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](#_Toc41736859)

[2 MOTIVACIJA ZA UVOĐENJE KUBERNETES SUSTAVA 2](#_Toc41736860)

[2.1 Prelazak na mikro-servis arhitekturu 2](#_Toc41736861)

[2.2 Uvod u container tehnologiju 4](#_Toc41736862)

[2.3 Potreba za Kubernetes sustavom 6](#_Toc41736863)

[3 UVOD U KUBERNETES SUSTAV 7](#_Toc41736864)

[3.1 Pojednostavljeni prikaz rada Kubernetes sustava 7](#_Toc41736865)

[3.2 Arhitektura Kubernetes sustava 7](#_Toc41736866)

[3.3 Izvršavanje aplikacije na Kubernetes klasteru 9](#_Toc41736867)

[3.4 Prednosti korištenja Kubernetes sustava 10](#_Toc41736868)

[4 PRVA APLIKACIJA NA KUBERNETES SUSTAVU 12](#_Toc41736869)

[5 TEMELJNI POJMOVI KUBERNETES SUSTAVA 19](#_Toc41736870)

[6 ZAKLJUČAK 20](#_Toc41736871)

[LITERATURA 21](#_Toc41736872)

[POPIS OZNAKA I KRATICA 22](#_Toc41736873)

[SAŽETAK 23](#_Toc41736874)

# UVOD

Novi trend razvijanja aplikacija su svakako mikro-servisi. Mikro-servis 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.

Mikro-servis 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 mikro-servis 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 mikro-servisa 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. Rad će biti fokusiran na Kubernetes sustavu u oblaku (engl. cloud) radije nego na Kubernetes sustavu koji se nalazi na vlastitom klasteru s ograničenim hardverom.

# MOTIVACIJA ZA UVOĐENJE KUBERNETES SUSTAVA

## Prelazak na mikro-servis 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 posjeduju 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 mikro-servis arhitekturu

Budući da je svaki mikro-servis neovisni proces, moguće ih je migrirati, razvijati i održavati neovisno o drugim mikro-servisima. Promjena na jednom mikro-servisu ne uzrokuje „redeploy“ ostalih mikro-servisa. Posljedično tome, kao rezultat dobijemo mogućnost skaliranja svakog mikro-servisa zasebno, neovisno o drugim mikro-servisima (Slika 2.2). Na taj način možemo skalirati samo one mikro-servise koji zahtijevaju više resursa, a ne cijelu aplikaciju.

A close up of a logo

Description automatically generated

Slika 2.2 Svaki mikro-servis 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 mikro-servis 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 mikro-servis 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“. 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 [1]

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 mikro-servis 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 mikro-servis arhitekture te se odlučile prebaciti na istu. Arhitektura kompanija kao što su Amazon i Netflix se sastoji od nekoliko tisuća mikro-servisa koji međusobno komuniciraju (Slika 2.1). Gotovo nemoguće je upravljati s nekoliko tisuća mikro-servisa koji komuniciraju i zajedno obavljaju neki složeniji posao. Uzročno tome, došlo je do potrebe za sustavom koji će olakšati razvoj aplikacije temeljene na mikro-servis arhitekturi.

A close up of a logo

Description automatically generated

Slika 2.4 Mikro-servis arhitektura Amazon-a i Netflix-a [2]

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.

# UVOD U KUBERNETES SUSTAV

## Pojednostavljeni prikaz rada Kubernetes sustava

Najveći nivo apstrakcije Kubernetes sustava izgleda kao na slici (Slika 3.1). Sustav se sastoji od glavnog čvora (Kubernetes master) i radnih čvorova. Broj radnih čvorova nije ni na koji način ograničen. Može se kretati od samo jednog čvora pa do preko tisuću radnih čvorova, ovisno o potrebi. Developer pri migraciji aplikacije na server specificira koliko instanci pojedine komponente aplikacije želi, sve ostalo je zadaća Kubernetes sustava. Kubernetes potom, na osnovu specificiranih zahtjeva, komponente aplikacije migrira na klaster radnih čvorova. Nebitno je na koji će radni čvor Kubernetes smjestiti komponentu aplikacije. Bitno je samo da su ispunjeni uvjeti koje je developer specificirao. Kubernetes sustav će samostalno zaključiti koji je radni čvor najpogodniji za izvršavanje pojedine komponente. Također, developer može naznačiti ukoliko je potrebno da se neke komponente nalaze na istom čvoru te će sukladno tome Kubernetes navedene komponente rasporediti na isti radni čvor. Ukoliko nije posebno naglašeno, Kubernetes će imati potpunu slobodu rasporeda komponenti na dostupne radne čvorove. Komponente će i dalje moći međusobno komunicirati iako se nalaze nasumično raspoređene na klasteru radnih čvorova.

