Infrastruktura kao kôd : Ansible i Terraform u praksi

Stojić, Matko

VELEUČILIŠTE SUVREMENIH INFORMACIJSKIH TEHNOLOGIJA

STRUČNI PRIJEDIPLOMSKI STUDIJ INFORMACIJSKIH TEHNOLOGIJA

Matko Stojić

ZAVRŠNI RAD

INFRASTRUKTURA KAO KÔD : ANSIBLE I TERRAFORM U PRAKSI

Zagreb, listopada 2024.

Veleučilište suvremenih informacijskih tehnologija
10000 Zagreb, Ulica Vjekoslava Klaića 7

Studij:	Stručni prijediplomski studij informacijskih tehnologija smjer programiranje
Student:	**Matko Stojić**
Matični broj:	2015032

Zadatak završnog rada

Predmet:	Programski alati na UNIX računalima
Naslov:	**Infrastruktura kao kôd : Ansible i Terraform u praksi**
Zadatak:	Opisati tehnologije Ansible i Terraform na Linux operacijskom sustavu. Usporediti te tehnologije u kontekstu komunikacije s cloud servisima. Pomoću njih posložiti infrastrukturu i osnovnu web aplikaciju s bazom na najmanje dva virtualna poslužitelja u nekom javnom cloud servisu.
Mentor:	Ivan Capan, pred.
Zadatak uručen kandidatu:	11.10.2023.
Rok za predaju rada:	29.10.2024.
Rad predan:	_____

Povjerenstvo:

Dalibor Bužić, v. pred.	član predsjednik	_____
Ivan Capan, pred.	mentor	_____
Edmond Krusha, v. pred.	član	_____

SADRŽAJ

POPIS SLIKA

POPIS KôDOVA

1. UVOD

U modernom svijetu informacijskih tehnologija sposobnost upravljanja IT infrastrukturom te brza i efikasna implementacija su jedni od najvažnijih faktora uspjeha za organizacije svih veličina. Kako poslovanje postaje sve više oslonjeno na tehnologiju, potreba za brzim, sigurnim i efikasnim implementacijama raste. Klasični pristupi upravljanja infrastrukturom, koji se često oslanjaju na ručne procese, sve više ustupaju mjesto automatiziranim metodama zbog njihove bolje učinkovitosti, preciznosti i ekonomičnosti. Razvoj i održavanje ekonomičnih, pouzdanih i sigurnih IT sustava u takvom okruženju zahtijeva inovativne pristupe i alate. Jedan od takvih pristupa je koncept "Infrastruktura kao kôd" (engl. Infrastructure as a Code), koji bitno mijenja način upravljanja IT infrastrukturom omogućavajući brže, konzistentnije i pouzdanije postavljanje infrastrukturnih resursa kroz automatizaciju. Ovaj koncept omogućuje kompanijama ne samo da upravljaju svojim IT resursima na način koji je brži i konzistentniji, već također omogućuje bolju integraciju između različitih razvojnih i operativnih timova, poboljšavajući time DevOps prakse unutar organizacija. Automatizacijom, koja je središnji element IaC-a, smanjuje se potreba za ručnim intervencijama, a IT timovi mogu brže reagirati na promjene i zahtjeve, što im omogućuje da se usredotoče na složenije probleme i inovacije.

Ideja iza IaC (engl. Infrastructure as a Code) je da se piše i izvršava kôd kako bi se definirala, implementirala, ažurirala i na kraju uklanjala infrastruktura. Omogućuje automatizaciju konfiguracije i upravljanja IT infrastrukturom preko različitih okruženja i platformi. Značajno smanjuje mogućnost ljudske pogreške te povećava reproducibilnost i skalabilnost IT operacija.

Terraform i Ansible ističu se kao dva najpopularnija alata u svijetu IT industrije na području Infrastrukture kao kôda, pružajući snažne mehanizme za automatizaciju infrastrukture i konfiguraciju aplikacija. Terraform, alat razvijen od strane HashiCorpa omogućava korisnicima da deklarativnim jezikom kroz konfiguracijske datoteke definiraju i upravljaju infrastrukturom preko različitih cloud pružatelja usluga. Ansible je alat, razvijen od strane Red Hata, za automatizaciju postavljanja i konfiguriranja aplikacija na infrastrukturnim resursima.

U praktičnom dijelu ovog rada, demonstrirat će se proces podizanja dvije instance na AWS (Amazon Web Services) pomoću Terraforma. Na jednoj instanci instalirat će se Apache web server i PHP aplikacija online knjižare, dok će se na drugoj postaviti baza podataka, koristeći Ansible za automatizaciju ovih zadataka. Time će se pružiti dublji uvid u praktičnu primjenu ovih alata te demonstrirati kako se Ansible i Terraform mogu koristiti zajedno za

implementaciju aplikacija.

2. AUTOMATIZACIJA INFRASTRUKTURE: ANSIBLE I TERRAFORM

Ansible je alat za upravljanje konfiguracijom otvorenog kôda čija je svrha automatizacija različitih zadataka u upravljanju konfiguracijom, opskrbom softvera i implementaciji aplikacija na različitim računalnim sustavima. Prednosti korištenja alata za upravljanje konfiguracijom su ušteda vremena, smanjenje rizika, povećanje kontrole i automatizacija procesa koji se učestalo ponavljaju. Terraform je besplatni alat otvorenog kôda koji se koristi za automatizaciju procesa implementacije i upravljanja infrastrukturom. Infrastruktura koja se može implementirati i kojom se može upravljati pomoću Terraforma obuhvaća resurse u Cloudu, mrežne resurse, virtualne strojeve, baze podataka te razne druge usluge i aplikacije.

2.1. Ansible

Tvorac Ansiblea je Michael DeHaan, a prva stabilna verzija izašla je 2012. godine od tvrtke Ansible,Inc koju je u listopadu 2015. preuzeo Red Hat. Ansible podržava upravljanje infrastrukturom kao kôdom (IaC), potpuno je besplatan te ne zahtijeva instalaciju agenata ili dodatnog softvera na ciljanim sustavima (engl. agentless arhitektura). Koristi YAML sintaksu za definiranje konfiguracije i zadataka što olakšava čitanje i pisanje Ansible skripti (engl. playbook) budući da je sintaksa čista i lako razumljiva. Ansible se koristi direktno iz komandne linije terminala. U ovom radu će se koristiti primjeri Ansiblea na Linux platformi u Ubuntu (22.04 LTS) distribuciji.

```
ansible all -i "localhost," -m shell -a 'echo Hello Ansible'
```

Kôd 1.Primjer osnovne Ansible naredbe

U primjeru (Kôd 1), naredba ansible će se izvršiti na cijelom inventaru (popisu hostova, engl. inventory) jer se navodi all, dok se s opcijom -i radi lista u kojoj je naveden samo localhost. Opcija -m omogućava specificiranje modula koji će se koristiti. U ovom slučaju je to shell modul pomoću kojeg se izvršavaju shell komande, a pomoću opcije -a se pridaju argumenti shell modulu što je u ovom slučaju naredba echo koja ispisuje "Hello Ansible " na konzolu. Kada bi se htjelo izvršiti ovu naredbu na više sustava, iza "localhost," bi se nadodala njihova imena ili IP adrese.

Odlikuje ga idempotentnost što znači da se ista operacija može izvesti više puta bez straha od

nepotrebnih promjena (ako je sustav u željenom stanju Ansible neće iznova promijeniti konfiguraciju). Ansible je usmjeren na stanje (engl. State Driven) što znači da prvo analizira stanje pa potom odlučuje koje operacije treba odnosno ne treba izvršiti kako bi doveo sustav u željeno stanje. Velika i aktivna zajednica korisnika pruža podršku, dijeljenje iskustava i mnogo gotovih rješenja što uz jednostavnost korištenja, sigurnost, lakoću svladavanja sintakse te efikasnost čini Ansible trenutno jednim od najpopularnijih alata za automatizaciju na svijetu. Aktualna verzija u trenutku pisanja ovog rada je Ansible 8.1.0. za Community verziju i 2.15.1 za Core verziju.

2.1.1. Ansible Playbook: Automatizacija konfiguracije i zadataka

YAML (YAML Ain't Markup Language) je sintaksni jezik koji se koristi za označavanje podataka i konfiguracije. Prednost mu je laka čitljivost u usporedbi s nekim drugim formatima konfiguracijskih datoteka kao što su npr. XML ili JSON. Playbookovi u Ansibleu koriste YAML sintaksu za definiranje konfiguracije i zadataka. Playbookovi su u osnovi skripte koje Ansible izvršava kako bi automatizirao različite operacije na sustavima. Datoteka je u YAML formatu ako ima .yml ili .yaml ekstenziju i obavezno mora počinjati s tri crtice (---). Tako aplikacije koje učitavaju datoteku znaju da se radi o YAML formatu. Indentacija (uvlačenje) i razmaci označavaju strukturu datoteke. Za indentaciju se obično koristi 2 ili 4 razmaka, a važno je održavati dosljedan stil indentacije unutar iste datoteke odnosno pridržavati se jedne razine uvlačenja za cijelu datoteku. Ako je indentacija neregularna datoteka neće biti ispravno učitana. Osim toga, ne bi trebalo koristiti tabulator za uvlačenje. Komentari se pišu iza znaka # u istom retku. Kod liste članova, svaki član mora biti na istoj razini indentacije i započinjati s crticom

(-

```
---
Operating Systems:
  - Ubuntu
  - Debian
  - Red Hat
  - CentOS
  - Windows
  - Mac
```

Kôd 2.Primjer liste članova u YAML sintaksi

)(Kôd 2). Rječnike čine parovi „Ključ:Vrijednost". Ispod imena rječnika se uvlači vrijednost te dodaje ključ-vrijednost par u svaki redak (Kôd 3).

```
---
matko:
- full_name: Matko Stojić
  - age: 31
  - height: 180
   - bio: |
      Matko Stojić je rođen u Zagrebu.
      Stekao je status prvostupnika IT na Vsite-u.
  - quote: <
     " Novi početci su često prerušeni
      kao bolni završetci."
```

Kôd 3.Primjer rječnika u YAML sintaksi

Kada treba dodati više linija podataka za jedan unos koristi se znak | (engl. pipe) dok će korištenje znaka < zanemariti nove linije. YAML sintaksa ima podršku u raznim programskim

jezicima te se odlikuje lakoćom čitanja. To čini YAML popularnim izborom u skriptama za automatizaciju i konfiguracijskim datotekama.

Playbookovi su skripte pisane u YAML sintaksi koje definiraju korake koje Ansible treba slijediti kako bi postigao željeno stanje sustava. Predstavljaju način korištenja Ansiblea u obliku skripti umjesto izvođenja pojedinačnih naredbi putem naredbenog retka. Koriste se za različite svrhe: konfiguriranje poslužitelja, instalaciju softvera, ažuriranje aplikacija i dr. Deklarativni pristup kakav se koristi u Ansible playbookovima omogućava da je potrebno samo definirati željeno stanje sustava, a Ansible će se pobrinuti da ga postigne.

