KOMPARATIVNA ANALIZA ANSIBLE I TERRAFORM AUTOMATIZACIJSKIH ALATA

Kocijan, David

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Komparativna analiza Ansible i Terraform automatizacijskih alata

David Kocijan

Zagreb, veljača 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

Zahvaljujem svom mentoru prof. Zlatanu Moriću na pomoći i vodstvu pri izradi ovog završnog rada.

U Zagrebu, 20.02.2023.

Sažetak

U ovom su radu uspoređena dva najpopularnija alata za automatizaciju na današnjem tržištu - Ansible i Terraform. Ansible je alat za IT automatizaciju otvorenog kôda koji pomaže u upravljanju sustavima. Može se koristiti za upravljanje mrežama, poslužiteljima, aplikacijama ili klasterima. Terraform je infrastruktura otvorenog kôda kao Infrastruktura kao kôd. Terraform i Ansible nude različite alate za automatizaciju infrastrukture. Terraform je fokusiran na upravljanje konfiguracijom infrastrukture, a Ansible primarno pokriva pokretanje aplikacija i servisa.

Ova dva alata razlikuju se po sintaksi i ciljanim platformama. U ovom radu uspoređena su dva alata na temelju četiri kriterija: upotrebljivost (jednostavan jezik), mogućnosti automatizacije (fleksibilnost konfiguracije), upravljanje resursima (sposobnost pouzdanog upravljanja resursima) i skalabilnost (mogućnost jednostavnog upravljanja velikim infrastrukturama).

Ključne riječi: Ansible, Terraform, automatizacija, sintaksa, platforma, učinkovitost

Abstract

In this work, we will compare the two most popular automation tools on the market today - Ansible and Terraform. Ansible is an open-source IT automation tool that helps manage systems. It can be used to manage networks, servers, applications, or clusters. Terraform is an open-source Infrastructure as Code. Terraform and Ansible offer different automation tools for infrastructure provisioning. Terraform is focused on infrastructure configuration and management, while Ansible primarily covers running applications and services.

The two tools differ in syntax and target platforms. In this paper, we will compare the two tools based on four criteria: usability (simple language), automation capabilities (configuration flexibility), resource management (ability to reliably manage resources), and scalability (ability to easily manage large scales).

Keywords: Ansible, Terraform, automation, syntax, platform, efficiency

Sadržaj

1. Uvod

Kako bi se lakše razumjela tematika automatizacijskih alata poput Ansiblea i Terraforma, mora se prvo razumjeti način na koji funkcioniraju i što im je sve potrebno za rad.

U prvom dijelu ovog rada odredit će se što je Infrastruktura kao kôd (engl. Infrastructure as Code (IaC)), objasnit će se kako se koriste Ansible i Terraform alati te koje su im prednosti i nedostaci. Ansible je razvijen 2012. godine, a Terraform je došao na tržište dvije godine kasnije.

Ansible je alat za automatizaciju koji se koristi za provođenje konfiguracija, instalaciju aplikacija, postavljanje mreže, provjeru sigurnosti i mnoge druge svrhe na mrežnim uređajima i poslužiteljskim računalima. Razvio se kako bi se smanjilo vrijeme i troškovi potrebni za izvršavanje ovih zadataka te kako bi se smanjila potreba za ručnim radom i pogreškama. Brzo se proširio i sada se koristi diljem svijeta u različitim industrijama, od financijskih i telekomunikacijskih do proizvodnje i zdravstva.

Terraform je alat za upravljanje infrastrukturom koji omogućava automatizaciju kreiranja resursa, što je dovelo do veće učinkovitosti i skalabilnosti te smanjenja rizika u radu s infrastrukturom.

Kako bi usporedili ova dva alata, osmišljeno je i provedeno testiranje u virtualnom okruženju. Okolina na kojoj se odvijao rad bila je segmentirana na dva dijela. Prvi dio sadržavao je Ansible virtualno okruženje, a drugi dio okoline sadržavao je Terraform alat. U zadnjem dijelu rada izneseni su rezultati testiranja na temelju kojih su uspoređeni alati.

2. Opis primijenjenih tehnologija

Infrastruktura kao kôd (engl. Infrastructure as Code (IaC)) je koncept koji se temelji na ideji da se infrastruktura (npr. mreža, baze podataka, poslužitelji itd.) opisuje kao kôd, što omogućuje automatizaciju i lakše upravljanje. IaC se često koristi u DevOps kulturi kako bi se osigurala jednakost okruženja, a također može pomoći u učinkovitom upravljanju infrastrukturom kroz različite faze životnog ciklusa aplikacije. IaC pomaže IT timovima u povećanju efikasnosti, smanjenju vremena potrebnog za izvršavanje rutinskih poslova i osiguravanju konzistentnosti u procesima.