A picture containing drawing

Description automatically generated

Slika 3.1 Proces „deploy-a“ na Kubernetes cluster

## Arhitektura Kubernetes sustava

Arhitektura kubernetes sustava se sastoji od puno čvorova (engl. nodes). Dva su glavna tipa čvorova:

1. Master čvor – sadrži Kubernetes upravljačku ploču koja upravlja cijelim sustavom.
2. Radni čvorovi – čvorovi na kojima se izvršava aplikacija koju „deploy-amo“.

A screenshot of a cell phone

Description automatically generated

Slika 3.2 Arhitektura Kubernetes sustava - glavni čvor i radni čvorovi [4]

Master čvor sastoji se od sljedećih komponenti:

* API server – korisnici (developeri, sistem administratori i drugi…) upravljaju Kubernetes klasterom komunicirajući s API serverom. U prethodnom poglavlju (Slika 3.1) prikazano je da korisnik opis zahtjeva aplikacije šalje master čvoru. Komunikacija između korisnika i master čvora se odvija upravo preko API servera.
* Kontroler (engl. Controller manager) – kontrolira cjelokupni klaster. Zadužen je za replikaciju komponenti, praćenje radnih čvorova, poduzima radnje u slučaju prestanka rada pojedinih čvorova i slično.
* Raspoređivač (engl. Scheduler) – Zadužen je za raspoređivanje komponenti aplikacije na radne čvorove kako bi resursi bili što bolje iskorišteni i kako bi bili zadovoljeni svi specificirani zahtjevi.
* etcd – pouzdana baza podataka koja sprema konfiguraciju i stanje klastera.

Radni čvor sastoji se od sljedećih komponenti:

* Kubelet – komunicira s API serverom i upravlja komponentama radnog čvora na kojem se nalazi.
* Kube-Proxy – zadužen je za optimalni raspored opterećenja između radnih čvorova.
* Docker – zadužen za dohvaćanje „container image-a“ i pokretanje container-a.

## Izvršavanje aplikacije na Kubernetes klasteru

Nakon što smo objasnili arhitekturu Kubernetes sustava, možemo detaljnije objasniti proces migriranja i izvršavanja aplikacije na Kubernetes klasteru nego što li je to objašnjeno u poglavlju 3.1.   
Prvi korak migriranja aplikacije na Kubernetes kluster (u nastavku „deploy“) je „pakiranje“ aplikacije u jednu ili više „container“ slika (engl. container images). Te se slike potom objave na registru slika (engl. image registry). Zatim developer objavi opis aplikacije na Kubernetes API server (Slika 3.1). U opisu aplikacije specificirano je koliko treba biti replika pojedine komponente, koje komponente se moraju nalaziti na istom radnom čvoru (ukoliko ima takvih komponenti), od koje container slike će nastati pojedina komponenta itd.   
API server obradi zahtjev te posao preuzima Raspoređivač (engl. Scheduler). Raspoređivač raspoređuje containere na slobodne radne čvorove uzimajući u obzir opterećenje radnog čvora, zahtijevane resurse pojedinog containera i sl. Nakon što Raspoređivač rasporedi gdje će ići koji container, Kubelet radnog čvora na kojem je container smješten naredi Dockeru da preuzme container sliku s registra slika te pokrene container na osnovu preuzete slike. Na slici (Slika 3.2) ostao je neobjašnjen termin „Pod“. Pod sadrži jedan ili više pokrenutih slika – containera. Pod će detaljno biti objašnjen u nadolazećim poglavljima.

Nakon što je aplikacija uspješno pokrenuta, Kubernetes sustav će i dalje kontinuirano nadgledati rad naše aplikacije vodeći računa da je broj svakog containera jednak broju kojeg je developer specificirao u opisu aplikacije. Ukoliko neki od containera prestane s radom Kubernetes će automatski restartirati problematični container. Slično tome, ukoliko neki od radnih čvorova prestane s radom, Kubernetes će odabrati drugi radni čvor pogodan za premještanje svih containera koji se nalaze na problematičnom radnom čvoru.

