DRŽAVNI UNIVERZITET U NOVOM PAZARU



TEHNIČKO-TEHNOLOŠKE NAUKE

SOFTVERSKO INŽENJERSTVO

PERFORMANSE RAČUNARSKIH SISTEMA

Analiza performansi aplikacije u monolitnoj i mikroservisnoj verziji

Studenti: Mentor:

Hamić Ensar 036-037/20 doc. dr Irfan Fetahović

Crnovršanin Adnan 036-011/20

Ibrahimović Tarik 036-015/20

Sadržaj

[Uvod 3](#_Toc169560785)

[AWS 4](#_Toc169560786)

[Apache JMeter 4](#_Toc169560787)

[Opis aplikacije 5](#_Toc169560788)

[Postupak rada 5](#_Toc169560789)

[Rezultati 6](#_Toc169560790)

[Analiza 7](#_Toc169560791)

[Zaključak 8](#_Toc169560792)

## Uvod

U savremenom softverskom inženjeringu, arhitektura aplikacija igra ključnu ulogu u postizanju visokih performansi, skalabilnosti i fleksibilnosti sistema. Tradicionalno, aplikacije su bile razvijane u monolitnoj arhitekturi, gde su sve funkcionalnosti integrisane u jedinstvenu celinu. Međutim, sa sve većim zahtevima za brzinom razvoja, održavanjem i skalabilnošću, mikroservisna arhitektura postaje sve popularniji pristup. Mikroservisi omogućavaju razbijanje aplikacije na manje, nezavisne komponente koje se mogu razvijati, testirati i distribuirati autonomno.

Cilj ovog rada je da eksperimentiše sa performansama i troškovima infrastrukture aplikacije razvijene za Veterinarsku kliniku u oba arhitektonska pristupa - monolitnom i mikroservisnom. U tu svrhu, koristićemo AWS (Amazon Web Services) kao okruženja za implementaciju i testiranje aplikacije. Za merenje performansi koristićemo alat JMeter, prateći eksperimentalnu proceduru opisanu u radu „Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation“.

Eksperiment će se fokusirati na ključne metriku: broj obrađenih zahteva po sekundi. Performanse će se mereni kao throughput, odnosno broj zahteva koje server može da obradi u sekundi.

Testiranje će se obaviti u cloud (AWS) okruženju. Aplikacija koju ćemo testirati je rađena u Java Spring Boot tehnologiji.

Ovaj rad ima za cilj da pruži uvid u prednosti i mane oba pristupa, pružajući tako smernice za buduće projekte i implementacije. U nastavku rada biće detaljno opisani eksperimenti, rezultati poređenja i zaključci do kojih smo došli tokom istraživanja.

## AWS

AWS (Amazon Web Services) je vodeća platforma za cloud computing, pružajući širok spektar servisa i alata za skaliranje i upravljanje aplikacijama u cloud-u. Sa početkom 2006. godine, AWS je postao ključan igrač u industriji cloud usluga, omogućavajući organizacijama globalnu prisutnost i efikasno upravljanje IT resursima.

AWS infrastruktura je raspoređena globalno u regijama sa više dostupnih zona, obezbeđujući visoku dostupnost i otpornost na greške. Korisnici mogu birati lokaciju svojih servisa u skladu sa regulativama o podacima i potrebama za niskom latencijom.

Cene AWS usluga se određuju prema korišćenju, bez unaprednih troškova ili dugoročnih ugovora. Ova fleksibilna cena omogućava korisnicima da efikasno upravljaju troškovima IT infrastrukture.

AWS nudi širok spektar servisa pogodnih za razvoj i deploy mikroservisnih arhitektura. Koristeći servise kao što su Amazon ECS i Amazon EKS, developeri mogu efikasno upravljati kontejnerizovanim aplikacijama i orkestracijom kontejnera. AWS Lambda omogućava serverless pristup, smanjujući operativne troškove i kompleksnost upravljanja infrastrukturom.

U zaključku, AWS pruža pouzdanu i skalabilnu platformu za deploy aplikacija svih veličina. Njegova globalna infrastruktura i bogata ponuda servisa omogućavaju organizacijama da brzo inoviraju, smanje troškove i unaprede performanse svojih aplikacija u cloud-u.

## Apache JMeter

Apache JMeter je popularan alat za testiranje performansi i opterećenja aplikacija, koji omogućava precizno merenje kako aplikacije reaguju na različite opterećenja. Koristi se za simulaciju stvarnog korisničkog saobraćaja i analizu performansi aplikacija.

JMeter omogućava kreiranje različitih test planova koji uključuju scenarije testiranja, broj korisnika (threads), opterećenje (load), i merenje vremena odziva servera. Koristeći JMeter, razvijači mogu detaljno pratiti throughput aplikacije, odnosno broj obrađenih zahteva po sekundi, što je ključno za ocenu performansi aplikacije pod stvarnim uslovima upotrebe.