U primjeru u playbooka (Kôd 4), prvu liniju kôda čine tri crtice (---) čime je deklarirana YAML sintaksa. Drugom linijom kôda je određeno da će se zadatci izvršiti samo na lokalnom računalu, odnosno dodan je samo localhost u popis hostova. U trećoj liniji počinje popis zadataka koje treba izvršiti. Direktiva "-name" daje ime i opis zadatka, a naredba u liniji ispod nje ispisuje "Hello Ansible" na konzolu baš kao i Kôd 1.

```
---
- hosts: localhost
  tasks:
    - name: Pokreni Ansible naredbu iz prvog primjera
      debug: msg=" Hello Ansible"
```

Kôd 4.Primjer Ansible playbooka koji daje isti rezultat kao i Kôd 1

Ako se playbook spremi u datoteku primjer.yml pokreće ga se naredbom:

```
ansible-playbook primjer.yml -i "localhost,"
```

Umjesto da se u samoj naredbi specificiraju sustavi nad kojima će se naredba izvršiti može se kreirati inventarna datoteka s popisom računala ili poslužitelja nad kojima će se izvršavati zadatci (Kôd 5). Takva datoteka se u pravilu naziva "hosts". U inventarnoj datoteci se hostovi

```
[MojaKuća]
localhost
192.168.1.100
192.168.1.101

[webserver]
webserver1.primjer.com
webserver2.primjer.com

[database]
db1.primjer.com
db2.primjer.com
```

Kôd 5.Primjer tipične hosts datoteke

grupiraju po kategorijama koje se nalaze unutar uglatih zagrada npr. [webserver], [database], [development] itd. Ispod svake kategorije je popis IP adresa ili imena domena od računala ili poslužitelja. Ovakva "hosts" datoteka s popisom računala i poslužitelja, može se koristiti kao izvor podataka u naredbama umjesto da se ručno navode IP adrese ili nazivi uređaja. Naredba sada izgleda ovako:

```
ansible-playbook -i hosts primjer.yml
```

Ansible moduli su komponente koje omogućavaju izvršavanje različitih zadataka i operacija na ciljanim računalima ili poslužiteljima. Svaki modul predstavlja određenu funkcionalnost i zadatke koje Ansible može izvesti. Ansible dolazi s preko 1300 ugrađenih modula, ali korisnici također mogu pisati vlastite module kako bi prilagodili automatizaciju svojim specifičnim potrebama. Dokumentaciju o modulima i pluginovima raspoređenu po kategorijama moguće je

vidjeti u službenoj dokumentaciji. Evo nekoliko primjera uz objašnjenja pojedinih modula:

1. Shell

Modul shell je modul upotrijebljen u osnovom primjeru, a omogućava izvršavanje različitih operacija putem shell naredbi na ciljanim sustavima. U primjeru je shell modul korišten izravno iz naredbenog retka, a ovo je primjer korištenja unutar .yml datoteke:

```
  -name: instaliraj neki-paket
   shell: apt-get install neki-paket
```

2. Ping

Ovaj modul izvodi ping test na ciljanom sustavu odnosno provjerava je li dostupan i uključen. Primjer pinganja lokalnog računala:

```
  ansible localhost -m ping
```

Uspješna izvedba će rezultirati sljedećim ispisom:

```
localhost | SUCCESS => {
   "changed": false,
   "ping": "pong"
}
```

3. Setup

Setup modul pruža informacije o sustavu. Informacije su prikazane u JSON formatu, pri čemu vrijednosti počinju s "ansible_". Setup je koristan za prikupljanje podataka o sustavu i zatim izvođenje određenih promjena ovisno o prikupljenim informacijama. Na primjer instalacija neke aplikacije ovisno o arhitekturi operacijskog sustava. Budući da modul setup pruža preko 1000 linija informacija o sustavu u primjeru ispod je samo isječak:

```
"ansible_lsb": {
           "codename": "jammy",
           "description": "Ubuntu 22.04.3 LTS",
           "id": "Ubuntu",
```

```
        "major_release": "22",
        "release": "22.04"
    },
    "ansible_lvm": "N/A",
    "ansible_machine": "x86_64",
    "ansible_machine_id": "664209dbc4984e5badcdc85a31f8b7c0",
    "ansible_memfree_mb": 658
```


4. Apt (ili Yum)

Većina modula se može izvršavati na različitim verzijama Linuxa, ali poneki poput sustava upravljanja paketima se razlikuju za određene sustave. Apt modul se koristi za upravljanje paketima na distribucijama Linuxa koje su bazirane na Debianu (npr. Ubuntu korišten u primjerima) dok se Yum koristi kod distribucija baziranih na Red Hat-u (npr. CentOS). Ako korisnik nema administratorske ovlasti neki moduli zahtijevaju korištenje opcije "--ask-become-pass". Ova opcija omogućava Ansibleu da stekne "root" ovlasti, što znači da će tražiti lozinku kako bi dobio dozvole za instalaciju programa na sustavu.

```
ansible localhost -m apt -a "name=apache2 state=present" –ask-become-pass
```

U primjeru, naziv paketa koji će se instalirati je apache2, a stanje je "present". Za uklanjanje aplikacije koristila bi se ista naredba, ali bi se stanje moralo promijeniti u "absent".

5. Service

Modul service omogućava inicijalno i ponovno pokretanje te zaustavljanje servisa instaliranih i pokrenutih na sustavu. Argumenti za modul service su "started" (pokrenuto), "restarted" (ponovno pokrenuto) i "stopped" (zaustavljeno). Apache2 će se pokrenuti sljedećom naredbom:

```
ansible localhost -m service -a "name=apache2 state=started"
```

6. File

Za stvaranje direktorija ili datoteke u nekom okruženju koristi se modul file. Potrebno je navesti putanju direktorija ili datoteke koja će se stvoriti, a kao argumente stanje datoteke, vlasnika i

grupu. Primjer:

```
ansible localhost -m file -a "path=/tmp/primjer owner=root group=root
state=directory"
```

Osim ovih navedenih treba spomenuti: git modul pomoću kojeg se izvršavaju git naredbe putem Ansiblea, get_url modul koji dohvaća sadržaj HTTP, HTTPS i drugih internetskih protokola, user modul kojim se upravlja korisničkim računima (stvaranje, mijenjanje, brisanje, postavljanje lozinki itd.) te find modul za pronalaženje datoteka i direktorija na ciljanom sustavu (po starosti, veličini, tipu itd.).

Svaki task je pojedinačni zadatak. Zadaci se izvršavaju redoslijedom kojim su napisani, jedan po jedan, na ciljanom računalu. Navode se jedan ispod drugog nakon oznake "tasks:". Svaki zadatak bi trebao imati opciju "-name", koja se koristi za opisivanje promjena koje će se izvršiti, te odgovarajuće module koje će koristiti. Radi jasnoće poželjno je ispod modula navesti sve argumente kao listu (Kôd 6). Uvijek treba voditi računa o pravilnoj indentaciji (dosljednim razmacima) kako ne bi bilo problema prilikom izvršavanja playbooka.

```
---
- name: Konfiguracija web poslužitelja
- hosts: webserver
tasks:
- name: Instaliraj Apache web server
  apt:
    name: apache2
    state: present

- name: Postavi osnovnu konfiguraciju Apache web servera
  template:
    src: templates/apache.conf.j2
    dest: /etc/apache2/apache.conf

- name: Pokreni Apache web server
  service:
    name: apache2
    state: started
```

Kôd 6.Primjer liste playbook taskova

Notify i handleri su mehanizmi za upravljanje događajima koji reagiraju na promjene u okruženju, odnosno izvode određene akcije na ciljanim sustavima kao odgovor na promjenu prilikom izvođenja playbooka. Ključna riječ "notify" se koristi unutar zadatka (engl. task) kako bi se definiralo koji događaj će biti okidač za izvršavanje određene akcije (engl. handler) ako se dogodi promjena. Handleri su sekcija koja se nalazi obično na kraju playbooka u kojoj su definirane akcije koje će se izvršiti ako su okidači (engl. notify) aktivirani. Svaki handler ima jedinstveno ime i definira akciju koja će se izvršiti (najčešće restart nekog servisa ili ponovno pokretanje sustava). Handleri se izvršavaju samo ako su okidači aktivirani tijekom izvođenja playbooka. U primjeru (Kôd 7) je definiran handler koji će pomoću "service" modula ponovno pokrenuti Apache poslužitelj ako se primijeni konfiguracijska datoteka. Korištenje handlera uz "notify" omogućuje da se određene akcije izvrše isključivo kad su potrebne čime se poboljšava učinkovitost i kontrola nad Ansible playbookom.

```
---
- name: Konfiguracija web poslužitelja
- hosts: webserver
tasks:
- name: Postavi osnovnu konfiguraciju Apache web servera
    template:
      src: templates/apache.conf.j2
      dest: /etc/apache2/apache.conf
notify: Restart Apache

handlers:
- name: Restart Apache
    service:
      name: apache2
      state: restarted
```

Kôd 7.Primjer handlera kojeg aktivira "notify"

U Ansibleu varijable su ključni elementi koji omogućuju dinamičko ponašanje. Koriste se za pohranu i prenošenje vrijednosti poput putanja, imena datoteka, IP adresa, lozinki i dr. Varijable se deklariraju i dodjeljuje im se vrijednost unutar playbooka, rola i konfiguracijskih datoteka. Primjer:

```
varijabla: "vrijednost"
```

Korite se u raznim dijelovima playbooka, taskovima, templateima, uvjetima, putanjama, raznim modulima i dr. Primjer upotrebe:

```
-name: Primjer taska s varijablom
 debug:
   msg: "Vrijednost varijable: {{ varijabla }}"
```

Postoje unaprijed deklarirane varijable (tipično informacije o sustavima i okolini), globalne varijable, varijable role, varijable modula i dr. Korištenje varijabli pruža fleksibilnost i

omogućuje korištenje iste konfiguracije na različitim sustavima s različitim vrijednostima varijabli.

2.1.2. Organizacija playbooka: Roles, include, import

Načini na koje se Ansible može upotrijebiti za višekratnu upotrebu su korištenjem includea i importa te putem rolesa. Ciljani sustavi se onda mogu konfigurirati na temelju specifičnih potreba ili uvjeta.