Svaka aplikacija ima periode kada posjećenost raste ili pada. Na već spomenutom primjeru Internet stranice krema za sunčanje, porast prometa će biti u ljetnim mjesecima, dok će u zimskom periodu promet biti vrlo mali. Koristeći Kubernetes sustav, ne moramo se brinuti da će dolaskom ljetnih mjeseci, naša aplikacija prestati s radom zbog velike posjećenosti. Isto tako, u zimskim mjesecima će Kubernetes sustav zaključiti kako imamo previše resursa nego što nam u tom trenutku treba te će automatski skalirati aplikaciju da ne trebamo plaćati dodatne, nepotrebne resurse. Na taj način ćemo uštedjeti novac, a isto tako osigurati da aplikacija radi i u periodima nagle velike posjećenosti. Kada potreba za pojedinim servisom pređe dozvoljenu specificiranu vrijednost, Kubernetes će automatski za nas napraviti još jednu kopiju kritičnog servisa.

## Prednosti korištenja Kubernetes sustava

Osim već spomenutog automatskog skaliranja u prethodnom poglavlju, Kubernetes sustav donosi i niz drugih prednosti. Automatskim skaliranjem postiže se ušteda novca reduciranjem nepotrebnih resursa i konzistentni rad naše aplikacije čak i u periodima povećane posjećenosti. Kubernetes sustav vrši konstantno nadgledanje cjelokupnog klastera. Ukoliko neki od čvorova ili containera prestane s radom, automatski će se poduzeti potrebna radnja. Dok nije bilo Kubernetes sustava, sistem administratori su morali konstantno nadgledati opterećenje pojedinih komponenti aplikacije i sukladno tome vršiti radnje koje bi rezultirale novim brojem instanci pojedine komponente, premještanjem svih komponenti na drugi radni čvor itd. Ovo možda i ne zvuči toliko strašno dok vam u 2 sata ujutro ne zazvoni mobitel da aplikacija nije dostupna u srcu sezone i da ste izgubili nekoliko tisuća ili milijuna potencijalne zarade. Kubernetes sustav omogućava sistem administratorima da mirno spavaju noću tako što automatski zamjenjuje pokvarene instance novim instancama. Ujutro, kada se naspavaju, u okviru radnog vremena mogu proučiti zbog čega je nastao problem i raditi na otklanjanju problema.

Koristeći Kubernetes sustav posao migriranja aplikacije na server više ne treba raditi Ops tim. Iz razloga što su aplikacije zapakirane u containere (engl. containerized), posjeduju sve potrebne biblioteke i okolinu za njihovo izvršavanje. Nije potrebno instalirati ništa dodatno na servere. Sukladno tome, programeri mogu samostalno aplikaciju migrirati na klaster. Svi radni čvorovi predstavljeni su kao jedna cjelina te nije potrebno imati nikoju informaciju u karakteristika servera za izvršavanje aplikacije. Radni čvorovi Kubernetes klastera ne moraju biti svi isti. Određen broj čvorova npr. može imati SSD diskove, dok drugi posjeduju HDD diskove. Ukoliko je potrebno, može se npr. specificirati zahtjev da se određena komponenta izvršava samo na čvorovima koji posjeduju SSD disk. Kubernetes sustav će samostalno rasporediti tu komponentu na čvor koji posjeduje SSD disk bez da mu mi točno navedemo koji je to čvor. S ovim smo postigli da ne trebamo voditi brigu na koji će se čvor pojedina komponenta smjestiti sve dok su zadovoljeni zahtjevi koje smo naveli u opisu aplikacije. Bez Kubernetes sustava, sistem administrator bi samostalno morao odabrati radni čvor koji posjeduje SSD disk od svih mogućih čvorova sa SSD diskom i na njega smjestiti komponentu.

Ljudi, za razliku od računala, su vrlo loši u pronalaženju optimalnih kombinacija pogotovo kad je broj takvih kombinacija velik. Zbog toga možemo biti sigurni da će Kubernetes sustav bolje i optimalnije rasporediti komponente aplikacije na klaster nego što bi to napravili ljudi. Kod odabira svakog radnog čvora uzimaju se u obzir razni parametri kao što su opterećenost radnog čvora, ispravnost rada, zahtijevani resursi i sl.

