkog interfejsa aplikacije: stranica dobrodošlice, izbor profila, pregledanje sadržaja, lista za praćenje korisnika i upravljanje profilom i informacijama o plaćanju.

3.2. SERVERSKI DEO

Serverski deo se sastoji od ukupno sedam mikroservisa, a to su:

- korisnički servis,
- servis za plaćanje,
- servis za pristup sadržajima,
- API za mrežni prolaz,
- registar servisa,
- kontrolna tabla,
- distribuirano praćenje.

Cela pozadinska arhitektura počiva na Spring Cloud ekosistemu, koji pruža ogromnu količinu alata za razvoj, praćenje i kontrolu ponašanja mikroservisnih aplikacija [14]. Pored Spring Cloud-a, postoji još pozadinskih okvira koji se koriste za neke servise, a svaki od njih opisan je u nastavku. Klijentski deo i serverski deo komuniciraju preko REST API-ja. Četiri poslednja sevisa su deo Spring Cloud infrastrukture koja ima veliku ulogu u dodavanju sloja apstrakcije servisima, rukovanju greškama i nadgledanju sistema. Kao što nazivi sugerišu, svi ovi servisi su napisani u Spring Boot-u.

3.3. KORISNIČKI SERVIS

Korisnički servis je odgovoran za sve stvari koje se tiču korisnika i njihovih ličnih podataka. Ovaj servis obrađuje zahteve za autentifikaciju, autorizaciju, upravljanje profilima, sviđanje/nedopadanje sadržaja, dodavanje sadržaja na listu za praćenje, ali i komunicira sa servisom za plaćanje kada korisnik želi da promeni svoj plan plaćanja. Dve najzanimljivije karakteristike koje ovaj servis pruža su autentifikacija (putem e-pošte i lozinke, sa potvrdom e-pošte), autorizacija JVT (JSON veb token) i interna komunikacija sa servisom za plaćanje.

Kada se korisnik registruje prvi put ili kad god se ponovo prijavi, klijentski deo dobija token koji se zove JSON veb token, koji sadrži šifrovane informacije o tom korisniku, tako da kasnije može da pristupi različitim stvarima na platformi. Ovaj token proverava serverski deo na svaki zahtev u korisničkom i servisu za plaćanje, i kontroliše da li korisnik može da pristupi određenim podacima u aplikaciji ili ne.

Drugi interesantan deo je interna komunikacija sa servisom za plaćanje. Kao što znamo, servisi za plaćanje su često veoma spori sa transakcijama, jer sve mora da se proveri i obriše, te se implementacija ovde zasniva na redovima poruka. Servis za plaćanje je odgovoran za dobijanje i brisanje informacija o korisnikovom trenutnom planu, ali korisnički servis ima ulogu kada se korisnik prvi put registruje. Kada korisnik podnese zahtev da promeni svoju trenutnu pretplatu (plan), korisnički servis će pozvati servis za plaćanje preko AMPQ protokola, koristeći CloudAMPQ (RabbitMq klaud rešenje). On će tu porukustaviti u red koristeći odgovarajuću temu, a servis za plaćanje će je obraditi i odgovoriti. Dakle, korisnički servis ovde deluje kao proizvođač, dok je servis za plaćanje potrošač. Važan zaključak je da je ova komunikacija asinhrona, tako da neće biti blokiranja u sistemu. Korisnički servis je implementiran sa Fastify Node.js okvirom i koristi NoSQL MongoDB baze podataka.

3.4. SERVIS ZA PLAĆANJE

Kao što je već pomenuto, ovaj mikroservis obrađuje sve zahteve vezane za plan plaćanja – pretplatu. Pruža API za preuzimanje, ažuriranje i brisanje informacija o plaćanju datog korisnika. Takođe, prima poruke od korisničkih servisa kada se kreira pretplata. Implementiran je korišćenjem Python-a, sa

mikrookvirom Flask, i koristi PostgresSQL. Razlog za to je što je ovaj servis prilično jednostavan, pa se Flask ovde pojavio kao prirodan izbor.

