SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**Kontinuirana integracija i kontinuirana isporuka aplikacija visoke raspoloživosti i skalabilnosti postavljenih unutar Kubernetesa u oblaku**

Mislav Jelušić

Zagreb, lipanj 2023.

Mentor: prof. dr. sc. Željko Ilić

Voditelj rada: prof. dr. sc. Željko Ilić

.

Sadržaj

[1. Uvod 1](#_Toc132109464)

[1.1. Naslov potpoglavlja 1](#_Toc132109465)

[1.1.1. Naslov potpoglavlja 1](#_Toc132109466)

[2. Kubernetes u oblaku 2](#_Toc132109467)

[2.1. Mikroservisi 2](#_Toc132109468)

[2.2. Kontejneri 3](#_Toc132109469)

[2.3. Orkestracija kontejnera 4](#_Toc132109470)

[2.3.1. Kubernetes 5](#_Toc132109471)

[2.3.2. Alat Helm chart 5](#_Toc132109472)

[2.4. Računalstvo u oblaku 6](#_Toc132109473)

[2.4.1. Karakteristike platforme Amazon Web Services 7](#_Toc132109474)

[2.4.2. AWS virtualni privatni oblak 9](#_Toc132109475)

[2.4.3. Usluga AWS Elastic Kubernetes Service 10](#_Toc132109476)

[2.4.4. AWS komandna linija 11](#_Toc132109477)

[2.5. Infrastruktura kao kod 12](#_Toc132109478)

[2.5.1. Alat Terraform 13](#_Toc132109479)

[2.5.2. Tok rada Terraforma 14](#_Toc132109480)

[2.6. Postavljanje infrastrukture 15](#_Toc132109481)

[2.6.1. Izgradnja infrastrukture uz pomoć Terraforma 15](#_Toc132109482)

[2.6.2. Spajanje na Kubernetes grozd 17](#_Toc132109483)

[2.6.3. Postavljanje pomoćnih alata na Kubernetes uz pomoć Helm Charta 18](#_Toc132109484)

[3. Agilni razvoj 19](#_Toc132109485)

[3.1. Kontinuirana integracija i kontinuirana isporuka 19](#_Toc132109486)

[3.1.1. Alat GitHub Actions 19](#_Toc132109487)

[3.2. GitOps 19](#_Toc132109488)

[3.2.1. Alat ArgoCD 19](#_Toc132109489)

[4. Razvoj cjevovoda 20](#_Toc132109490)

[Sažetak 21](#_Toc132109491)

[Summary 22](#_Toc132109492)

[Zaključak 23](#_Toc132109493)

[Literatura 24](#_Toc132109494)

[Skraćenice 25](#_Toc132109495)

# Uvod

Tekst

## Naslov potpoglavlja

Tekst

### Naslov potpoglavlja

**Primjer ubacivanje slike u tekst i referenciranja na istu.**

Na slici (Slika 1) predočen je ...

Slika 1-1.

**Primjer ubacivanje tablice u tekst i referenciranja na istu.**

U tablici (Tablica 1) su navedeni rezultati...

Tablica . Tekst

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**Primjer ubacivanje reference u tekst i referenciranja na istu.**

...što je navedeno u [1].

# Kubernetes u oblaku

## Mikroservisi

Mikroservisna arhitektura je način razvoja programa kod kojeg se aplikacija sastoji od više malih, neovisnih usluga koje međusobno komuniciraju putem API-ja (*engl. Application Programming Interface*) ili komunikacija porukama (*engl. Message-queuing systems*). Nastala je na temeljima monolitne arhitekture u kojoj su usluge ili komponente bile čvrsto povezane. [2]

Prednosti ovakvog razvoja programa su poboljšana skalabilnost, fleksibilnost i tolerancija na pogreške, te mogućnost neovisne implementacije i ažuriranja usluga. Skalabilnost je moguća zbog same neovisnosti pojedine usluge, pa je tako moguće horizontalno ili vertikalno skalirati uslugu ovisno i njenom opterećenju unutar aplikacije. Aplikacijske usluge imaju različit prioritet i stadij razvoja. Odvojenost usluga unutar aplikacije omogućuje programerima neovisan razvoj svake pojedine usluge što dovodi do bržeg razvoja i izlaska na tržište čitave aplikacije. U slučajevima kvara jedne od usluga, cijela aplikacija može nastaviti raditi, a programeri se mogu fokusirati na popravak te usluge bez da je čitav sustav neaktivan. Neovisnost se također ogleda u pogledu korištenja tehnologija. Moguće je koristiti različite tehnologije unutar svake pojedine usluge zasebno. [2]

Ipak, postoje mane i izazovi ovakvih arhitektura. Jedna je složenost, iako je sve odvojeno veliki sustavi imaju jako puno usluga kojima treba na kraju upravljati i povezati što povećava složenost sustava. Također svaka mikrousluga zahtijeva vlastite resurse za rad što može dovesti do povećane potražnje za resursima u odnosu na monolitnu aplikaciju. To povećava troškove, posebno u slučaju povećanog opterećenja. Osim toga, može doći do dodatnih troškova zbog komunikacije između usluga i potrebe za učinkovitim nadzorom i testiranjem aplikacije u cjelini. Razvoj i implementacija aplikacije temeljene na mikrouslugama može biti složena i zahtijeva specijalizirano znanje i infrastrukturu. Planiranje i dizajniranje sustava je kompliciranije zbog odluka o strategijama implementacije, upravljanju podacima, korištenju komunikacijskih protokola i drugih. [3]