2.1. Uloga Terraforma

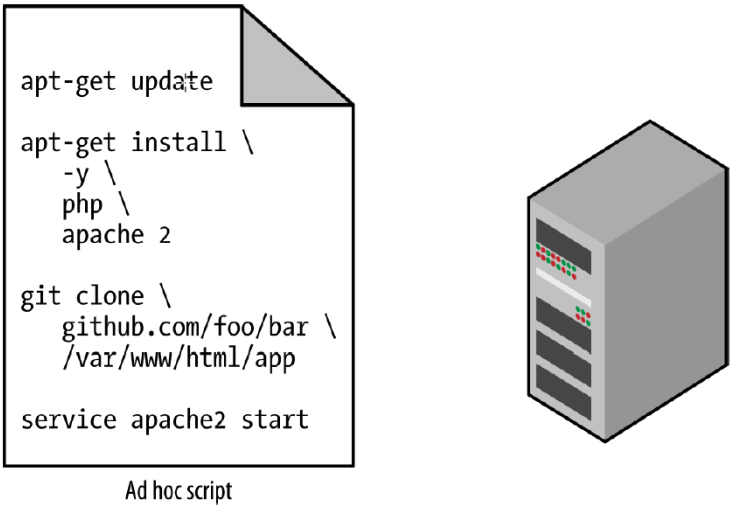
„Ideja infrastrukture kao kôda (IaC) je da se piše i izvršava kôd koji definira, instalira, ažurira i uništava korisnikovu infrastrukturu. Ovo predstavlja važan preokret u mentalitetu u kojem se svi aspekti rada tretiraju kao softver - čak i oni aspekti koji predstavljaju hardver (npr. postavljanje fizičkih poslužitelja). Činjenica je da je ključni zaključak DevOpsa da se gotovo sve može upravljati kôdom, uključujući poslužitelje, baze podataka, mreže, datoteke, konfiguracije aplikacija, dokumentaciju i slično.

Postoji pet kategorija alata za infrastrukturu kao kôd:

- Ad hoc skripte (engl. Ad hoc scripts)
- Alati za upravljanje konfiguracijom (engl. Configuration management tools)
- Alati za izradu predložaka poslužitelja (engl. Server templating tools)
- Orkestracijski alati (engl. Orchestration tools)
- Alati za kreiranje resursa (engl.. Provisioning tools).“ (Brikman, 2019, p. 24)

Ad Hoc skripte

Najjednostavniji pristup automatizaciji bilo čega je napisati ad hoc skriptu. Uzme se bilo koji zadatak koji se radio ručno, podijeli se u konkretne korake, koristeći omiljeni jezik za skriptiranje (npr. Bash, Ruby, Python) definira se svaki od tih koraka u kôdu i izvrši se ta skripta na poslužitelju, kao što je prikazano na Slika 2.1. (Brikman, 2019, p. 24)



Slika 2.1 Pokretanje ad hoc skripte na poslužitelju (Brikman, 2019, p. 25)

Najbolje kod ad hoc skripti je da korisnik može koristiti popularne programske jezike opće namjene (engl. general-purpose) i pisati kôd na način koji mu odgovara. Nedostatak kod ad hoc skripti je da se mogu koristiti popularni programski jezici opće namjene i pisati kôd na bilo koji način . Alati koji su specijalno namijenjeni za IaC nude kratke API-e za postizanje složenih zadataka, a ako se koristi programski jezik opće namjene, mora se napisati potpuno prilagođeni kôd za svaki zadatak. Osim toga, alati dizajnirani za IaC obično nameću određenu strukturu, a s općenito namijenjenim programskim jezikom svaki korisnik ima svoj stil i nešto drugačije radi. Nijedan od ovih problema nije veliki problem za osam linija kôda koji instalira Apache, ali to postaje neuredno ako se pokušaju koristiti ad hoc skripte za upravljanje desetinama poslužitelja, bazama podataka, mrežnim konfiguracijama i slično. (Brikman, 2019, p. 25)

Ako je korisnik ikada morao održavati veliki repozitorij Bash skripti, zna da se gotovo uvijek pretvori u zbrku neodrživog kôda nalik špagetima. Ad hoc skripte su namijenjene za male i jednokratne zadatke, ali ako će se upravljati svojom infrastrukturom kao kôdom, onda bi se trebao koristiti alat za IaC specijalno namijenjen za taj posao. (Brikman, 2019, p. 25)

Konfiguracijski alati za upravljanje

„Chef, Puppet, Ansible i SaltStack su konfiguracijski alati za upravljanje, što znači da su dizajnirani za instaliranje i upravljanje softverom na postojećim poslužiteljima. Kôd izgleda slično Bash skripti, ali korištenjem alata poput Ansiblea pružaju se brojne prednosti:

- Konvencije kôda
  Ansible primjenjuje konzistentnu, predvidljivu strukturu, uključujući dokumentaciju, raspored datoteka, jasno imenovane parametre, upravljanje tajnim informacijama i sl.
- Idempotentnost
  Pisanje ad hoc skripte koja jednom funkcionira nije previše teško; pisanje ad hoc skripte koja funkcionira ispravno čak i ako se pokreće više puta, puno je teže.
- Distribucija
  Ad hoc skripte su dizajnirane za pokretanje na jednom, lokalnom računalu. Ansible i ostali konfiguracijski alati za upravljanje su dizajnirani posebno za upravljanje velikim brojem udaljenih poslužitelja.“ (Brikman, 2019, p. 26)

Alati za izradu predložaka poslužitelja

Alternativa upravljanju konfiguracijom kojoj je popularnost u porastu su alati za stvaranje predložaka poslužitelja poput Dockera, Packera i Vagranta. Umjesto da se pokrene veliki broj poslužitelja koji će se konfigurirati tako da se isti kôd pokrene na svakome od njih, ideja iza alata za stvaranje predložaka poslužitelja je stvoriti sliku poslužitelja koja sadrži potpuno samostalnu presliku operacijskog sustava, softvera, datoteka i svih ostalih relevantnih detalja. Zatim, može se koristiti neki IaC alat da se instalira taj obrazac na sve korištene poslužitelje. (Brikman, 2019, p. 28)

Orkestracijski alati

„Alati za izradu predložaka poslužitelja su dobri za izradu VM-a i kontejnera, ali kako njima upravljati? Za većinu primjena treba se naći način za sljedeće:

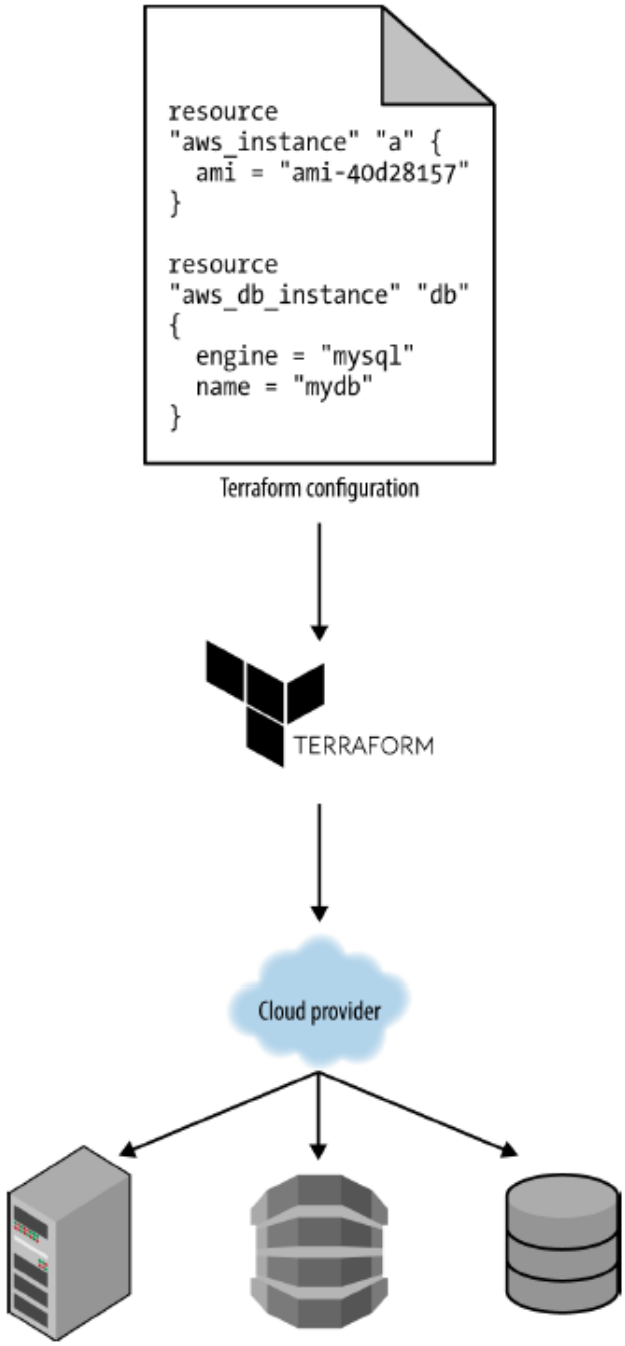
- kreirati VM i kontejner učinkovito koristeći vlastiti hardver.
- Instaliranje ažuriranja postojećih VM-a i kontejnera koristeći strategije kao što su *rolling deployment*, *blue-green deployment* i *canary deployment*.
- Pratiti ispravnost vlastitih VM-ova i kontejnera i automatski zamijeniti neispravne.
- Regulirati broj VM-a (više ili manje) ovisno o opterećenju.

- Distribuirati promet kroz VM-ove i kontejnere (balansiranje opterećenja).
- Dopustiti VM-ovima i kontejnerima da se međusobno pronađu i komuniciraju preko mreže.

Rukovanje ovim zadacima je područje alata za orkestraciju poput Kubernetesa, Marathon/Mesos, Amazon Elastic Container Service (Amazon ECS), Docker Swarm i Nomad. Na primjer, Kubernetes omogućuje da korisnik definira kako će upravljati svojim Docker kontejnerima kao kôdom.“ (Brikman, 2019, p. 32)

Alati za kreiranje resursa

Alati za upravljanje konfiguracijom, izradu predložaka poslužitelja i orkestracijom odgovorni su za definiranje kôda koji se izvodi na svakom poslužitelju, a alati za kreiranje resursa poput Terraforma, CloudFormationa i OpenStack Heata su odgovorni za stvaranje poslužitelja. Naime, alati za kreiranje mogu se koristiti ne samo za stvaranje poslužitelja, već i za stvaranje baza podataka, praćenje sustava, konfiguracije mreža, postavke vatrozida, pravila usmjeravanja, certifikate SSL-a i gotovo svaki aspekt korisnikove infrastrukture, kako je prikazano na Slika 2.2. (Brikman, 2019, p. 34)



Slika 2.2 Primjer uporabe Alata za kreiranje resursa (Brikman, 2019, p. 35)

