Kreiranje mrežne infrastrukture koristeći infrastrukturu otvorenog koda i pružatelja usluga u oblaku

Grdović, Toni

Sveučilište u Zadru

Odjel za informacijske znanosti

Stručni prijediplomski studij

Informacijske tehnologije

Toni Grdović

Kreiranje mrežne infrastrukture koristeći infrastrukturu otvorenog koda i pružatelja usluga u oblaku

Završni rad

Zadar, 2024.

Sveučilište u Zadru
Odjel za informacijske znanosti
Stručni prijediplomski studij
Informacijske tehnologije

Kreiranje mrežne infrastrukture koristeći infrastrukturu otvorenog koda i pružatelja usluga u oblaku

Završni rad

Student/ica:
Toni Grdović

Mentor/ica:
mag. ing. inf. et comm. tech. Marko Buterin

Zadar, 2024.

Izjava o akademskoj čestitosti

Ja, **Toni Grdović,** ovime izjavljujem da je moj **završni** rad pod naslovom **Kreiranje mrežne infrastrukture koristeći infrastrukturu otvorenog koda i pružatelja usluga u oblaku** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 30. rujna 2024.

Sadržaj

Sažetak

Ovaj rad istražuje izradu pouzdane, dinamički skalirajuće mrežne infrastrukture koristeći infrastrukturu kao kod (IaC) i poslužitelja u oblaku, s fokusom na Terraform i Amazon Web Servise. Rješenje je izrađeno prema industrijskim najboljim praksama, kombinirajući tehnologije temeljene na klasicima , linux poslužiteljima i tehnologiji bez poslužitelja, radi maksimalne učinkovitosti i fleksibilnosti.

Terraform se koristi kao glavni IaC alat za automatizaciju konfiguracije i upravljanja resursima u oblaku. AWS je odabran kao pružatelj usluga u oblaku zbog svojih širokih usluga i globalnog dosega. Infrastruktura je dizajnirana s dva odvojena virtualna privatna oblaka, jedan za korisnike i jedan za razvoj, kako bi se osigurala pouzdanost usluge.

Automatsko skaliranje je implementirano kako bi se alokacija resursa dinamično prilagodila potražnji, osiguravajući optimalnu pouzdanost, performanse i isplativost. Koriste se EC2 instance, temeljene na linux operacijskom sustavu za web poslužitelje i računanje bez poslužitelja poput AWS Lambda za stvaranje hibridne infrastrukture koja može podnijeti različita opterećenja. Git se koristi za verzioniranje i omogućuje suradnju u upravljanju IaC kodom.

Naglasak je na pouzdanost, skalabilnost i sigurnost uz pridržavanje industrijskog standarda za arhitekturu u oblaku. Primjenom IaC osigurava se konzistentnost i agilnost infrastrukture u razvojnim i produkcijskim okruženjima.

Ključne riječi: Infrastruktura kao kod (IaC), Terraform, AWS, Poslužitelj u oblaku, Web poslužitelj

Uvod

U doba koje je definirano digitalnom transformacijom i neprestanim razvojem tehnologije, uspostava snažne i prilagodljive mrežne infrastrukture je ključna za svaku organizaciju koja želi uspjeti u digitalnom krajoliku. Pojava infrastrukture otvorenog koda pokrenula je revoluciju u načinu na koju pristupamo dizajniranju, implementaciji i upravljanju mrežnom infrastrukturom, posebice unutar dinamičnih okolina koje pružaju davatelji usluga u oblaku. Ovaj rad bavi se istraživanjem izrade virtualne mrežne infrastrukture na Amazon Web Service koristeći (Infrastructure as Code Infrastruktura otvorenog koda). Dublje proučavanje načela, prednosti i izazova ugrađenih u ovaj inovativni pristup ima za cilj prikazati kako IaC (eng. Infrastructure as Code) omogućava organizacijama da postignu nevjerojatnu agilnost, skalabilnost, operativnu učinkovitost i sigurnost u oblikovanju svoje digitalne budućnosti. Odabirom AWS (eng. Amazon Web Services) kao svojeg radnog okruženja, cilj nam je prikazati ogroman potencijal stvaranja mrežne infrastrukture koja ne samo da se prilagođava, već se uspješno nosi s okruženjem ispunjenim stalnim promjenama.

Popis korištenih kratica

IaC - (Infrastructure as Code/Infrastruktura otvorenog koda)

DevOps - (Development and Operations/Razvoj i operacije)

AWS - (Amazon Web Services/Amazon Web usluge)

IT - (Information Technology/Informacijske tehnologije)

HCL - (HashiCorp Configuration Language/HashiCorp jezik za konfiguraciju)

JSON - (JavaScript Object Notation/JavaScript notacija objekata)

API - (Application Programming Interface/Sučelje za programiranje aplikacija)

S3 - (Simple Storage Service/Jednostavna usluga pohrane)

ID - (Identification/Identifikacija)

VPC - (Virtual Private Cloud/Virtualna privatna mreža)

IP - (Internet Protocol/Internetski protokol)

EC2 - (Elastic Compute Cloud/Elastični računalni oblak)

SSD - (Solid State Drive/Pogon na čvrstom stanju)

HDD - (Hard Disk Drive/Tvrdi disk)

AMI - (Amazon Machine Image/Amazon strojna slika)

ELB - (Elastic Load Balancer/Elastični uravnoteživač opterećenja)

HTTP - (Hypertext Transfer Protocol/Protokol za prijenos hiperteksta)

HTTPS - (Hypertext Transfer Protocol Secure/Sigurni protokol za prijenos hiperteksta)

TCP - (Transmission Control Protocol/Protokol za kontrolu prijenosa)

UDP - (User Datagram Protocol/Protokol za korisnički datagram)

CLI - (Command Line Interface/Sučelje naredbenog retka)

SDK - (Software Development Kit/Komplet za razvoj softvera)

DVCS - (Distributed Version Control System/Distribuirani sustav za kontrolu verzija)

SHA - (Secure Hash Algorithm/Sigurni algoritam sažetka)

HTML - (Hypertext Markup Language/Jezičak za označavanje hiperteksta)

BASH - (Bourne Again Shell/Bourne Again ljuska)

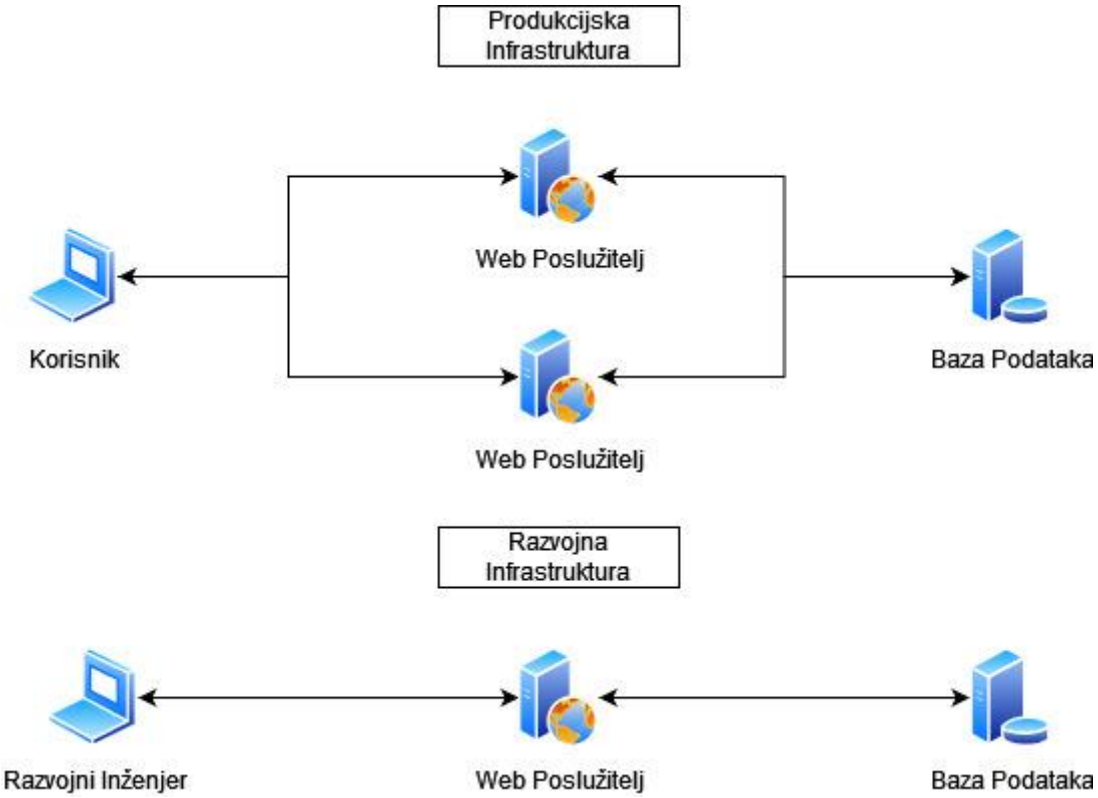
AES256 - (Advanced Encryption Standard 256-bit/Napredni standard enkripcije sa 256-bitnim ključem)

1. Ciljevi i svrha

Cilj ovog završnog rada je postaviti neophodnu infrastrukturu za jedan web servis u oblaku, koristeći industrijski prihvaćenu najbolju praksu. Za infrastrukturu su potrebna dva odvojena virtualna privatna oblaka, jedan za razvoj i testiranje, a drugi za produkciju i krajnje korisnike. Detaljno će biti obrađen svaki potreban servis za izradu infrastrukture kao što su Internet pristupnik, virtualni poslužitelji, tablice ruta, automatsko skaliranje i sigurnost sustava. Pokretanjem koda bit će moguće kompletnu infrastrukturu postaviti i ugasiti u nekoliko minuta.

2. Opis problema

Da bismo omogućili korištenje određenog web servisa, potrebna je infrastruktura koja će poslužiti tom servisu. IaC (Infrastructure as Code) nam omogućuje da tu cijelu infrastrukturu napišemo u kodu koji šaljemo skripti, a koja će nam potom izraditi infrastrukturu kakvu smo mi definirali. Infrastruktura jednog web servisa mora biti dostupna krajnjim korisnicima, ali ista ta infrastruktura mora biti dostupna i razvojnim inženjerima za testiranje i daljnji razvoj tog servisa.



Slika 1 - Osnovna Arhitektura Web

U višeslojnoj arhitekturi svoje funkcije proizvoda dijelite na više slojeva, kao što su prezentacija, posao, baza podataka i usluge, tako da se svaki sloj može neovisno

implementirati i skalirati.[1] Najbolja praksa je izrada dva odvojena privatna oblaka s međusobnom komunikacijom, gdje se promjene izvršavaju prvo na razvojnom oblaku, a zatim nakon testiranja prebacuju u rad na produkcijskom oblaku. Radi smanjenja troškova infrastrukture potrebno je postaviti automatsko skaliranje web poslužitelja, koji će resurse paliti i gasiti ovisno o broju korisnika našeg servisa u tom trenutku. Potrebno je osigurati svu infrastrukturu od neželjenih upada, kao i pratiti unutarnju i vanjsku komunikaciju svih poslužitelja. Na kraju, treba omogućiti da se kompletna infrastruktura može ponovno izraditi u slučaju katastrofe i gubitka podataka na lokaciji poslužitelja.

3. Teorijski Okvir uz problematiku IaC

IaC je način automatizacije dodjele resursa na informacijskoj infrastrukturi koristeći određeni jezik za skriptiranje. Automatizacijom eliminiramo potrebu za ručnim postavljanjem poslužitelja, operacijskih sustava, baza, pohrane i ostalih elemenata informacijske infrastrukture. U današnjem dobu agilnog razvoja aplikacija, kod većih tvrtki nastala je potreba da se izmjene stotine aplikacija dnevno. Potrebna je infrastruktura koja može pratiti te promjene te skalirati infrastrukturu ovisno o potrebama razvojnih inženjera i krajnjih korisnika. Sve to je potrebno na neki način automatizirati, što nam omogućuje IaC.