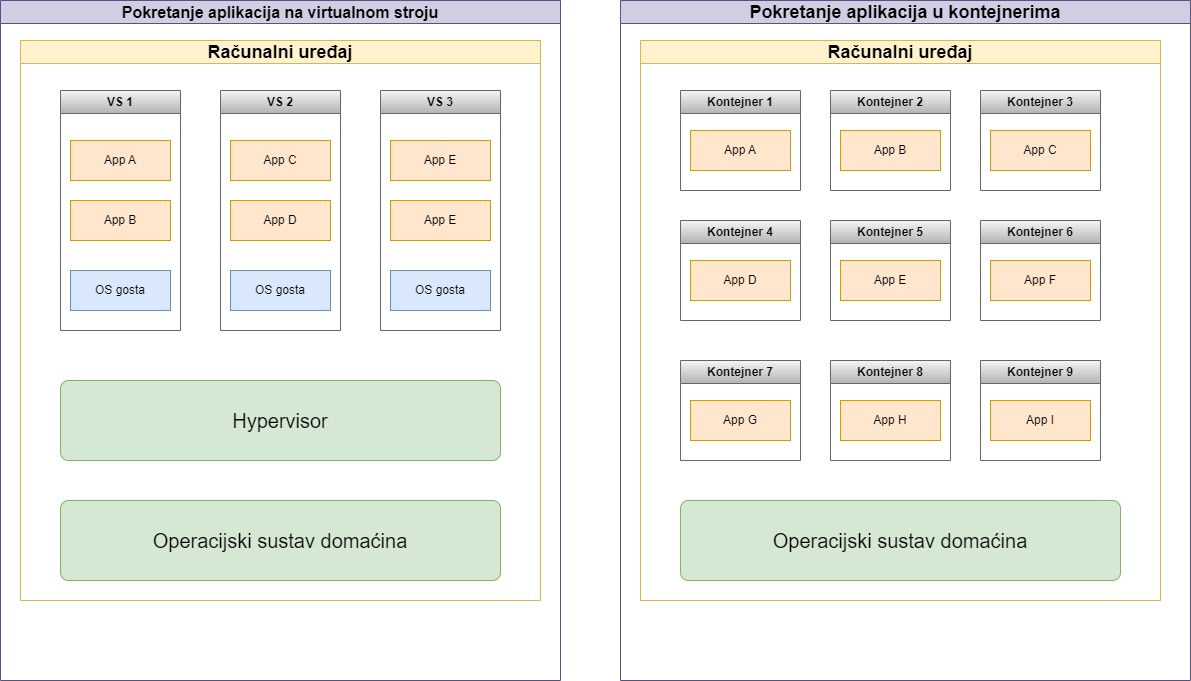
Kako bi se olakšao početni razvoj te iskoristila fleksibilnost i skalabilnosti, mikroservisna arhitektura se postavlja na oblak, koji je opisan u kasnijim poglavljima. Osim toga, alati kao što su Kubernetes, Istio i Docker postali su popularni za upravljanje i implementaciju mikroservisnih aplikacija. [3]

## Kontejneri

Kontejneri su popularna tehnologija za implementaciju aplikacija temeljenih na mikroservisnoj arhitekturi. Pružajući lagan i prenosiv način za pakiranje i distribuciju aplikacija i njihovih ovisnosti. Mogućnost brzog pokretanja i uništavanja kontejnera, programerima olakšava testiranje i promjenu unutar koda na pojedinačne usluge bez utjecaja na cijelu aplikaciju. [5]

Kontejneri su oblik virtualizacije operacijskog sustava koji omogućuje aplikacijama da se izvode izolirano od osnovnog operativnog sustava. Oni pružaju lagano i prijenosno okruženje za rad koje se može lako premještati između različitih sustava. Kontejneri se sastoje od aplikacije i svih njezinih ovisnosti, pakiranih zajedno u jednu izvršnu sliku (*engl. image*). Ova se slika može jednostavno distribuirati i pokrenuti na bilo kojem sustavu koji podržava kontejnerizaciju putem registara (*engl. registries*). Registri su repozitoriji koji pohranjuju slike, koje se onda kasnije mogu preuzeti na bilo koje računalo povezano na Internet. [4]

Na slici (Slika 1) je prikazano pokretanje aplikacija unutar virtualnog stroja i unutar kontejnera. Kontejneri su izbacili potrebu za hipervizorom (*engl. Hypervisor*). Hipervizor virtualizira fizičke računalne resurse glavnog računala, kao što su CPU, memorija i pohrana, i dodjeljuje ih virtualnim strojevima koji mogu neovisno pokretati vlastite operativne sustave i aplikacije. [6] Dodatno, slika prikazuje učestalu praksu gdje se samo jedna aplikacija postavlja unutar kontejnera dok se kod virtualnih strojeva više aplikacija nalazi unutar jednog virtualnog stroja zbog nedostatka resursa koji se trebaju zauzeti za pojedini virtualni stroj. Što znači da su kontejneri mnogo „lakši“ od virtualnih strojeva. Odnosno, pomoću kontejnera je moguće pokretati puno više aplikacija na istom hardveru. Važno je napomenuti da svi kontejneri na računalu koriste jedan kernel domaćina, dok virtualni strojevi koriste vlastiti kernel. To čini virtualni stroj više izoliranim u odnosu na kontejnere. [4]



Slika . Izolacija aplikacija na virtualnom stroju (lijevo), na kontejnerima (desno) [4]

Nekoliko je prednosti korištenja kontejnera u razvoju softvera. Kontejneri pružaju konzistenciju i mogućnost ponovnog stvaranja okruženja za pokretanje aplikacija, bez obzira na temeljni sustav. Lako prenošenje i pokretanje olakšava implementaciju i skaliranje aplikacija. Kontejneri također omogućuju lakše upravljanje aplikacijama, omogućujući programerima da lakše testiraju, ispravljaju pogreške i implementiraju svoje aplikacije. [5]

Docker je najpopularniji sustav za izvođenje kontejnera i upravljanje slikama. Omogućuje jednostavan i intuitivan način za izradu, pakiranje i distribuciju slika kontejnera. Docker slike mogu se jednostavno prenijeti u registar kontejnera, što olakšava dijeljenje i distribuciju slika u različitim sustavima. Docker je razvio svoj registar kontejnera koji se zove Docker Hub. Kako bi se izradile Docker slike potrebno je napisati Dockerfile odnosno tekstualnu datoteku koja sadrži niz instrukcija za izradu slike. Docker je moguće instalirati na svim popularnim operacijskim sustavima kao što su Linux, Microsoft Windows ili MacOS. [5]

