SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

IZRADA SKALABILNE CHAT APLIKACIJE KOJA SE IZVRŠAVA NA KUBERNETES SUSTAVU

Ivo Kovačević

Split, rujan 2020.

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Diplomski studij: **FESB**

Smjer/Usmjerenje: **Računarstvo**

Oznaka programa: 250

Akademska godina: 2019./2020.

Ime i prezime: **IVO KOVAČEVIĆ**

Broj indeksa: 744-2018

**ZADATAK DIPLOMSKOG RADA**

Naslov: **IZRADA SKALABILNE CHAT APLIKACIJE KOJA SE IZVRŠAVA NA KUBERNETES SUSTAVU**

Zadatak: Izučiti principe dizajna i izrade skalabilnih web aplikacija. Izučiti način korištenja Kubernetes sustava kao i prednosti koje donosi u rada sa skalabilnim aplikacijama. Izradom chat aplikacije demonstrirati način i prednosti korištenja Kubernetes sustava.

Prijava rada: 05.10.2020. (početak semestra u kojem se prijavljuje rad)

Rok za predaju rada: 05.02.2020. (deset dana prije završetka semestra u kojem je rad prijavljen)

Rad predan:

Predsjednik   
Odbora za diplomski rad: Mentor:

prof. dr. sc. Ime Prezime izv. prof. dr. sc. Marin Bugarić

**IZJAVA**

Ovom izjavom potvrđujem da sam završni rad s naslovom (Izrada skalabilne chat aplikacije koja se izvršava na Kubernetes sustavu) pod mentorstvom (prof. dr. sc. Marin Bugarić) pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo/la u završnom radu citirao/la sam i povezao/la s korištenim bibliografskim jedinicama.

Student

Ivo Kovačević

**SADRŽAJ**

[1 UVOD 1](#_Toc41470011)

[2 MOTIVACIJA ZA UVOĐENJE KUBERNETES SUSTAVA 2](#_Toc41470012)

[2.1 Prelazak na mikroservis arhitekturu 2](#_Toc41470013)

[2.2 Uvod u container tehnologiju 4](#_Toc41470014)

[2.3 Potreba za Kubernetes sustavom 6](#_Toc41470015)

[3 KUBERNETES ARHITEKTURA 7](#_Toc41470016)

[4 ZAKLJUČAK 8](#_Toc41470017)

[LITERATURA 9](#_Toc41470018)

[POPIS OZNAKA I KRATICA 10](#_Toc41470019)

[SAŽETAK 11](#_Toc41470020)

# UVOD

Novi trend razvijanja aplikacija su svakako mikroservisi. Mikroservis arhitektura se uveliko razlikuje od tradicionalnog razvoja aplikacija gdje je cijela aplikacija „zapakirana“ kao jedna velika cjelina. Takve aplikacije nazivaju se monolitne aplikacije.

Monolitan razvoj je i dalje vrlo raširen iako ima mnogo nedostataka. Samim tim što je aplikacija jedna velika cjelina gdje su svi „servisi“ strogo povezani, jako je teško nadograđivati i mijenjati aplikaciju pa su nove verzije aplikacija vrlo rijetke. Nakon što se takve aplikacije naprave, programeri aplikaciju proslijede Ops timu. Ops tim potom aplikaciju ručno migrira na „zdravi“ (engl. healthy) server. Ukoliko dođe do prekida rada servera, aplikaciju je potrebno opet ručno migrirati na drugi „zdravi“ server.

Mikroservis arhitektura je skup više nezavisnih, samostalnih, odvojenih servisa. Svaki servis se razvija zasebno, najčešće od strane manjeg tima unutar organizacije te se potom migrira (engl. deploy) i održava neovisno o ostalim servisima. Ovakva arhitektura donosi mnoge prednosti u odnosu na tradicionalni razvoj. Glavna prednost je svakako mogućnost učestalijeg mijenjanja pojedinih komponenti ovisno o poslovnim zahtjevima.

Kod velikih aplikacija mikroservis arhitektura može rezultirati velikim brojem zasebnih servisa. Kod kompanija kao što su Amazon i Netflix taj broj prelazi nekoliko tisuća. Upravljati s nekoliko tisuća zasebnih mikroservisa bi svakako bilo nemoguće da ne postoje alati koji nam olakšavaju i omogućavaju taj posao. Jedan od tih alata je i Kubernetes sustav koji će biti objašnjen u okviru ovog diplomskog rada.