JMeter podržava ponovljeno izvršavanje testova i analizu rezultata, omogućavajući da se identifikuju potencijalne slabosti u aplikaciji ili infrastrukturi. Ovaj alat je neophodan za svakog developera ili tim koji želi da optimizuje performanse svojih aplikacija pre nego što ih puste u produkciju.

Ukratko, Apache JMeter pruža moćne mogućnosti za testiranje performansi aplikacija, pomažući developerima da osiguraju stabilnost, skalabilnost i efikasnost svojih softverskih rešenja.

## Opis aplikacije

U okviru ovog istraživanja, uzeta je aplikacija veterinarske klinike koja je implementirana kako u monolitnoj, tako i u mikroservisnoj arhitekturi. Aplikacija omogućava podršku za upravljanje veterinarima, ljubimcima i njihovim vlasnicima.

Aplikacija je razvijena u Java Spring Boot tehnologiji, koja je popularna za razvoj robustnih i skalabilnih web aplikacija. Java Spring Boot omogućava brz razvoj, olakšano upravljanje zavisnostima, i podršku za integraciju sa raznim tehnologijama i alatima.

Monolitna verzija aplikacije integriše sve funkcionalnosti u jedinstvenu celinu, dok mikroservisna verzija razbija aplikaciju na manje, nezavisne servise (API Gateway, Configuration Server, Discovery Server, Customer Service, Veterinarian Service, Visits Service), što omogućava fleksibilnije upravljanje i skaliranje pojedinačnih delova aplikacije.

Ovo istraživanje će analizirati performanse i troškove infrastrukture obe verzije aplikacije, koristeći AWS cloud platformu za implementaciju i testiranje. Korišćenje JMeter alata za testiranje opterećenja omogućavaće merenje performansi i throughput-a obe arhitekture, pružajući dublji uvid u njihovu efikasnost i skalabilnost.

## Postupak rada

U ovom istraživanju izvršeno je poređenje između monolitne i mikroservisne arhitekture iste aplikacije, koja je implementirana i testirana na AWS platformi, koristeći Amazon EC2 instancu (t3.micro – 2vCPUs, 1GiB RAM). Cilj istraživanja bio je analizirati performanse ovih arhitektura putem testiranja opterećenja primenom Apache JMeter alata.

Za testiranje, lokalno je korišćen JMeter alat konfigurisan da izvršava 3 GET i 1 POST zahtev. Testovi su prethodno dizajnirani u JMeter GUI-u, ali su izvršavani isključivo preko JMeter CLI-a, u skladu sa preporukama JMeter tima radi pouzdanijih rezultata.

Svaki test run je uključivao slanje zahteva od strane 10 threads (korisnika), svaki put ponavljajući test 100 puta u loopu. Ovaj ciklus je ponovljen 20 puta kako bi se osigurala konzistentnost rezultata. Za svaki test run je izračunata medijana throughput-a, kako bi se dobila reprezentativna vrednost performansi za tu iteraciju.

Ovaj proces je ponovljen 5 puta za obe arhitekture (monolitna i mikroservisna), sa ciljem dobijanja stabilnih i uporedljivih rezultata. Nakon pet iteracija, izračunata je medijana throughput-a za svaku arhitekturu. Ovaj pristup omogućio je eliminisanje eventualnih varijacija u performansama usled fluktuacija u cloud okruženju.

Kombinacija korišćenja JMeter alata za testiranje opterećenja i AWS platforme za hostovanje aplikacija omogućila je detaljnu analizu performansi monolitne i mikroservisne arhitekture. Konačni rezultati će pružiti uvide u prednosti i nedostatke svakog pristupa, korisne za donošenje informisanih odluka u budućim razvojnim projektima.

## Rezultati

**Monolitna arhitektura**

Tabela prikazuje propusnost za svaki zahtev na aplikaciju, kao i ukupnu propusnost

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Testovi | Propusnost za GET jednog vlasnika | Propusnost za GET tabele veterinara | Propusnost za GET pocetne stranice | Propusnost za POST dodavanje vlasnika | Ukupna propusnost |
| Od 1 do 20 | 49.8 | 43.2 | 53.8 | 34.1 | 180.9 |
| Od 21 do 40 | 49.7 | 43.7 | 53.7 | 34.2 | 181.3 |
| Od 41 do 60 | 49.7 | 43.5 | 53.6 | 33.9 | 180.7 |
| Od 61 do 80 | 50.2 | 43.2 | 53.6 | 34.1 | 181.1 |
| Od 81 do 100 | 49.9 | 43.3 | 53.8 | 34.1 | 181.1 |
| Prosek | 49.86 | 43.38 | 53.7 | 34.08 | 181.02 |