## Orkestracija kontejnera

Korištenjem mikroservisne arhitekture zajedno s kontejnerima, broj kontejnera raste i javlja potreba za lakšim upravljanjem i postavljanjem kontejnera odnosno aplikacijskih usluga unutar kontejnera. Rješenje se nalazi u orkestraciji kontejnera odnosno alatima za orkestraciju. Orkestracija kontejnera je programsko rješenje koje pomaže u implementaciji, skaliranju i upravljanju kontejnerskom infrastrukturom. Omogućuje jednostavnu implementaciju aplikacija u više kontejnera rješavanjem izazova upravljanja pojedinačnim kontejnerima. Ova arhitektura omogućuje automatizaciju životnog ciklusa aplikacije pružajući jedinstveno sučelje za stvaranje i orkestriranje kontejnera. Pomoću ovih alata moguće je stvaranje, ažuriranje i uklanjanje aplikacije bez brige za infrastrukturu koja ih pokreće. Sposobnost jednostavnog skaliranje je još jedan od prednosti, a moguće je skalirati na dva načina. Horizontalno skaliranje dodaje više replika aplikacije. Dok se vertikalnim skaliranjem aplikaciji povećavaju resursi koje aplikacija može koristiti, kao što su procesorska snaga i memorija. Također, olakšano je upravljanje mrežnih dijelova infrastrukture kao i čuvanje od sigurnosnih rizika. [7]

### Kubernetes

Kubernetes je najpopularniji alat za orkestraciju kontejnera i alat koji će se koristiti u ovom diplomskom radu. Početno je to bio projekt koji je razvijao Google, no kasnije je doniran organizaciji Cloud Native Computing Foundation (CNCF). CNCF razvija projekte otvorenog koda te je Kubernetes ujedno jedan od njih.

Kubernetes pruža platformu za implementaciju, upravljanje i skaliranje aplikacija u kontejneru. U svojoj srži, Kubernetes se sastoji od skupa glavnih čvorova i radnih čvorova. Glavni čvorovi odgovorni su za upravljanje radničkim čvorovima i raspoređivanje radnih opterećenja za njih. Radnički čvorovi odgovorni su za pokretanje kontejnerskih aplikacija. Kubernetes pruža skup API-ja i alata za implementaciju i upravljanje kontejnerskim aplikacijama, kao i skup kontrolera za upravljanje infrastrukturom potrebnom za pokretanje aplikacija. Kubernetes koristi deklarativne konfiguracijske datoteke, zvane manifesti, za definiranje željenog stanja infrastrukture. Ovi manifesti opisuju željeno stanje sustava, uključujući broj replika usluge, resurse potrebne za kapsulu i mrežna pravila. Kubernetes kontinuirano prati trenutno stanje sustava i radi na tome da trenutno stanje odgovara željenom stanju. [8]

Kubernetes je široko prihvaćen među pružateljima usluge oblaka, a većina pružatelja usluga sada nudi upravljane Kubernetes usluge. Upravljani Kubernetes omogućuje jednostavan i učinkovit način za upravljanje Kubernetes klasterima bez brige o temeljnoj infrastrukturi. Temeljna infrastruktura Kubernetesa se odnosi na komponente upravljačke ravnine koje se nalaze na glavnom čvoru i komponente radničkih čvorova. Što znači da se korisnik Kubernetesa ne mora brinuti oko instalacije Kubernetes komponenti na infrastrukturi oblaka, nego to pružatelj usluge radi automatski pri instalaciji upravljanog Kubernetesa od strane korisnika. Ovime se zaobilazi dugi put učenja o arhitekturi Kubernetesa i instalacije komponenti Kubernetesa. Također je olakšana nadogradnja na novije verzije, pa tako većina pružatelja usluge ima jednostavni način podizanja verzije Kubernetesa u par koraka.

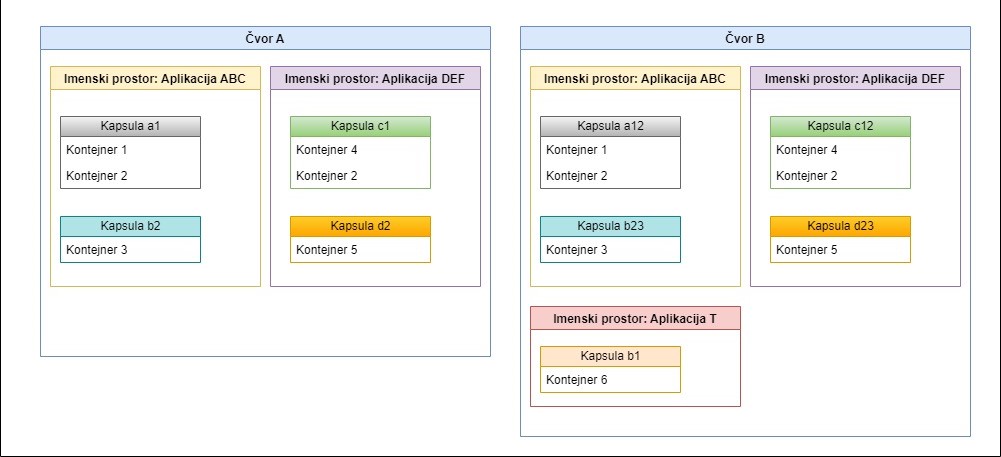
### Glavni koncepti Kubernetesa

Razumijevanje Kubernetesa počinje od njegovih koncepata, iako ih ima mnogo, u ovom radu će biti opisani i definirani samo glavni za osnovno razumijevanje rada u Kubernetesu.

Radni čvorovi mogu biti fizički ili virtualni strojevi. Svaki od čvorova sadrži usluge potrebne za pokretanje kapsula. Kubernetes grozd je skup glavnih i radnih čvorova. Svaki grozd se sastoji od minimalno jednog glavnog i jednog radnog čvora. Komunikacija korisnika s grozdom se obavlja preko API-ija na glavnom čvoru.

Kapsule (*engl. Pods*) su najmanje jedinice unutar Kubernetesa koje je moguće stvoriti. Jedan ili više kontejnera je moguće staviti unutar jedne kapsule, ali najčešće je to samo jedan kontejner.

Na slici (Slika 2)



Slika . Prikaz odnosa čvorova, kapsula, kontejnera i imenskog prostora

### Alat Helm chart

Helm je upravitelj paketa za Kubernetes koji pojednostavljuje implementaciju i upravljanje aplikacijama i uslugama na Kubernetes grozdovima. Helm chart su paketi koji sadrže sve potrebne resurse i konfiguracije za implementaciju aplikacije ili usluge na Kubernetes grozdu.