3.5. SERVIS ZA PRISTUP SADRŽAJIMA

Ovaj servis je poslednji koji se odnosi na kupca. Pruža API za pregledanje sadržaja. Ima različite rute koje vraćaju TV emisije i filmove na osnovu odabrane kategorije iz zahteva. Ovaj servis je razvijen korišćenjem Spring Boot Java okvira, a komunicira sa eksternom TMDB bazom podataka u kojoj se čuvaju svi podaci o TV emisijama i filmovima. TMDB pruža besplatan javni API dostupan sa API ključem.

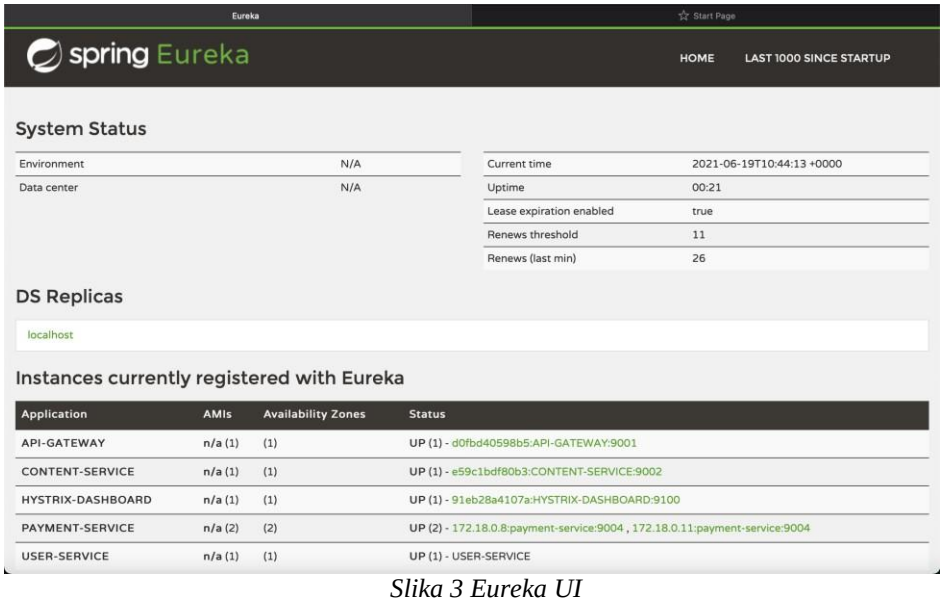
3.6. API ZA MREŽNI PROLAZ

API za mrežni prolaz služi kao ulazna tačka celog serverskog dela sistema, gde počinje komunikacija između klijentskog dela i serverskog dela servisa. On usmerava zahtev korisnika na osnovu rute, kao što je prikazano na dijagramu arhitekture. Imamo tri rute – „/users“, za slanje zahteva korisničkom servisu, „/payment“ za slanje servisu za plaćanje i „/content“, za slanje servisu za pristup sadržajima. Takođe je ograničeno iz kog izvora se mogu slati zahtevi, i u ovom slučaju to može učiniti samo klijentski deo aplikacije. Ovim se upravlja konfigurisanjem CORS-a (Cross Origin Resource Sharing). Postoji mnogo više funkcija koje Spring Cloud Gateway uključuje, kao što su integracija dizajn šablona prekid kola, prepisivanje putanje, ograničavanje brzine itd.

3.7. EUREKA

Eureka je registar servisa koja sadrži informacije o svim mikroservisima. Svaki mikroservis će se registrovati na Eureka server i on ima informacije o svim servisima koji rade na svakom portu i IP adresi. Eureka server je takođe poznat i kao Discovery Server. To znači da se svaki od pomenutih servisa registruje kod Eureke kada se pokrene. Mehanizam otkrivanja servisa pomaže API-ju za mrežni prolaz da otkrije servise koji su mu potrebni za usmeravanje zahteva na odgovarajući način.

Eureka UI je prikazan na slici 3, gde se mogu videti sve registrovane aplikacije (servisi).