IaC je također bitna DevOps (spoj razvoj i operacije) praksa, nezamjenjiva u životnom ciklusu isporuke softvera s konkurentnim tempom. Omogućuje DevOps timovima brzu izradu i verziju infrastrukture na isti način na koji verziraju izvorni kod i praćenje tih verzija kako bi se izbjegla nedosljednost među IT (informacijske tehnologije) okruženjima koja mogu dovesti do ozbiljnih problema tijekom implementacije.[2]

DevOps je skup praksi koje kombiniraju razvoj softvera i operativni dio informacijskih tehnologija kako bi se skratio ciklus razvoja informacijskog sustava. DevOps je baziran na automatizaciji, a IaC omogućuje da sva infrastruktura bude automatizirana zbog brzine, verzioniranja radi sigurnosti, da se pritom prate promjene i izbjegnu ljudske greške tijekom izrade infrastrukture.

Dodjela i postavljanje IT resursa je dugotrajno i skupo. Postavljanje fizičkog hardvera, instalacija i konfiguracija operacijskih sustava, izrada fizičke mreže i postavljanje parametara te postavljanje pohrane je ogroman posao. Virtualizacija i infrastruktura u oblaku omogućila je izradu kompletne virtualne infrastrukture na već postojećoj fizičkoj

[1] Saurabh Shrivastava. Solutions Architect's Handbook - Second Edition. January 2022.
[2] https://www.ibm.com/cloud/learn/infrastructure-as-code

infrastrukturi. Koristeći IaC moguće je još lakše izraditi virtualnu infrastrukturu jer sav ručni posao obavljaju automatizirane skripte. IaC omogućuje da se kompletna infrastruktura definira u kodu, i jednom naredbom izvrše sve operacije potrebne za izmjenu ili postavljanje infrastrukture.

3.1. Terraform

Terraform je alat za IaC koji vam omogućuje da sigurno i učinkovito gradite, mijenjate i verzionirate resurse u oblaku i on-premu.[3] Terraform je alat otvorenog koda za izradu infrastrukture u oblaku. Koristan je za automatizaciju, upravljanje postavljanjem i konfiguracijom resursa u oblaku. Omogućuje definiranje i deklariranje infrastrukture u jeziku visoke razine, a zatim postavljanje i upravljanje tim resursima na različitim poslužiteljima u oblaku ili lokalnoj mrežnoj infrastrukturi. Terraform je izradio HashiCorp 2014. godine koristeći programski jezik GO. HashiCorp je poznat po nizu alata za automatizaciju infrastrukture.

Za kod se koristi HCL (eng. HashiCorp Configuration Language) kao deklarativni jezik konfiguracije, dosta sličan JSON-u (eng. JavaScript Object Notation). U tekstu zvanom *Terraform konfiguracija* potrebno je specificirati koje resurse treba i kako ih treba konfigurirati da bi ih Terraform mogao stvoriti u oblaku. Možete koristiti ovu binarnu datoteku za implementaciju infrastrukture sa svog prijenosnog računala, poslužitelja za izgradnju ili bilo kojeg drugog računala, a ne trebate pokretati nikakvu dodatnu infrastrukturu da biste to ostvarili. To je zato što Terraform u pozadini automatski upućuje API (eng. Application Programming Interface) pozive u vaše ime na jednog ili više pružatelja usluga, kao što su AWS, Azure, Google Cloud, DigitalOcean, OpenStack i drugi.[4]

Terraform podržava različite poslužitelje oblaka, infrastrukturne platforme i usluge te konfiguraciju s njima radimo pomoću davatelja usluga. Svaki davatelj usluga je odgovoran za upravljanje resursima i konfiguracijama za svoju specifičnu platformu. Tako postoje davatelji usluga za AWS, Microsoft Azure, Google Cloud Platform i još mnoge druge.

Resursi su pojedinačne komponente infrastrukture kojom želite upravljati. Definiraju se u Terraform konfiguraciji i povezuju se s određenim davateljem usluga. Primjeri resursa na AWS-u su instance, virtualni privatni oblaci ili DynamoDB tablice. Resurse možemo definirati pomoću varijabli. Varijable koristimo u Terraform konfiguraciji kako bi postala dinamična i ponovno upotrebljiva. Dinamična alokacija vrijednosti pomoću varijabli nam

[3] https://developer.hashicorp.com/terraform/intro
[4] Yevgeniy Brikman. Terraform: Up and Running, 3rd Edition. September 2022

daje mogućnost da u konfiguraciji resursa dodamo prethodno nepoznate vrijednosti koje će Terraform popuniti za vrijeme primjene na infrastrukturi. Druga mogućnost varijabli je ponavljanje, što nam omogućuje da vrijednost jednom definiramo i koristimo za više resursa.

Moduli su primjer na kojem možemo unaprijed definirati konfiguraciju cijelog jednog resursa koja će se više puta ponavljati u konfiguraciji infrastrukture. Module možemo sami definirati ili preuzeti s HashiCorp-ovog registra za module, gdje se nalaze HashiCorp-ovi moduli, kao i moduli kreirani od strane zajednice otvorenog koda.

Stanje infrastrukture se prati u Terraformovoj datoteci stanja. Svaka promjena inicirana od strane Terraforma se bilježi u toj datoteci. Stanje se pod zadanim pohranjuje u "terraform.tfstate" datoteci, a poželjno je da se sprema negdje gdje se može dijeliti s timom radi verzioniranja. Pomoću datoteke stanja Terraform određuje koje promjene treba napraviti na infrastrukturi. Svrha stanja je pohraniti veze između objekata u udaljenom sustavu i resursa deklariranih u konfiguraciji. Kad Terraform stvori udaljene objekte iz konfiguracije, uspoređuje ih s postojećim stanjem na infrastrukturi i na temelju toga određuje potrebu za ažuriranjem ili brisanjem objekata kao odgovor na promjene u konfiguraciji.

"Terraform init" je temeljna naredba za početak rada s Terraform konfiguracijom. Ova naredba inicijalizira projekt tako da preuzima i instalira potrebne datoteke i providere za našu konfiguraciju. Nakon toga postavlja konfiguraciju udaljenog spremišta i preuzima vanjske module navedene u našoj konfiguraciji. Generira se i datoteka zaključavanja (lock file) koja bilježi verzije providera i modula kako bi se osigurala dosljednost u različitim okruženjima tijekom suradnje s drugima. Ovo je obično jedan od prvih koraka za početak rada s Terraformom, nakon čega je spreman za korištenje drugih naredbi kao što su plan i apply.

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.66"...
- Finding latest version of hashicorp/http...
- Installing hashicorp/aws v5.66.0...
- Installed hashicorp/aws v5.66.0 (signed by HashiCorp)
- Installing hashicorp/http v3.4.4...
- Installed hashicorp/http v3.4.4 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

Slika 2 - Terraform Inicijalizacija

„Terraform plan" je naredba u Terraformu koja omogućuje planiranje promjena koje će se dogoditi na našoj infrastrukturi prije nego što se te promjene stvarno primijene. Ova naredba radi analizu naše Terraform konfiguracije i trenutnog stanja infrastrukture kako bi utvrdila koje resurse treba dodati, ažurirati ili ukloniti. Tijekom izvođenja naredbe generira se detaljni izvještaj o predloženim promjenama, uključujući što treba dodati, ažurirati ili ukloniti te plan akcija koje će se poduzeti. Planiranje pomaže developerima da procijene kakav će utjecaj planirane promjene imati na infrastrukturu kako bi se izbjegle neželjene posljedice. Osim toga, naredba „plan" provjerava jesu li svi resursi i veze u konfiguraciji ispravno definirani kako bi se osigurala ispravnost konfiguracije prije primjene, čime se smanjuje rizik od nepoželjnih posljedica tijekom izvođenja promjena na infrastrukturi.

„Terraform apply" je naredba za primjenu promjena na infrastrukturi prema planu koji generira naredba „plan". Ova naredba izravno utječe na infrastrukturu i izvršava promjene na temelju naše Terraform konfiguracije. Ovaj korak rezultira stvaranjem novih resursa, ažuriranjem postojećih ili brisanjem nepotrebnih resursa prema našoj konfiguraciji. U slučaju da planiranje nije izvršeno prije izvršenja naredbe „apply", Terraform će prvo izvršiti planiranje, generirati plan po kojem će izvršavati promjene i zatim zatražiti konačnu potvrdu prije izvođenja promjena na infrastrukturi.

„Terraform destroy" je naredba koja se koristi za uništavanje infrastrukture koju smo prethodno definirali u Terraform konfiguraciji. Ova naredba automatski detektira resurse u konfiguraciji i zatim ih isključuje na ciljanoj infrastrukturi. Važno je pažljivo koristiti naredbu „destroy", jer će se ukloniti svi resursi definirani u konfiguraciji, što može rezultirati gubitkom podataka i nepovratnim promjenama na infrastrukturi. Kako bismo izbjegli

slučajno brisanje, Terraform traži potvrdu prije nego započne proces uništavanja. Nakon izvođenja, infrastruktura će biti vraćena u stanje u kojem je bila prije nego što se Terraform koristio za upravljanje njome.

Naredbu „terraform fmt" koristimo za oblikovanje datoteke s Terraform konfiguracijom. Ona primjenjuje dosljedan i standardiziran stil na našu konfiguraciju, omogućavajući da naš kod bude dobro organiziran i čitljiv. Pokretanjem naredbe automatski se prilagođavaju razmaci, uvlačenja i drugi aspekti oblikovanja, čime se olakšava održavanje i suradnja s drugim članovima tima.

3.2. AWS

Računalstvo u oblaku model je za omogućavanje sveprisutnog, praktičnog mrežnog pristupa na zahtjev zajedničkom skupu konfigurabilnih računalnih resursa (npr. mreže, poslužitelji, pohrana, aplikacije i usluge) koji se mogu brzo osigurati i osloboditi uz minimalan napor upravljanja ili interakcija pružatelja usluga.[5] AWS je Amazonova platforma za računalstvo u oblaku koja nudi širok spektar raznih usluga uključujući procesorsku snagu, pohranu, baze podataka, strojno učenje, analitiku, mrežno povezivanje i još mnogo toga. Omogućuje tvrtkama i pojedincima lako skaliranje i upravljanje IT infrastrukturom i aplikacijama na fleksibilan i ekonomičan način. S podatkovnim centrima koji su smješteni diljem svijeta, AWS nudi globalnu dostupnost i snažne sigurnosne značajke, što ga čini vrhunskim izborom za posluživanje web stranica, web aplikacija, izvođenje kompleksnih računanja i upravljanje podatcima u oblaku.

3.2.1. Amazon S3

Amazon Simple Storage Service, ili skraćeno Amazon S3 (eng. Simple Storage Service) je usluga pohrane podataka u oblaku. S3 je brz, siguran i skalabilan način pohrane podataka. Amazon je s uslugom S3 započeo 2006. godine te je među prvim uslugama koje su nudili.

Dizajnirano za izdržljivost 99,999999999% objekata u više zona dostupnosti[6] ili po Amazonu "jedanaest devetki" izdržljivost za objekte pohranjene u S3 usluzi, što znači da je izrazito mala vjerojatnost da će se neki objekt izgubiti ili biti kompromitiran. S3 podržava enkripciju i integraciju s ostalim AWS uslugama te je moguće pratiti sve podatke s uslugama nadzora kao što su CloudTrail i CloudWatch.

[5] https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf
[6] https://aws.amazon.com/s3/storage-classes/

Prijenos podataka u S3 može se obaviti na nekoliko različitih načina. Direktno putem interneta, podatci se mogu poslati putem API konzole ili preko web sučelja. S3 pruža i uslugu AWS Direct Connect koja stvara direktnu privatnu vezu između Amazonove infrastrukture i vašeg vlastitog data centra. Postoji i usluga AWS Snowball gdje se šalje fizički uređaj koji sadrži nekoliko tvrdih diskova na koji se prebacuju podatci, a potom ga se vraća u Amazon dana centar za prijenos u željeni privatni oblak.

S3 koriste organizacije različitih veličina, od malih poduzeća s web stranicama do velikih korporacija za obradu podataka. Uobičajeni slučajevi korištenja S3 su: pohrana podataka, arhiviranje podataka, isporuka softvera, sigurnosne kopije, hosting aplikacija, mobilne aplikacije, hosting multimedije kao što su slike, videozapisi ili glazbene datoteke.