Helm koristi YAML konfiguracijske datoteke za definiranje resursa koji bi trebali biti postavljeni kao dio aplikacije unutar Kubernetesa. Konfiguracija tih datoteka je deklarativna, kao u već spomenutim Terraform datotekama. Ovo omogućuje korisnicima da specificiraju željeno stanje svojih resursa, a Helm će osigurati da su resursi kreirani i konfigurirani kako je navedeno.

Kao i s Terraformom snaga Helma je u ponovnom korištenju. Helm chartovi se mogu koristiti za implementaciju višestrukih instanci aplikacije ili usluge i mogu se lako dijeliti s drugima. To olakšava implementaciju i upravljanje aplikacijama na dosljedan i ponovljiv način te pomaže u smanjenju vremena i truda potrebnog za implementaciju i upravljanje aplikacijama na Kubernetes grozdovima.

Helm chartovi koriste semantičko određivanje verzija za praćenje promjena resursa i konfiguracija koje definiraju. To olakšava praćenje i vraćanje promjena te nadogradnju na novije verzije aplikacije ili usluge. Svakim novim postavljanjem ili ažuriranjem aplikacije, povećava se broj koji se naziva revizija (eng. *Revision*). U svakom trenutku moguće je vraćanje na stare revizije. Uz reviziju, istovremeno se prate verzija charta i verzija aplikacije.

## Računalstvo u oblaku

Računalstvo u oblaku (eng. *Cloud Computing*) je pojam koji se koristi za opisivanje isporuke računalnih usluga kao što su pohrana, obrada, umrežavanje, softver, analitika i ostalih usluga putem interneta. Ove se usluge pružaju na zahtjev, obično na osnovi tekućeg plaćanja.

Računalstvo u oblaku revolucioniralo je način na koji tvrtke i pojedinci pristupaju i koriste tehnologiju. Uklonio je potrebu za skupim hardverskim i softverskim instalacijama i omogućio korisnicima pristup svojim podacima i aplikacijama s bilo kojeg mjesta, u bilo koje vrijeme, na bilo kojem uređaju.

Postoji nekoliko vrsta modela računalstva u oblaku, podjela je moguća na javne oblake, privatne oblake i hibridne oblake. Javni oblaci su u vlasništvu tvrtki koje nude svoje usluge i dostupni su široj javnosti. Tvrtke održavaju fizičku infrastrukturu te klijentima nude računalne usluge putem interneta. Privatnim oblacima upravlja jedna organizacija te ih koristi isključivo ta organizacija. Javnosti nije dostupan pristup privatnim oblacima te nije moguće koristiti njihove usluge. Hibridni oblaci su kombinacija javnih i privatnih oblaka, omogućujući organizacijama da iskoriste prednosti obaju modela.

Podjela prema modelu usluga koji neki oblak nudi su infrastruktura kao usluga (eng. *Infrastructure as a Service - IaaS)*, platforma kao usluga (eng. *Platform as a Service – PaaS*) i softver kao usluga (eng. *Softver as a Service – SaaS*). Infrastruktura kao usluga označava skup usluga gdje pružatelj upravlja dijelom infrastrukture zadužene za pohranu podataka, servere i mrežni dio te korisniku isporučuje pristup virtualnim strojevima ili jednostavnom skladištenju podataka. Platforma kao usluga je model u kojem se pruža platforma za razvoj, pokretanje i upravljanje aplikacijama i uslugama. Uključuje niz alata i usluga koje programeri mogu koristiti za izradu, testiranje i implementaciju aplikacija, bez brige o temeljnoj infrastrukturi ili upravljanju infrastrukturom. Primjer takve usluge bi bila Amazonova implementacija Kubernetesa na njihovom oblaku - EKS (eng. *Elastic Kubernetes Service*). Softver kao usluga, model je računalstva u oblaku u kojem se softverske aplikacije pružaju kao usluga putem interneta, umjesto da se instaliraju i izvode na pojedinačnim računalima ili poslužiteljima. Uz SaaS, korisnici mogu pristupiti softveru i koristiti ga putem web-preglednika ili druge klijentske aplikacije, a davatelj usluge odgovoran je za hosting, održavanje i podršku. SaaS omogućuje korisnicima korištenje softverskih aplikacija na temelju pretplate, umjesto da ih moraju kupiti i instalirati na vlastite sustave.

Jedna od glavnih prednosti računalstva u oblaku je njegova skalabilnost. Uz mogućnost brzog i jednostavnog dodavanja ili uklanjanja resursa, organizacije mogu povećati ili smanjiti svoje računalne potrebe prema potrebi, bez potrebe za skupim ulaganjima u hardver. To im omogućuje da budu agilniji i osjetljiviji na promjenjive poslovne potrebe.

Još jedna prednost računalstva u oblaku je njegova isplativost. Plaćajući samo resurse koje koriste, organizacije mogu uštedjeti novac na hardveru, održavanju i troškovima energije. Osim toga, računalstvo u oblaku može smanjiti potrebu za IT osobljem, budući da mnoge zadatke povezane s upravljanjem lokalnom infrastrukturom obavlja pružatelj usluga u oblaku.

Sigurnost fizičke infrastrukture i ponekih softverskih stavki isto tako preuzima pružatelj. Zbog toga, postoje i neki potencijalni nedostaci računalstva u oblaku. Jedna od briga je sigurnost jer se osjetljivi podaci pohranjuju i obrađuju izvan mjesta organizacije. Kako bi se to riješilo, organizacije moraju odabrati uglednog i sigurnog pružatelja usluga oblaka i implementirati odgovarajuće sigurnosne mjere za zaštitu svojih podataka.

### Karakteristike platforme Amazon Web Services

Amazon Web Services (AWS) je platforma za računalstvo u oblaku koja pruža širok raspon usluga i alata za tvrtke svih veličina. Ove su usluge osmišljene kako bi omogućile tvrtkama izgradnju, implementaciju i upravljanje aplikacijama odnosno radnim opterećenjima u oblaku, kao i pohranjivanje i upravljanje podacima.

AWS nudi preko 200 različitih proizvoda i usluga, koji su grupirani u nekoliko kategorija: računalstvo, pohrana i isporuka sadržaja, baza podataka, umrežavanje, sigurnost, analitika, strojno učenje, umjetna inteligencija, integracija aplikacija, razvojni alati i Internet stvari (eng. *Internet of Things -* IoT). Neke od najpopularnijih i najčešće korištenih usluga uključuju Amazon Elastic Compute Cloud (EC2) za računalstvo, Amazon Simple Storage Service (S3) za pohranu i Amazon Relational Database Service (RDS) za baze podataka.