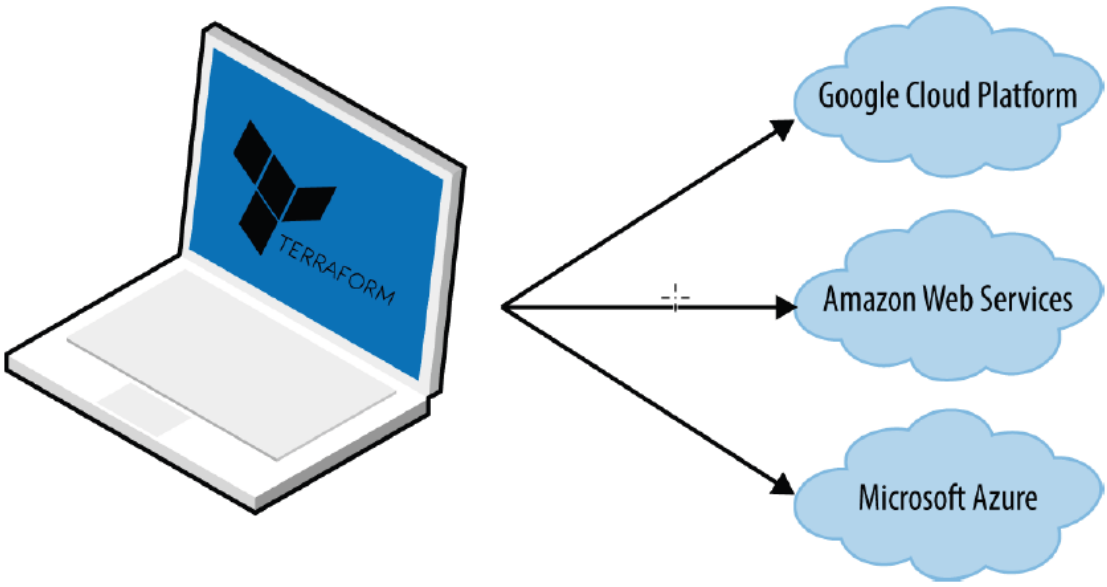
„Terraform je alat otvorenog kôda kreiran od tvrtke HashiCorp i napisan je u programskom jeziku Go. Programski jezik Go se kompilira u jedan binarni dokument (jedan binarni dokument za svaki od podržanih operativnih sustava) nazvan terraform. Binarni dokument može se koristiti za raspoređivanje infrastrukture s laptopa ili servera ili bilo kojeg drugog računala i ne mora se pokretati nikakva dodatna infrastruktura. To je zato što terraform binarni dokument izvršava API pozive u korisnikovo ime (engl. on your behalf) prema jednom ili više pružatelja usluga, poput AWS-a, Azurea, Google Clouda, DigitalOceana,

OpenStacka i još mnogim drugima. Ovo znači da Terraform može iskoristiti infrastrukturu koju ti pružatelji već pokreću za svoje API servere, kao i mehanizme autentifikacije koje već korisnik upotrebljava s tim pružateljima usluga (npr. API ključevi koje već ima za AWS). Kako Terraform zna koje API pozive da izvrši? Odgovor je da se kreira Terraform konfiguracija, u kojoj se navodi što se želi kreirati u infrastrukturi. Ove konfiguracije su „kôd" u „infrastrukturi kao kôd". Ovo je primjer Terraform konfiguracije

```
resource "aws_instance" "example" {
ami = "ami-0c55b159cbfafe1f0"
instance_type = "t2.micro"
}
resource "google_dns_record_set" "a" {
name = "demo.google-example.com"
managed_zone = "example-zone"
type = "A"
ttl = 300
rrdatas = [aws_instance.example.public_ip]
}
```

Kôd 2.1 Primjer Terraform konfiguracije

Čak i ako korisnik nikada prije nije vidio Terraform kôd, ne bi trebao imati previše problema kod njegovog čitanja. Ovaj dio kôda navodi Terraform da izvršava API pozive prema AWS-u kako bi kreirao server, a zatim da izvršava API pozive na Google Cloud da kreira stavku DNS-a koja pokazuje na IP adresu servera na AWS-u. U samo jednoj, jednostavnoj sintaksi, Terraform omogućuje da korisnik raspoređuje resurse povezane preko više pružatelja usluga u oblaku. Može definirati cijelu svoju infrastrukturu - servere, baze podataka, balansere opterećenja, topologiju mreže i tako dalje - u Terraform konfiguracijskim dokumentima i te dokumente dodati u kontrolu verzija. Zatim, pokreće određene Terraform naredbe, kao što su `terraform apply`, da bi primijenio tu infrastrukturu. Terraform binarni dokument analizira korisnikov kôd, prevodi ga u seriju API poziva prema platformi za računarstvo u oblaku navedenima u kôdu i izvršava te API pozive u korisnikovo ime što je efikasnije moguće, kako je prikazano na Slika 2.3.



Slika 2.3 Primjer pozivanja API poziva (Brikman, 2019, p. 38)

Kada netko u timu korisnika želi napraviti izmjene na infrastrukturi, umjesto da ručno i direktno ažurira infrastrukturu na serverima, on napravi svoje promjene u Terraform konfiguracijskim datotekama, provjeri te promjene putem automatiziranih testova, pošalje ažurirani kôd u kontrolu verzija i zatim pokrene naredbu `terraform apply` kako bi Terraform izvršio potrebne API pozive za izvršavanje izmjena.“ (Brikman, 2019, p. 37/38)

2.2. Uloga Ansiblea

Ansible je alat kojeg je u veljači 2012. godine razvio Michaela DeHaana, zaposlenik tvrtke Red Hat. Michaela je inspiriralo nekoliko alata i njegovo iskustvo u industriji sistemskog administratora. Postojala je jaka potreba kako bi se omogućilo korištenje iste konfiguracije prilikom upravljanja na cijeloj Linux floti te sposobnost da se konfiguracijske datoteke konzistentno mijenjaju svaki put. Michael je započeo izradu alata u Pythonu, najzanimljivijem računalnom jeziku tada i izgradio je temelj bezagentne (engl. Agentless) arhitekture bazirane na modulima (engl. module-based). Svaki modul izvodi specifičan zadatak i stvara Python bytecode koji se šalju izravno za izvršavanje putem SSH-a za Linux, Unix i macOS sustave. Za Windows sustave Ansible proizvodi PowerShell ili CMD deliverable koji se dijeli putem WinRM-a kako bi se postigao očekivani rezultat. Ovaj jednostavan i proširiv pristup formirao je početnu zajednicu. Michael je osnovao tvrtku kako

bi podržao početnu potražnju i ta tvrtka je 2015. godine kupljena od strane Red Hata. (Berton, 2022).

