|  |  |
| --- | --- |
|  | Zbornik radova Fakulteta tehničkih nauka, Novi Sad |

**UDK: 004.9**

**DOI:** [**https://doi.org/10.24867/10BE30Aleksic**](https://doi.org/10.24867/10BE30Aleksic)

**IMPLEMENTACIJA MIKROSERVISNE ARHITEKTURE POMOĆU SPRING, JENKINS I OPENSHIFT PLATFORME**

**IMPLEMENTATION OF MICROSERVICE ARCHITECTURE USING SPRING, JENKINS AND OPENSHIFT PLATFORM**

Isidora Aleksić, *Fakultet tehničkih nauka, Novi Sad*

**Oblast – ELEKTROTEHNIKA I RAČUNARSTVO**

**Kratak sadržaj –** *Tema rada jeste analiziranje osnovnih koncepata mikroservisne arhitekture web aplikacija. Poseb­no je proučen Jenkins alat za kontinuiranu integraciju, kao i OpenShift platforma za implementaciju i eksploataciju softvera. Komponente prikazane arhitekture su višeslojno organizovane a navedeni su i principi organizacije kompo­nenti.*

**Ključne reči:** *Mikroservisi, API, deploy*

**Abstract** – *This paper analyzes the basic concepts of microservice web application architecture. The Jenkins tool for continuous integration, as well as the OpenShift plat­form for software deployment, are presented in more detail. Components of this architecture are organized into multiple layers, while component organization principles are also defined.*

**Keywords:** *Microservices, deployment, software architecture*

**1. UVOD**

Mikroservisi se sastoje od malih, samostalnih servisa koji rade zajedno [1]. Cilj ovih pojedinačnih servisa je da razlože kompleksnu aplikaciju na manje jednostavnije jedinice koje će biti grupisane na osnovu njihovih zajed­ničkih funkcionalnosti (*Single Responsibility Principle*). Komunikacija između ovih servisa se obezbeđuje uz pomoć API-a (*Application Programming Interface*).

Aplikacije koje su izgrađene korišćenjem mikroservisne arhitekture imaju određene karakteristike. Aplikacija je podeljena na više komponenti, od kojih svaka vrši određenu funkciju. Svaka komponenta je napravljena tako da predstavlja nezavisnu uslugu koja se može menjati, ažurirati i brisati bez ometanja ostatka aplikacije. U projektima, svaki mikroservis ima jasnog vlasnika. Svaki tim ima jasan, nepreklapajući niz odgovornosti. U praksi neke usluge mogu doći sa drugih strana putem API-ja i možda neće biti u potpunosti vidljive vlasniku aplikacije. Osnovno pravilo prilikom izmene servisa jeste deploy tih izmena neće uticati ni na jedan drugi servis [1]. Potrebno je na odgovarajući način sistem razdvojiti u samostalne servise kako bismo obezbedili aplikaciju koja će biti jednostavna za dalji razvoj. Mnoge organizacije su ustanovile da korišćenjem mikroservisne arhitekture omogućava brže isporučivanje softvera.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NAPOMENA:**

**Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.**

Mnoge organizacije su ustanovile da korišćenjem mikro­servisne arhitekture omogućava brže isporučivanje soft­vera i brže prihvatanje novijih tehnologija.

Mikroservisi nam daju znatno više slobode reagovanja i donošenja različitih odluka.

Ovo nam omogućava brži odgovor na neizbežne promene. Kroz primere i implementaciju analizirane su funkcio­nalnosti takve arhitekture.

**2.** **KORIŠĆENE TEHNOLOGIJE I TERMINOLOGIJE**

**2.1 Spring**

Spring [2] je Java platforma koja pruža širok opseg mogućnosti za podršku prilikom razvoja veb aplikacija. *Spring* rukuje infrastrukturom, tako da programer može usmeriti svoj fokus na razvoj aplikacije. *Spring* nudi kontejner koje se odnose na *Spring appliation context* uz pomoć kojeg se kreiraju i rukuju aplikacione komponente koje čine sistem [3].