Jedna od ključnih prednosti korištenja AWS-a je njegova skalabilnost. Tvrtke mogu jednostavno i brzo povećati ili smanjiti svoje resurse kako bi zadovoljile promjenjive potrebe svojih aplikacija i radnih opterećenja. To im omogućuje uštedu novca plaćajući samo resurse koje koriste, umjesto održavanja fiksne infrastrukture. AWS također nudi model određivanja cijena po principu "pay-as-you-go", što znači da tvrtke plaćaju samo resurse koje potroše.

AWS također ima globalnu infrastrukturu koja je dizajnirana za visoku dostupnost i pouzdanost. Ima više podatkovnih centara smještenih diljem svijeta, što pomaže osigurati da aplikacije i radna opterećenja ostanu dostupni čak i ako postoje problemi s određenim podatkovnim centrom. Infrastruktura odnosno podatkovni centri su raspoređeni po lokacijama. Ove lokacije sastoje se od AWS regija, zona dostupnosti i lokalnih zona. Svaka AWS regija je zasebno geografsko područje. Svaka AWS regija ima više izoliranih lokacija poznatih kao zone dostupnosti. Korištenjem lokalnih zona, moguće je smjestiti resurse na više lokacija bliže svojim korisnicima. Amazon RDS vam omogućuje da resurse, kao što su instance baze podataka, i podatke smjestite na više lokacija. Resursi se ne repliciraju u AWS regijama osim ako to ne učinite izričito. Svaka regija označena je svojim identifikacijskim imenom, primjerice podatkovni centar koji se nalazi u Frankfurtu, za svoj identifikacijski kod koristi „eu-central-1“. Identifikacijski kodovi su bitni prilikom dizanja nekih resursa.

Graphical user interface

Description automatically generated with low confidence

Slika : Lokacijska raspodjela

Na slici (Slika 1) se vidi u kojem su odnosu regija, zone dostupnosti i lokalne zone.

AWS također nudi razne alate i usluge za nadzor i upravljanje aplikacijama, kao i za rješavanje problema i otklanjanje grešaka.

Uz svoje tehničke mogućnosti, AWS također nudi niz opcija podrške kako bi pomogao tvrtkama da izvuku najviše iz njegovih usluga. To uključuje dokumentaciju, forume i resurse zajednice, kao i sveobuhvatnije planove podrške koji omogućuju pristup tehničkim stručnjacima i drugim resursima.

Međutim, postoje i neki izazovi u korištenju AWS-a. Jedan od izazova je trošak korištenja AWS-a. Iako model određivanja cijena po principu "pay-as-you-go" može biti isplativ za tvrtke koje koriste samo malu količinu resursa, može postati skuplji za tvrtke koje koriste veliku količinu resursa. To može biti posebno izazovno za tvrtke koje imaju fluktuirajuće potrebe za resursima, jer će možda trebati platiti za resurse koje ne koriste.

Drugi izazov je složenost korištenja AWS-a. AWS nudi širok raspon proizvoda i usluga, što može biti neodoljivo za tvrtke koje tek počinju koristiti oblak. Zbog toga poduzećima može biti teško odrediti koji su im proizvodi i usluge potrebni i kako ih učinkovito koristiti. Problem se može javiti i kod tvrtki koje nemaju obučene ljude za korištenje AWS-a što može dodatno koštati organizaciju u vidu vremena i novca.

### AWS virtualni privatni oblak

Virtualni privatni oblak (eng. *Virtual Private Cloud* – VPC) je logički izolirani dio AWS oblaka koji korisnicima omogućuje pokretanje resursa u virtualnoj mreži koju sami definiraju. VPC omogućuje korisnicima stvaranje virtualne mreže koja je izolirana od ostatka AWS oblaka i kontrolu mrežnih konfiguracija svojih resursa.

VPC je podijeljen u jednu ili više podmreža, koje su grupe instanci Amazon Elastic Compute Cloud (EC2) koje se nalaze u određenoj zoni dostupnosti (AZ). Podmreže mogu biti javne ili privatne, ovisno o tome imaju li ili nemaju direktni put do interneta.

Svaki VPC ima sigurnosni dio koji štiti cijelu ili dijelove te mreže. Stoga postoje usluge kao što su sigurnosna grupa (eng. *Security groups*) i liste kontrole mrežnog pristupa (eng. *Network Access Control Lists –* NACLs). Sigurnosna grupa je virtualni vatrozid koji kontrolira dolazni i odlazni promet za grupu instanci Amazon EC2. Sigurnosne grupe mogu se koristiti za dopuštanje ili odbijanje prometa na temelju protokola, pristupa (eng. *Port*) i izvornih/odredišnih IP adresa. NACL je virtualni vatrozid koji kontrolira dolazni i odlazni promet za VPC podmrežu. NACL se mogu koristiti za dopuštanje ili odbijanje prometa na temelju protokola, priključaka i izvornih/odredišnih IP adresa.

Ostale komponente koje se mogu koristiti unutar VPC-a su internetski poveznik (eng. *Internet Gateway – IGW*) i tablice usmjeravanja (eng. *Route Tables*). Internetski poveznik je VPC komponenta koja omogućuje resursima u VPC-u da se povežu na internet. Djeluje kao most između VPC-a i interneta i omogućuje protok odlaznog prometa s VPC-a na Internet. Tablica usmjeravanja je VPC komponenta koja kontrolira protok prometa između podmreža u VPC-u, kao i između VPC-a i interneta. Tablica usmjeravanja sastoji se od skupa pravila, koja određuju kamo promet treba biti usmjeren na temelju odredišta prometa.

### Usluga AWS Elastic Kubernetes Service

AWS EKS (eng. *Elastic Kubernetes Service –* EKS) potpuno je upravljana Kubernetes usluga koja olakšava implementaciju, pokretanje i upravljanje kontejnerskim aplikacijama na AWS oblaku. Uz AWS EKS, tvrtke mogu brzo i jednostavno kreirati, konfigurirati i upravljati Kubernetes grozdovima, bez brige o temeljnoj infrastrukturi ili upravljanju infrastrukturom. Korisnik se brine o radnim čvorovima te resursima koji se nalaze na njima kao što su kapsule, servisi, imenski prostori i ostalo. Dok su glavni čvorovi skriveni i održavani od strane pružatelja usluge odnosno AWS-a.