Ansible se klasificira kao alat za automatizaciju infrastrukture, tako da se mogu jednostavno automatizirati zadaci sistemskog administratora. Omogućuje proces nazvan Infrastruktura kao kôd, način obrade osiguranja i upravljanja strojevima korištenjem jednostavnih definicijskih datoteka čitljivih za ljude. To je iznimno korisno u okruženju s centrom podataka kada se isti zadatak često ponavlja među skupinom strojeva. Prije Ansiblea, ručna konfiguracija je zahtijevala složene i specijalizirane tehničare za konfiguraciju hardvera i interaktivne konfiguracijske alate. Ovaj stil programiranja, također, se naziva deklarativnim jer se fokusira na rezultat izvođenja, a ne na pojedinačne radnje kao u tradicionalnim imperativnim programskim jezicima (C, C++, Java, Python itd.). (Berton, 2022).

Prema Bertonu (2022), tri glavne primjene su:

1. Osiguranje

   Svi sistemski administratori znaju koliko je važno upravljati flotom računala. Neki ljudi još uvijek se oslanjaju na softver za stvaranje preslike (engl. Image) radne stanice. Ali postoji nedostatak jer korištenjem tehnologije preslika samo se uzima snimka u trenutku pa se svaki put mora ponovno instalirati softver zbog moderne aktivacije ključeva ili se moraju ručno ažurirati najnovije sigurnosne zakrpe. Ansible pomaže automatizirati ovaj proces stvaranjem jednostavnijeg postupka.

2. Upravljanje konfiguracijom

   Proces održavanja sustava i softvera u željenom i konzistentnom stanju. Drugi ključni primjer korištenja je upravljanje konfiguracijama: održavanje ažurnog i konzistentnog načina preko korisnikove flote u koordinaciji s ažuriranjima i planiranjem vremena gašenja. S Ansibleom se može provjeriti status upravljanih poslužitelja i može se poduzeti akcija u njihovoj grupi. Dostupna je velika raznolikost modula za najčešće upotrebe, uključujući provjeru sukladnosti korisnikove flote nekim međunarodnim standardima i primjenu planova.

3. Razvoj aplikacije

   Proces objavljivanja korisnikovog softvera između testiranja i primjene za korištenje u proizvodnji. Treći ključni primjer korištenja gdje je Ansible koristan je kreiranje aplikacija. Primjerice, može automatizirati radni tok cjelovitog integriranja korisnikovih aplikacija.

Berton (2022) navodi da se Ansible koristi za primjenu DevOps principa u svjetskim organizacijama. Metodologija DevOpsa sastoji se od skupa najboljih praksi visoke razine koje se trebaju slijediti tijekom cjeloživotnog ciklusa softvera, od početne dizajnerske faze, kôdiranja, testiranja, objavljivanja, aktivnog korištenja i umirovljenja s tržišta. Ovi principi se odnose i na male i na velike projekte i koriste se danas od strane IT stručnjaka u malim i velikim organizacijama diljem svijeta. Općenito govoreći, primjenjuju se inženjerski principi na proces stvaranja softvera. DevOps je tako popularan zbog sljedećih snaga: performanse, mogućnost ponovnog korištenja, izdržljivosti i smanjenja troškova. Osim toga, softver korišten u DevOps okruženju se definira kao alatni lanac (engl. *toolchains*). Alatni lanci su obično posebno dizajnirani za jedan određeni zadatak. Često se klasificiraju korištenjem DevOps kategorija. Svaka kategorija odražava ključni aspekt procesa dizajna, razvoja, testiranja i isporuke softvera. Osam DevOps kategorija su:

1. Plan: menadžer analizira zahtjeve proizvoda/projekta i povratne informacije od unutarnjih i vanjskih sudionika i stvara putanju proizvoda. Softver koji se obično koristi je Jira, Azure DevOps ili Asana.
2. Kôd: Razvojni proces koji stvara kôd projekta. Alatni lanci variraju ovisno o korištenom računalnom jeziku i alatima za upravljanje verzijama izvornog kôda.
3. Izgradnja: Alatni lanci su vrlo specifični ovisno o računalnom jeziku korisnikovog tima. Primjenjuju se metodologije kontinuirane integracije/kontinuirane isporuke i statusa izgradnje.
4. Testiranje: Alatni lanci za testiranje osiguravaju brze i ažurirane rezultate na temelju poslovnih rizika, često baziranih na Infrastrukturi kao kôdu (engl. Infrastructure-as-Code) i mnogim DevOps ciljevima.
5. Objavljivanje: Skup alata koji pakira korisnikov softver u repozitorij.
6. Razmještaj: Softver ulazi u fazu proizvodnje. Koristi se infrastruktura kao kôd (engl. Infrastructure-as-Code) za objavljivanje u dvije faze implementacije kako bi se koristile nove proizvodne usluge bez prekida usluge.
7. Upravljanje: Konfiguracija infrastrukture i upravljanje. Skaliranje infrastrukture i prikupljanje povratne informacije o usluzi.
8. Praćenje: Praćenje performansi aplikacije i povratne informacije krajnjeg korisnika.

Prema Bertonu (2022), četiri ključna načela Ansiblea su:

- Deklarativno
- Bezagentno (engl. Angentless)
- Idempotentno
- Vođeno zajednicom.