Roles su način organizacije i strukturiranja Ansible playbooka. Prednosti rolesa su: modularnost, ponovna upotreba podataka, čitljivost i održivost. Roles omogućuju organizaciju taskova i konfiguracije u logičke jedinice. Svaka se može fokusirati na specifičan aspekt sustava ili zadatka (modularnost) što playbooke čini čitljivijima i lakšima za održavanje. Mogu se lako ponovno koristiti u sklopu različitih playbooka ili projekata što znači dijeljenje i primjenu konfiguracije između različitih okruženja. Njihovo pravilno imenovanje i dokumentiranje čini playbooke čitljivijima i lakšima za razumijevanje. Tipična role ima strukturu direktorija podijeljenog na specifične poddirektorije za različite vrste komponenti: default, files, handlers, meta, tasks, templates, vars.

```
roles/web_server/
├──defaults
├──files
│ ├── 000-default.conf
│ └── index.py
├── handlers
│ └── main.yml
├── meta
├── tasks
│ └── main.yml
├── templates
└── vars
```

Kôd 8.Primjer poddirektorija roles web server

Default sadrži zadane varijable za role, files sadrži datoteke koje se kopiraju na ciljanje sustave,

u handlers direktoriju se definiraju handleri koji reagiraju na događaje, u meti se definiraju meta podatci npr. ovisnosti, tasks sadrži taskove koji se izvode unutar role, templates služi za pohranu predložaka a u varsu se mogu definirati dodatne varijable. U primjeru (Kôd 8) roles/web_server/tasks/main.yml sadrži isključivo zadatke, dok ~/handlers sadrži isključivo handlere , a ~/files sadrži konfiguracijske datoteke. Kako bi se pokrenula role potrebno je kreirati playbook npr. pokretac.yml:

```
---
- hosts: localhost
    roles:
      -web_server
```

Playbook se pokreće sa:

```
ansible-playbook -i hosts pokretac.yml
```

Ansible Galaxy je online platforma putem koje korisnici mogu pretraživati, dijeliti i ponovno koristiti već gotova rješenja koja se mogu brzo integrirati u postojeće playbooke i projekte. Korisnici mogu jednostavno instalirati roles pomoću naredbe "ansible-galaxy". Ansible Galaxy roles su često povezane s Git repozitorijima pa je sadržaj moguće pratiti i ažurirati direktno iz Git repozitorija. Mogućnost ocjenjivanja rolesa pomaže korisnicima da prepoznaju kvalitetne i pouzdane resurse. Ansible Galaxy olakšava upotrebu Ansiblea za konfiguraciju, upravljanje i automatizaciju, ubrzava razvoj, štedi vrijeme, čini ga pristupačnijim za korisnike i omogućuje im da koriste tuđi rad, ali i da doprinose zajednici.

Drugi način ponovne upotrebe kôda su opcije include i import. S rastom kompleksnosti okruženja raste i kôd unutar playbooka i dolazi do nepreglednosti i problema s organizacijom. Obje opcije omogućuju izdvajanje kôda u manje module ili datoteke radi lakšeg održavanja i ponovne upotrebe što bitno poboljšava organizaciju kôda unutar playbooka. Razlika je u načinu obrade tijekom izvođenja playbooka. Opcija "include" se obrađuje tijekom izvođenja. Omogućuje dinamičko odlučivanje koji taskovi će se izvršiti na temelju zadanih uvjeta ili varijabli. Primjer "include" opcije:

```
- name: Izvrši task ovisno o uvjetu
```

```
  include_tasks: task.yml
  when: neka_varijabla='određena_vrijednost'
```

Opcija "import" se obrađuje prilikom parsiranja playbooka, kad playbook učita kôd prije izvršenja. Koristi se za uvoz cijelih playbooka ili YAML datoteka kako bi se bolje strukturirao kôd. Česta upotreba kod većih projekata, gdje svaki playbook ima svoju specifičnu svrhu. Primjer opcije "import":

```
- import_playbook: task.yml
```

U obje opcije se mogu definirati dodatni uvjeti, varijable i argumenti. Odabir opcije ovisi o potrebama projekta i složenosti playbookova.

2.2. Terraform

Terraform definira i upravlja infrastrukturom koristeći jednostavnu deklarativnu sintaksu. Stvorila ga je kompanija HashiCorp 2014. godine i napisan je u programskom jeziku Go. Deklarativnim jezikom zadaje se krajnje stanje koje se želi postići, a alat se sam pobrine kako će to postići. Piše se u HCL (HashiCorp Configuration Language) jeziku u bilo kojem tekst editoru, a većina ih ima podršku za HCL sintaksu (vim, emacs, Atom, Visual Studio Code, IntelliJ). Datoteke imaju nastavak .tf. Može kreirati infrastrukturu putem raznih providera (pružatelja usluga), kao što su: AWS, Azure, Google Cloud, DigitalOcean i drugi. Čitljiv format i čisti izgled kôda olakšavaju korisnicima definiranje resursa i njihovih atributa te odnosa između njih. Isto kao i Ansible, Terraform ne zahtijeva instalaciju agenata na ciljanim sustavima (engl. agentless) i podržava IaC pristup, što omogućava reproducibilnost, skalabilnost, te lakše upravljanje infrastrukturom. je primjer konfiguracije za virtualne instance

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "primjer" {
  ami           = "ami-008fe2fc65df48dac"
  instance_type = "t2.micro"
}
```

Kôd 9.Primjer Terraform konfiguracije za stvaranje virtualne mašine na AWS-u

na AWS-u. Kôd 9 je primjer konfiguracije virtualne mašine na AWS-u.

Poput Ansiblea, i Terraform odlikuje idempotentnost. Ista konfiguracija se može izvršiti više puta bez neželjenih promjena ako je resurs već u željenom stanju. Prije nego se primijeni promjena Terraform generira plan promjena u kojem se jasno vidi što će se točno promijeniti prije nego što se one zaista dogode. Terraform podržava modularnost putem modula koji omogućavaju ponovno korištenje konfiguracijskih blokova. Kôd je tako bolje organiziran i olakšano je upravljanje velikim projektima. Terraform ima veliku i aktivnu zajednicu koja je u stalnom porastu. Zajednica pruža podršku, daje gotova rješenja i dijeli iskustva i time čini Terraform vodećim alatom za upravljanje infrastrukturom kao kôdom u industriji. Aktualna verzija u trenutku pisanja ovog rada je Terraform v1.6.6.

2.2.1. Prikaz osnovnih naredbi kroz implementaciju virtualnog servera

U ovom radu će se koristiti primjeri Terraforma na Linux platformi u Ubuntu (verzije 22.04 LTS) distribuciji i AWS kao provider (potrebno je imati otvoren račun na AWS-u). Da bi Terraform mogao vršiti promjene na AWS računu potrebno je postaviti AWS credentials za željenog korisnika u obliku environment varijabli AWS_SECRET_ACCESS_KEY i AWS_ACCESS_ID. Postavke će biti privremene i vrijedit će samo za trenutnu sesiju terminala.

$export AWS_ACCESS_KEY_ID= (vaš ID pristupnog ključa)
$export AWS_SECRET_ACCESS_KEY= (vaš tajni pristupni ključ)

U praznom folderu kreira se datoteka "main.tf". Prvi korak je konfiguriranje providera koji će se koristiti. U primjeru ispod provider "aws" je konfiguracija za AWS (Amazon Web Services) u Terraformu. Unutar iste se navodi regija u kojoj će se izvršavati operacije. AWS ima podatkovne centre diljem svijeta, grupirane u regije. U primjeru ispod koristi se "us-west-2".

```
provider "aws" {
  region = "us-west-2"
}
```

Za svaki tip providera postoje različite vrste resursa koji se mogu kreirati (npr. serveri, load balanceri, baze podataka itd.). Opća sintaksa za kreiranje resursa u Terraformu je:

```
resource "<PROVIDER>_<TIP>" "<IME>" {
```

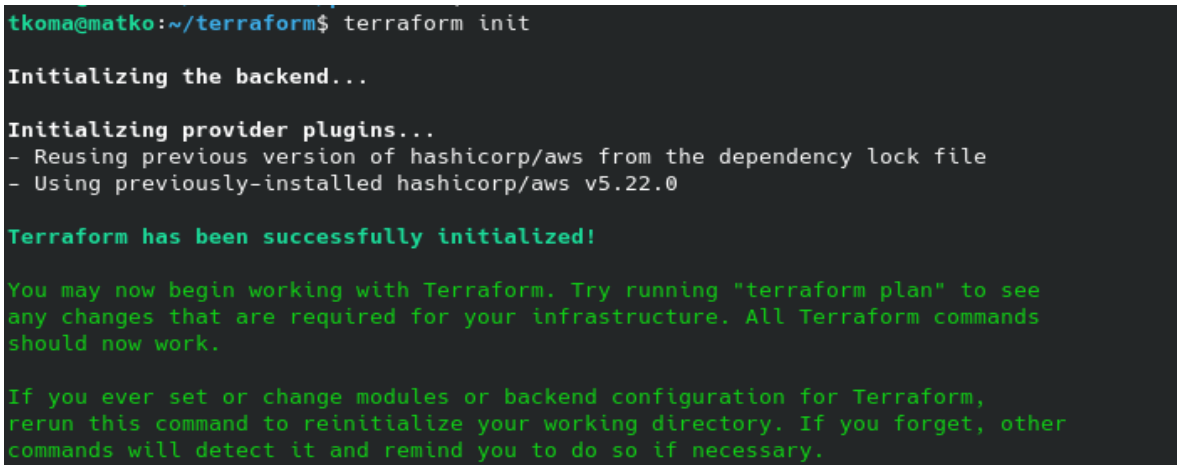
```
  [CONFIG ...]
}
```

PROVIDER je ime providera (npr. AWS). TIP je tip resursa koji će se kreirati putem tog providera (npr. instance). IME je identifikator koji se koristi u kôdu kako bi se referirao ovaj resurs. CONFIG je konfiguracija specifična za pojedini resurs koja se sastoji od jednog ili više argumenata. Primjer:

```
resource "aws_instance" "primjer" {
  ami           = "ami-008fe2fc65df48dac"
  instance_type = "t2.micro"
  tags = {
    Name = "terraform-primjer"
  }
}
```

Resource definira resurs koji će se koristiti, u ovom slučaju "aws_instance" (virtualni server), a "primjer" je odabrani identifikator za resurs. Ami specificira ID Amazon Machine Image, predložak na osnovu kojeg će se stvoriti nova EC2 instanca (virtualni server). Svaka AMI ima jedinstveni ID koji predstavlja različite konfiguracije operativnih sistema (razne verzije Linuxa, Windowsa ili drugih operativnih sistema) s unaprijed instaliranim softverom. Za specifični ID u konfiguraciji Terraforma stvorit će se nova EC2 instanca koristeći taj predložak kao osnovu. ID "ami-008fe2fc65df48dac", koji je korišten u primjeru je parametar koji će stvoriti "sliku" s Ubuntu 22.04 LTS operativnim sustavom i njegovim softverskim komponentama za "us-east-2" regiju. Parametar instance_type određuje vrstu EC2 instance. Ovdje zadani "t2.micro" označava tip instance s određenim karakteristikama procesora i memorije (1 GB RAM-a i jedna virtualna procesna jedinica), prostora na disku i kapaciteta mreže. Tags služe kao metapodaci koji služe za identifikaciju, organizaciju i označavanje resursa unutar infrastrukture. Par ključ-vrijednost u primjeru (Name="terraform-primjer") će dodijeliti ime instanci. Na AWS-ovim stranicama nalazi se popis svih vrsta instanci. Terraform podržava desetke pružatelja usluga, od kojih svaki podržava desetke resursa, a svaki resurs ima desetke argumenata. Nemoguće ih je sve zapamtiti, stoga se preporučuje redovito koristiti Terraform dokumentaciju kako bi se znalo koji resursi stoje na raspolaganju i kako koristiti svaki od njih.

Binarna datoteka terraform sadrži osnovne funkcionalnosti, ali ne dolazi s kôdom za pružatelje usluga (razne providere), stoga, kada se prvi put koristi neki Terraform kôd potrebno je prvo pokrenuti "terraform init" naredbu kako bi Terraform pregledao kôd, otkrio koji pružatelj usluga se koristi i preuzeo njegov kôd. Kôd pružatelja usluga bit će preuzet u direktorij ".terraform" (radni direktorij Terraforma) i u "terraform.lock.hcl" datoteci. Terminal se pozicionira u direktorij u kojem se nalazi main.tf datoteka, zatim se pokreće naredba "terraform init" (Slika 1):



Slika 1: Primjer Terraform init naredbe.

Nakon što se inicijalizira direktorij, dobro je pokrenuti naredbu "validate". Naredba će napraviti validaciju, odnosno uhvatit će sintaksne pogreške, pogreške verzija i druge probleme. Ne može se pokrenuti prije naredbe "terraform init". Ako je sve u redu ispisat će se poruka "The configuration is valid.".

Naredba "terraform plan" (Slika 2) prikazuje što će Terraform napraviti prije nego što zapravo napravi bilo kakve promjene. To je jedna od prednosti Terraforma koja pruža mogućnost provjere ispravnosti kôda prije što se primjeni na stvarni sustav. Izlaz naredbe "terraform plan", sličan je izlazu naredbe "diff" na Linuxu ili Gitu: sve uz što stoji znak (+) bit će kreirano, sve uz što stoji znak (-) bit će uklonjeno, a sve što stoji uz znak (~) bit će modificirano.

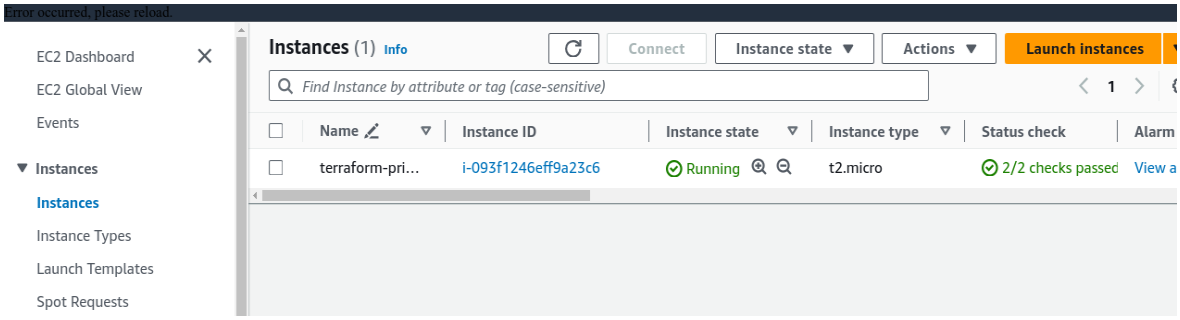
```
tkoma@matko:~/terraform$ terraform plan
data.aws_vpc.default: Reading...
data.aws_vpc.default: Read complete after 1s [id=vpc-06550f41a63774e4c]
data.aws_subnets.default: Reading...
data.aws_subnets.default: Read complete after 1s [id=us-east-2]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_autoscaling_group.primjer will be created
  + resource "aws_autoscaling_group" "primjer" {
      + arn                               = (known after apply)
      + availability_zones                = (known after apply)
      + default_cooldown                  = (known after apply)
      + desired_capacity                  = (known after apply)
      + force_delete                      = false
      + force_delete_warm_pool            = false
      + health_check_grace_period         = 300
      + health_check_type                 = "ELB"
      + id                                = (known after apply)
      + ignore_failed_scaling_activities = false
      + launch_configuration              = (known after apply)
      + load_balancers                    = (known after apply)
      + max_size                          = 10
```

Slika 2: Primjer naredbe terraform plan.

Kako bi se zaista kreirao virtualni server pokreće se naredba "terraform apply" (Slika 3). Ona će prikazati isti izlaz kao i naredba "terraform plan" samo što će tražiti potvrdu da nastavi s planom. Ako je korisnik suglasan s promjenama treba upisati "yes":

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_security_group.alb: Creating...
aws_security_group.instance: Creating...
aws_lb_target_group.asg: Creating...
aws_lb_target_group.asg: Creation complete after 2s [id=arn:aws:elasticloadbalancing:us-east-2:228339
231:targetgroup/terraform-asg-example/b5087d0c2447282e]
aws_security_group.instance: Creation complete after 4s [id=sg-0a3dd44ce6faaae3e]
aws_launch_configuration.primjer: Creating...
aws_security_group.alb: Creation complete after 4s [id=sg-0b4afc0896e6a858a]
aws_lb.example: Creating...
aws_launch_configuration.primjer: Creation complete after 1s [id=terraform-20240616204605631400000001
aws_autoscaling_group.primjer: Creating...
aws_lb.example: Still creating... [10s elapsed]
aws_autoscaling_group.primjer: Still creating... [10s elapsed]
aws_lb.example: Still creating... [20s elapsed]
aws_autoscaling_group.primjer: Still creating... [20s elapsed]
aws_lb.example: Still creating... [30s elapsed]
aws_autoscaling_group.primjer: Still creating... [30s elapsed]
aws_lb.example: Still creating... [40s elapsed]
aws_autoscaling_group.primjer: Still creating... [40s elapsed]
aws_autoscaling_group.primjer: Creation complete after 40s [id=terraform-20240616204607108400000002]
aws_lb.example: Still creating... [50s elapsed]
```

Slika 3: Primjer terraform apply naredbe.



Slika 4. AWS konzola prikazuje EC2 instancu koju smo implementirali

Slika 4 prikazuje kreiranu EC2 instancu u AWS konzoli. Ako se želi ukloniti infrastruktura, upotrijebit će se naredba "`terraform destroy`" (Slika 5). Ovom naredbom Terraform analizira svoju konfiguraciju i identificira sve resurse koje je stvorio na temelju te konfiguracije. Opcija "`-target`" omogućava uklanjanje samo pojedinih resursa, inače će svi biti uklonjeni. Nakon pokretanja naredbe prvo će na izlazu biti prikazan popis svih resursa i tražit će se potvrda od korisnika da ih želi ukloniti. Nakon potvrde, Terraform će početi s rušenjem svih identificiranih resursa.

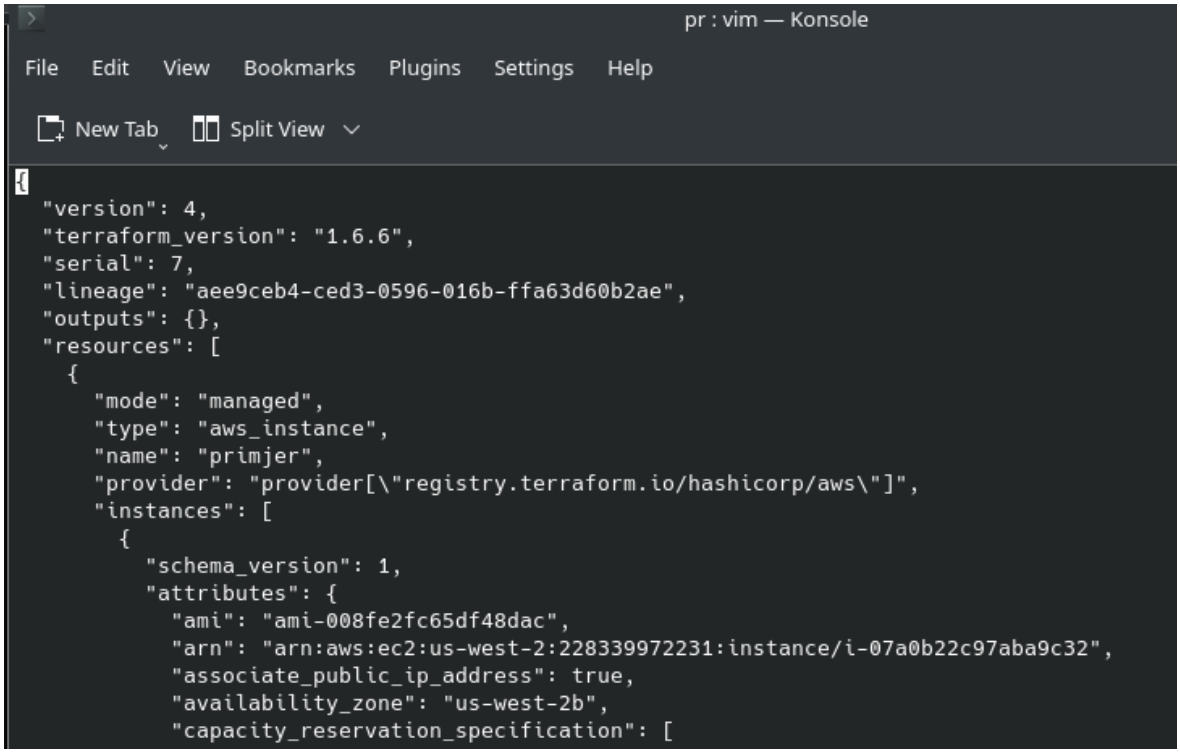
```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_lb_listener_rule.asg: Destroying... [id=arn:aws:elasticloadbalancing:us-east-2:228339972231:liste
-rule/app/terraform-asg-example/7faffabe2f9c4378/5712d01c9cb07153/90bd337a15e5379c]
aws_autoscaling_group.primjer: Destroying... [id=terraform-20240616204607108400000002]
aws_lb_listener_rule.asg: Destruction complete after 1s
aws_lb_listener.http: Destroying... [id=arn:aws:elasticloadbalancing:us-east-2:228339972231:listener/a
/terraform-asg-example/7faffabe2f9c4378/5712d01c9cb07153]
aws_lb_listener.http: Destruction complete after 0s
aws_lb.example: Destroying... [id=arn:aws:elasticloadbalancing:us-east-2:228339972231:loadbalancer/app
erraform-asg-example/7faffabe2f9c4378]
aws_lb.example: Destruction complete after 4s
aws_security_group.alb: Destroying... [id=sg-0b4afc0896e6a858a]
aws_autoscaling_group.primjer: Still destroying... [id=terraform-20240616204607108400000002, 10s elaps
]
aws_security_group.alb: Still destroying... [id=sg-0b4afc0896e6a858a, 10s elapsed]
aws_autoscaling_group.primjer: Still destroying... [id=terraform-20240616204607108400000002, 20s elaps
]
aws_security_group.alb: Still destroying... [id=sg-0b4afc0896e6a858a, 20s elapsed]
aws_autoscaling_group.primjer: Still destroying... [id=terraform-20240616204607108400000002, 30s elaps
]
```