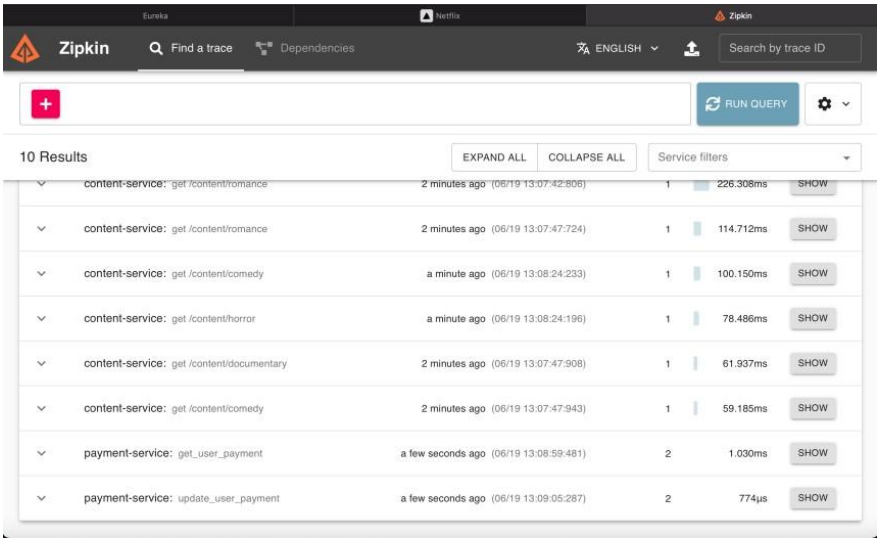


Slika 3 Eureka UI

3.8. DISTRIBUIRANO PRAĆENJE

Zipkin pomaže u prikupljanju podataka o vremenu potrebnom za rešavanje problema sa kašnjenjem u arhitekturi servisa. Funkcije obuhvataju i prikupljanje i traženje ovih podataka. On identifikuje neuspele mikroservise ili servise koji imaju probleme sa performansama kada u okviru zahteva postoji

mnogo servisnih poziva. Distribuirano praćenje je veoma korisno kada treba da pratimo zahtev koji prolazi kroz više mikroservisa. Takođe se koristi za merenje performansi mikroservisa. Postoji integrisani korisnički interfejs gde se može direktno ući u datoteku evidencije ako imamo ID praćenja, ili napraviti različiti upiti na osnovu atributa kao što su servis, oznake i trajanje. Na slici 4 prikazan je primer praćenja nekih zahteva iz aplikacije. Može se videti koji servis je obradio zahtev, kada, koliko dugo ga je obrađivao, i ostali detalji.

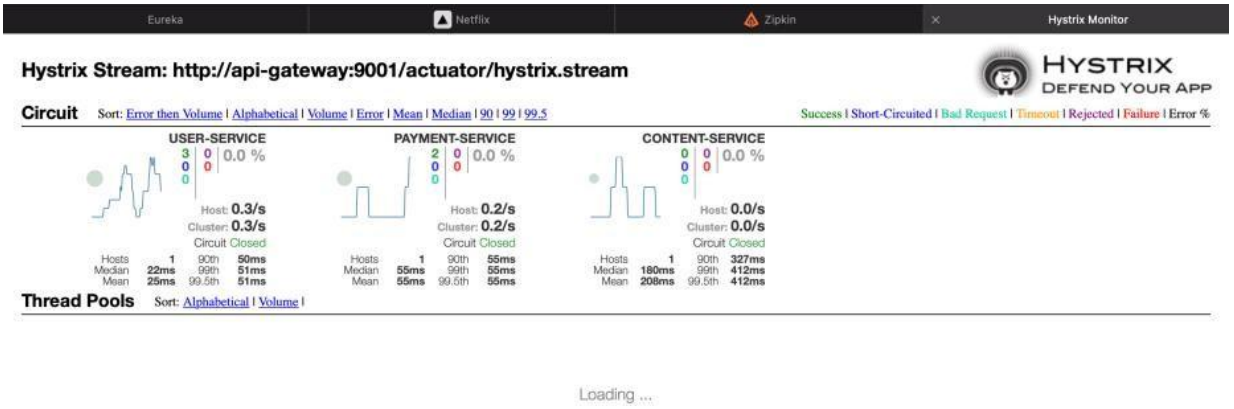


Slika 4 Distribuirano praćenje

3.9. KONTROLNA TABLA

Histrik kontrolna tabla, prikazana na slici 5, omogućava praćenje Histrik metrike u realnom vremenu.