Deklarativno znači da se može koristiti na način sličan programskom jeziku za primjenu sekvenciranja, odabira i iteracije u protoku kôda. Bezagentno (engl. Angentless) znači da Ansible djeluje na način koji ne zahtijeva instaliranje aktivnog procesa (agenta) na ciljnom računalu, već koristi SSH vezu. Za Linux i Unix strojeve to znači korištenje SSH-a, korištenjem OpenSSH ili u ograničenim okruženjima Paramiko (Python OpenSSH biblioteka). Za Windows računala to znači korištenje Windows Remote Managementa putem PowerShell Remotinga. Jezik je sam po sebi idempotentan, što znači da će kôd provjeriti stanje na upravljanom računalu. To znači da, na primjer, prvi put kada se pokrene, kôd će promijeniti nešto, a prilikom sljedećeg pokretanja će se samo provjeravati ako se je nešto promijenilo. Vođen zajednicom znači da je Ansible tehnologija bazirana na otvorenom kôdu (engl. Open-source). Postoji javna arhiva nazvana Ansible Galaxy gdje se može preuzeti kôd napravljen od strane drugih razvijatelja (Berton, 2022).

„Slika 2.4 prikazuje primjer uporabe Ansiblea. Korisnik koji je nazvan Stacy koristi Ansible za konfiguraciju tri web poslužitelja na bazi Ubuntua kako bi pokrenuo nginx servis. Korisnik je napisao Ansible skriptu pod nazivom webservers.yml koja se u Ansible terminologiji naziva playbook. Playbook opisuje koje je poslužitelje (što Ansible naziva udaljeni poslužitelji) potrebno konfigurirati i sadrži sortirani popis zadataka za izvršavanje na tim poslužiteljima. U ovom primjeru poslužitelji su web1, web2 i web3, a zadaci su stvari poput:

- Instaliranje nginxa
- Stvaranje nginx konfiguracijske datoteke
- Kopiranje sigurnosnog certifikata
- Pokretanje nginx usluge.

Stacy izvršava playbook koristeći naredbu `ansible-playbook`. U ovom primjeru, playbook se naziva *webservers.yml* i pokreće se upisivanjem:

```
$ ansible-playbook webservers.yml
```

Ansible će paralelno napraviti SSH vezu s web1, web2 i web3 poslužiteljem. Izvršit će prvi zadatak s popisa na sva tri poslužitelja istodobno. U ovom primjeru, prvi zadatak je instaliranje apt paketa nginxa (jer Ubuntu koristi apt paket) pa bi zadatak u playbooku izgledao ovako:

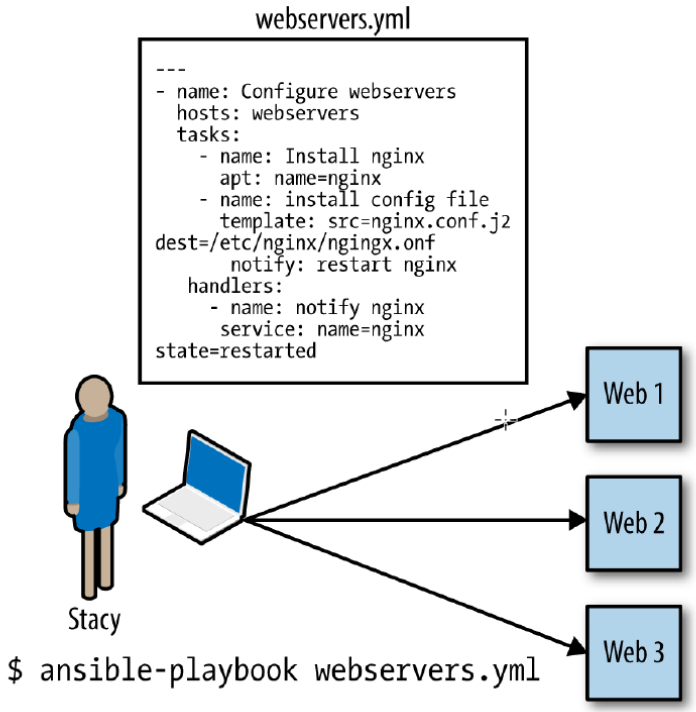
```
- name: instaliraj nginx
apt: name=nginx
```

Ansible će:

1. Generirati Python skriptu koja instalira nginx paket.
2. Kopirati skriptu na web1, web2 i web3.
3. Izvršiti skriptu na web1, web2 i web3.
4. Čekati dok se skripta ne izvrši na svim poslužiteljima.

Ansible će zatim preći na sljedeći zadatak s popisa i proći kroz ista ta četiri koraka. Važno je napomenuti da:

- Ansible pokreće svaki zadatak paralelno na svim poslužiteljima.
- Ansible čeka dok svi poslužitelji ne završe zadatak prije nego pređe na sljedeći zadatak.
- Ansible pokreće zadatke u redoslijedu u kojem su specificirani“. (Hochstein, 2015, p. 3)



Slika 2.4 Primjer uporabe Ansiblea (Hochstein, 2015, p. 4)

2.3. Prednosti i nedostaci Ansiblea

Korištenjem Ansiblea IT timovi mogu brzo i jednostavno implementirati i održavati kompleksne infrastrukture, međutim, kao i sa svakim alatom, Ansible ima i svoje prednosti i nedostatke. U nastavku su prikazane prednosti, ali i neki nedostaci ovog alata.