AWS EKS integrira se s drugim AWS uslugama kao što su Amazon Elastic Container Service (ECS) i Amazon Elastic Container Registry (ECR), što olakšava izgradnju, implementaciju i upravljanje kontejnerskim aplikacijama na AWS platformi. AWS EKS je visoko skalabilan i visoko dostupan, s više zona dostupnosti i više glavnih čvorova kako bi se osigurala dostupnost aplikacija i radnih opterećenja. AWS EKS kompatibilan je s Kubernetes zajednicom otvorenog koda i podržava sve najnovije značajke i ažuriranja Kubernetesa.

Graphical user interface, application

Description automatically generated

Slika : Primjer mrežne infrastrukture

Na slici (Slika 2) je prikazan primjer AWS EKS koji je smješten unutar VPC-a. Kubernetes čvorovi su smješteni unutar privatnih podmreža (eng. *Subnet*), a svaka privatna podmreža je smještena u zasebnu zonu dostupnosti. To omogućuje zalihost i naposljetku visoku raspoloživost sustava. Unutar javnih podmreža smještena je još jedna grupa čvorova koja je direktno izložena internetu. Pomoću nje je moguće izlagati neke pristupe internetu ili može poslužiti kao bastion prema privatnim podmrežama.

### AWS komandna linija

Komunikacija s oblakom AWS moguća je putem komandne linije (eng. *Command Line Interface* – CLI), što može automatizirati i olakšati procese, a ponekad je nužna kod komunikacije s pojedinim resursima. Postoji nekoliko koraka prije nego je moguće ispravno koristiti AWS CLI. Na AWS stranicama potrebno je pronaći instalacijske upute za AWS CLI te ih slijediti. Provjera instalacije je moguća pomoću naredbe „aws –version“.

Nakon instalacije nužno je konfigurirati svoje AWS vjerodajnice. To se može učiniti pomoću naredbe "aws configure", koja će zatražiti ID ključ za pristup AWS-u (eng. *AWS access key ID*) i tajni ključ za pristup (eng. *secret access key*). Ključevi se generiraju prilikom stvaranja AWS korisnika.

Sada je moguća komunikacija s AWS-om putem komandne linije.

## Infrastruktura kao kod

Infrastruktura kao kod (eng. *Infrastructure as Code -* IaC) je metodologija za upravljanje i postavljanje infrastrukture i povezanih resursa na oblak koristeći standardizirani i automatizirani način. Uključuje korištenje konfiguracijskih datoteka i skripti za definiranje i postavljanje infrastrukture, umjesto ručnog konfiguriranja resursa putem korisničkog sučelja (eng. *User Interface* – UI) ili sučelja naredbenog retka (eng. *Command Line Interface –* CLI). Infrastruktura kao kod moćan je i učinkovit pristup upravljanju i postavljanju infrastrukture i povezanih resursa.

Jedna od ključnih prednosti korištenja IAC-a jest to što organizacijama omogućuje upravljanje svojom infrastrukturom na učinkovitiji i konzistentniji način. Korištenjem konfiguracijskih datoteka i skripti, organizacije mogu osigurati da je njihova infrastruktura dosljedno konfigurirana i postavljena u više okruženja. To pomaže smanjiti rizik od pogrešaka i povećava pouzdanost njihovih sustava.

IaC također olakšava praćenje i promjene verzija infrastrukture. Korištenjem konfiguracijskih datoteka, organizacije mogu koristiti sustave kontrole verzija (git) za praćenje i vraćanje promjena, što pomaže smanjiti rizik od neželjenih posljedica i olakšava rješavanje problema.

Uz svoje prednosti za upravljanje infrastrukturom, IaC se također može koristiti za automatizaciju postavljanja aplikacija i usluga. Korištenjem konfiguracijskih datoteka i skripti za definiranje i postavljanje aplikacijskih okruženja, organizacije mogu pojednostaviti svoj proces postavljanja i smanjiti vrijeme i trud potreban za postavljanje novih aplikacija i usluga.

Jedan od glavnih alata koji se koristi za implementaciju IaC-a je alat za upravljanje konfiguracijom kao što su Puppet, Chef ili Ansible. Ovi alati omogućuju organizacijama da definiraju svoju infrastrukturu pomoću konfiguracijskih datoteka i skripti, a zatim koriste te datoteke za postavljanje i upravljanje svojom infrastrukturom na dosljedan i automatiziran način. Korištenjem konfiguracijskih datoteka i skripti za definiranje i implementaciju infrastrukture, organizacije mogu povećati učinkovitost, pouzdanost i agilnost, dok također smanjuju rizik od pogrešaka te vrijeme i trud koji su potrebni za upravljanje i implementaciju sustava.

Postoje dva načina kojim možemo pisati kod, deklarativni i imperativni način. Deklarativni pristup definira željeno stanje sustava, uključujući resurse koji su vam potrebni i sva svojstva koja bi trebala imati, a IaC alat će to konfigurirati kako je deklarirano kodom. Deklarativni pristup također čuva popis trenutnog stanja vaših sistemskih objekata, što uklanjanje infrastrukture čini jednostavnijim za izvršiti. Imperativni pristup definira specifične naredbe potrebne za postizanje željene konfiguracije, a te se naredbe zatim moraju izvršiti ispravnim redoslijedom. Mnogi IaC alati koriste deklarativni pristup i automatski će osigurati željenu infrastrukturu. Ako napravite promjene u željenom stanju, deklarativni IaC alat će primijeniti te promjene umjesto vas. Imperativnim pristupom potrebno je unaprijed smisliti kako treba primijeniti promjene. IaC alati često mogu raditi oba pristupa, ali imaju tendenciju da daju prednost jednom pristupu nad drugim.

### Alat Terraform

Terraform je alat otvorenog koda za infrastrukturu kao kod (IaC) koji korisnicima omogućuje definiranje i pružanje infrastrukture i povezanih resursa na već poznati i automatizirani način. Razvio ga je i održava HashiCorp i naširoko ga koriste organizacije svih veličina za upravljanje svojom infrastrukturom i povezanim resursima u oblaku ili lokalno.