*Spring* radi sa *JavaBean* klasama gde su sva polja privatna (*private*), serijalizabilna, imaju podrazumevani konstruk­tor kao i get/set metode pomoću kojih se pristupa poda­cima. *Spring* je osmišljen da bude dopuna Java standarda za poslovne aplikacije.

*Spring* predstavlja bolju alternativu za kompleksni J2EE standarda, naročito deo koji se tiče rada sa JavaBeans.

**2.2 Swagger**

Swagger [4] je jednostavan alat za upotrebu API progra­mera (timove i pojedince). Omogućava razvoj tokom celog životnog ciklusa API-ja, od dizajna i dokumentacije, do testiranja i primene. Swagger se sastoji od mešavine otvorenog koda, besplatnih i komercijalno dostupnih alata koji omogućavaju da se naprave API-jevi.

Swagger omogućava da opišemo strukturu API-ja tako da ih mašine mogu čitati. Sposobnost API-ja da opišu sopstve­nu strukturu je osnova u Swaggeru. Čitajući strukturu API-ja može se automatski napraviti interaktivna API dokumen­tacija.

Takođe može automatski generisati klijentske biblio­teke za API na mnogim jezicima i tražiti druge mogućnosti poput automatizovanog testiranja. Swagger to radi tako što traži od API-ja da vrati YAML ili JSON koji sadrži detaljan opis celokupnog API-ja.

Specifikacija traži da se uključi informacije poput operacije koje API podržava, koji su parametri API-ja i šta vraća i da li je potrebna neka autorizacija.

Swagger spec. možemo ručno da napišemo ili ga možemo automatski generisati iz anotacija u našem izvornom kodu.

**2.3 Jenkins**

Jenkins [5] je server za automatizaciju otvorenog koda koji se može koristiti za automatizaciju svih vrsta zadataka koji se odnose na izgradnju, testiranje i isporuku ili upotrebu softvera.

Može se instalirati preko sistemskih paketa, Docker-a ili se može čak samostalno pokretati na bilo kojoj mašini sa instaliranim Java Runtime Environment (JRE).

Jenkins postiže kontinuiranu integraciju uz pomoć dodata­ka. Dodaci omogućavaju integraciju različitih DevOps faza. Ukoliko želimo da integrišemo određeni alat, mora­mo prvo da instaliramo dodatak za taj alat.

Za na primer: Git, Maven 2 projekat, Amazon EC2, HTML izdavač itd., slika 1.

Najosnovniji kontinuirani *delivery pipeline* imaće, u najmanju ruku, tri faze koje bi trebale definisati u Jenkinsfile-u: Build, Test i Deploy.



Slika 1. *Jenkins integriše različite DevOps faze*

**2.4 Openshift container platform**

Openshift container platform [6] treba da omogući programerima da se usredsrede na svoj kod koristeći postojeće alate i poznate tokove rada, uključujući CI/CD.

Konzistentna kontejnerska platforma veoma je važna za Ops timove koji žele da sa jednog mesta, sa potpunom vidljivošću od nivoa aplikacije do kontejnera, OS-a, vir­tualizacije i hardverskog sloja, upravljaju tradicionalnim mikroservisima, slika 2.

Ops timovi (operativni timovi) takođe brinu o bezbednosti, modularnosti, kontroli pristupa i polisama.



Slika 2. *Prikaz Red Hat Container Stack i alata*

**3. MIKROSERVISNA ARHITEKTURA**

Mikroservisi su tehnika razvoja softvera - varijanta servisno-orijentisanog arhitekturalnog (SOA) stila koji struktuira aplikaciju kao kolekciju labavo povezanih servisa.