# PRVA APLIKACIJA NA KUBERNETES SUSTAVU

Prije nego što se objasne temeljni pojmovi Kubernetes sustava, svakako bi bilo dobro prvo na jednostavnom primjeru vidjeti kako izgleda proces izvršavanja aplikacije na Kubernetes-u. Mnogi autori literatura o Kubernetes sustavu slično pristupaju objašnjavanju istog navodeći sličnost s vožnjom auta. Prvo naučite voziti auto, a tek potom krenete učiti što se nalazi ispod haube i kako promijeniti ulje.

U ovom poglavlju cilj će biti napraviti „deploy“ jednostavne NodeJs aplikacije na Kubernetes klaster. Kompletan kod aplikacije može se naći na Github linku: <https://github.com/ikovac/kubernetes-hello-world>.  
Aplikacija se sastoji od svega par linija koda:

const express = require('express');

const app = express();

app.use('/', (req, res) => {

    res.send('Hello world!');

});

app.listen(3000, () => {

    console.log('App is listening on port 3000');

});

Sve što aplikacija radi je kreira server koji sluša sav dolazni promet na portu 3000 i odgovara sa „Hello world!“.

U poglavlju 3 već smo opisali kako izgleda ovaj proces, stoga će biti mnogo sličnosti ali s dodanim praktičnim primjerima. Kako je već rečeno prije nego što započnemo sa samim migriranjem na server, moramo aplikaciju zapakirati u Docker container. Da bi to napravili moramo kreirati Docker sliku.  
Unutar projekta kreirajmo datoteku naziva Dockerfile sa sljedećim naredbama:

FROM node:12.17.0

WORKDIR /usr/src/app

COPY package\*.json ./

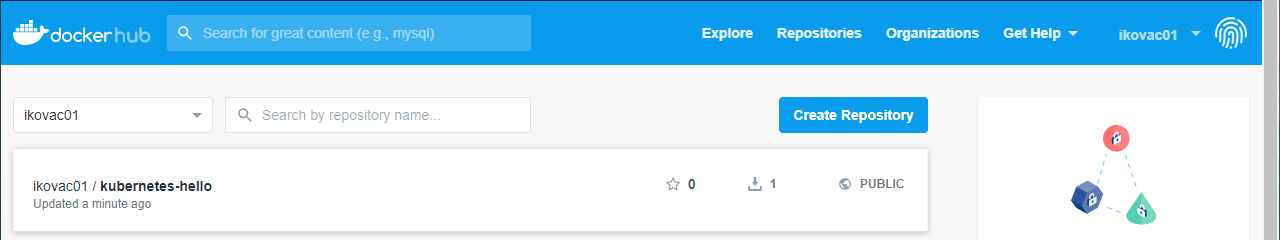
RUN npm install

COPY ./ ./

EXPOSE 3000

CMD ["node", "index.js"]

S ovim kažemo Dockeru da kreira sliku na temelju Nodejs slike. Time ćemo imati mogućnost izvršavanja NodeJs aplikacija bez da samostalno instaliramo NodeJs. Potom odaberemo radni direktoriji i kopiramo package.json datoteku unutar containera te instaliramo sve biblioteke.  
Naredbom „EXPOSE 3000“ naznačimo koji PORT koristimo unutar Docker containera. Naredba ništa ne radi nego služi samo kao smjernica. Na posljetku pokrenemo NodeJs aplikaciju sa naredbom „node index.js“ kao što bi to napravili i da izvršavamo aplikaciju izvan Docker-a. Da bi kreirali Docker sliku na temelju datoteke Dockerfile moramo izvršiti sljedeću naredbu:  
docker build -t image-name .  
Za image-name odabrao sam ikovac01/kubernetes-hello, gdje ikovac01 označava ime mog korisničkog računa na Docker Hub-u, a kubernetes-hello je ime same slike. Sliku je potrebno potom objaviti na registar slika – Docker Hub. To se postiže komandom: docker push ikovac01/kubernetes-hello. Ukoliko vam objava slike nije uspjela, morate se prijaviti na Docker Hub račun sa naredbom docker login. Pri uspješnoj objavi, Docker slika se treba pojaviti na Docker Hub-u (Slika 4.1).



Slika 4.1 Objava slike na Docker Hub

Prethodnim korakom našu aplikaciju smo zapakirali unutar Docker slike i to objavili na registar slika. Kubernetes će dohvatiti našu sliku i na temelju nje kreirati Docker container unutar kojeg će se izvršavati naša aplikacija.