Jedna od ključnih značajki Terraforma je njegova sposobnost podrške za više pružatelja usluga oblaka i lokalnih okruženja. Podržava preko 100 različitih pružatelja usluga, uključujući glavne pružatelje usluga oblaka kao što su Amazon Web Services (AWS), Microsoft Azure i Google Cloud Platform, kao i lokalna okruženja kao što je VMware vSphere. To korisnicima omogućuje upravljanje i dodjelu resursa u širokom rasponu okruženja pomoću jednog alata.

Terraform koristi konfiguracijske datoteke napisane u HashiCorp Configuration Language (HCL) za definiranje infrastrukture i povezanih resursa. Ove konfiguracijske datoteke nazivaju se "Terraform moduli" i mogu se koristiti za definiranje resursa kao što su računalne instance, spremnici za pohranu i mrežne konfiguracije.

Jedna od prednosti korištenja Terraforma je njegova sposobnost upravljanja resursima na deklarativan način. To znači da korisnici definiraju željeno stanje svojih resursa u konfiguracijskim datotekama, a Terraform se brine o osiguravanju i konfiguriranju resursa kako bi odgovarali tom stanju. To pomaže osigurati da su resursi dosljedno i predvidljivo konfigurirani te olakšava praćenje i promjene verzija infrastrukture.

Terraform se također integrira s brojnim drugim alatima i uslugama, uključujući sustave kontrole verzija, cjevovode kontinuirane integracije/kontinuirane isporuke (CI/CD) i alate za praćenje i upozoravanje. Ovo omogućuje korisnicima da iskoriste Terraform u svoje postojeće procese te pomaže u pojednostavljenju upravljanja i implementacije infrastrukture i povezanih resursa.

### Tok rada Terraforma

Kako bi se koristio Terraform prvo ga je potrebno instalirati lokalno na računalu. To se može učiniti tako da se slijede upute za instalaciju na službenim stranicama Hashicorp Terraforma (<https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>).

Nakon što se Terraform instalira i provjeri valjanost putem komandnog retka, potrebno je izraditi konfiguracijske datoteke koje će definirati željene resurse. Takve datoteke pišu se u jeziku HCL posebno dizajniranom za Terraform. Svaka takva datoteka koja postoji u nekom direktoriju čini skup koji nazivamo modul. Terraform modul omogućuje stvaranje logičke apstrakcije na nekom skupu resursa. Moduli omogućuju grupiranje resursa i ponovno korištenje grupe resursa zajedno u nekom budućem projektu više puta. Postoje korijenski moduli koji je obično i glavni modul iz kojega povlačimo ostale module odnosno module djecu, čija lokacija može biti lokalna, iz službenih Terraform registara, Git repozitorija ili direktno iz HTTP URL-ova.

Nakon izrade modula moguće je dalje koristiti Terraform CLI za inicijalizaciju projekta. Uz pomoć komande „terraform init“ postiže se inicijalizacija konfiguracijskih datoteka te se preuzimaju sve potrebne datoteke i ovisnosti.

Nakon što se konfiguracija inicijalizira, moguće je dalje koristiti Terraform CLI za stvaranje i upravljanje resursima. Za stvaranje resursa koristi se naredba "terraform apply" koja će stvoriti resurse definirane u Terraform modulu. Kako bi se upravljalo i imalo više kontrole nad resursima. Koriste se naredbe kao što su „terraform validate“ i „terraform plan". Naredba „terraform validate“ služi kako bismo provjerili konfiguracijske datoteke, provjerava se je li sintaksa valjana i interno dosljedna. Dok se naredba „terraform plan“ koristi za pregled promjena koje bi se eventualno izvršile naredbom „terraform apply“. Ako je u nekom trenutku potrebno obrisati sve resurse stvorene preko modula dovoljno je pokrenuti naredbu „terraform destroy" za brisanje resursa.

Terraform sva stanja „pamti“ i zapisuje u datoteku stanja koju kreira. Datoteka je u json formatu te ju je moguće spremati lokalno ili udaljeno. Udaljeno spremanje omogućava da više korisnika istovremeno mogu nadograđivati i koristiti Terraform skripte nad istim resursima. To je moguće zato što Terraform u slučaju pokretanja komandi od strane jednog korisnika, zaključava datoteku stanja i drugi korisnici ne mogu u tom trenutku izvršavati naredbe nad tim resursima. Datoteka stanja sadrži informacije o resursima koji su kreirani, uključujući njihov tip, ID i svojstva. Prilikom pokretanja Terraform komandi, Terraform uspoređuje stanje resursa u datoteci stanja sa željenim stanjem definiranim u konfiguraciji. Ako postoje bilo kakve razlike, Terraform će stvoriti, ažurirati ili izbrisati resurse prema potrebi kako bi odgovarali željenom stanju. Moguće je koristiti Terraform CLI za izravni pregled i manipuliranje datotekom stanja. Na primjer, postoji mogućnost koristiti naredbu "terraform state show" za pregled trenutnog stanja resursa ili naredbu "terraform state rm" za brisanje resursa iz datoteke stanja.

## Postavljanje infrastrukture

### Izgradnja infrastrukture uz pomoć Terraforma

Cilj je uz pomoć Terraforma izgraditi infrastrukturu na oblaku pružatelja naziva AWS, potrebnu za razvoj aplikacija na Kubernetes grozdu. Za to su potrebni sljedeći resursi VPC s pripadajućim podmrežama i sigurnosnim grupama u kojima će se naći EKS grozd.

Text

Description automatically generated

Slika : Datoteke početnog Terraform direktorija

Na slici (Slika 3) se vide datoteke koje za nastavak imaju „.tf“. To su konfiguracijske datoteke u kojima ćemo podesiti željene parametre.

Unutar datoteke „eks-cluster.tf“ nalaze se vrijednosti koje grade Kubernetes grozd. Bitne vrijednosti koje se mogu podesiti su tip instanci koje grade grozd, minimalni i maksimalni broj instanci, sigurnosne grupe instanci, VPC i podmreže u kojima će biti stvorene, verzije Kubernetes grozda i ime grozda.

Datoteka „main.tf“ sadržava vrijednosti bitne za pružatelje modula i resurs koji generira nasumični niz. Svi pružatelji modula i njihove verzije su definirane unutar „terraform.tf“ datoteke.