Cilj razvoja servisa u mikroservisnoj arhitekturi je da servisi budu što manji, optimalni i fokusirani isključivo na svoj domen. Servisi su alternativno rešenje monolitnim sistemima koji u sebi sadrže jednu aplikaciju zaduženu za kompletnu interakciju sa korisnikom i bazom podataka. Sa vremenom, kako se radi na aplikaciji, povećava se količina programskog koda i time promene postaju sve teže i raste kompleksnost. Iako se pri razvoju koristi primeri za odvajanje delova aplikacije te za strukturiranje koda unutar programske komponente, te komponente postaju sve komplikovanije. Desi se i da funkcije i metode povezuju više domena odjednom, što otežava da se implementira nova funkcionalnost ili ukloni greška. Ukoliko se pri razvoju monolitne aplikacije ima na umu da programski kod čini koheziju, odnosno da su zajed­nički koncepti čvrsto povezani, tada je prelazak iz posto­jeće monolitne aplikacije u mikroservisnu arhitekturu nešto jednostavniji.

Ukoliko servis deli previše svoje funkcionalnosti, koris­nici servisa mogu da postanu zavisni o internoj logici toga servisa. Servisi koriste aplikacijski programski interfejs (*Application programming interface-API*) za komuni­ciranje i sarađivanje sa drugim servisima putem mreže. Svaki servis pruža svoj interfejs drugim servisima pa odabir tehnologije interfejsa nije trivijalan zadatak.

Pošto postoji nezavisnost između servisa, potpuno je svejedno koje tehnologije koristi servis, sve dok ti servisi imaju „zajednički jezik“, API. To omogućava odabir odgovarajuće tehnologije za svaki zadatak, skraćivanje vremena razvoja i poboljšane performanse aplikacije.

Servisi mogu biti implementirani pomoću različitih pro­gramskih jezika, baza podataka i softverskih okruženja. To omogućava svakom servisu da bude ponovo izgrađen, ponovo isporučen i upravljan nezavisno i bez promene njihovih potrošača (*consumer*).

Kod tradicionalnih monolitnih aplikacija stvari se kom­plikuju ako tehnologija koja se koristi u celoj aplikaciji nije najbolji alat za neki zadatak ili ta funkcionalnost uopšte nije podržana. U tom slučaju potrebno je napraviti svoju implementaciju potrebnog algoritma u korišćenoj tehnologiji ili spajati dve tehnologije, što otežava daljnji razvoj [1].

**3.1 Višeslojna arhitektura**

Kod razvoja web aplikacija često se sama aplikacija struk­turira u više delova ili slojeva. Primer takve arhitekture je višeslojna ili N-slojna arhitektura, gde N označava broj slojeva. Kada se priča o višeslojnoj arhitekturi, najčešće se misli na troslojnu arhitekturu koja aplikacijske kompo­nente razdvaja na prezentacijski, aplikacijski i sloj podataka, slika 3.

Prezentacijski sloj je u direktnom kontaktu sa korisnikom, prikazuje korisniku interfejs i informacije i održava interakciju sa korisnikom.

Za komunikaciju sa korisnikom, prezentacijski sloj koristi preglednik korisnika. Sloj koji se nalazi u sredini naziva se aplikacijski sloj koji je zadužen za upravljanje aktivnostima koje određena web aplikacija treba da obavlja na osnovu interakcije korisnika. Taj sloj je takođe zadužen za komunikaciju prezentacijskog sloja i sloj podataka i njime upravlja tokom aplikacije. Treći sloj troslojne arhitekture je sloj podataka. Sloj podataka sadrži podatke i logiku pristupa i ĉuvanje podataka. [7].



Slika 3. *Prikaz višeslojne arhitekture*

**3.2 Servisno modeliranje**

Posmatrajući usluge u mikroservisnoj arhitekturi, može se reći da su njihove zajedničke karakteristike slaba povezanost (loose coupling) i velika kohezija

Slabo povezivanje (*loose coupling*) u mikroservisnoj arhitekturi koristi se kako bismo imali mogućnost menjanja jednog servisa bez uticaja ili izmena drugog servisa te je to jedna od najvažnijih prednosti servisa. Slabo povezani servisi ne znaju puno jedni o drugima, čak i kada neposredno sarađuju.