**Mikroservisna arhitektura**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Testovi | Propusnost za GET jednog vlasnika | Propusnost za GET tabele veterinara | Propusnost za GET pocetne stranice | Propusnost za POST dodavanje vlasnika | Ukupna propusnost |
| Od 1 do 20 | 45.8 | 40.3 | 49.6 | 31.3 | 167 |
| Od 21 do 40 | 44.1 | 40.1 | 51.7 | 29.3 | 165.2 |
| Od 41 do 60 | 46.3 | 39.9 | 50.5 | 30.1 | 166.8 |
| Od 61 do 80 | 44.3 | 42.1 | 51.1 | 32.3 | 169.8 |
| Od 81 do 100 | 45.9 | 42.3 | 51.9 | 30.1 | 170.2 |
| Prosek | 45.28 | 40.94 | 50.96 | 30.62 | 167.8 |

## Analiza

U ovom projektu upoređena je propusnost dve različite arhitekture – monolitne i mikroservisne – hostovane na Amazon Web Services (AWS) platformi. Propusnost sistema je kritičan pokazatelj performansi koji se meri brojem zahteva koje sistem može da obradi u jedinici vremena. U ovom istraživanju, testirana je propusnost za četiri tipa operacija:

1. GET jednog vlasnika
2. GET tabele veterinara
3. GET početne stranice
4. POST dodavanje vlasnika

Propusnost za GET jednog vlasnika

Monolitna arhitektura pokazuje konzistentno veću propusnost za GET jednog vlasnika, sa prosekom od 49.86 zahteva u sekundi u poređenju sa 45.28 zahteva u sekundi kod mikroservisne arhitekture. To ukazuje da monolitna arhitektura efikasnije obrađuje jednostavne GET zahteve, verovatno zbog manjeg overhead-a u komunikaciji između komponenti.

Propusnost za GET tabele veterinara

Sličan trend se primećuje i kod GET zahteva za tabelu veterinara. Monolitna arhitektura ostvaruje prosečnu propusnost od 43.38 zahteva u sekundi, dok mikroservisna arhitektura postiže 40.94 zahteva u sekundi. Ovo može biti rezultat optimizacije baze podataka i boljeg keširanja unutar monolitnog sistema.

Propusnost za GET početne stranice

Kod GET zahteva za početnu stranicu, monolitna arhitektura takođe nadmašuje mikroservisnu sa prosečnom propusnošću od 53.7 u poređenju sa 50.96 zahteva u sekundi. Razlika je nešto manja, ali ipak značajna, što sugeriše da se statički sadržaji bolje poslužuju iz monolita.

Propusnost za POST dodavanje vlasnika

Najveća razlika primećuje se kod POST zahteva za dodavanje vlasnika. Monolitna arhitektura ostvaruje prosečnu propusnost od 34.08 zahteva u sekundi, dok mikroservisna arhitektura ima samo 30.62 zahteva u sekundi. Ova razlika može biti uzrokovana dodatnim overhead-om u mikroservisnoj arhitekturi zbog orkestracije i komunikacije između servisa.

Ukupna propusnost

Kao rezultat svih ovih pojedinačnih merenja, ukupna propusnost monolitne arhitekture je znatno veća, sa prosekom od 181.02 zahteva u sekundi, u poređenju sa 167.8 zahteva u sekundi kod mikroservisne arhitekture. Ovo ukazuje da, u pogledu čistih performansi na AWS-u, monolitna arhitektura pruža bolju propusnost.

## Zaključak

Na osnovu analize performansi monolitne i mikroservisne arhitekture hostovanih na AWS-u, utvrđeno je da monolitna arhitektura pruža veću propusnost za sve ispitivane operacije. Prosečna ukupna propusnost monolitne arhitekture iznosi 181.02 zahteva u sekundi, dok mikroservisna arhitektura postiže 167.8 zahteva u sekundi.

Monolitna arhitektura pokazuje prednosti u smislu nižeg overhead-a i optimizovanih performansi unutar jedinstvenog procesa. Međutim, mikroservisna arhitektura nudi značajne benefite u smislu skalabilnosti i fleksibilnosti. Ključna prednost mikroservisa je mogućnost horizontalnog skaliranja, što može značajno poboljšati rezultate performansi. Sa horizontalnim skaliranjem, pojedinačni servisi mogu biti skalirani nezavisno, što omogućava bolje iskorišćavanje resursa i veću ukupnu propusnost sistema.

Iako monolitna arhitektura trenutno pokazuje bolje performanse, izbor arhitekture treba da se zasniva na specifičnim potrebama projekta. Za projekte koji zahtevaju visoku performansu i gde su komponente usko povezane, monolitna arhitektura može biti pogodnija. S druge strane, za projekte koji zahtevaju agilnost, skalabilnost i brzo reagovanje na promene, mikroservisna arhitektura, uz adekvatno horizontalno skaliranje, može pružiti superiorne performanse i fleksibilnost.

U konačnici, oba pristupa imaju svoje prednosti i mane, i izbor treba biti pažljivo donet na osnovu poslovnih ciljeva i tehničkih zahteva projekta.