U datoteci „outputs.tf“ definiraju se vrijednosti koje želimo iskoristiti kasnije. To je bitno kako bi se kasnije mogli povezati na Kubernetes grozd.

Datoteka „security-groups.tf“ je zadužena za izgradnju sigurnosnih grupa i tu se nalaze sve vrijednosti za ograničavanje i dopuštanje pristupa.

Varijable i njihove vrijednosti dodajemo u datoteku „variables.tf“, tu se dodavaju varijable koje se često izmjenjuju.

Na kraju, u datoteci „vpc.tf“ moguće je izmijeniti vrijednosti podmreža ili samog VPC-a, to jest njihove IP adrese.

Kada se podese svi željeni parametri. Dovoljno je prvo inicijalizirati projekt tako što se pozicionira unutar direktorija infrastruktura i izvrši komanda „terraform init“. Nakon toga je proizvoljno izvršiti komandu „terraform validate“ i „terraform plan“. Nužna komanda nakon inicijalizacije je „terraform apply“ koja će stvoriti resurse na oblaku nakon samo nekoliko minuta.

Graphical user interface, text, application

Description automatically generated

Slika : Direktorij nakon izvršenja naredbi

Slika (Slika 4) prikazuje direktorij „infrastruktura“ nakon izvršenja naredbi „terraform init“ i „terraform apply“. Terraform je dodao nekoliko novih datoteka i jedan direktorij. Datoteka „terraform.tfstate“ je datoteka stanja. Ostale datoteke su datoteke pružatelja modula.

Graphical user interface, application

Description automatically generated

Slika : Skica infrastrukture

Slika (Slika 5) prikazuje skicu arhitekture infrastrukture koja se podigla na AWS oblaku. Unutar VPC-a stvorile su se privatne i javne podmreže svaka u svojoj zoni dostupnosti. Unutar privatnih podmreža je EKS grozd koji ima tri EC2 instance. Dvije slabije instance su tipa „t3.small“ i imaju svoju grupu čvorova dok je treća jača instanca u zasebnoj grupi čvorova i tipa je „t3.medium“.

### Spajanje na Kubernetes grozd

Kada se sve uspješno instaliralo, potrebno je konfigurirati kubectl koji vrši komunikaciju s glavnim Kubernetes čvorovima. U tome pomažu vrijednosti koje se definiraju unutar datoteke „outputs.tf“, parametar regija „region“ i parametar imena grozda „cluster\_name“. Naredbom „aws eks --region $(terraform output -raw region) update-kubeconfig --name $(terraform output -raw cluster\_name)“ dodajemo konfiguraciju unutar „kubeconfig“ datoteke iz koje kubectl izvlači vrijednosti.

Sada je dovoljno s naredbama „kubectl cluster-info“ i „kubectl get nodes“ provjeriti ispravnost spajanja na grozd. U slučaju postojanja više konfiguracija grozdova unutar datoteke „kubeconfig“ potrebno je promijeniti kontekst. To se radi uz pomoć naredbe „kubectl config use-context CONTEXT\_NAME“.

### Postavljanje pomoćnih alata na Kubernetes uz pomoć Helm Charta

Nakon izgradnje infrastrukture i spajanja na Kubernetes grozd. Cilj je postavljanje pomoćnog alata za razvoj aplikacija ArgoCD pomoću Helma.

Prvi korak je pronalazak ArgoCD Helm chart projekta. Na github profilu argoproj moguće je pronaći repozitorij argo-helm, koji sadrži svu dokumentaciju za instalaciju alata ArgoCD na Kubernetes grozd pomoću Helma.

Drugi korak je spremanje datoteke „values.yaml“ lokalno u direktorij pod nazivom alati-helm. Dobra je praksa izmijeniti podrazumijevani naziv „values.yaml“ u „values-argo.yaml“. Slijedi podešavanje željenih parametara unutar same datoteke. Bitno je izmijeniti dio „argocdServerAdminPassword:“, jer je to lozinka koja je potrebna za prijavu u aplikaciju ArgoCD.

Idući korak je dodati prostor imena naziva „argocd“ u koji ćemo smjestiti aplikaciju. To se radi uz pomoć naredbe „kubectl create namespace argocd“.

Posljednji korak je izvršenje naredbe „helm -n argocd upgrade --install argocd argo/argo-cd -f values-argo.yaml“. Zastavica „-n“ označava u koji prostor imena će se instalirati Helm chart. Dio naredbe „upgrade –install“ zadužen je za prvu instalaciju ili ažuriranje postojeće instalacije. Nakon toga ide ime Helm charta koji će se prikazivati unutar Kubernetesa te poslije toga ime repozitorija i ime aplikacije. Zadnja stavka koja se dodaje u naredbu označava iz koje datoteke se izvlače parametri za instalaciju.

# Agilni razvoj

## Kontinuirana integracija i kontinuirana isporuka

### Alat GitHub Actions

## GitOps

### Alat ArgoCD

# Razvoj cjevovoda

# Sažetak

# Summary

# Zaključak

# Literatura

1. IETF RFC 3261 – SIP: *Session Initiation Protocol*; June 2002 (available at: <http://www.ietf.org/rfc/rfc3261.txt>)
2. “*Microservices: a definition of this new architectural term*”, Web link: <https://martinfowler.com/articles/microservices.html> , Datum pristupa: 2.4.2023.
3. *Microservices Architecture*, Web link: <https://microservices.io/> , Datum pristupa: 2.4.2023.
4. M. Lukša, “Kubernetes in Action”, Manning Publications, 2019
5. “*What are containers?*”, Web link: <https://www.netapp.com/devops-solutions/what-are-containers/> , Datum pristupa: 3.4.2023.
6. “*What is hypervisor*”, Web link: <https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor> , Datum pristupa: 3.4.2023.
7. “*What is Container orchestration*” by Middleware Team, Web link: <https://middleware.io/blog/what-is-container-orchestration/> , Datum pristupa: 5.4.2023
8. “*Kubernetes dokumentacija*”, Web link: <https://kubernetes.io/docs> , Datum pristupa: 11.4.2023.
9. Ref
10. Ref

Skraćenice

PSTN Public Switched Telephone Network

API Application Programming Interface