Slabo povezani servisi nemaju puno informacija jedni o drugima, čak i kada neposredno sarađuju. Ako se pri raz­voju ne uzme u obzir da se sva komunikacija odvija kroz mrežu, tada može doći do problema sa performansama, pa je iz tog razloga važno pripaziti na obim razmene poruka između servisa. Povećana složenost komunikacije između dva servisa dovodi i do čvršće povezanosti između njih, što je isto suprotno od onoga što se želi postići.

Cilj visoke kohezije (High cohesion) je grupisanje logike programa u povezane celine. Svaka celina sadrži logiku i ponašanje koje je usko vezano za njen kontekst, dok je sve ostalo što nije direktno vezano za taj kontekst, u drugoj celini.

Pronalaženjem granica celina obezbeđuje da je ponašanje povezano sa jednim kontekstom u jednoj celini, a drugačije ponašanje u drugoj celini, drugim reĉima, granice celina kreiraju se prema njihovim kontekstima. Te se celine pretvaraju u servise pa se kod zahteva za promenom ponašanja pojedinog servisa ono promeni samo na jednom mestu, usled čega je servis znatno brže spreman za produkcijsko okruženje (production environment).

Ukoliko se funkcionalnost proteže kroz više servisa, nekoliko servisa će se morati izmeniti da bi se promenila funkcionalnost, dok će se oni možda u isto vreme proizvoditi. Menjanjem većeg broja servisa odjednom, drastično se povećava rizik od grešaka te je sam postupak izmene puno teži, s obzirom da se izvršava na nekoliko servisa. [7].

**4. ZAKLJUČAK**

Mikroservisna arhitektura popularan je izbor za implementaciju web aplikacija. Detaljno su analizirani i opisani osnovni koncepti mikroservisne arhitekture i tehnologije, koje su korišćenje u toku izrade aplikacije. Mikroservisna arhitektura omogućava fleksibilnost koja je potrebna za prilagođavanje rastućem broju korisnika Internet i Web aplikacija kao i broju tehnologija i alata dostupnih za razvoj istih. Cilj je pojednostavljenje svakog servisa i implementacionih rešenja koja neće zavisiti od tehnologije koja se trenutno koristi.

Neophodno je upoznati se sa domenom monolitne aplikacije kako bi se mogao pojednostaviti veliki broj tesno povezanih funkcionalnosti u samostalne servise. Tek tada se možemo odlučiti za mikroservisnu arhitekturu kao moguće rešenje. Postoji dosta pogodnih razloga za implementaciju web aplikacije uz pomoć mikroservisa. Meutim, razvojni proces takvih aplikacija je komplikovaniji i traži više angažovanja, koordinacije, znanja i rada na projektu. Javlja se i problematika rada sa velikim brojem alata i tehnologija u isto vreme.

Ukoliko je jedan od osnovnih zahteva web aplikacije skalabilnost, tj. podrška za veliki broj korisnika ili podrška za veliki broj tehnologija koje aplikacija koristi, tada je mikroservis pogodno iskoristiti kao softversko rešenje.

Održavanje i razvoj web aplikacije zasnovane na mikroservisima ne mora biti teže od razvoja aplikacije u jednostavnijem skupu tehnologija.

**5. LITERATURA**

1. Sam Newman, Building Microservices, O’Reilly Media, 2015.
2. Spring Framework Documentation. Dostupno na: <http://docs.spring.io/springframework/docs/current/spring-framework-reference/html/index.html>
3. Craig Walls, Spring in Action, Manning Publication Co., 2018.
4. Swagger Documentation. Dostupno na: <https://swagger.io/docs/specification/about/>
5. Jenkins Documentation. Dostupno na: <https://www.jenkins.io/doc/>
6. Openshift Documentation. Dostupno na: <https://docs.openshift.com/container-platform/3.3/architecture>
7. Chapter 4 Layered Architecture for Web Applications, 2018

**Kratka biografija:**

|  |  |
| --- | --- |
|  | **Isidora Aleksić** rođena je u Novom Sadu 1995. god. Master rad na Fakul­tetu teh­ničkih nauka iz oblasti Računarstva i automatika – Elektronsko poslovanje odbranila je 2020.god.kontakt: isidoraxaleksic@gmail.com |