Slika 5: Primjer terraform destroy naredbe

2.2.2. Terraform state

Svaki put kada se pokrene neka Terraform naredba bilježe se informacije o infrastrukturi koju je kreirana i pohranjuje ih se u datoteku stanja, odnosno u *Terraform state file*. U svakom direktoriju u kojem se izvršava Terraform stvorit će se datoteka terraform.tfstate, npr. u

direktoriju /primjer stvorit će se datoteka /primjer/terraform.tfstate. Terraform state file je strukturiran u JSON formatu i služi kao mapa resursa definiranih u Terraform konfiguracijskim datotekama (Slika 6). Svaki resurs ima svoj jedinstveni identifikator i povezan je s odgovarajućim elementom u stvarnom svijetu, npr. ID resursa u određenom cloud okruženju. Na taj način Terraform prati stanje infrastrukture i uspoređuje ga s definicijama u kôdu. Kada se ponovno pokrene, Terraform koristi informacije iz datoteke stanja kako bi procijenio postojeće resurse i utvrdio koje promjene treba primijeniti. Pomoću ove datoteke Terraform prati i održava usklađenost između željenog stanja definiranog u kôdu i trenutačnog stanja infrastrukture. Izlaz naredbe "terraform plan" zapravo predstavlja razliku između kôda na računalu i infrastrukture implementirane u stvarnom svijetu, otkrivene putem identifikatora u datoteci stanja. Terraform State tako postaje ključni element u procesu planiranja i izvršavanja promjena, pružajući transparentnost i kontrolu korisnicima.



Slika 6: Primjer Terraform state datoteke.

Radi bolje organizacije, Terrafom omogućuje stvaranje odvojenih datoteka stanja za svako okruženje. Razvojno, testno i produkcijsko okruženje mogu imati vlastite datoteke stanja, što pridonosi boljoj izolaciji promjena i smanjuje rizik od grešaka u infrastrukturnim promjenama.

2.2.3. Terraform moduli

U općenitom programskom jeziku kao što je npr. C++ ili Python, kada se želi izbjeći gomilanje kôda koji se ponavlja on se stavlja funkciju, umjesto da se kopira isti ili sličan dio kôda. Kod

koji se želi ponovno koristiti stavlja se unutar Terraform modula. Bilo koji skup Terraform konfiguracijskih datoteka u određenom direktoriju smatra se modulom. Konfiguracija iz prijašnjeg primjera također se smatra modulom. Kada se naredba "`terraform apply`" pokrene direktno na modulu, on se naziva root modul. Ako se želi stvoriti ponovno upotrebljiv modul, tada je potrebno stvoriti reusable modul, odnosno modul namijenjen korištenju unutar drugih modula. Svi se moduli obično drže u direktoriju "`/modules`". Osnovna sintaksa modula:

```
module "<NAME>" {
  source = "<SOURCE>"

  [CONFIG...]
}
```

NAME je identifikator koji se koristi u daljnjem kôdu kako bi se referencirao pojedini modul. SOURCE je putanja na kojoj se nalazi kôd modula, a CONFIG se sastoji od argumenata koji su specifični za taj modul. Kada bi se ponovno upotrijebio modul u nekoj drugoj datoteci npr. stage/virtual-machine/main.tf:

```
provider "aws" {
  region = "us-west-2"
}

module "instance" {
  source = "./modules/virtual-machine"
}
```

Svaki puta kada se doda modul u Terraform konfiguraciju ili promijeni parametar source modula, potrebno je prvo pokrenuti naredbu "init" prije naredbi "plan" i "apply".

3. RAZLIKE I USPOREDBA TERRAFORMA I ANSIBLEA

Ansible je alat za upravljanje konfiguracijom, a Terraform alat za isporuku infrastrukture. Unatoč toj temeljnoj razlici, ovi alati dijele mnogo sličnosti. Oba alata su sposobna obavljati dobavljanje infrastrukture i konfiguriranje iste s potrebnim komponentama i aplikacijama. Oba alata su agentless, odnosno ne trebaju implementirati agente na ciljanim sustavima kako bi djelovali. Oba koriste SSH autentikaciju pri komunikaciji s ciljanim sustavima. Nijedan ne koristi neki oblik centralnog upravljačkog čvora (engl. masterless). To znači da ne zahtijevaju sustav za upravljanje stanjem, odnosno da koriste lokalne ili distribuirane resurse za praćenje i održavanje trenutnog stanja infrastrukture. Postoje i značajne razlike.

3.1. Razlike

Unatoč brojnim sličnostima, Ansible i Terraform se razlikuju u svojim primarnim funkcijama i pristupima upravljanju infrastrukturom. Terraform je primarno usmjeren na deklarativno definiranje infrastrukture i upravljanje stanjem kroz datoteke, dok Ansible koristi proceduralni pristup za konfiguriranje postojećih resursa. Terraform stvara i održava infrastrukturu kroz definiciju željenog stanja, dok Ansible izvršava zadatke u stvarnom vremenu, često bez potrebe za praćenjem stanja infrastrukture.

3.1.1. Mutabilna nasuprot nepromjenjivoj infrastrukturi

Implementacija neke aplikacije obuhvaća stvaranje infrastrukture i instalaciju odgovarajuće verzije izvornog kôda te pripadajućih ovisnosti (engl. dependencies) na prethodno stvorenoj infrastrukturi. Mutabilnost je svojstvo koje određuje način implementacije novih verzija aplikacija i usluga. Implementacija se može vršiti na postojećoj infrastrukturi ili se može osigurati potpuno nov skup infrastrukturnih elemenata. Praksa implementacije određuje je li infrastruktura mutabilna ili nepromjenjiva. Kada se nove verzije aplikacija objavljuju na istoj infrastrukturi, ona se naziva mutabilnom. Ako je za implementaciju potrebno osigurati potpuno novu infrastrukturu kaže se da je nepromjenjiva. Iako se mutabilna infrastruktura doima praktičnijom, nosi veći rizik od neuspjeha. Kada se konfiguracije neke aplikacije ponovno primjenjuju na istoj infrastrukturi postoji dodatan korak deinstalacije prijašnje verzije pa tek onda instalacije željene verzije. Više koraka nosi i veće šanse za neuspjeh. Obavljanje ovakvih radnji na skupu servera može rezultirati neujednačenim konfiguracijama. Ako se umjesto toga instalacija vrši na novim resursima infrastrukture, može se provesti testiranje, i po potrebi, poništiti nova implementacija. Tako se dobiva veća kontrola nad unošenjem promjena. Nijedan spominjani alat nije sam po sebi niti mutabilan niti nepromjenjiv, već to ovisi o pristupu i strategiji koju korisnik primjenjuje koristeći ove alate. Terraform bolje podržava nepromjenjivu

infrastrukturu budući da puno lakše stvara potpuno nov skup infrastrukture i uklanja stari. Taj postupak troši vrijeme i resurse te nije praktičan za česte zadatke poput npr. ažuriranja vatrozida. Ansible vrlo brzo popravlja ili primjenjuje promjene i ovisnosti na ciljanim sustavima bez potrebe implementiranja svega ispočetka što ga čini vrlo mutabilnim. Nepromjenjiva infrastruktura je pouzdanija i dosljednija, ali ju je teže modificirati kada to postane potrebno. Stoga je najbolji pristup s nepromjenjivom infrastrukturom u kojem Terraform opskrbljuje resursima, a Ansible primjenjuje promjene u konfiguraciji na postojećoj infrastrukturi.

3.1.2. Deklarativni nasuprot proceduralnom jeziku

Ansible koristi proceduralni pristup u kojem kôd specificira, korak po korak, kako postići određeno stanje. Terraform koristi deklarativni stil u kojem kôd specificira samo krajnje željeno stanje, a sam alat je odgovoran za određivanje načina kako postići isto. U primjeru će biti demonstrirana razlika. Ako je potrebno implementirati 10 virtualnih servera (EC2 na AWS-u) za pokretanje AMI (Amazon Machine Image) s ID-om Ubuntua 22.04 LTS, proceduralni pristup koristeći Ansible izgleda ovako:

```
- ec2:
    count: 10
    image:  ami-008fe2fc65df48dac
    instance_type: t2.micro
```

Postizanje istoga s Terraformom izgleda ovako:

```
resource "aws_instance" "primjer" {
  count         = 10
  ami           = "ami-008fe2fc65df48dac"
  instance_type = "t2.micro"
}
```

Ova dva pristupa polučit će isti rezultat. Do razlike dolazi kada se napravi promjena. Kao npr. povećanje broja servera na 15. S Ansibleom, ako se ažurira broj servera na 15 i ponovno pokrene kôd stvorit će se novih 15 servera. Dakle ukupno 25. Zato je potrebno uzeti u obzir već implementirane servere i napisati novi kôd za dodavanje 5 novih servera.

```
- ec2:
```

```
    count: 5
    image:  ami-008fe2fc65df48dac
    instance_type: t2.micro
```

S deklarativnim kôdom, potrebno je samo definirati krajnje stanje koje želimo postići budući da je Terraform svjestan svakog stanja koje je stvorio u prošlosti. Za implementaciju dodatnih 5 servera, potrebno je u istoj konfiguraciji ažurirati broj s 10 na 15.

```
resource "aws_instance" "primjer" {
  count         = 15
  ami           = "ami-008fe2fc65df48dac"
  instance_type = "t2.micro"
}
```

Ako se primijeni ova konfiguracija, Terraform će prepoznati da je već implementirano 10 servera i da treba samo dodati 5 novih. Za implementaciju drugog AMI-a s Ansibleom, potrebno je stvoriti novi predložak koji će pronaći tih 10 prethodno implementiranih servera i ažurirati ih na novu verziju. S deklarativnim pristupom Terraforma, u istoj konfiguracijskoj datoteci bi samo trebalo promijeniti parametar ami. Terraformov kôd, kada se aplicira, uvijek predstavlja najnovije stanje infrastrukture. Lakše je utvrditi što je implementirano i kako je konfigurirano, što olakšava stvaranje ponovno upotrebljivog kôda. Iz tog razloga Terraform kôdovi obično ostaju mali i lako razumljivi. U konačnici, deklarativni pristup čini Terraform superiornijim rješenjem za implementaciju infrastrukture.