Za pohranu podataka u S3, Amazon koristi objekte. Svaki objekt pohranjuje se kao datoteka s meta podacima. Svaki objekt dobiva svoj ID (Identifikacijski) broj i to omogućuje direktan pristup podatcima. Koristeći ID možemo putem REST API-a doći do podataka. Ograničenje veličine jedne datoteke je 5 terabajta, što je sasvim dovoljno za većinu slučajeva.

Amazon S3 sve objekte sprema u takozvane Kante (tzv. Buckets). Kante su globalna usluga i svaka kanta na svijetu mora imati svoj jedinstven naziv. Nemoguće je da dvije kante imaju isto ime bilo gdje na svijetu, neovisno o korisniku ili regiji. Kante mogu biti javne ili privatne. Javne kante omogućuju korisnicima da dohvate podatke bez autentifikacije na Amazonu, kao u slučaju web hostinga. Privatne kante su zaštićene unutar korisnikovog Amazon računa te radi povećane sigurnosti mogu biti šifrirane, tako da u slučaju da netko provali u Amazonov data centar i kopira vaše podatke, neće ih moći pročitati.

3.2.2. Amazon VPC

Amazon VPC (eng. Virtual Private Cloud) ili virtualni privatni oblak jedna je od glavnih usluga na Amazon Web Services platformi. VPC je virtualno logički izolirani privatni oblak unutar samog Amazonova oblaka. Unutar virtualnog oblaka, korisnik ima potpunu kontrolu nad virtualnim mrežnim okruženjem. Korisnik postavlja vlastiti raspon IP (eng. Internet Protocol) adresa, može stvarati podmreže te postavlja i konfigurira tablice ruta i prolaze. VPC omogućuje stvaranje vlastite mreže u oblaku koju je moguće direktno povezati s postojećom fizičkom infrastrukturom. VPC pruža kontrolu pristupa mreži, omogućava formiranje javno dostupnih mreža koje imaju pristup internetu te privatnih mreža za aplikacije u pozadini. Mrežna sigurnost se postavlja na više slojeva kako biste mogli kontrolirati pristup korisnicima i virtualnim instancama.

Amazon VPC omogućuje izgradnju informacijske infrastrukture bez potrebe za vlastitim hardverom ili fizičkim podatkovnim centrom. Unutar VPC-a nalaze se svi ostali resursi potrebni za rad. Sve se virtualno postavlja na hardverskoj infrastrukturi Amazona. Postavljanje servera, što je nekada bio posao koji je zahtijevao nekoliko dana i fizičkog hardvera, unutar Amazon VPC-a se može postaviti u roku od nekoliko minuta. Prilikom korištenja VPC-a, Amazon se brine za svu fizičku infrastrukturu u pozadini, kao i za sigurnost i dostupnost te infrastrukture.

3.2.2.1. Regije

Regije na AWS-u su geografski odvojene cjeline. Svaka se regija sastoji od najmanje tri zone dostupnosti. Ovakva arhitektura pruža redundantnost i visoku dostupnost aplikacije. Regije su ključan dio strategije oporavka od katastrofe i visoke dostupnosti putem geografske raznolikosti i redundancije ključnih servisa. AWS Cloud obuhvaća 102 zone dostupnosti u 32 geografske regije diljem svijeta, s najavljenim planovima za još 12 zona dostupnosti i još 4 AWS regije u Kanadi, Maleziji, Novom Zelandu i Tajlandu.[7]

Slika 3 - AWS Global Infrastructure (https://aws.amazon.com/about-aws/global-infrastructure/)

3.2.2.2. Availability Zone

Dostupne zone su izolirani podatkovni centri unutar AWS regije. Dizajnirane su za pružanje redundancije i visoke dostupnosti. Svaka dostupna zona ima svoju vlastitu opskrbu energijom, hlađenje i mrežnu infrastrukturu, što osigurava neovisno funkcioniranje. Distribuiranje resursa preko više dostupnih zona gradi otporne i pouzdane aplikacije. Unutar

[7] https://aws.amazon.com/about-aws/global-infrastructure/

regije, dostupne zone se povezuju niskom latencijom i visokim propusnim opsegom veze koje omogućavaju rad računalnih skupova unutar jedne regije.

3.2.2.3. Podmreže

Podmreže su segmenti virtualne mreže unutar oblaka koji omogućavaju izolaciju i organizaciju resursa. Svaka podmreža obuhvaća određeni raspon IP adresa unutar svog virtualnog privatnog oblaka. Podmreže nam omogućavaju izradu složene i sigurne mreže tako da se grupiraju i segmentiraju resursi za specifične namjene. Svaka podmreža može imati svoja pravila kako bi osigurala usklađenost i sigurnost te omogućavala različitim resursima u oblaku komuniciranje ili ograničavanje komunikacije prema potrebi.

3.2.2.4. Internet pristupnik

U kontekstu AWS usluga, internet pristupnik je usluga koja omogućava komunikaciju između virtualnih privatnih oblaka i globalne mreže, odnosno interneta. Internet pristupnik je bitna komponenta koja omogućava resursima unutar VPC-a da razmjenjuju podatke s resursima izvan oblaka. Ova funkcionalnost je ključna za resurse koji moraju biti javno dostupni, kao što su web aplikacije, a istovremeno omogućava kontrolirani pristup i primjenu sigurnosnih politika za komunikaciju sa svijetom izvan oblaka.

3.2.2.5. Tablica usmjeravanja

Tablica usmjeravanja je komponenta virtualnog privatnog oblaka koja igra ključnu ulogu u upravljanju prometom između resursa unutar VPC-a i prema vanjskim mrežama. Svaki VPC ima svoju tablicu usmjeravanja koja definira kako će se promet usmjeriti između različitih mrežnih segmenata unutar oblaka. Tablica sadrži pravila koja definiraju kamo će se promet usmjeriti, uključujući destinacijske IP adrese i putanje prema drugim VPC-ima, internetu ili drugim resursima unutar istog VPC-a. Tablice usmjeravanja omogućuju kontrolu nad mrežnim prometom u oblaku prema specifičnim zahtjevima aplikacije i sigurnosnim politikama.

3.2.3. Amazon EC2

Amazon EC2 (eng. Elastic Compute Cloud) je usluga koja nam omogućuje korištenje virtualnih poslužitelja nazvanih „instance”. Kod EC2 koristimo već postavljen hardver od Amazona u pozadini, a na nama je da instaliramo potrebni softver i postavimo svoju aplikaciju. Plaćamo samo kapacitet koji postavimo, što nam omogućuje povećavanje i smanjivanje kapaciteta prema potrebi, što s fizičkim hardverom nije moguće. EC2 je

podijeljen na više lokacija što nam omogućuje redundanciju i smanjenje latencije kod krajnjih korisnika.

3.2.3.1. Instance

Instance su virtualni poslužitelji na Amazonovoj platformi. Instance se dijele na različite tipove. Tipovi instanci ovise o količini procesorske snage, memorije, prostoru za pohranu, mrežnim adapterima i grafičkim karticama. Oznaka svake instance dijeli se na četiri dijela. Prvo slovo je oznaka tipa instance, zatim broj koji označava generaciju instance, nakon toga dolazi dodatna specijalna oznaka, i nakon točke veličina instance. Primjerice, M6g.large označava M za "general purpose", šesta generacija, "g" jer je Graviton procesor, i "large" što označava veličinu od 4 vCPU i 8 GiB RAM-a.

Svaka vrsta instance nudi različite mogućnosti računanja, memorije i pohrane te je grupirana u obitelj instanci na temelju tih mogućnosti.[8] Instance se mogu grupirati u sljedeće podgrupe:

General purpose – Balans između procesora, memorije i mrežnih resursa. General purpose instance su namijenjene širokom spektru upotrebe. Osim specijalnih instanci, najčešće varijante dolaze iz M ili T kategorija instanci. M instance postoje već u šest generacija, dostupne su u Graviton, Intel i AMD varijantama ovisno o procesoru u pozadini, i služe za aplikacije s ravnomjernom raspodjelom radnog opterećenja. T instance su jeftinija varijanta M instanci i namijenjene su situacijama gdje je radno opterećenje neravnomjerno tijekom dana. T instance, kao i M, dolaze s tri različite procesorske varijante, ali razlika je u tome što T instance imaju postavljeno bazno opterećenje izraženo u postotku procesorske snage. Dok rade ispod tog postotka, stvaraju takozvane kredite, koje kasnije koriste da bi radile iznad baznog opterećenja.

Compute optimized – ili C varijanta su instance koje imaju veći omjer CPU prema memoriji. Idealne su za aplikacije koje koriste više procesora kao što su obrada slike, web servisi s visokim performansama, znanstveni izračuni i serveri za video igre.

Memory optimized – instance, najčešće R varijante, su instance koje imaju veći omjer radne memorije prema procesoru. Ove instance su idealne za obradu većih količina podataka ili velikih skupova podataka. Postoje i X, Z i U varijante koje imaju ogromne količine radne memorije, kao što je u-24tb1.metal koja sadrži 24,576 GiB radne memorije na jednoj instanci.

[8] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

Accelerated Computing –su instance s nekom vrstom hardverskog akceleratora, bilo to grafička kartica, video transkoder ili procesor za umjetnu inteligenciju. Namijenjene su za strojno učenje i procesiranje ogromnih količina podataka. Dijele se na P, G, V i F varijante ovisno o akceleratoru na instanci.

Storage optimized – instance su namijenjene za rad s velikim količinama pohranjenih podataka. Svaka ima svoj vlastiti podatkovni hardver, bilo SSD (eng. Solid State Drive) ili HDD (eng. Hard Disk Drive) spojen na instanci, što omogućuje brži zapis u usporedbi s mrežnim diskovima, kao i na prethodnim obiteljima.

3.2.3.2. AMI

Amazon Machine Image, skraćeno AMI, je dio EC2 servisa na kojem pohranjujemo osnovnu konfiguraciju za EC2 instancu. AMI je slika operativnog sustava koju koristimo za stvaranje instanci. AMI se dijele na javne i privatne. Javni AMI su dostupni svima, a uključuju već gotove Amazon AMI instalacije koje održava Amazon, pružajući dobru početnu konfiguraciju, te Community AMI koje održavaju korisnici AWS sustava. Amazon AMI su različiti ovisno o operativnom sustavu koji je instaliran na njima. Neke AMI su besplatne, kao što su Amazon Linux i Ubuntu, dok je za druge potrebno platiti licencu, kao što je slučaj s Windows Serverom, MacOS-om ili Oracleom. Privatni AMI su instalacije koje sami kreiramo i dostupne su samo na našem AWS računu. Obično se koristi postojeći AMI, instalira na instancu, konfigurira sa svim potrebnim postavkama i softverom, zatim se instanca zaustavi i stvori slika njenog diska kako bi se kreirao novi AMI.

3.2.3.3. Grupe automatskog skaliranja

Grupe automatskog skaliranja su AWS usluga koja omogućava dinamičko prilagođavanje kapaciteta resursa kako bi se automatski nosile s promjenjivim opterećenjima aplikacije. Ova grupa omogućuje automatsko povećanje ili smanjenje broja instanci ovisno o unaprijed definiranim metrikama. To pomaže u održavanju stabilnosti, performansi i dostupnosti aplikacije jer se kapacitet automatski prilagođava u stvarnom vremenu. Grupe omogućavaju brzu zamjenu neispravnih instanci, čime osiguravaju pouzdanost cijelog sustava.

3.2.3.4. Elastični balanser opterećenja

Visoka dostupnost i otpornost na pogreške bitne su značajke svake suvremene aplikacije. Bitno je raspodijeliti opterećenje i također izbjeći pojedinačne točke kvara kada je u pitanju

pristup aplikaciji.[9] „Elastic Load Balancer“ je upravljana usluga koja omogućuje distribuciju mrežnog prometa prema više Amazon EC2 instanci, kontejnerima ili drugim resursima unutar AWS okoline. ELB (eng. Elastic Load Balancer) je ključan za poboljšanje dostupnosti, skalabilnosti i otpornosti na greške aplikacija koje se hostaju u AWS oblaku.
ELB nudi više algoritama za balansiranje opterećenja, od klasičnog (round-robin) koji kreće od prvog do zadnjeg pa ponovo u krug, do odabira instance s najmanjim opterećenjem ili najbržim putem. Pomoću ELB možemo automatski skalirati svoj kapacitet ovisno o promjeni prometa jer se ulazna putanja ne mijenja, a ELB će automatski preusmjeravati promet na nove instance koje se pokreću, osiguravajući optimalnu dostupnost. ELB kontinuirano provjerava zdravlje pozadinskih instanci periodičkim slanjem jednostavnih HTTP/HTTPS (eng. Hypertext Transfer Protocol/ Hypertext Transfer Protocol Secure) zahtjeva i procjenom njihovih odgovora. Provjera zdravlja se koristi kako bi se utvrdilo jesu li pozadinske instance u ispravnom stanju i sposobne obrađivati promet. Ako instance ne prođu provjeru zdravlja, ELB prestaje usmjeravati promet prema toj instanci dok ponovno ne postane zdrava ili dok je druga ne zamijeni.