2.3.1. Prednosti

Jednostavnost

Jedna od glavnih prednosti Ansiblea je njegova jednostavnost. Nije namijenjen samo stručnjacima, već je namijenjen i početnicima. Skripte za upravljanje konfiguracijom u Ansibleu nazvane su playbooksima. Sintaksa playbooksa je izgrađena na temelju YAML-a, koji je jezik za oblikovanje podataka dizajniran da ljudima bude lak za čitanje i pisanje. (Hochstein, 2015, p. 5)

Bez potrebe za agentima (engl. Agentless)

Još jedna važna prednost Ansiblea odnosi se na činjenicu da mu nije potreban agent. Za upravljanje serverom potrebna je samo SSH veza i Python verzije 2.5 ili novije. Nema potrebe za instalacijom programa na udaljenom poslužitelju. (Hochstein, 2015, p. 5)

Alat temeljen na guranju (engl. Push-Based)

Neki sistemi za upravljanje konfiguracijom koji koriste agente, poput, Chef i Puppet, su bazirani na povlačenju (engl. pull-based). Agenti instalirani na poslužiteljima redovito se javljaju centralnom servisu i preuzimaju informacije o konfiguraciji s tog servisa nakon isteka postavljenog vremenskog okidača. U usporedbi, Ansible je temeljen na guranju te do promjene na poslužiteljima dolazi odmah pri primjeni novog playbooka. Pristup temeljen na guranju ima značajnu prednost: korisnik kontrolira kada se promjene događaju na poslužiteljima. Ne mora čekati da istekne vremenski okidač. (Hochstein, 2015, p. 5)

Korištenje na velikom broju poslužitelja

Ansible se može koristiti za upravljanje stotinama ili čak tisućama poslužitelja. Koristiti ga za konfiguriranje jednog poslužitelja je jednostavno; jednostavno se napiše jedan playbook. Ansible poštuje Alan Kayevu uzrečicu: „Jednostavne stvari bi trebale biti jednostavne, složene stvari bi trebale biti moguće.“ (Hochstein, 2015, p. 6)

Ugrađeni moduli

Ansible sadrži kolekciju modula koja se može koristiti za obavljanje zadataka poput instaliranja paketa, ponovnog pokretanja usluge ili kopiranja konfiguracijske datoteke. Moduli su deklarativni; koriste se za opis stanja u kojem bi trebao biti poslužitelj. Na primjer, može se pozvati modul korisnik kako bi se osiguralo postojanje računa s imenom „deploy“ u grupi „web“: `user: name=deploy group=web`. Moduli su, također, idempotentni. Ako korisnik „deploy“ ne postoji, Ansible će ga stvoriti. Ako postoji, Ansible neće ništa učiniti. Idempotencija je korisna jer to znači da je sigurno pokrenuti Ansible playbook više puta na istom poslužitelju. (Hochstein, 2015, p. 6)

Tanak sloj apstrakcije

Neki alati za upravljanje konfiguracijom pružaju sloj apstrakcije kako bi se mogle koristiti iste skripte za upravljanje poslužiteljima koji rade na različitim operacijskim sustavima. Na primjer, umjesto da se koriste yum ili apt paketi za upravljanje možete se koristiti apstrakcija „paket.“ Ansible nije takav te se mora koristiti apt modul za instaliranje paketa na apt temeljenim sustavima i yum modul za instaliranje paketa na yum temeljenim sustavima. Iako ovo može zvučati kao nedostatak, u praksi to čini Ansible lakšim za upotrebu. (Hochstein, 2015, p. 7)

2.3.2. Nedostaci

Korisničko sučelje

Ansible je početno bio samo alat s naredbenom linijom. Prvi pokušaj Ansiblea da napravi korisničko sučelje bio je s grafičkim sučeljem AWX. Naknadno se AWX pretvorio u Ansible Tower, koji je web sučelje za upravljanje i pruža vizualne značajke upravljanja. Međutim, ima još mjesta za napredovanje jer još uvijek ima zadataka koji se mogu dovršiti samo putem naredbenog retka.

Ograničena podrška za Windows

Ansible verzija 1.7 podržava Windows i Linux/Unix. U slučaju Windowsa, Ansible koristi PowerShell Remoting umjesto SSH-a. Kao rezultat, za upravljanje Windows poslužiteljima potreban je Linux kontrolni poslužitelj.

2.4. Prednosti i nedostaci Terraforma

Kako Terraform može konkurirati velikanima poput Amazona, Microsofta i Googlea? U nastavku su prikazane prednosti, ali i neki nedostaci ovog alata.

2.4.1. Prednosti

„Šest ključnih karakteristika čine Terraform jedinstvenim i daju mu konkurentsku prednost:

Alati za kreiranje resursa

Terraform je alat za kreiranje resursa, a ne za upravljanje konfiguracijom. Alati za kreiranje resursa izrađuju i upravljaju infrastrukturom, a alati za upravljanje konfiguracijom, poput, Ansiblea, Pupeta, SaltStacka i Chefa instaliraju softvera na postojećim poslužiteljima. Neki alati za upravljanje konfiguracijom, također, mogu djelomično obavljati izradu i upravljanje infrastrukturom, ali ne tako dobro kao Terraform jer to nije zadatak za koji su izvorno dizajnirani. Razlika između alata za upravljanje infrastrukturom i alata za kreiranje resursa je pitanje filozofije. Alati za upravljanje infrastrukturom favoriziraju mutabilnu infrastrukturu, a Terraform i ostali alati za kreiranje resursa favoriziraju imutabilnu infrastrukturu. Mutabilna infrastruktura znači da se softver ažurira na postojećim poslužiteljima. S druge strane, imutabilna infrastruktura se ne brine za postojeće poslužitelje, već infrastrukturu tretira kao potrošnu robu. Razlika između ta dva paradigma se može sažeti kao mentalitet recikliranja i jednokratne uporabe.