3.1.3. Upravljanje stanjem

Terraform upravlja cjelokupnim životnim ciklusom resursa koji su pod njegovim nadzorom. Datoteke stanja (engl. state file) sadrže presliku resursa trenutne konfiguracije datoteka. Terraform uspoređuje podatke iz datoteke stanja s postojećim resursima i prilagođava resurse infrastrukture prema potrebi. Sva odstupanja i promjene infrastrukture pohranjuju se u datoteku. U svakom trenutku moguće je napraviti upit u datoteku stanja Terraforma kako bi se saznale komponente infrastrukture i njihovi trenutni atributi. Za razliku od toga, Ansible ne podržava upravljanje životnim ciklusom i ne prati postojeće stanje konfiguracije. Promjene koje su namijenjene izvršavaju se automatski na svim ciljanim sustavima. Potrebno je izvršiti upit u trenutno stanje kako bi se saznale pojedinosti.

U konačnici, Terraform pruža sveobuhvatno rješenje za upravljanje životnim ciklusom infrastrukture putem datoteka stanja, dok Ansible, fokusiran na konfiguracijsko upravljanje, automatski primjenjuje promjene bez praćenja stanja.

4. PRAKTIČNI RAD – TERRAFORM I ANSIBLE U PRAKSI

Ovaj dio rada demonstrira upotrebu oba alata u praksi. Uz pomoć Terraforma stvorit će se dvije virtualne instance putem AWS providera. Nakon toga će se povezati s njima putem SSH autentifikacije s lokalnog stroja. Na jednoj instanci izvršit će se Ansible playbook koji će instalirati Apache web server, PHP i druge potrebne komponente. Nakon toga klonirat će se aplikacija s GitHuba, osigurati prava pristupa potrebnim direktorijima te pokrenuti web server. Također, bit će napravljene potrebne promjene vezane za bazu podataka i ažurirat će se connectDB.php datoteka s novim podacima o bazi podataka. Ovaj playbook instalira, konfigurira i osigurava rad PHP Bookstore aplikacije na Apache web serveru, omogućujući razvoj i testiranje PHP aplikacije na virtualnoj instanci. Na drugoj instanci izvršit će se Ansible playbook koji će instalirati MySQL server i Python3-mysql modul, a zatim pokrenuti MySQL server i omogućiti mu automatsko pokretanje. Slijedi kreiranje .my.cnf datoteke uz pomoć template modula koja sadrži lozinku root korisnika. Nakon toga pomoću mysql_user modula postavit će se lozinka za root korisnika, a zatim dodati novog korisnika pod imenom 'bookstore_user', te MySQL bazu podataka s nazivom 'bookstore'. Nakon toga preuzet će se SQL 'dump' s GitHub repozitorija na instancu. Potom će se napraviti izmjene na SQL 'dump' datoteci i ažurirati mysql.conf.d datoteka kako bi se omogućilo povezivanje s bazom izvan lokalnog računala. MySQL server će se ponovno pokrenuti. Završni korak je uvoz SQL 'dumpa' u MySQL bazu podataka 'bookstore'. Ovaj playbook automatski postavlja bazu podataka, novog korisnika, konfiguracije potrebne za pristup bazi te uvozi podatke iz SQL 'dumpa' u bazu. Time je MySQL baza podataka potrebna za funkcioniranje PHP Bookstore aplikacije uspješno postavljena i konfigurirana prema potrebama aplikacije. Direktorij u kojem se nalaze datoteke i poddirektoriji za konfiguraciju infrastrukture uz pomoć Terraforma i upravljanje istom s Ansibleom sadrži: ansible.cfg, main.tf, /roles, security_group.tf, terraform.tfstate.backup, terraform.tfstate, install_apache_and_php.yml, mysql.yml i root_cnf.j2. Direktorij /roles sadrži dvije role: apache_php i mysql.

4.1. Terraform konfiguracija

Ovaj skup datoteka sadrži glavnu konfiguraciju za stvaranje AWS resursa, uključujući definiranje virtualnih instanci i određivanje sigurnosne grupe. Ovdje će se i lokalno računalo povezati s instancama putem SSH protokola i izvršiti Ansible playbooke na istima.

4.1.1. Datoteka main.tf

```
provider "aws" {
 region = "us-west-2"
}
resource "aws_instance" "apache" {
   ami = "ami-073ff6027d02b1312"
   count = 2
   instance_type = "t2.micro"
   associate_public_ip_address = true
   vpc_security_group_ids = [aws_security_group.apache.id]
   key_name = "web_server_key"
}
```

Kôd 10.Konfiguracija providera i definicija AWS instance resursa

U prvom bloku (Kôd 10) odabran je AWS kao provider i određena je regija u kojoj će se stvoriti resursi. U drugom bloku (Kôd 10) definirani su atributi AWS instanci: ami ID je postavljen na Ubuntu 22.04 LTS, count = 2 označava da će se stvoriti dvije instance, tip instance je t2.micro, associate_public_ip_address je postavljen na true što znači da će instance dobiti javnu IP adresu, vpc_security_groups_id je lista sigurnosnih grupa koje će biti dodijeljene instancama, gdje je korištena referenca na sigurnosnu grupu aws_security_group.apache.id, a key_name je naziv ključa (engl. key pair) koji će se koristiti za SSH pristup instancama.

Ovaj blok kôda (Kôd 11) definira null resurs "wait_for_ssh". Null resursi su resursi koji ne stvaraju stvarne resurse, već se koriste za izvršavanje naredbi lokalno na Terraform hostu ili udaljenim resursima. U ovom slučaju to će biti uspostava SSH veze s obje instance. Terraform će pričekati dok svaka instanca ne bude dostupna za SSH konekciju prije nego što nastavi s daljnjim izvršavanjem naredbi.

```
  resource "null_resource" "wait_for_ssh" {
    count = 2
    provisioner "remote-exec" {
      inline = ["echo 'Wait until SSH is ready!'"]
```

```
    connection {
       type = "ssh"

       user = "ubuntu"
       private_key=file("/home/tkoma/AWS_keys/
       web_server_key.pem")
       host = aws_instance.apache[count.index].public_ip
       }
    }
}
```

Kôd 11.Null resurs za čekanje i spajanje s instancama putem SSH protokola

Ponekad je potrebno više vremena da instanca bude dostupna putem SSH protokola, npr. ako se radi o većem broju instanci ili ako je sustav pod većim opterećenjem. Parametar count je postavljen na 2, što znači da će se null resurs primijeniti na svaku od dvije stvorene instance. U remote-exec bloku definiran je "provisioner" koji će izvršiti naredbe na udaljenom računalu. Inline predstavlja naredbe koje će se izvršiti, u ovom slučaju će echo naredba ispisati tekst. U bloku connection su postavke veze za SSH: type je vrsta veze ovdje postavljeno na "ssh", user je korisničko ime koje će se koristiti prilikom autentifikacije ovdje postavljeno na "ubuntu" jer je "ubuntu" zadano ime za ovu vrstu instance, private key predstavlja putanju ključa na lokalnom računalu koji će se koristiti za uspostavu veze, host je javna IP adresa instance na koju će se uspostaviti SSH veza. Korištenje count indexa omogućava dinamički odabir adrese svake instance prema njenom indeksu. Na primjer za prvu instancu će vrijediti count.index=0, odnosno koristit će se prva javna IP adresa.

```
output "second_instance_ip"{
    value = aws_instance.apache[1].public_ip
}
```

Ovaj blok kôda predstavlja izlaznu varijablu koja će biti prikazana nakon izvršavanja kôda. Naziv second_instance_ip koristit će se kada bude potrebno dohvatiti vrijednost varijable. Vrijednost, odnosno value, postavljena je na javnu IP adresu druge instance (indeks [1]). Ovo će biti korisno kada bude potrebno proslijediti tu vrijednost kao argument Ansible playbooku.

```
resource "null_resource" "delay" {
    depends_on = [aws_instance.apache]

    provisioner "local-exec" {
      command = "sleep 60"
   }
resource "null_resource" "delay" {
    depends_on = [aws_instance.apache]

    provisioner "local-exec" {
      command = "sleep 60"
    }
}
```

Kôd 12.Null resurs delay koji služi za vremenski odmak prije daljenjeg izvršavanja

Još jedan null resurs koji će ovog puta poslužiti da stvori vremenski odmak prije daljnjeg izvršavanja naredbi (Kôd 12). Linija depends_on definira ovisnost resursa. Ona uvjetuje da se ovaj null resurs izvrši tek nakon što se izvrši sve u bloku aws_instance.apache. "Provisioner" local-exec služi za izvršavanje lokalnih naredbi na Terraform hostu. U naredbi command="sleep 60" je definirano da null resurs mora pričekati 60 sekundi prije nastavka izvršavanja daljnjih naredbi. Ovaj pristup se koristi kada je potrebno pričekati određeno vrijeme prije nego se nastavi s postupkom izgradnje ili konfiguracije kako bi se resursi pokrenuli ili postali dostupni. U ovom slučaju nam treba vremenska odgoda kako bi sve instance postale dostupne putem njihovih IP adresa.

```
resource "null_resource" "ansible_provisioning" {
  depends_on = [null_resource.delay]
  provisioner "local-exec" {
```

```
      command = "ansible-playbook -
i{aws_instance.apache[0].public_ip}, --private-key
${"/home/tkoma/AWS_keys/web_server_key.pem"} --extra-vars
\"mysql_host=$(terraform output -no-color -raw second_instance_ip)\"
install_apache_and_php.yml"
    }

     provisioner "local-exec" {
       command = "ansible-playbook -i $
{aws_instance.apache[1].public_ip}, --private-key
${"/home/tkoma/AWS_keys/web_server_key.pem"} mysql.yml"
    }
}
```

Kôd 13.Null resurs u kojem će se izvršiti ansible playbookovi na instancama

Null resurs pod nazivom ansible_provisioning poslužit će za konfiguriranje AWS instanci pomoću Ansiblea (Kôd 13). Ovaj resurs također ovisi o prethodnom resursu delay i neće se izvršiti ako resurs delay nije izvršen. U dva local-exec "provisionera" izvršit će se dvije ansible-playbook naredbe, svaka na svojoj instanci. Prva naredba izvršava playbook "install_apache_and_php.yml" na prvoj instanci, dok druga naredba izvršava playbook "mysql.yml" na drugoj instanci. Za svaku instancu, u naredbama se koristi opcija -i kako bi se specificirala adresa hosta na kojem će se izvršiti playbook. Ovdje se koristi vrijednost javne IP adrese svake instance, dobivena interpolacijom "${aws_instance.apache[0].public_ip}" i "${aws_instance.apache[1].public_ip}". Kod oba playbooka, parametar --private-key definirat će putanju ključa na lokalnom računalu koji će se koristiti za autentifikaciju prilikom spajanja na instance putem SSH veze. Na prvoj instanci, putem opcije --extra-vars, prosljeđuje se dodatna varijabla u playbook. Varijabla mysql_host definirana je vrijednošću IP adrese druge instance koja se dinamički dobiva iz Terraform outputa second_instance_ip. Opcija -raw služi za dobivanje čiste vrijednosti bez dodatnog formatiranja i metapodataka, a opcija -no-color kako bi se onemogućilo bojanje outputa komande Terraforma, odnosno kako bi se dobio čisti tekstualni output.