Postoje tri vrste ELB na AWS:

1. ALB (Aplikacijski balanser opterećenja) je idealan za usmjeravanje HTTP/HTTPS prometa na temelju sadržaja na aplikacijskoj razini. Koristi se za mikroservise i arhitekture temeljene na kontejnerima.
2. NLB (Mrežni balanser opterećenja) dizajniran je za ultra visoke performanse i nisku latenciju prometa. NLB upravlja s TCP(eng. Transmission Control Protocol) i UDP (eng. User Datagram Protocol) prometom i pogodan je za visoko performantska radna opterećenja, kao što su video igre i aplikacije u stvarnom vremenu.
3. CLB (Klasični balanser opterećenja) je originalni balanser na AWS, nudi osnovne mogućnosti balansiranja i često se zamjenjuje novijim balanserima za složenije i zahtjevnije aplikacije.

3.2.4. IAM

Amazon Identity and Access Management je sustav kontrole pristupa za AWS. IAM nam omogućava kontrolu načina autentikacije i pristupa svim korisnicima i servisima na našim AWS resursima. IAM je ključan za održavanje sigurnosti naše infrastrukture.

[9] Mitesh Soni. Practical AWS Networking: Build and manage complex networks using services such as Amazon VPC, Elastic Load Balancing, Direct Connect, and Amazon Route 53. January 2018

IAM nam omogućava centralno upravljanje pristupom našim AWS resursima unutar cijele organizacije. Kada pošaljete zahtjev AWS API-ju, IAM provjerava vaš identitet i provjerava imate li dopuštenje za izvođenje radnje. IAM kontrolira tko (autentifikacija) može učiniti što (autorizacija) na vašem AWS računu.[10]

3.2.4.1. User

Korisnici su osobe ili entiteti unutar naše organizacije kojima je potreban pristup određenim AWS resursima. Svaki AWS korisnik ima jedinstveno ime i svoje podatke za prijavu, bilo da se radi o lozinki, uređaju s više faktora autentikacije ili AWS ključu za pristup.

3.2.4.2. Groups

Grupe su kolekcija korisničkih računa. Grupe nam omogućuju dodjelu ovlasti za više korisnika koji imaju slične zahtjeve za pristup. Umjesto konfiguriranja ovlasti za svakog korisnika ili servis pojedinačno, možemo ih dodavati u grupe i upravljati njima kao skupinom.

3.2.4.3. Role

Uloge su slični entiteti kao korisnici, samo što nisu definirani za određenu osobu. Uloge nam omogućuju dodjelu prava pristupa AWS uslugama ili eksternim entitetima, poput korisnika s drugog računa. Uloge se privremeno preuzimaju i možemo dalje definirati prava koja daju.

3.2.4.4. Politike

Politike su dokumenti u JSON formatu koji definiraju pristup i ovlasti. U politikama definiramo što je dozvoljeno ili zabranjeno za određene resurse na AWS. Politike se zatim mogu dodijeliti korisnicima, grupama ili ulogama. AWS nudi skup predefinirani politika za korištenje, ali radi sigurnosti bolje je samostalno definirati politike za svaki zahtjev.

3.2.4.5. Pristupni ključevi

Pristupni ključevi se koriste od strane korisnika i AWS usluga za izvođenje API poziva prema AWS resursima. Sastoje se od dva dijela, identifikacijskog ključa i tajnog pristupnog ključa. Identifikacijski ključ možemo koristiti u politikama kako bismo ograničili prava samo na taj ključ. Korisnici pristupnih ključeva koriste ih s AWS CLI (eng. Command

[10] Andreas Wittig. Amazon Web Services in Action, Third Edition. May 2023

Line Interface), SDK-ovima (eng. Software Development Kit) i drugim alatima kako bi komunicirali s AWS uslugama.

3.2.5. Lambda

Amazonov servis AWS Lambda omogućuje pokretanje koda bez potrebe za pružanjem ili upravljanjem poslužiteljima. Kod se izvršava kao odgovor na različite događaje bez potrebe za upravljanjem poslužiteljima koristeći takozvano računalstvo bez poslužitelja. Vi učitavate svoj kod, definirate okidače, a Lambda servis automatski skalira i upravlja infrastrukturom za izvođenje koda. Na ovaj način je Lambda isplativ i učinkovit način za izradu i implementaciju aplikacija koje reagiraju na događaje poput API-ja, prijenosa datoteka ili promjena na bazi podataka. Kod Lambde se plaća samo vrijeme izvođenja koda, što je pogodno za aplikacije koje ne moraju konstantno biti dostupne.

Lambda podržava više jezika putem korištenja runtime-ova, kao što su Java, Go, PowerShell, Node.js, C#, Python i Ruby, i API-jem podržava i druge tradicionalne jezike.[11] Funkcija je temeljna izvršna jedinica u AWS Lambda. To je komad koda koji obavlja zadatak. Izvor događaja pokreće izvođenje funkcija, što može biti različite usluge na AWS-u poput S3, DynamoDB, API Gateway ili slično. Okidač definira kako se izvor događaja povezuje s funkcijom u Lambdi. Lambda se nakon izvođenja gasi i ponovo pokreće po potrebi, Lambde se mogu izvršavati paralelno i mogu se povezivati s drugim Lambda funkcijama. Najveća prednost je da se za sve brine Amazonov servis u pozadini.

3.2.6. Dynamodb

Amazon DynamoDB je potpuno upravljana NoSQL usluga baze podataka koja pruža brzu i predvidljivu izvedbu uz besprijekornu skalabilnost. Za aplikacije gdje se količina podataka rapidno povećava, DynamoDB nudi skalabilnost, visoko performansno i pouzdano rješenje za besprijekoran i nisko latentni pristup podatcima.[12]

Ključna karakteristika DynamoDB-a je njegova serverska arhitektura. DynamoDB je još jedan od servisa bez poslužitelja, što znači da korisnici ne moraju brinuti o osiguranju ili upravljanju poslužiteljima baze podataka. Ovaj aspekt pojednostavljuje operativne zadatke koji su tradicionalno povezani s administracijom baza podataka i čini ga privlačnim izborom za developere i organizacije koje žele usredotočiti se na izgradnju aplikacija umjesto održavanju baza podataka.

[11] https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html
[12] https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/

DynamoDB je dizajniran kako bi imao visoku dostupnost i otpornost na greške. Radi u više dostupnih zona unutar regije, što osigurava da će podatci ostati dostupni čak i u slučaju hardverskog kvara ili drugih prekida. Skalabilnost je još jedna značajka DynamoDB-a jer podržava automatsko skaliranje te se lako nosi s opterećenjima za čitanje i pisanje koje variraju od nekoliko zahtjeva do nekoliko milijuna zahtjeva po sekundi. Ovo omogućuje da se, kako opterećenje na aplikaciji raste s povećanjem broja korisnika, tako i baza u pozadini povećava bez potrebe za ručnim intervencijama. S ovim modelom skaliranja kapaciteta, ovisno o opterećenju baze, možemo biti ekonomičniji za nepredvidiva radna opterećenja jer nema potrebe držati visok kapacitet aktivno za vrijeme niskog opterećenja.

Podatkovni model DynamoDB-a temelji se na tablicama, stavkama i atributima. Tablice su zbirke podataka, stavke su pojedinačni zapisi unutar tih tablica, a atributi su polja podataka unutar tih stavki. DynamoDB podržava strukturirane i polustrukturirane podatke i ne zahtijeva fiksni okvir za svaku tablicu što omogućava developerima da razvijaju svoje modele podataka kako se zahtjevi aplikacije mijenjaju, povećavajući agilnost u razvojnom procesu. DynamoDB se besprijekorno integrira s drugim uslugama na AWS-u i podržava kontrolu s preciznim pravilima u AWS IAM, što olakšava spajanje s drugim servisima i povećava sigurnost podataka unutar aplikacije.

3.3. GIT

Git je temeljni alat za moderni razvoj softvera, omogućavajući učinkovitu kontrolu verzija, suradnju i upravljanje kodom. Linux jezgra je softverski projekt otvorenog koda prilično velikog opsega. Tijekom ranih godina održavanja Linux jezgre (1991. – 2002.), promjene softvera prosljeđivane su kao zakrpe i arhivirane datoteke. Godine 2002. projekt jezgre Linuxa počeo je koristiti vlasnički DVCS (eng. Distributed Version Control System) nazvan BitKeeper. Godine 2005. prekinut je odnos između zajednice koja je razvila Linux jezgru i komercijalne tvrtke koja je razvila BitKeeper, a status besplatnog alata je opozvan. To je potaknulo razvojnu zajednicu Linuxa (a posebno Linusa Torvaldsa, tvorca Linuxa) da razviju vlastiti alat na temelju nekih lekcija koje su naučili koristeći BitKeeper.[13] Git je distribuirani sustav za kontrolu verzija, namijenjen je učinkovitom upravljanju i praćenju promjena u izvornom kodu i drugim datotekama temeljenim na tekstu. Git je poznat po brzini, sigurnosti, fleksibilnosti i sposobnosti upravljanja projektima bilo koje veličine.

Git radi na temelju jednostavnog principa prateći promjene u datotekama i direktorijima u repozitoriju tijekom vremena. Svaki programer koji radi na projektu kod sebe

[13] Scott Chacon, Ben Straub. ProGit 2nd Edition. 2014

ima lokalni Git repozitorij. U tom lokalnom repozitoriju se pohranjuje cjelokupna povijest projekta, sve verzije svake datoteke i svaka promjena na datotekama. Commitom programeri vrše promjene u svom kodu. Commit je snimka promjena koje su napravljene od zadnjeg commita. Svaki commit ima jedinstveni identifikator kao SHA-1(eng. Secure Hash Algorithm) hash.

Git omogućava programerima stvaranje takozvanih grana, koje su odvojene linije razvoja. Grane omogućuju rad na novim značajkama i ispravcima grešaka bez utjecaja na glavni projekt. Spajanjem u glavnu granu se mogu kombinirati izmjene i dovesti do promjena na glavnoj grani nakon što je sporedna grana prošla sve provjere ispravnosti.

Programeri mogu surađivati i na udaljenom repozitoriju koji je baziran na glavnoj grani, ali se nikada ne spaja s njom.

3.3.1. Github

GitHub je platforma u oblaku koja je izgrađena na Gitu, a na kojoj možete pohraniti, dijeliti i surađivati s drugima na pisanju koda.[14] GitHub poboljšava sposobnosti Gita dodajući značajke poput alata za suradnju, upravljanje projektima i praćenje problema. GitHub nudi uslugu pohrane Git repozitorija na jednom mjestu tako da su dostupni na globalnoj razini.

GitHub nudi nekoliko poboljšanja za Git. Hostiranje u oblaku olakšava dijeljenje i suradnju s drugima. Programeri mogu klonirati, gurnuti i povući svoj kod direktno iz oblaka, neovisno gdje se nalaze. Zahtjevi za povlačenje omogućuju programerima da predlože promjene koje njihovi kolege mogu pregledati i komentirati prije spajanja u glavnu granu.