# MOTIVACIJA ZA UVOĐENJE KUBERNETES SUSTAVA

## Prelazak na mikroservis arhitekturu

Monolitne aplikacije sastoje se od više strogo povezanih aplikacija ili servisa koji se razvijaju, migriraju i održavaju kao cjelina. Takva aplikacija je u biti jedan proces koji se vrti unutar servera. Za svaku i najmanju promjenu neke od komponenti, cijela aplikacija se treba iznova migrirat na server (engl. redeploy).   
Ovakva arhitektura predstavlja ograničenje u skaliranju aplikacije pri promjeni opterećenja. Postoje dva tipa skaliranja: vertikalno i horizontalno skaliranje. Vertikalno skaliranje podrazumijeva nadogradnju servera dodavanjem procesorske snage, memorije, povećanje frekvencije procesora itd. Horizontalno skaliranje odnosi se na dodavanje više servera koji su najčešće slabijih performansi nego kod vertikalnog skaliranja, ali je ključ u njihovom broju. Ti serveri imaju istu instancu aplikacije te se opterećenje raspoređuje između servera. Ukoliko je neki server u velikom opterećenju novi zahtjevi će se prebaciti na drugi, manje opterećen server.   
Monolitne aplikacije vrlo se lako vertikalno skaliraju. Zamislimo scenarij da posjedujemo Internet trgovinu koja prodaje kreme za sunčanje. Lako je zaključiti da će u periodu od jeseni do proljeća posjećenost naše Internet trgovine biti vrlo mala. Za takvo opterećenje će vrlo vjerojatno biti dovoljni prosječni serveri. Pri dolasku sunčanijih ljetnih mjeseci, za očekivati je da će posjećenost naše stranice naglo porasti. Moguć je scenariji da naš prosječni server neće biti u mogućnosti podnijeti nagli porast opterećenja. Jedno od rješenja bi bilo vertikalno skalirati našu aplikaciju dodavanjem procesorske snage i memorije. Ukoliko se radi o manjoj Internet trgovini ovo će vrlo vjerojatno riješiti problem. U ovoj priči radi se o velikoj Internet trgovini koja drži monopol prodaje krema za sunčanje u cijelom svijetu. Može se pomisliti: „Pa zašto nebi dodali još, još i još procesorske snage i memorije?“. Naime, vertikalno skaliranje ima svoja ograničenja. Glavno ograničenje svakako bi bila cijena. Snažni i moćni hardver je vrlo skup. Drugo ograničenje je proizvodnja i dostupnost hardvera na tržištu. Iako danas na tržište vrlo brzo izlaze nove i bolje verzije hardvera opet ćemo doći do točke gdje više nećemo moći dobiti bolji server nego što ga imamo. Kao ograničenje svakako treba izdvojiti i rizik prestanka rada servera. Ukoliko imamo samo jedan server, prestankom rada tog servera usluga koju pružamo će biti nedostupna. Može se naslutiti da smo dosegli vrhunac vertikalnog skaliranja i da moramo pronaći neko drugo rješenje. To bi bilo horizontalno skaliranje. Međutim, monolitne aplikacije je najčešće vrlo teško horizontalno skalirati jer su komponente strogo povezane. Ukoliko samo jedan dio monolitne aplikacije nije skalabilan, cijela aplikacije će biti neskalabilna. Uvjet za horizontalno skaliranje je neovisnost komponenti. Da bi ispunili taj preduvjet monolitnu aplikaciju moramo „razbiti“ na više neovisnih servisa gdje svaki servis predstavlja jedan nezavisni proces (Slika 2.1). Procesi mogu međusobno komunicirati. Komunikacija među procesima može se ostvariti velikim brojem protokola koji nisu strogo vezani za određeni programski jezik nego su implementirani gotovo u svakom programskom jeziku. To su HTTP, TCP, AMQP itd.

A screenshot of a cell phone

Description automatically generated

Slika 2.1 Prelazak s tradicionalnog razvoja na mikroservis arhitekturu

Budući da je svaki mikroservis neovisni proces, moguće ih je migrirati, razvijati i održavati neovisno o drugim mikroservisima. Promjena na jednom mikroservisu ne uzrokuje „redeploy“ ostalih mikroservisa. Posljedično tome, kao rezultat dobijemo mogućnost skaliranja svakog mikroservisa zasebno, neovisno o drugim mikroservisima (Slika 2.2). Na taj način možemo skalirati samo one mikroservise koji zahtjevaju više resursa, a ne cijelu aplikaciju.

A close up of a logo

Description automatically generated

Slika 2.2 Svaki mikroservis može biti skaliran zasebno

## Uvod u container tehnologiju

Kod migracije aplikacija na server, može se dogoditi da različite aplikacije koriste različite verzije biblioteka. Kako je već rečeno, kod mikroservis arhitekture, najčešće se razvoj aplikacije radi unutar manjih timova od kojih svaki tim razvija jedan zasebni servis. To također može rezultirati korištenjem različitih verzija biblioteka između timova što će rezultirati brojnim problemima kod migracije na isti server.   
Kako bi se riješio spomenuti problem, pribjegava se korištenju virtualizacijskih tehnologija kao što su virtualne mašine i „container“ tehnologije. Virtualizacijom se postiže neovisne, izolirane okoline pogodne za izvršavanje aplikacija. Kod aplikacija koje se sastoje od manjeg broja komponenti, moguće je svaku komponentu izolirati unutar vlastite virtualne mašine (VM). Svaka VM ima svoj operacijski sustav (OS). Kada se broj ovakvih komponenti počne povećavati, dodjeljivanjem VM svakoj komponenti predstavlja veliki gubitak hardverskih resursa iz razloga što VM imaju veliki „overhead“ budući da se unutar svake VM izvršava vlastiti OS. Osim hardverskih resursa, upravljanje (engl. manage) VM odvija se zasebno što predstavlja i veliki trošak ljudskih resursa.  
Umjesto korištenje virtualnih mašina za osiguravanje izolirane cjeline za svaki mikroservis koristi se Linux „container“ tehnologija. Container tehnologija ima znatno manji „overhead“ od virtualnih mašina jer se izvršavaju na operacijskom sustavu domaćina (engl. host) umjesto da svaki container ima vlastiti OS unutar kojeg se izvršava proces. Proces unutar containera je također izoliran kao i kod VM. Korištenje container tehnologije naspram VM rezultira većim brojem aplikacija koje je moguće smjestiti na jedan server.