Njegova glavna svrha je pomoć u smanjenju vremena potrebnog za otkrivanje i oporavak od operativnih događaja. Trajanje većine proizvodnih incidenata (retki zbog Histrik-a) postaje daleko kraće, sa smanjenim uticajem, zbog uvida u ponašanje sistema u realnom vremenu koji pruža Histrik kontrolna tabla.



Slika 5 Hystrix kontrolna tabla

4. ZAKLJUČAK

U eri borbe kompanija da zadovolje potražnju za automatizacijom servisa podataka kako bi se postigla efikasnost poslovanja i bolje odgovorilo na sve veću složenost i komunikacione zahteve sve većeg broja aplikacija i uređaja, mikroservisi mogu pružiti fleksibilnost, skalabilnost i ekonomičnost

koja je potrebna da bi njihova IT infrastruktura bila pouzdana i efikasna. Ova vrsta arhitekture je bez konkurencije kada je u pitanju razdvajanje poslovne logike na male komponente koje komuniciraju jedna sa drugom, i ima mnogo vidljivih prednosti kao što su bolja skalabilnost, modularnost i lakši razvoj. S druge strane, odabir odgovarajućih alata, protokola, okvira i jezika koji dobro funkcionišu jedni sa drugima, je komplikovan. Međutim, ako se to uradi na pravi način, ovaj stil softverske arhitekture se pokazuje kao veoma zahvalan pri razvoju aplikacija.

5. LITERATURA

[1] De Lauretis L.: *From Monolithic Architecture to Microservices Architecture*, IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 2019, str. 93-96.

[2] Knoche H.; Hasselbring W.: *Using Microservices for Legacy Software Modernization*.IEEE Software, vol. 35, no. 3, 2018, str. 44-49.

[3] Sill A.: *The Design and Architecture of Microservices*. IEEE Cloud Computing, vol. 3,no. 5, 2016, str. 76-80.

[4] Huang B., Liang Z., Qin M., Zhou H. and Z. Li G. Liu, *Microservices: architecture, container, and challenges*, in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 2020, str. 629-635.

[5] Fidge C., Zimmermann O., Kelly W., Barros A. and Furda A.: "*Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency*, IEEE Software, vol. 35, no. 3, 2018, str. 63-72.

[6] Jha, D.N., Garg, S., Jayaraman, P.P., Buyya, R., Li, Z. and Ranjan, R.: *A Holistic Evaluation of Docker Containers for Interfering Microservices, in* 2018 IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, 2018, str. 33-40.

[7] Jaramillo, D., Nguyen, D.V. and Smart, R.: *Leveraging microservices architecture by using Docker technology*, In SoutheastCon, IEEE, 2016, str. 1-5.

[8] El Kholy, M. and El Fatatry, A.: *Framework for interaction between databases and microservice architecture,* IT Professional, 2019, str. 57-63.

[9] Aleksić I.: *Implementation of microservices architecture using Spring, Jenkins and Openshift platform*, Proceeding of the Faculty of Technical Sciences, vol. 35, no. 11, 2020.

[10] Inđić, T.: *Search microservice for Serbian language dictionary project*, Proceeding of the Faculty of Technical Sciences, vol. 36, no. 7, 2021.

[11] Boduch, A.: *Next.js in Action*. New York, USA: Manning Publications, 2020.

[12] Avdić, A., *Realizacija servisa pametnog zdravstva i njihova integracija u koncept pametnih gradova*, Универзитет у Нишу, 2021.

[13] Netfilx, https://www.netflix.com/app, decembar 2022.

[14] Minkowski, P. *Mastering Spring Cloud: Build self-healing, microservices-based, distributed systems using Spring Cloud*, Birmingham, United Kingdom: Packt Publishing, 2018.