Sljedeći korak je kreiranje Kubernetes klastera. Da bi mogli kreirati klaster prvo moramo imati račun na GKE. Iako se usluga plaća, dostupan je probni period od jedne godine koji je potpuno besplatan i bit će korišten u svrhu ovog diplomskog rada.

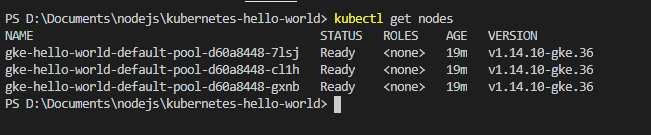
Kreiranje klastera je vrlo jednostavno. Stisnemo dugme „Create cluster“ i odaberemo ime našeg klastera (Slika 4.2).

A screenshot of a cell phone

Description automatically generated

Slika 4.2 Izrada klastera

Izrada klastera će potrajati neko vrijeme dok se ne kreiraju svi radni čvorovi koji posjeduju vlastite IP adrese. Klaster će se sastojati od 3 radna čvora, ukoliko niste promijenili standardne postavke pri kreiranju klastera. Nakon što se klaster kreira, možemo se iz komandne linije računala spojiti na novo-kreirani klaster. To se radi pritiskom na dugme „Connect“ i kopiranjem danog linka u komadnu liniju. Da bi se uvjerili kako klaster stvarno sadrži 3 radna čvora možemo izvršiti naredbu „kubectl get nodes“ (Slika 4.3). Kao rezultat dobijemo listu 3 radna čvora s nazivom, statusom i ostalim specifikacijama.



Slika 4.3 Lista radnih čvorova klastera

Sada konačno možemo objaviti našu aplikaciju na Kubernetes klaster. U prethodnim poglavljima rečeno je da developer mora objaviti opis aplikacije Kubernetes glavnom čvoru. Opis aplikacije se piše u yaml datotekama. Za objaviti aplikaciju kreirat ćemo datoteku deployment.yaml sa sljedećim kodom:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: kubernetes-hello

spec:

  replicas: 1

  selector:

    matchLabels:

      app: kubernetes-hello

  template:

    metadata:

      labels:

        app: kubernetes-hello

    spec:

      containers:

      - image: ikovac01/kubernetes-hello

        name: kubernetes-hello

U opisu se nalazi specificirano na osnovu koje Docker slike želimo kreirati aplikaciju i koliko replika te aplikacije želimo. Iako sama datoteka može izgledati zbunjujuće to je doista sve što smo naveli. Kao vrstu opisa koju šaljemo glavnom čvoru stavili smo „Deployment“. Rečeno je da Kubernetes u slučaju prestanka rada neke komponente kreira novu komponentu. To upravo radi deployment resurs. Ukoliko trenutni broj replika ne odgovara specificiranom broju (u ovom slučaju samo jedna replika) onda deployment poduzme radnje kako bi se postigao specificirani broj replika. Iz razloga što se naša aplikacija može sastojati od velikog broja servisa sa selektorom smo odabrali koju komponentu želimo staviti u nadzor ovog deployment-a. Odabrali smo selektor „kubernetes-hello“ jer je to oznaka (engl. label) koju smo dodijelili toj komponenti. Deployment tip resursa će biti objašnjen detaljnije u narednim poglavljima.

Objava na glavni čvor se vrši naredbom „kubectl apply -f ./deployment.yaml“. Primjetit ćemo da se pojavila crvena oznaka jer jedna replika ove aplikacije još nije spremna (Slika 4.4). Imamo 0/1 pokrenutih Pod-ova. Pod će također detaljnije biti objašnjen kasnije. Za sada to zamislimo kao kutiju u kojoj se izvršava jedan ili više containera.

A screenshot of a computer

Description automatically generated

Slika 4.4 Replika je u procesu stvaranja

Nakon što Kubernetes dohvati objavljenu Docker sliku i kreira Doker container iz iste trebali bi dobiti zelenu oznaku koja govori kako je sve prošlo uspješno (Slika 4.5).

A screenshot of a social media post

Description automatically generated

Slika 4.5 Replika je kreirana

Ovo možemo provjeriti i ako dohvatimo sve podove na kreiranom klasteru naredbom „kubectl get po“ (Slika 4.6) što je skraćeno od „kubectl1 get pods“.