3.4. Python

Python je programski jezik opće namjene visoke razine. Njegova filozofija dizajna naglašava čitljivost koda uz korištenje značajnog uvlačenja. Razvijen krajem 1980-ih od strane Guida van Rossuma, Python je postao izrazito popularan s jakom zajednicom otvorenog koda. Python koristi zamke za označavanje blokova koda, što potiče čiste i dosljedne prakse kodiranja i poboljšava čitljivost koda. Python kod je jasan za čitanje i pisanje i jezgrovit je, ali nije zagonetan. Python je vrlo ekspresivan jezik, što znači da obično možemo napisati mnogo manje redaka Python koda nego što bi bilo potrebno za ekvivalentnu aplikaciju napisanu u, recimo, C++ ili Java.[15] Ovaj programski jezik podržava

[14] https://docs.github.com/en/get-started/start-your-journey/about-github-and-git
[15] Mark Summerfield. Programming in Python 3: A Complete Introduction to the Python Language. November 2008.

različite paradigme programiranja, uključujući objektno orijentirano, imperativno i funkcionalno programiranje, pružajući fleksibilnost u pristupu programiranju.

Python se široko koristi u razvoju web aplikacija, analizi podataka, znanstvenom računanju, strojnom učenju i automatizaciji. Njegova popularnost i svestranost pružaju brojne prilike za developere u različitim industrijama. U svrhu ovog projekta će se koristiti kao programski jezik za programiranje lambda funkcije.

3.5. HTML

HTML (eng. Hypertext Markup Language) je standardni jezik za označavanje i strukturiranje web sadržaja. Web preglednici primaju HTML dokumente s web poslužitelja ili iz lokalne pohrane te pretvaraju te dokumente u multimedijalne web stranice. HTML se bazira na oznakama koje označavaju različite dijelove sadržaja poput naslova, slike, teksta ili poveznice. Kombinacijom HTML-a i drugih jezika i tehnologija dobivamo interaktivne web stranice. U svrhu ovog projekta smo izradili demonstracijsku web stranicu za naše web poslužitelje.

3.6. Bash

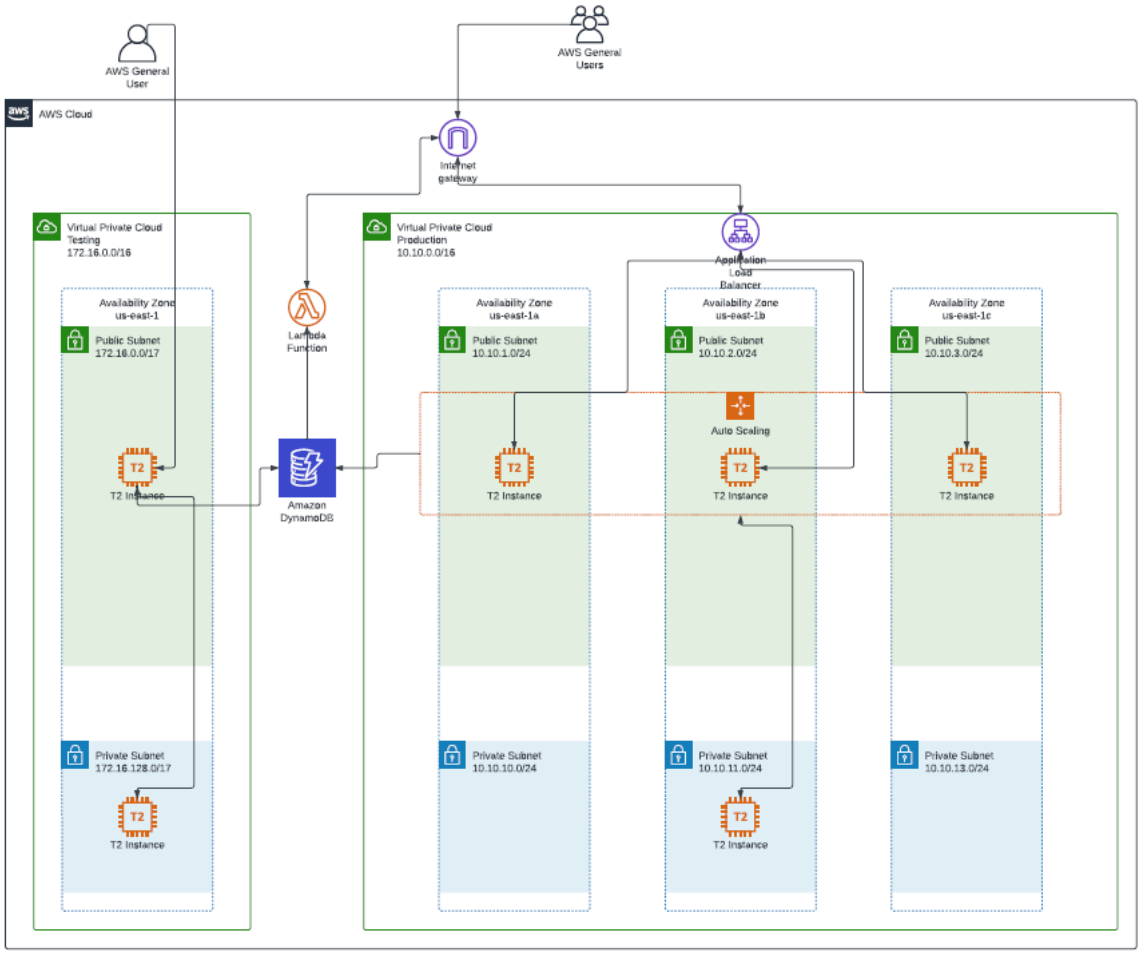
Bash (eng. Bourne Again Shell) je interpretor naredbenog retka koji se koristi na Linux i Unix operativnim sustavima. To je moćan alat za interakciju s operativnim sustavom putem naredbi. Bash omogućava korisnicima automatizaciju zadataka, manipulaciju datotekama i direktorijima te izvođenje skripti kako bismo olakšali upravljanje sustavom. Bash podržava varijable, petlje i uvjete što omogućava izradu kompleksnih skripti za automatizaciju zadataka. U ovom projektu koristimo Bash skripte za automatizaciju instalacije servisa za posluživanje web stranica na web poslužitelju.

4. Razrada teme

Za potrebe ovog rada izrađujemo arhitekturu jednog web servisa u AWS oblaku. Putem Terraform koda izrađujemo sve potrebno za prikaz jednog web servisa. Izvorni kod se nalazi na GitHub repozitoriju projekta što omogućuje provjeru i verzioniranje. Implementacijom koda na AWS računu stvara se sve potrebno za pružanje usluge web servisa.

4.1. Arhitektura

Osnova arhitekture je AWS račun u kojem se nalazi virtualni privatni oblak za projekt. Unutar našeg oblaka postavljamo šest podmreža: jednu privatnu i jednu javnu podmrežu u tri različite dostupne zone. Dvije EC2 instance smještene u javnim podmrežama služe kao poslužitelji korisničkog sučelja web servisa. Na S3 pohranjujemo potrebnu konfiguraciju za poslužitelje, kao i HTML kod korisničkog sučelja. Instance se nalaze iza elastičnog balansera opterećenja koji će ravnomjerno usmjeravati promet na obje instance.



Slika 4 - Plan Arhitekture

Jedna instanca sa SQL bazom u privatnoj podmreži služit će kao simulacija veze između korisničkog sučelja i baze podataka putem sigurnosnih grupa. Podižemo i tablicu na AWS DynamoDB servisu za simulaciju prometa između EC2 instanci i upravljanih baza od strane Amazona putem uloga koje dodjeljujemo instancama. Sva prava su ograničena korištenjem uloga i politika s najmanje potrebnim pravima.

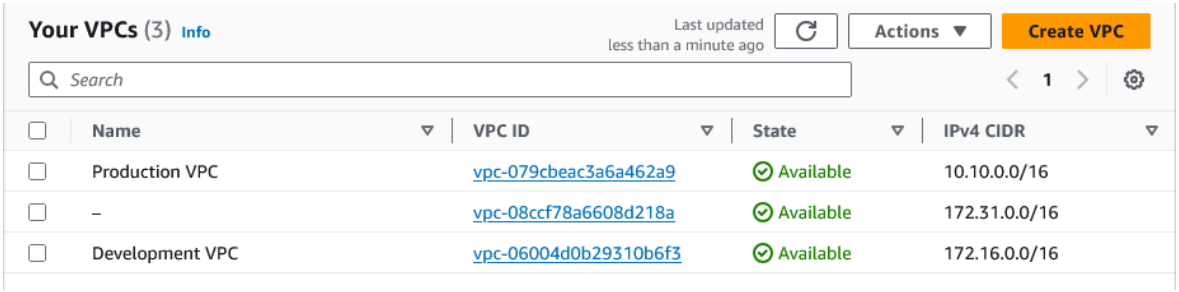
4.2. Terraform Code

4.2.1. VPC

Kao osnovu arhitekture koristimo produkcijski VPC. Za ovaj VPC koristit ćemo adresni prostor 10.10.0.0/16. Ovo je privatni adresni blok s oznakom /16, što znači da prvih 16 bitova služe za označavanje adrese. To nam omogućuje da koristimo sve adrese od 10.10.0.0 do 10.10.255.255, što nam daje 65534 IP adrese.

```
resource "aws_vpc" "prod_vpc" {
  cidr_block = "10.10.0.0/16"
  tags = {
    Name = "Production VPC"
  }
}
```

Kod 1- VPC



Slika 5 - VPC

VPC dijelimo na tri privatne i tri javne pod mreže, svaka smještena u različitim dostupnim zonama. Ovaj pristup se koristi kako bi osigurali redundanciju i visoku dostupnost naše mrežne arhitekture.

Podmreža	Dostupna Zona	CIDR
Javna Podmreža 1	us-east-1a	10.10.1.0/24
Javna Podmreža 2	us-east-1b	10.10.2.0/24
Javna Podmreža 3	us-east-1c	10.10.3.0/24
Privatna Podmreža 1	us-east-1a	10.10.11.0/24
Privatna Podmreža 2	us-east-1b	10.10.12.0/24
Privatna Podmreža 2	us-east-1c	10.10.13.0/24

U javnim podmrežama dijelit ćemo IP adresni raspon 10.10.x.0/24 gdje je x redni broj od 1 do 3. To znači da će svaka javna pod mreža imati vlastiti raspon IP adresa unutar svog bloka. Javne podmreže će služiti za resurse koji moraju biti dostupni prema internetu, kao što su web poslužitelji.

```
resource "aws_subnet" "public_subnet" {
  count                   = 3
  vpc_id                  = aws_vpc.prod_vpc.id
  cidr_block              = "10.10.${1 + count.index}.0/24"
  availability_zone       = var.availability_zones[count.index]
  map_public_ip_on_launch = true
  tags = {
    Name = "Public Subnet ${var.availability_zones[count.index]}"
  }
}
```

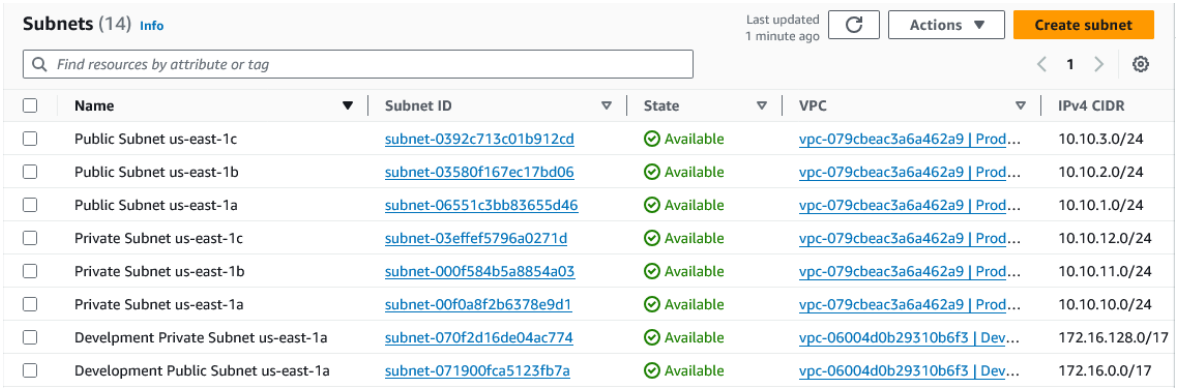
Kod 2- Javna podmreža

S druge strane, privatne podmreže koristit će IP adresni raspon od 10.10.x.0/24 gdje je x redni broj od 11 do 13. Kao i kod javnih podmreža, svaka podmreža imat će vlastiti jedinstveni raspon IP adresa unutar svog bloka. Privatne podmreže koristimo za resurse koje ne želimo izložiti internetu, kao što su baze podataka. Ova podjela pomaže u poboljšanju sigurnosti i kontroli nad našim mrežnim prometom.

```
resource "aws_subnet" "private_subnet" {
  count             = 3
  vpc_id            = aws_vpc.prod_vpc.id
  cidr_block        = "10.10.${10 + count.index}.0/24"
  availability_zone = var.availability_zones[count.index]
  tags = {
    Name = "Private Subnet ${var.availability_zones[count.index]}"
  }
}
```

Kod 3- Privatna podmreža

Koristeći tri različite dostupne zone, osiguravamo da čak i ako jedna dostupna zona ima probleme ili prekid rada, naše usluge će ostati dostupne unutar druge zone.