4.1.2. Datoteka security_group.tf

Sigurnosna grupa (engl. Security Group) je virtualni vatrozid koji kontrolira dolazni i odlazni

promet na instanci. Definira pravila za dozvoljene vrste protokola i njihove izvore te tako osigurava veću razinu sigurnosti za resurse (Kôd 14).

```
resource "aws_security_group" "apache" {
  name = "apache_acess"

  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 3306
    to_port = 3306
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
    ingress {
      from_port = 443
      to_port = 443
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }

    egress {
      from_port = 0
      to_port = 0
      protocol = "-1"
      cidr_blocks = ["0.0.0.0/0"]
    }
}
```

Kôd 14.Sigurnosna grupa instanci

Parametar name određuje naziv sigurnosne grupe. Ingress blokovi kôda definiraju pravila za dolazni promet prema instancama. Raspon portova s kojih je odobren dolazni promet je određen parametrima from_port (početak raspona) i to_port (kraj raspona), ako su oba postavljena npr. na 22 to znači da je dozvoljen promet samo na port 22. Port 80 dopušta dolazni promet za HTTP pristup, port 22 za SSH pristup, port 443 za HTTPS komunikaciju, a port 3306 dolazni promet za MySQL bazu podataka. Parametar protocol određuje koji protokol je dozvoljen za promet obuhvaćen ovim pravilom. U ovom slučaju se radi o tcp protokolima za dolazni promet. Raspon IP adresa s kojih je dozvoljen dolazni promet definiran je u cidr_block parametru. Ako je postavljen na 0.0.0.0/0 to znači da je sav promet dozvoljen s bilo koje IP adrese. Egress blok definira pravila za odlazni promet s instanci. Ovdje postavljeno pravilo ima port 0 što znači da se odnosi na sav odlazni promet s instance, protocol je postavljen na -1 što pokriva sve protokole, a cidr_block je opet 0.0.0.0/0 što označava sav izlazni promet prema svim IP adresama. Ovo egress pravilo dopušta odlazni promet s instanci prema bilo kojoj destinaciji, na bilo koji port i koristeći bilo koji protokol.

4.2. Ansible konfiguracija

Ansible konfiguracija se sastoji od playbookova i rola koji će se koristiti za automatiziranu konfiguraciju virtualnih servera, definirajući konkretne zadatke i resurse potrebne za njihovo postavljanje i upravljanje.

4.2.1. Playbook install_apache_and_php.yml

```
---
- name: Install Apache and PHP
hosts: all
vars: mysql_host: "{{ mysql_host }}"
  mysql_user: "bookstore_user"
  mysql_password: "S3cur3L0gin!"
  remote_user: ubuntu
  become: yes
roles: - apache_php
```

Kôd 15.Ansible playbook koji će se izvršiti na prvoj instanci

Naziv playbooka je "Install Apache and PHP" (Kôd 15). Parametar hosts je postavljen na all, što znači da su svi hostovi navedeni u inventaru obuhvaćeni ovim playbookom. Pod vars su definirane varijable, u ovom slučaju mysql_user i mysql_password s time da je varijabla mysql_host već definirana prilikom pokretanja naredbe. Korisnička uloga remote_user je postavljena na ubuntu, za udaljeno izvršavanje naredbi. Parametar become: yes dozvoljava Ansibleu izvršavanje naredbi s root ovlastima. Pod roles su definirane role koje će se izvršiti na hostovima, u ovom slučaju samo apache_php. Inače, u Ansible playbookovima lozinke i druge osjetljive informacije ne bi trebale biti vidljive.

4.2.2. Rola roles/apache_php/tasks/main.yml

Ova rola sadrži seriju taskova za automatsko postavljanje i konfiguriranje web aplikacije PHP knjižare na web poslužitelju koji koristi Apache web server (Kôd 16).

```
---
- name: Install Apache and PHP
```

```
    apt:
    update_cache: yes
    cache_valid_time: 3600
    name:
      - apache2
      - php
      - php-mysql
      - libapache2-mod-php
    state: latest

  - name: Delete existing index.html
    file:
      path: /var/www/html/index.html
      state: absent

  - name: Clone PHP Bookstore Website Example from GitHub
    git:
   repo: https://github.com/blue-sky-dev/PHP-Bookstore-Website-
Example.git
      dest: /var/www/html/
      force: yes

  - name: Make sure Apache can read the PHP Bookstore files
    file:
      path: /var/www/html
      state: directory
```

```
      mode: "0755"
      recurse: yes

  - name: Make sure Apache is running
    systemd:
      name: apache2
      state: started

  - name: Update connectDB.php with new database host, user, and password
    replace:
      path: /var/www/html/bookstore/connectDB.php
     regexp:"mysql:host=.*?;port=3306;dbname=bookstore','.*?', '.*?'"                   replace:"mysql:host={{ mysql_host }};port=3306;dbname=bookstore','{{ mysql_user }}', '{{ mysql_password }}'"


  - name: Update database connection details in PHP files
ansible.builtin.shell: |
    sed -i \
    -e 's/book/Book/g' \
    -e 's/cart/Cart/g' \
    -e 's/Register.php/register.php/g' \
    -e 's/users/Users/g' \
    -e 's/order/Order/g' \
    -e 's/customer/Customer/g' \
```

```
    -e 's/Bookstore/bookstore/g' \
   -e 's/\$servername=".*";/\$servername="{{ mysql_host }}";/g' \
   -e 's/\$username =".*";/\$username = "{{ mysql_user }}";/g' \
   -e's/\$password=".*";/\$password="{{ mysql_password }}";/g' \
  {{ item }}
  loop:
    - /var/www/html/bookstore/index.php
    - /var/www/html/bookstore/register.php
    - /var/www/html/bookstore/edituser.php
   - /var/www/html/bookstore/checkout.php
   become: yes
```

Kôd 16.Ansible rola koja automatski postavlja i konfigurira PHP web aplikaciju

Prvi task "Install Apache and PHP" instalira Apache web poslužitelj i PHP uključujući module PHP-MySQL i libapache2-mod-php i ažurira na najnovije verzije. Update_cache parametar služi za update paketa, a cache_valid_time za definiranje vremenskog perioda tijekom kojeg se prethodno preuzeti podaci o paketima smatraju aktualnima. Idući task koristi modul file kako bi izbrisao datoteku index.html iz direktorija /var/www/html/, odnosno kako bi uklonio zadanu Apache početnu stranicu. Sljedeći task će u navedeni direktorij klonirati web stranicu knjižare s odgovarajućeg github repozitorija koristeći modul git. Opcija force omogućava da se po potrebi prepiše preko postojećeg sadržaja u odredišnom direktoriju. Nakon toga daju se dozvole direktoriju /var/www/html/ kako bi Apache mogao čitati datoteke i primjenjuje se opcija recurse, što znači da će se dozvole rekurzivno primijeniti i na sve poddirektorije. Task nakon toga osigurava da je Apache servis pokrenut. Zatim slijedi task u kojem se ažurira connectDB.php datoteka tako da se izmjenjuju postojeće postavke hosta, korisnika i lozinke baze podataka s varijablama mysql_host, mysql_user i mysql_password. Ova datoteka služi za uspostavljanje veze između PHP aplikacije i baze podataka. Posljednji task služi za ažuriranje detalja veze s bazom podataka i ispravljanje određenih naziva i putanja unutar kôda. Izmjene postiže nizom sed operacija unutar ansible.builtin.shell modula koji služi za izvršavanje shell naredbi na ciljanim sustavima, a vrši ih iteracijom po petlji koja prolazi kroz skup određenih datoteka, primjenjujući izmjene na svaku od njih.

4.2.3. Playbook mysql.yml

```
---
- name: Install and configure MySql
  hosts: all
  vars:
    mysql_root_password: "jelly22fi$h"
  remote_user: ubuntu
  become: yes


  roles:
  - mysql
```

Kôd 17.Ansible playbook koji će se izvršiti na drugoj instanci

Naziv playbooka je "Install and configure MySql" (Kôd 17). Kao i u playbooku koji se izvršava na prvoj instanci i u ovaj će se izvršiti na svim hostovima navedenim u inventaru, ima jednu definiranu varijablu "mysql_root_password", remote_user je također ubuntu i postavljen je become: yes parametar. Rola koju će izvršiti ovaj playbook je "mysql". Inače, u Ansible playbookovima lozinke i druge osjetljive informacije ne bi trebale biti vidljive.

4.2.4. Rola roles/mysql/tasks/main.yml

Rola pruža automatizaciju za postavljanje i konfiguriranje MySQL servera, kreiranje novog korisnika i baze podataka te inicijalizaciju baze (Kôd 18).

```
---
- name: Install MySQL Server and Python3-mysql
  apt:
    update_cache: yes
    cache_valid_time: 3600
    name:
      - python3-pymysql
```

```
      - mysql-server
    state: latest

- name: Start MySQL Service and enable on start
  service:
    name: mysql
    state: started
    enabled: yes

- name: Save root password in .my.cnf
  template:
    src: root_cnf.j2
    dest: /root/.my.cnf
    owner: root
    mode: '0600'

- name: Set password for root user
  mysql_user:
    name: "root"
    password: "{{ mysql_root_password }}"
    priv: '*.*:ALL,GRANT'
    host: 'localhost'
    login_unix_socket: /var/run/mysqld/mysqld.sock
    state: present

- name: Create MySQL User
  mysql_user:
```

```
      name: "bookstore_user"
      password: "S3cur3L0gin!"
      host: '%'
      priv: '*.*:ALL'
      state: present

  - name: Create MySQL Database
    mysql_db:
      name: "bookstore"
      state: present

  - name: Download SQL dump from GitHub
    get_url:
      url: "https://raw.githubusercontent.com/blue-sky-dev/PHP-Bookstore-Website-Example/main/bookstore/database.sql"
      dest: "/home/ubuntu/"
  - name: Modify and remove first two lines from SQL dump file
command: "sed -i '1,2d; s/`book`/Book/g' /home/ubuntu/database.sql"
  - name: Update mysql.conf.d file
    ansible.builtin.command:
      cmd: >
        sed -i
        -e 's/mysqlx-bind-address\s*=\s*127\.0\.0\.1/mysqlx-bind-address = 0.0.0.0/g'
        -e 's/bind-address\s*=\s*127\.0\.0\.1/bind-address = 0.0.0.0/g'
```

```
      /etc/mysql/mysql.conf.d/mysqld.cnf
    become: yes

- name: Restart MySQL Service
  service:
    name: mysql
    state: restarted

- name: Import SQL dump into MySQL database
  mysql_db:
    name: "bookstore"
    state: import
    target: "/home/ubuntu/database.sql"
```