A close up of a screen

Description automatically generated

Slika 4.6 Lista svih podova

Završetkom ovog koraka aplikacija je spreman i izvršava se na Kubernetes klasteru.

Kako pristupiti aplikaciji i uvjeriti se da stvarno radi?  
Pri kreiranju Pod-a, svaki dobije vlastitu IP adresu. Ta IP adresa je interna adresa i moguće joj je pristupiti samo unutar klastera. Da bi pristupili aplikaciji moramo nekako izložiti pristup aplikaciji i izvan klastera. To se radi pomoću novog Kubernetes resursa nazvanog „Service“ koji će također biti detaljnije objašnjen kasnije.

Da bi kreirali Service, kreirajmo novu datoteku service.yaml sa sljedećim kodom:

apiVersion: v1

kind: Service

metadata:

  name: kubernetes-hello-service

spec:

  type: LoadBalancer

  ports:

  - port: 3000

    targetPort: 3000

  selector:

    app: kubernetes-hello

Service možemo nazvati kako hoćemo. U primjeru je nazvan kubernetes-hello-service. Sljedeće što moramo napraviti je specificirati tip servisa. LoadBalancer je tip servisa koji će biti dostupan izvan klastera jer će imati vanjsku IP adresu na koju ćemo moći pristupiti aplikaciji. Uz IP adresu moramo specificirati i na kojem portu želimo da aplikacija bude dostupna. Odabran je port 3000. Ovo može biti bilo koji drugi slobodni i dostupni port samo je bitno da kao „targetPort“ bude navedeno 3000 jer je to port na kojem aplikacija sluša dolazeće zahtjeve. Naposljetku, sa selektorom kažemo Kubernetes-u koju točno komponentu aplikacije želimo izložiti da bude javno dostupna. Naša aplikacija ima samo jednu komponentu kojoj smo dodijelili oznaku „kubernetes-hello“.

Nakon što objavimo servis na glavni čvor već poznatom naredbom „kubectl apply -f ./service.yaml“ u listi servisa moći ćemo vidjeti novo-kreirani servis (Slika 4.7).

A screenshot of a social media post

Description automatically generated

Slika 4.7 Servis je kreiran, ali još nije spreman

Aplikacija još neće biti dostupna jer proces dodjele javne IP adrese traje neko vrijeme. Nakon što se dodijeli javna IP adresa možemo pristupiti servisu (Slika 4.8).

A screenshot of a social media post

Description automatically generated

Slika 4.8 Javna IP adresa je dodijeljena servisu

Ovim korakom aplikacija je u potpunosti spremna i možemo joj pristupiti. Ukoliko odemo na dobivenu vanjsku IP adresu na port 3000 vidjet ćemo „Hello world!“ kao rezultat (Slika 4.9).

A screenshot of a social media post

Description automatically generated

Slika 4.9 Aplikacija je dostupna izvan klastera

Bilo tko u svijetu može pristupiti aplikaciji na dobivenoj IP adresi. Aplikacija je napokon objavljena i radi.

Iako se proces možda u početku pari kompliciran, u biti je vrlo jednostavan. Ukoliko napravimo sažetak, sve što smo napravili je zapakirali aplikaciju u Docker sliku i tu sliku objavili na registar slika. Potom smo glavnom čvoru poslali opis aplikacije sa navedenim imenom Docker slike i koliko replika želimo i na kraju smo kreirali servis da naša aplikacija bude dostupna i izvan klastera. Aplikacija je sada u rukama Kubernetes sustava koji će biti zadužen da uvijek imamo željeni broj replika i u slučaju prestanka rada kreirati nove replike. Moguće je i naknadno mijenjati broj replika. Samo uredite datoteku tako što promijenite broj iz 1 u novi željeni broj replika i spremite promjene. Kubernetes sustav će potom vidjeti da željeni broj replika nije jednak postojećem stanju te će kreirani nove replike.

# TEMELJNI POJMOVI KUBERNETES SUSTAVA

# 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. svibnja 2020.

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

[4] „What is Kubernetes?“, s interneta, <https://linuxhint.com/what-is-kubernetes/>, 28. svibnja 2020.

# 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  
API – Application Programming Interface  
itd – i tako dalje  
sl – slično  
SSD - Solid-state Drive  
HDD – Hard Disk Drive  
GKE – Google Kubernetes Engine

# SAŽETAK