Slika 6 - Podmreže

Internet pristupnik nam omogućuje komunikaciju između resursa u javnim podmrežama i interneta. On djeluje kao most između naših internih resursa i svjetske mreže. Bez internet pristupnika, resursi u javnim podmrežama ne bi mogli uspostavljati vezu s internetom ili primati dolazne zahtjeve s interneta.

```
resource "aws_internet_gateway" "prod_internet_gateway" {
  vpc_id = aws_vpc.prod_vpc.id
  tags = {
    Name = "Internet Gateway Prod"
  }
}
```

Kod 4- Internet pristupnik

Za određivanje putanja kojima će se podatci kretati između različitih resursa ili mrežnih segmenata, potrebno nam je definirati tablice usmjeravanja. Potrebne su nam dvije tablice, jedna privatna i jedna javna. U tablici koja usmjerava promet javnih podmreža definiramo dvije rute: sav promet u adresnom bloku 0.0.0.0/0, što znači sve javne IP adrese, bit će usmjeren prema internet brani. Ovo omogućuje resursima u javnim podmrežama da šalju svoj promet prema internetu. Druga ruta je sav promet koji se nalazi u privatnom bloku 10.10.0.0/16, a smatra se lokalnim prometom i preusmjerava se prema odgovarajućim resursima. Ovo nam omogućava lokalnu komunikaciju unutar podmreža, bile one javne ili privatne. Za privatne podmreže tablica usmjeravanja je nešto jednostavnija jer sadrži samo putanju za sav promet unutar privatnog bloka 10.10.0.0/16 koji se preusmjerava lokalno.

```
resource "aws_route_table" "prod_public_route_table" {
  vpc_id = aws_vpc.prod_vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.prod_internet_gateway.id
  }
  route {
    cidr_block = "10.10.0.0/16"
    gateway_id = "local"
  }
  tags = {
    Name = "Public Routing Table Prod"
  }
}
```

Kod 5 - Tablica usmjeravanja

U Terraformu moramo nakon stvaranja ovih tablica usmjeravanja asocirati te tablice na odgovarajuće podmreže: javna tablica usmjeravanja na javne podmreže i privatna tablica usmjeravanja na privatne podmreže. Ova asocijacija osigurava da se primjenjuju odgovarajuća pravila usmjeravanja prometa na odgovarajuće podmreže.

4.2.2. S3

U projektu za dugotrajnu pohranu koristimo AWS S3 servis. Potrebno je stvaranje spremišta s jedinstvenim imenom koje će služiti za pohranu naših podataka. Podatke u spremištu pohranjujemo u standardnoj klasi skladištenja u glavnoj mapi. Standardna klasa je najskuplja, ali nam osigurava najbržu dostupnost jer se podatci nalaze u nekoliko instanci unutar više dostupnih zona.

```
resource "aws_s3_bucket" "web_files" {
  bucket = "zd-web-files-uni"
  tags = {
    Name        = "Web Files"
    Environment = "Production"
  }
}
```

Kod 6- S3 bucket

Za enkripciju koristimo AES256 (eng. Advanced Encryption Standard 256-bit) koji je snažan algoritam šifriranja i industrijski standard danas. Ovaj oblik šifriranja je dodatni sloj zaštite od neovlaštenog pristupa i čitanja naših podataka. Nakon postavljanja AES256 šifriranja kao zadanu postavku, svi objekti koje učitamo u spremište bit će šifrirani od strane S3 servisa.

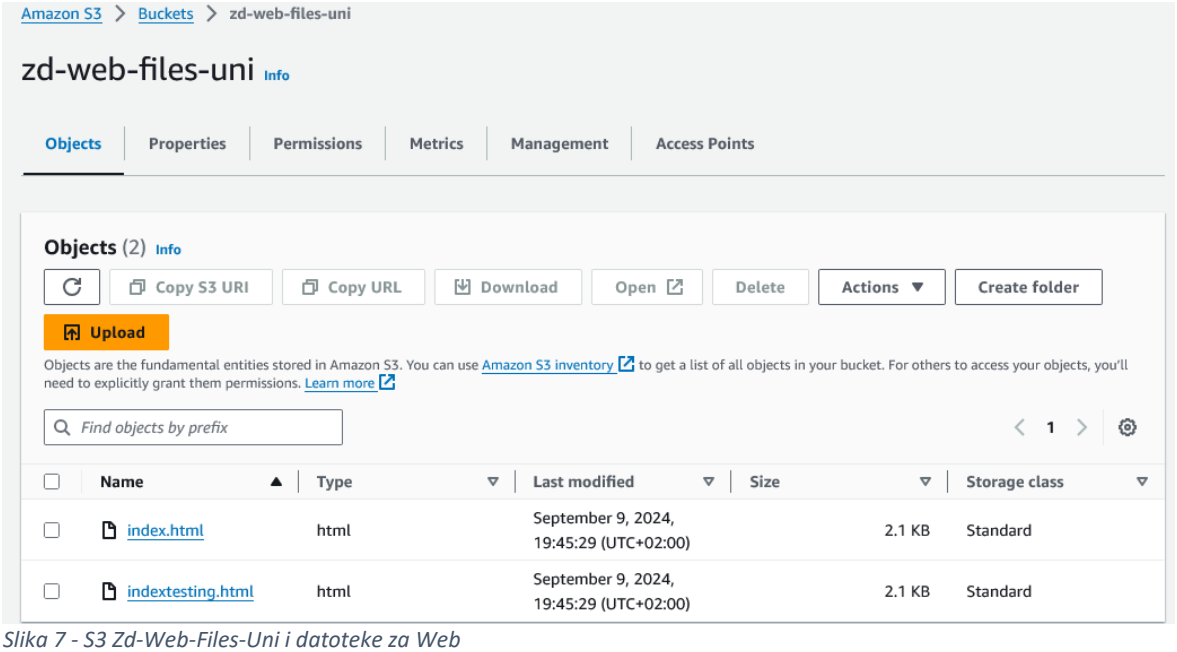
```
resource          "aws_s3_bucket_server_side_encryption_configuration"
"web_files_encryption" {
  bucket = aws_s3_bucket.web_files.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}
```

Kod 7- S3 enkripcija

Unutar spremišta učitavamo HTML stranicu za naše web poslužitelje kao objekt. Ova web stranica će biti korištena od strane naših web poslužitelja koji će je prikazivati korisnicima kad pristupe našoj stranici. S3 se brine za pohranu stranice, što omogućava da stranica bude neovisna o web poslužiteljima, što nam omogućava skaliranje i visoku dostupnost.

```
resource "aws_s3_object" "main_web_page" {
bucket = aws_s3_bucket.web_files.bucket
key    = "index.html"
source = ("scripts/index.html")
etag   = filemd5("scripts/index.html")
}
```

Kod 8- HTML datoteka



Slika 7 - S3 Zd-Web-Files-Uni i datoteke za Web

4.2.3. Autoskaliranje web poslužitelja

Kako bismo osigurali pouzdanost, skalabilnost i visoku dostupnost naše web aplikacije, moramo stvoriti grupu automatskog skaliranja instanci za naše Web poslužitelje. Stvaranje grupe automatskog skaliranja zahtijeva nekoliko koraka i komponenata.

Prvo smo morali definirati predložak za pokretanje, što je konfiguracija novih instanci koje će biti stvorene u grupi automatskog skaliranja. U ovom predlošku definiramo sve parametre instance. Za AMI (Amazon Machine Image) koristimo Amazon Linux, koji dolazi s unaprijed konfiguriranim paketima i postavkama prikladnim za pokretanje aplikacija na AWS-u. Vrsta instance je t2.micro, što je dovoljno za naše potrebe u pogledu resursa procesora i radne memorije.

```
resource "aws_launch_template" "web_server_template" {
  name = "Web_Server_Template"
  iam_instance_profile {
    name = aws_iam_instance_profile.web_instance_profile.name
  }
  image_id = "ami-067d1e60475437da2"
  instance_type = "t2.micro"
  network_interfaces {
    associate_public_ip_address = true
    security_groups             = [aws_security_group.ec2_cluster_sg.id]
  }
  user_data = base64encode(file("scripts/install_web.sh"))
  tags = {
    Name = "Web-Server"
  }
}
```

Kod 9- Predložak za pokretanje

Pri stvaranju instanci priložit ćemo mrežni uređaj s odgovarajućom sigurnosnom grupom kako bismo osigurali da su instance opremljene pravilnim mrežnim resursima i pristupom. Na kraju, u predlošku dodajemo skriptu za postavljanje koja će se izvršavati prilikom podizanja instance i postaviti web uslugu na instanci. U skripti za postavljanje instaliramo web servis „httpd“, koji nam služi za posluživanje web stranice i povlačimo našu HTML web stranicu iz S3 pohrane. Ovaj pristup osigurava da su naše instance pravilno konfigurirane i spremne za obavljanje zadataka odmah pri podizanju. Svaka instanca prilikom postavljanja se povezuje s DynamoDB bazom i upisuje svoje ime hosta u tablicu kako bismo mogli pratiti broj instanci.

S ovim predloškom za pokretanje možemo stvoriti grupu automatskog skaliranja. Za grupu moramo odabrati sve tri javne podmreže kako bismo rasporedili web poslužitelje unutar sve tri dostupne zone i omogućili im pristup internetu. Definiramo i predložak za pokretanje za web poslužitelje. U definiciji grupe automatskog skaliranja postavljamo maksimalni, minimalni i željeni broj instanci u grupi. U našem slučaju postavljamo maksimalni i željeni broj instanci na 3 i minimalno dvije instance, što nam omogućava da

zadržimo jednu instancu u svakoj zoni, ostavljajući nam prostora da smanjimo željeni broj na dvije u slučaju manjeg opterećenja.

```
resource "aws_autoscaling_group" "web_server_asg" {
  name = "Web_Server_ASG"
  vpc_zone_identifier = aws_subnet.public_subnet[*].id
  max_size         = 3
  min_size         = 2
  desired_capacity = 3
  launch_template {
    id      = aws_launch_template.web_server_template.id
    version = "$Latest"
  }
  depends_on = [aws_launch_template.web_server_template]
  tag {
    key                 = "Name"
    value               = "Web-Server"
    propagate_at_launch = true
  }
}
```

Kod 10- Grupa automatskog skaliranja

4.2.4. Aplikacijski balanser opterećenja

Da bismo ravnomjerno rasporedili dolazne zahtjeve na instance unutar naše grupe automatskog skaliranja, potreban nam je balanser opterećenja. U ovom slučaju koristimo aplikativni balanser opterećenja (ALB) koji će djelovati kao centralna točka za primanje dolaznih zahtjeva i njihovo usmjeravanje prema web poslužiteljima unutar ciljane grupe.