Na slici (Slika 2.3) vidimo spomenute razlike dvaju tehnologija. Uočavamo da svaka VM ima svoj vlastiti OS koji je na slici označen kao „Guest OS“ dok se aplikacije kod container tehnologije izvršavaju na OS hosta označenog kao „Host OS“ [1]. To je ujedno i glavna prednost VM u odnosu na container tehnologiju jer se postiže potpuna izoliranost što rezultira i većom sigurnošću.

A screenshot of a cell phone

Description automatically generated

Slika 2.3 Usporedba VM i container tehnologije

Najpopularnija container tehnologija je svakako Docker. Docker omogućava „pakiranje“ aplikacija, zavisnih biblioteka, pa čak i cijelog OS datotečnog sustava unutar jednog, jednostavnog paketa koji se lako prenosi na druge mašine. Kada se takav paket prenese na drugu mašinu, on vidi identični datotečni sustav kao i kada je zapakiran. To omogućava rad aplikacije čak i na kompletno različitom operacijskom sustavu od onoga na kojem je aplikacija razvijana. Na primjer, ukoliko razvijamo našu aplikaciju na Ubuntu Linux distribuciji, a server na kojem ćemo migrirati aplikaciju posjeduje Debian distribuciju, Docker tehnologija omogućava da zapakirani Docker container i dalje vjeruje kako se izvršava u Ubuntu. Jedini uvjet je da sve mašine između kojih se razmjenjuju Docker containeri imaju instaliran Docker.

## Potreba za Kubernetes sustavom

Iako je monolitni razvoj puno brži, dugoročno gledano, svakako se isplati uložiti vrijeme u mikroservis arhitekturu. Velika većina startup-ova u početku razvoja krene sa monolitnim pristupom te vrlo brzo naiđe na ograničenja. Mnoge velike kompanije su shvatile niz prednosti mikroservis arhitekture te se odlučile prebaciti na istu. Arhitektura kompanija kao što su Amazon i Netflix se sastoji od nekoliko tisuća mikroservisa koji međusobno komuniciraju (Slika 2.1) [2]. Gotovo nemoguće je upravljati s nekoliko tisuća mikroservisa koji komuniciraju i zajedno obavljaju neki složeniji posao. Uzročno tome, došlo je do potrebe za sustavom koji će olakšati razvoj aplikacije temeljene na mikroservis arhitekturi.

A close up of a logo

Description automatically generated

Slika 2.4 Mikroservis arhitektura Amazon-a i Netflix-a

Google kompanija je prva uvidjela ovaj problem te se odlučila na razvoj vlastitog, besplatnog i svima dostupnog rješenja koji su nazvali Kubernetes. Kubernetes je programsko rješenje koje omogućava jednostavno raspoređivanje i upravljanje container aplikacijama [3]. Zbog toga što su containeri izolirane cjeline, ne mogu utjecati na rad drugih aplikacija na istom serveru. To je vrlo važno svojstvo iz razloga što Cloud kompanije nastoje što bolje iskoristiti resurse, stavljajući različite aplikacije na isti server. Upravo na tom svojstvu se temelji i Kubernetes sustav. Migracija aplikacije na server kod Kubernetes sustava je vrlo jednostavna. Postupak je uvijek isti i ne razlikuje se o broju čvorova u klasteru. Dakle, isto je da li klaster ima jedan čvor ili 100 čvorova, što je uveliko doprinijelo prihvaćanju Kubernetes softvera i masivnom korištenju istog.

# KUBERNETES ARHITEKTURA

# ZAKLJUČAK

# LITERATURA

[1] „Dive Into Docker“, s interneta, <https://diveintodocker.com/?utm_source=nj&utm_medium=youtube&utm_campaign=virtual-machines-vs-docker-containers>, 26. svibnja 2020.

[2] „10 companies that implemented the microservice architecture and paved the way for others“, s interneta, <https://divante.com/blog/10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-others/>, 27.5.2020.

[3] Marko Lukša, „Kubernetes in Action“, Manning, NY, 2018.

# POPIS OZNAKA I KRATICA

Ops – Information-technology operations  
HTTP - HyperText Transfer Protocol  
TCP – Transmission Control Protocol  
AMQP – Advanced Message Queueing Protocol  
VM – Virtualna Mašina  
OS – Operacijski Sustav

# SAŽETAK