Kôd 18.Ansible rola za postavljanje i konfiguriranje MySql servera

Prvi task ažurira pakete, a zatim instalira MySQL server i Python3-pymysql i postavlja ih na najnovije verzije. Drugi task osigurava da je MySQL servis pokrenut i da mu je omogućeno automatsko pokretanje pri inicijalizaciji sustava (enabled: yes). Sljedeći task koristi template modul koji koristi datoteku root_cnf.j2 iz direktorija kao predložak kako bi stvorio datoteku .my.cnf u root direktoriju te joj postavlja odgovarajuće dozvole. Ta datoteka sadrži lozinku za root korisnika i time pruža siguran pristup MySQL-u. Nakon toga idući task koristi modul mysql_user za postavljanje lozinke root korisniku koristeći varijablu mysql_root_password. Istim modulom sljedeći task kreira novog korisnika "bookstore_user" i dodjeljuje mu lozinku, odgovarajuća prava (priv: '*.*:ALL') te mu omogućava dostupnost s bilo koje host adrese (host: '%'). Sljedeći task kreira bazu podataka modulom mysql_db i dodjeljuje joj naziv "bookstore". Task "Download SQL dump form GitHub" će pomoću modula get_url preuzeti SQL dump s GitHub repozitorija, a sljedeći task će ukloniti prve dvije linije dumpa i napraviti izmjene naredbom sed kako bi je pripremio za uvoz. Task nakon njega će ažurirati mysqld.cnf kako bi omogućio pristup MySql serveru s bilo koje adrese, a ne samo s localhosta. Zatim slijedi task koji će ponovno pokrenuti MySql server kako bi se sve promjene u konfiguraciji primijenile. Posljednji task će pomoću mysql_db modula izvršiti uvoz prethodno preuzetog SQL dumpa u

"bookstore" bazu podataka.

4.3. Ostalo

U ovom dijelu nalazi se konfiguracijska datoteka, Jinja2 predložak te datoteke stanja koje igraju ključnu ulogu u upravljanju infrastrukturom. Konfiguracijska datoteka definira postavke potrebne za optimalno izvođenje Ansible zadataka, dok datoteke stanja omogućavaju praćenje trenutnog stanja infrastrukture i sprečavaju nepotrebne promjene.

4.3.1. Datoteka ansible.cfg

Datoteka asnible.cfg je konfiguracijska datoteka za Ansible koja omogućava korisnicima da prilagode razne postavke koje utječu na njegovo ponašanje.

```
[defaults]
host_key_checking = False
```

Opcija host_key_checking određuje hoće li Ansible provjeravati SSH ključeve hosta prilikom prvog povezivanja sa SSH serverima. Postavljanjem na False, isključuje se provjera SSH ključa hosta. To znači da Ansible neće zahtijevati potvrdu identiteta udaljenog hosta pri prvom povezivanju što je korisno u automatiziranim okruženjima, posebno ako se često dodaju ili uklanjanju novi hostovi.

4.3.2. Datoteka root_cnf.j2

Datoteka root_cnf.j2 je Jinja2 predložak koji se koristi unutar Ansiblea za kreiranje konfiguracijske datoteke za MySQL klijent. Jinja2 je templating jezik za Python, koji omogućuje stvaranje dinamičkih datoteka s promjenjivim vrijednostima i često se koristi u Ansible playbookovima za stvaranje konfiguracijskih datoteka.

```
[client]
user=root
password={{ mysql_root_password }}
```

[client] označava početak sekcije koja se odnosi na MySQL klijent konfiguraciju, što znači da se nakon povezivanja s MySQL serverom mogu koristiti klijent alati. Postavlja korisniku root password koristeći varijablu "mysql_root_password". Rezultat je konfiguracijska datoteka koja se može koristiti za automatsku autentikaciju.

4.3.3. Datoteke stanja terraform.tfstate i terraform.tfstate.backup

Datoteka terraform.tfstate sadrži trenutno stanje infrastrukture kojom upravlja Terraform, što uključuje sve resurse, atribute i konfiguraciju. Navedeno stanje Terraform uspoređuje s definiranim stanjem u konfiguracijskim datotekama kako bi odredio koje promjene treba napraviti na infrastrukturi. Datoteka terraform.tfstate.backup je sigurnosna kopija datoteke terraform.tfstate koja se stvara automatski svaki put kada se ažurira stanje. Ona sadrži prethodno stanje prije posljednje promjene i služi za sigurnost u slučaju greški ili neželjenih promjena.

5. ZAKLJUČAK

Kroz detaljnu analizu Ansiblea i Terraforma, ovaj rad je istaknuo potencijal i raznolikost alata dostupnih za implementaciju koncepta "Infrastruktura kao kôd". Ova analiza je pokazala kako automatizacija nije samo sredstvo za smanjenje radnog opterećenja, već i ključni faktor u osiguravanju konzistentnosti i pouzdanosti u upravljanju IT sustavima. Proučavanje Ansiblea pokazalo je kako ovaj alat pruža fleksibilnost i jednostavnost u automatizaciji konfiguracije i upravljanja aplikacijama, dok se Terraform ističe svojom sposobnošću deklarativnog definiranja i upravljanja raznom infrastrukturom kroz cloud pružatelje usluga.

Usporedba ta dva alata razotkrila je kako, unatoč različitim pristupima i fokusima, oba alata nude komplementarne prednosti koje se mogu iskoristiti za optimizaciju razvojnih i operativnih procesa u IT projektima. Ansible je osobito koristan kada je potrebno brzo postavljanje i konfiguracija aplikacija na postojeću infrastrukturu, dok Terraform omogućava precizno i sigurno upravljanje infrastrukturom kroz cloud servise. Ova kombinacija omogućava organizacijama da istovremeno zadovolje visoke standarde u pogledu sigurnosti i učinkovitosti, što je od iznimne važnosti u modernim IT okruženjima.
Praktični dio rada je demonstrirao primjenu ovih alata na stvarnom projektu podizanja infrastrukture i aplikacija na AWS-u, a također je i pružio važne uvide u integraciju i sinergiju između Ansiblea i Terraforma. Kombiniranjem ova dva alata omogućeno je postavljanje cjelokupne infrastrukture i aplikacija s minimalnim ručnim intervencijama, čime su smanjeni rizici od ljudske pogreške, a ubrzani procesi razvoja i postavljanja. Kroz ovu praktičnu primjenu, jasno je kako kombiniranjem ova dva alata organizacije mogu ostvariti bržu implementaciju, veću pouzdanost i veću sigurnost svojih IT sustava.

Zaključno, koncept "Infrastruktura kao kôd" predstavlja jedan od najvažnijih modernih pristupa upravljanja IT resursima, omogućujući organizacijama bržu prilagodbu raznim tehničkim zahtjevima i tržišnim potrebama. Njegova primjena postaje sve rasprostranjenija kako se IT industrija suočava s potrebom za većom skalabilnošću, sigurnošću i fleksibilnošću u poslovanju. Ansible i Terraform, svaki sa svojim jedinstvenim prednostima, ključni su alati u ovom pristupu, omogućavajući efikasno upravljanje infrastrukturom i aplikacijama. Dok se tehnologija nastavlja razvijati, fleksibilnost, efikasnost i sigurnost koje ovaj pristup nudi postaju sve važniji, čineći znanje i vještine povezane s alatima poput Ansiblea i Terraforma neophodnim za buduće generacije IT profesionalaca. Osim toga, očekuje se da će se s vremenom pojaviti novi alati i metode koji će dodatno unaprijediti ovaj pristup, što će od

stručnjaka u IT industriji zahtijevati stalno učenje i prilagodbu.

LITERATURA

1. Brikman, Yevgeniy, (2022), *Terraform Up & Running: Third edition, Writing Infrastructure as Code*, Sebastopol: O'Reilly Media, Inc.
2. Greeling, Jeff, (2020), *Ansible for DevOps: Server and configuration managment for humans*, Leanpub
3. HashiCorp, Terraform About the Docs, https://developer.hashicorp.com/terraform/docs (pristupljeno 17.01.2024.)
4. Morgan, Derek (2022), Terraform vs. Ansible: Which Is Best for You?, https://thenewstack.io/terraform-vs-ansible-which-is-best-for-you/ (pristupljeno 17.01.2024)
5. Ninawe, Sumeet, (2023), Terraform vs. Ansible : Key Differences and Comparison of Tools, https://spacelift.io/blog/ansible-vs-terraform (pristupljeno 17.01.2024)
6. Sesto, Vincent, (2021), *Practical Ansible: Configuration Managment from Start to Finish*, Auckland, New Zealand: Apress
7. Red Hat, Inc., Ansible Documentation,

   https://docs.ansible.com/ (pristupljeno 17.01.2024.)

SAŽETAK

Ovaj rad detaljno istražuje koncept "Infrastruktura kao kôd", s posebnim fokusom na alate Terraform i Ansible. Rad pojedinačno analizira Terraform i Ansible, istražujući njihove ključne značajke, prednosti i potencijalne primjene. Nakon toga, usporedba ovih alata razotkriva kako, unatoč različitim pristupima, oba pružaju komplementarne prednosti koje se mogu iskoristiti u razvojnim i operativnim procesima IT projekata. U praktičnom dijelu rada, demonstrira se upotreba Terraforma za postavljanje dviju instanci na AWS-u, gdje se na jednoj instalira Apache web server i PHP aplikacija online knjižare, dok se na drugoj postavlja baza podataka, s Ansibleom koji automatizira ove zadatke. Kroz ovu praktičnu primjenu, rad nudi uvid u efikasnost i sinergiju između Terraforma i Ansiblea u realnom IT projektu.

Ključne riječi: Infrastruktura kao Kôd, Terraform, Ansible, AWS, automatizacija, Apache web server, PHP, baza podataka, cloud computing, IT infrastruktura

SUMMARY

This paper thoroughly investigates the concept of "Infrastructure as Code", with a special focus on the tools Terraform and Ansible. It individually analyzes Terraform and Ansible, exploring their key features, advantages, and potential applications. Subsequently, a comparison of these tools reveals that, despite their different approaches, both offer complementary benefits that can be leveraged in the development and operational processes of IT projects. In the practical part of the paper, the use of Terraform to set up two instances on AWS is demonstrated, where one instance is equipped with an Apache web server and a PHP online bookstore application, while the other hosts a database, with Ansible automating these tasks. Through this practical application, the paper provides insight into the efficiency and synergy between Terraform and Ansible in a real IT project.

Keywords: Infrastructure as Code, Terraform, Ansible, AWS, automation, Apache web server, PHP, database, cloud computing, IT infrastructure