Prvo definiramo ciljnu grupu za aplikativni balanser opterećenja. Ova ciljna grupa određuje kako će aplikativni balanser opterećenja usmjeravati promet prema našim web poslužiteljima. Postavljamo port na kojem će se posluživati aplikacija, koji je u ovom slučaju 80 za HTTP. Također smanjujemo provjeru zdravlja na minimalne postavke kako bismo omogućili što bržu izmjenu instanci u slučaju ispada ili preopterećenja jedne od njih.

```
resource "aws_lb_target_group" "web_server_target_group" {
  name        = "WebServerTG"
  port        = 80
  protocol    = "HTTP"
  vpc_id      = aws_vpc.prod_vpc.id
  target_type = "instance"
  health_check {
    healthy_threshold = 2
    timeout           = 2
    interval          = 5
  }
}
```

Kod 11- Ciljna grupu

Zatim definiramo slušatelja za aplikativni balanser opterećenja. Slušatelj određuje kako će aplikativni balanser opterećenja reagirati na dolazne zahtjeve. Postavljamo ga na port 80 i definiramo da će svi zahtjevi biti usmjereni prema ciljnoj grupi za web poslužitelje koju smo prethodno definirali.

```
resource "aws_lb_listener" "web_server_listener" {
  load_balancer_arn = aws_lb.web_alb.arn
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.web_server_target_group.arn
  }
}
```

Kod 12- Slušatelja

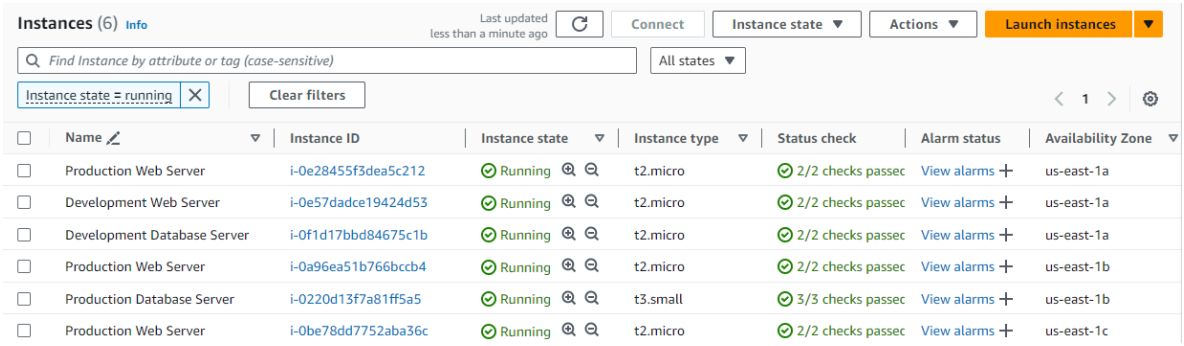
Konačno, definiramo postavke aplikativnog balansera opterećenja, kao što su naziv, sigurnosne grupe i podmreže. Prvo definiramo da aplikativni balanser opterećenja nije interni, što omogućuje promet prema internetu. Zatim ga dodajemo u sigurnosnu grupu za ALB, koja omogućuje promet na portu 80 za HTTP promet. Sve instance aplikativnog balansera opterećenja nalaze se unutar javno dostupnih podmreža, pa ih također dodajemo u postavke. Na kraju, isključujemo zaštitu brisanja kako bismo olakšali izmjenu instanci i postavki. Nakon što smo definirali sve što je potrebno za rad aplikativnog balansera opterećenja i grupe automatskog skaliranja, potrebno je stvoriti asocijaciju između njih.

```
resource "aws_lb" "web_alb" {
  name               = "WebALB"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.web_alb_sg.id]
  subnets            = aws_subnet.public_subnet[*].id

  enable_deletion_protection = false

  tags = {
    Name = "Web Aplication Load Balancer"
  }
}
```

Kod 13- Aplikativni balanser opterećenja



Slika 8 - Virtualne Instance

4.2.5. Politike

Da bismo pratili načelo najmanjeg privilegija, potrebno je stvoriti svoje vlastite ograničene politike u IAM centru. U ovom slučaju stvaramo tri politike, jednu za pristup DynamoDB tablici, drugu za pristup podacima na S3 i treća za Lambda funkcije.

Za pristup DynamoDB bazi ograničavamo politiku na tri moguće naredbe: "get" (dobivanje), "put" (postavljanje) i "scan" (skeniranje) nad DynamoDB tablicom "WebServers". Ovo osigurava mogućnost čitanja i pisanja samo unutar te jedne baze.

```
resource "aws_iam_policy" "dynamodb_servers_policy" {
  name        = "dynamodb_servers_policy"
  path        = "/"
  description = "EC2 Access to DynamoDB user Table"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        "Sid" : "VisualEditor0",
        "Effect" : "Allow",
        "Action" : [
          "dynamodb:PutItem",
          "dynamodb:GetItem",
          "dynamodb:Scan"
        ],
                        "Resource"      :     "arn:aws:dynamodb:us-east-
1:${data.aws_caller_identity.current.account_id}:table/WebServers"
      }
    ]
  })
}
```

Kod 14- Politika za dynamodb

Za pristup S3 ograničena politika dopušta samo dohvaćanje datoteka koje se nalaze unutar spremišta "zd-web-files-uni". Tako osiguravamo da korisnici ili uloge nemaju nepotrebne ovlasti koje bi mogle nanijeti ikakvu štetu sustavu.

```
resource "aws_iam_policy" "s3_web_policy" {
  name        = "s3_web_policy"
  path        = "/"
  description = "EC2 Access to S3 Web Bucket"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        "Sid" : "VisualEditor0",
        "Effect" : "Allow",
        "Action" : [
          "s3:GetObject"
        ],
        "Resource" : "arn:aws:s3:::zd-web-files-uni/*"
      }
    ]
  })
}
```

Kod 15- Politika za S3

Da bismo pokrenuli Lambda funkciju moramo omogućiti funkciji prava da zabilježi događaje. Bez tih prava se funkcija neće izvršiti. Događaje AWS sprema u servisu AWS Logs koji se kasnije mogu pohranjivati na S3 pohranu.

```
resource "aws_iam_policy" "lambda_access_policy" {
  name        = "lambda_access_policy"
  path        = "/"
  description = "Lambda access policies"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        "Effect" : "Allow",
        "Action" : [
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents"
        ],
        "Resource" : "*"
      }
    ]
  })
}
```

Kod 16- Politika za lambdu

Policy name	Type	Used as	Description
allow_all	Customer managed	Permissions policy (1)	-
dynamodb_servers_policy	Customer managed	Permissions policy (2)	EC2 Access to DynamoDB user Table
lambda_access_policy	Customer managed	Permissions policy (1)	Lambda access policies
Playground_AWS_Sandbox	Customer managed	Permissions policy (1)	-
s3_web_policy	Customer managed	Permissions policy (1)	EC2 Access to S3 Web Bucket

Slika 9 - Politike

4.2.6. Uloge

Nakon što smo definirali politike za pristup AWS resursima, trebamo ih dodati ulozi. Jedan od načina na koji entiteti unutar AWS-a mogu pristupiti drugim resursima je putem uloga. Kako bismo omogućili web poslužiteljima pristup resursima kao što su DynamoDB i S3, potrebno je stvoriti ulogu i dodijeliti odgovarajuće politike.

U definiciji uloge "web_instance_role" definiramo da svi entiteti iz usluge EC2 mogu preuzeti sve ovlasti koje se nalaze pod tom ulogom. Na ulogu "web_instance_role" povezujemo dvije politike koje smo sami napisali i jednu od Amazona. Amazonova politika "ssmmanagedec2instancepolicy" omogućuje instancama pristup Amazonovoj usluzi SSM (Upravljanje sustavima Amazona).

```
resource "aws_iam_role" "web_instance_role" {
  name = "Web_Instance_Role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      },
    ]
  })
  tags = {
    tag-key = "EC2 Role"
  }
}
```

Kod 17- Uloga web instance

Naše dvije politike, "dynamodb_servers_policy" i "s3_web_policy", služe tome da bi instance imale ograničen pristup DynamoDB i S3 uslugama. Kako bismo omogućili

instancama da preuzmu ovu ulogu, moramo je povezati s profilom koji se dodjeljuje instancama.

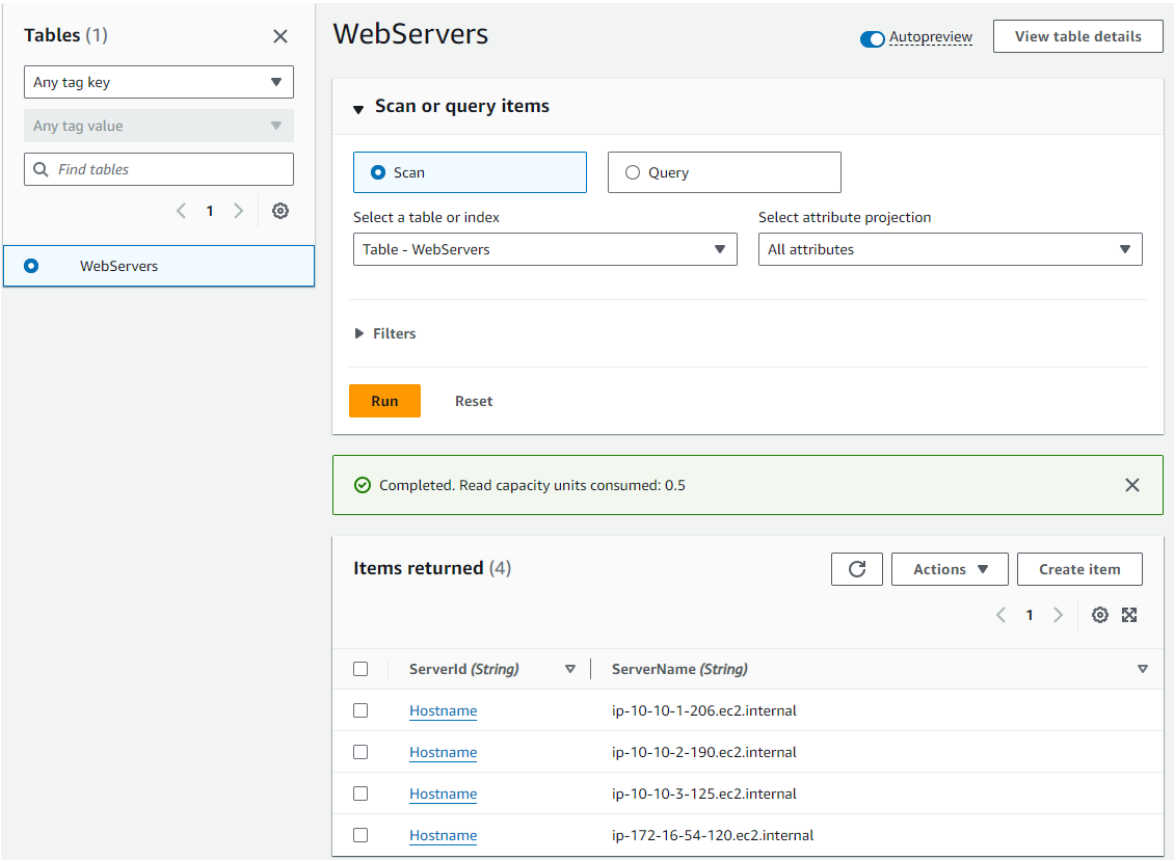
4.2.7. DynamoDB

Naša web aplikacija ima pristup bazi kao servis putem Amazonove DynamoDB usluge. Naziv tablice je "WebServers," a odabrali smo način naplate "Provisioned," što znači da ćemo ručno konfigurirati kapacitete čitanja i pisanja. Postavili smo kapacitete čitanja i pisanja na 5, što znači da naša tablica može podnijeti do 5 operacija čitanja i 5 operacija pisanja po sekundi.

```
resource "aws_dynamodb_table" "web_service_table" {
  name           = "WebServers"
  billing_mode   = "PROVISIONED"
  read_capacity  = 5
  write_capacity = 5
  hash_key       = "ServerId"
  range_key      = "ServerName"
  attribute {
    name = "ServerId"
    type = "S"
  }
  attribute {
    name = "ServerName"
    type = "S"
  }
  tags = {
    Name        = "dynamodb-server-table"
    Environment = "production"
  }
}
```

Kod 18- DynamoDB tablice

Ključevi tablice su definirani kao "ServerId" i "ServerName," što nam omogućuje identifikaciju i dohvaćanje stavki iz tablice. Atributi su također "ServerId" i "ServerName," oba u obliku stringa.



Slika 10 - DynamoDB WebServers Tablica

4.2.8. Zasebne instance

Za potrebe web aplikacije možemo sami konfigurirati i posluživati bazu podataka. Da bismo stvorili novu instancu, prvo je potrebno definirati AMI (Amazon Machine Image) koji će se koristiti i odabrati tip instance. U ovom slučaju biramo AMI "ami-053b0d53c279acc90," koji je pripremljen s instalacijom Ubuntu Server 22.04, i koristit ćemo t2.micro instancu. Instancu dodajemo u jednu od tri privatne podmreže i pridružujemo je posebnoj sigurnosnoj grupi za instance s bazom podataka. Sigurnosna grupa za instance s bazom podataka ima dozvoljen promet samo s instancama koje se nalaze u sigurnosnoj grupi web poslužitelja na portu "3306" za SQL.

```
resource "aws_instance" "sql_instance" {
  ami                    = "ami-053b0d53c279acc90"
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.private_subnet[1].id
                iam_instance_profile
aws_iam_instance_profile.web_instance_profile.name
  vpc_security_group_ids = [aws_security_group.sql_cluster_sg.id]

  tags = {
    Name = "Production Database Server"
  }
```

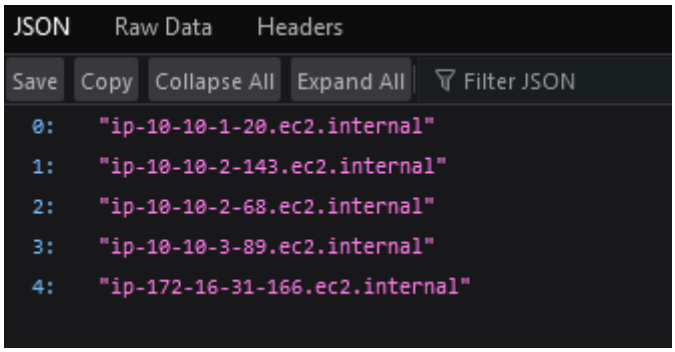
Kod 19- SQL instance

4.2.9. Lambda

U okviru našeg projekta razvili smo Lambda funkciju u Pythonu kako bismo izvukli listu servera iz DynamoDB tablice u JSON formatu i prikazali je putem poziva na URL adresi Lambda funkcije. Kako bismo stvorili Python funkciju pomoću Terraforma, prvo moramo arhivirati kod u ZIP datoteku. U definiciji Lambda funkcije određujemo datoteku Lambda funkcije s arhiviranim kodom, naziv funkcije, ulogu koja će se koristiti i okolinu, koja je u ovom slučaju Python 3.9. Uloga koju stvaramo daje AWS Lambda usluzi mogućnost pristupa politikama koje smo definirali, uključujući politiku za pristup DynamoDB tablici i politiku za pristup Lambda funkciji. To omogućava Lambda funkciji stvaranje zapisa u AWS CloudWatchu. Naposljetku, definiramo URL funkciju koja će biti povezana s Lambda funkcijom kako bi djelovala kao okidač i omogućila nam pokretanje Lambda funkcije izvana putem URL-a.

```
resource "aws_lambda_function" "web_servers_lambda" {
  filename      = "scripts/lambda_webservers.zip"
  function_name = "lambda_get_webservers"
  role          = aws_iam_role.lambda_role.arn
  handler       = "lambda_webservers.lambda_handler"
  runtime = "python3.9"
  environment {
    variables = {
      foo = "bar"
    }
  }
}
resource "aws_lambda_function_url" "web_servers_lambda_url" {
              function_name                                    =
aws_lambda_function.web_servers_lambda.function_name
  authorization_type = "NONE"
```

Kod 20- Lambda



Slika 11 – Odgovor Lambde

4.2.10. Development VPC

Uz produkcijsko okruženje i suvremeno programiranje, aplikacija zahtijeva razvojno ili testno okruženje gdje se promjene mogu testirati bez ometanja rada produkcijskog okruženja. U tu svrhu stvorit ćemo razvojni VPC (Virtual Private Cloud) unutar našeg računa. Budući da se razvojno okruženje koristi za testiranje, račun za razvoj će biti manja, ekonomičnija verzija web aplikacije koja je na developerskom okruženju. Sastojat će se samo od jedne javne i jedne privatne podmreže, s po jednim poslužiteljem u svakoj.

```
resource "aws_vpc" "dev_vpc" {
  cidr_block = "172.16.0.0/16"

  tags = {
    Name = "Development VPC"
  }
}
```

Kod 21- Dev VPC

Za VPC ćemo koristiti CIDR blok 172.16.0.0/16 kako bismo izolirali poslužitelje od produkcijskog VPC-a, budući da se radi o različitom rasponu privatnih IP adresa. Unutar tog bloka stvorit ćemo dvije pod mreže koristeći raspone 172.16.0.0/17 za javnu podmrežu i 172.16.128.0/17 za privatnu podmrežu. Obje podmreže bit će smještene u dostupnosti zone us-east-1a. Ovaj VPC zahtijeva internet vrata koji će biti definiran za javnu podmrežu, a rutne tablice bit će definirane kako bi ograničile promet samo prema ovom VPC-u.

```
resource "aws_subnet" "dev_public_subnet" {
  vpc_id                  = aws_vpc.dev_vpc.id
  cidr_block              = "172.16.0.0/17"
  availability_zone       = var.availability_zones[0]
  map_public_ip_on_launch = true
  tags = {
    Name = "Development Public Subnet ${var.availability_zones[0]}"
  }
}
resource "aws_subnet" "dev_private_subnet" {
  vpc_id            = aws_vpc.dev_vpc.id
  cidr_block        = "172.16.128.0/17"
  availability_zone = var.availability_zones[0]
  tags = {
    Name = "Develpment Private Subnet ${var.availability_zones[0]}"
  }
}
```

Kod 22- Dev podmreže

Web poslužitelj u razvojnom VPC-u bit će definiran isto kao i poslužitelji u produkcijskom okruženju, koristeći isti AMI i iste pristupne politike. Razlika je u tome što ćemo stvoriti samo jedan poslužitelj, pa nećemo definirati grupu za automatsko skaliranje, već ćemo ga definirati kao samostalnu instancu s nazivom Razvojni web poslužitelj. Također mu nećemo pridružiti aplikativni balanser opterećenja (ALB) jer se radi o samostalnoj instanci, a u svrhu testiranja omogućit ćemo pristup internetu preko port 80.

```
resource "aws_instance" "dev_web_instance" {
  ami                    = "ami-067d1e60475437da2"
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.dev_public_subnet.id
                iam_instance_profile                                    =
aws_iam_instance_profile.web_instance_profile.name
  vpc_security_group_ids = [aws_security_group.dev_ec2_cluster_sg.id]

  user_data = base64encode(file("scripts/install_web_development.sh"))

  tags = {
    Name = "Development Web Server"
  }
}
```

Kod 23- Dev Web instanca

Baza podataka će biti definirana na isti način kao i produkcijski poslužitelj baze podataka, ali s promijenjenim oznakama i imenom na "development". Pristup će biti ograničen samo na instance u grupi razvojnih web poslužitelja, i neće biti omogućen pristup internetu.

```
resource "aws_instance" "dev_sql_instance" {
  ami                    = "ami-053b0d53c279acc90"
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.dev_private_subnet.id
                iam_instance_profile                                    =
aws_iam_instance_profile.web_instance_profile.name
  vpc_security_group_ids = [aws_security_group.dev_sql_cluster_sg.id]

  tags = {
    Name = "Development Database Server"
  }
}
```

Kod 24- Dev SQL instanca

Za potrebe development okruženja učitavamo novu indeks_testing.html html datoteku na S3. Ta datoteka će se posluživati na web poslužitelju u development okruženju kako bismo omogućili developerima testiranje promjena bez da se ometa rad glavne produkcijske infrastrukture.

4.2.11. Git

Kako bismo u potpunosti iskoristili potencijal IaC, potrebno je pohraniti naš kod u Git repozitorij. Za ovaj projekt kreirali smo repozitorij pod nazivom Zav-TF na GitHubu. Za učitavanje cjelokupnog izvornog koda koristimo sljedeću naredbu: „git push origin main“.

Ova naredba će prenijeti sve promjene izravno u glavnu granu. Kod je sada verzioniran, što omogućava jednostavan pristup, ažuriranje i dijeljenje s onima kojima damo pristup. Ovakav pristup donosi sve prednosti IaC-a, uključujući lakšu suradnju, kontrolu verzija i upravljanje promjenama.

5. Zaključak

Creating network infrastructure using open-source infrastructure as code and cloud service providers

U sklopu ovog istraživanja o stvaranju mrežne infrastrukture korištenjem infrastrukture kao koda na cloud uslugama, uspjeli smo demonstrirati upotrebu IaC-a u stvaranju robusne i prilagodljive infrastrukture na AWS-u za potrebe poslužitelja web aplikacije. Kreiranjem više virtualnih privatnih oblaka uspjeli smo osigurati odvajanje između razvojnih i produkcijskih web poslužitelja, omogućujući programerima rad i testiranje bez utjecaja na korisnike na produkcijskim sustavima. Testirali smo sposobnost samog web poslužitelja i instanci baza podataka te primijenili upotrebu tehnologije bez poslužitelja za računanje, poslužitelja baza podataka i pohrane. Korištenjem samo stvorenih pristupnih politika, uloga i sigurnosnih grupa uspjeli smo stvoriti infrastrukturu koja je i sigurna i skalabilna. Cloud usluge koje nude više zona dostupnosti omogućile su nam stvaranje infrastrukture koja je otporna na kvarove hardvera i skalabilna po potrebi.

IaC nam je omogućio iskorištavanje svih prednosti koda kao što su verzioniranje, pregledi koda i sudjelovanje više sudionika dok smo i dalje radili na infrastrukturi kako bismo omogućili agilan pristup razmišljanju u operacijama. Jednostavna sposobnost kopiranja i lijepljenja koda te korištenje petlji i varijabli povećava produktivnost održavajući standardizaciju i preglednost. Na kraju, mogućnost ponovnog stvaranja cijelih infrastruktura prema potrebi omogućila nam je da u samo nekoliko minuta možemo implementirati našu aplikaciju u bilo kojem podatkovnom centru diljem svijeta. U usporedbi s tradicionalnim implementacijama na fizičkim poslužiteljima unutar tvrtke prije samo nekoliko godina, to je ništa manje nego izvanredno.

Popis literature

1. Saurabh Shrivastava. Solutions Architect's Handbook - Second Edition. January 2022.
2. https://www.ibm.com/cloud/learn/infrastructure-as-code
3. https://developer.hashicorp.com/terraform/intro
4. Yevgeniy Brikman. Terraform: Up and Running, 3rd Edition. September 2022
5. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf
6. https://aws.amazon.com/s3/storage-classes/
7. https://aws.amazon.com/about-aws/global-infrastructure/
8. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html
9. Mitesh Soni. Practical AWS Networking: Build and manage complex networks using services such as Amazon VPC, Elastic Load Balancing, Direct Connect, and Amazon Route 53. January 2018
10. Andreas Wittig. Amazon Web Services in Action, Third Edition. May 2023
11. https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html
12. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
13. Scott Chacon, Ben Straub. ProGit 2nd Edition. 2014
14. https://docs.github.com/en/get-started/start-your-journey/about-github-and-git
15. Mark Summerfield. Programming in Python 3: A Complete Introduction to the Pyth on Language. November 2008.

Summary

This paper explores the development of a reliable, dynamically scalable network infrastructure using Infrastructure as Code (IaC) and cloud providers, with a focus on Terraform and Amazon Web Services. The solution is built according to industry best practices, combining technologies based on classic Linux servers and serverless technology for maximum efficiency and flexibility.

Terraform is used as the primary IaC tool for automating the configuration and management of cloud resources. AWS was chosen as the cloud service provider due to its extensive services and global reach. The infrastructure is designed with two separate virtual private clouds, one for users and one for development, to ensure service reliability.

Auto-scaling is implemented to dynamically adjust resource allocation based on demand, ensuring optimal reliability, performance, and cost-efficiency. EC2 instances, based on the Linux operating system, are used for web servers, while serverless computing like AWS Lambda is employed to create a hybrid infrastructure capable of handling varying workloads. Git is used for version control, enabling collaboration in managing the IaC code.

The focus is on reliability, scalability, and security while adhering to industry standards for cloud architecture. The application of IaC ensures consistency and agility in infrastructure across both development and production environments.

Keywords: Infrastructure as Code (IaC), Terraform, AWS, Cloud Server, Web Server

Popis priloga