Lako se koristi

Osnove Terraforma se brzo i lako uče, čak i u slučaju korisnika koji nisu programeri. Usavršavanje je složenije, ali to vrijedi za većinu vještina. Glavni razlog zašto je Terraform tako lako koristiti je što je kôd napisan u jeziku konfiguracije namijenjenom za domene (engl. domain-specific) nazvan HashiCorp Configuration Language (HCL). To je jezik koji je stvorila tvrtka HashiCorp kao zamjenu za opširnije jezike konfiguracije, poput, JSON i XML. HCL pokušava postići balans između lakoće čitanja kôda za ljude i za računala, a izrađen je pod utjecajem ranijih pokušaja kao što su libucl i konfiguracija Nginxa. HCL je potpuno kompatibilan s JSON-om, što znači da se HCL može konvertirati 1:1 u JSON i obrnuto. To ga čini jednostavnim za interakciju sa sustavima izvan Terraforma ili za generiranje kôda konfiguracije u tijeku (engl. Code on the fly).

Besplatan softver otvorenog kôda

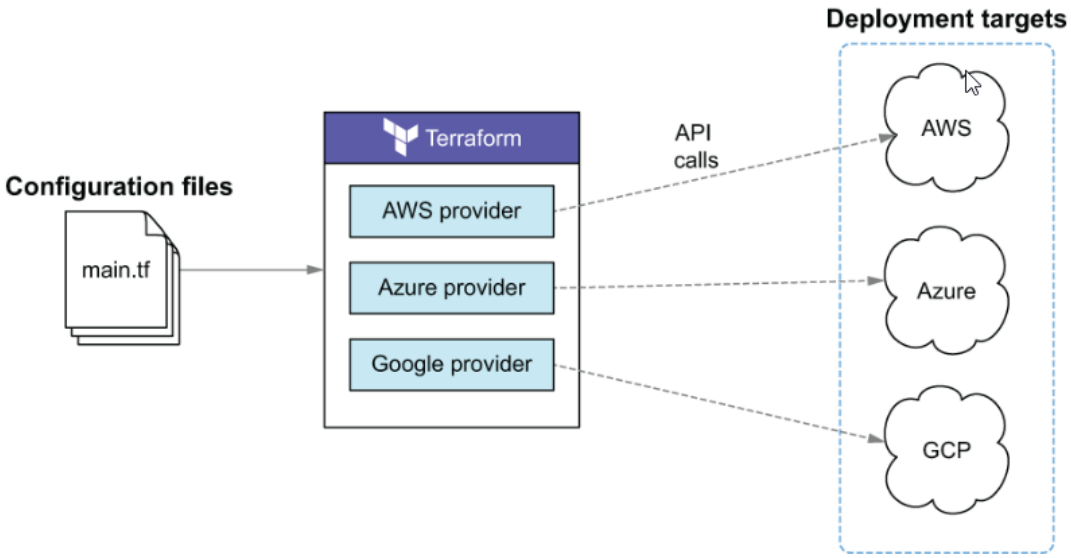
"Motor" koji pokreće Terraform se naziva Terraform core, besplatni softver otvorenog kôda ponuđen pod licencom Mozilla Public License v2.0. Ova licenca propisuje da bilo tko može koristiti, distribuirati ili mijenjati softver za privatne i komercijalne svrhe. Činjenica da je besplatan je velika prednost jer se korisnik ne mora brinuti o dodatnim troškovima prilikom korištenja Terraforma. Osim toga, korisnik ima potpunu transparentnost o proizvodu i načinu na koji radi. Nema premium verzije Terraforma, ali za velike korporacije postoji Terraform Cloud i Terraform Enterprise.

Deklarativno programiranje

Deklarativno programiranje predstavlja logiku računanja (što) bez opisivanja postupka računanja (kako). Umjesto pisanja uputa korak po korak, opisuje se što se želi. Primjeri deklarativnog programiranja uključuju upite baze podataka (SQL), funkcionalne jezike (Haskell, Clojure), jezike za konfiguraciju (XML, JSON) i većinu IaC alata (Ansible, Chef, Puppet). Deklarativno programiranje je u suprotnosti s imperativnim (ili proceduralnim) programiranjem. Imperativni programski jezici koriste grananje, petlje i izraze za kontrolu protoka sustava, spremanje stanja i izvođenje naredbi. Gotovo svi tradicionalni programski jezici su imperativni (Python, Java, C itd.).

Nespecifičan za oblak (engl. Cloud-agnostic)

Nespecifičan za oblak znači da se bez problema može izvoditi na bilo kojoj platformi korištenjem istog seta alata i radnih tokova. Terraform se smatra nespecifičnim jer se može kreirati infrastruktura na AWS-u jednako lako kao što bi se to moglo na GCP-u, Azureu ili čak privatnom poslužitelju (Slika 2.5 Kreiranje infrastruktura na više oblaka). Važno je biti nespecifičan jer to znači da korisnik nije vezan za određenog davatelja usluga i da ne mora učiti cijelu novu tehnologiju svaki put kada prelazi na drugog davatelja usluga.



Slika 2.5 Kreiranje infrastruktura na više oblaka

Terraform se integrira s različitim oblacima putem Terraform Providers. Terraform providers su dodaci za Terraform koji su dizajnirani za sučelje s vanjskim API-jima. Svaki davatelj usluge oblaka održava vlastiti Terraform provider, omogućavajući Terraformu upravljanje resursima u tom oblaku. Terraform providersi su napisani u jeziku Go i distribuirani su kroz Terraform Registry. Oni rukuju svim proceduralnim logikama za autentifikaciju, izradu zahtjeva za API-jem, te upravljanje vremenskim ograničenjima i pogreškama.“ (Winkler, 2021, Chapter 1)

2.4.2. Nedostaci

Grafičko sučelje

Terraform nema grafičko korisničko sučelje. Koristi se putem naredbene linije (engl. command line) ili pomoću skripte za upravljanje infrastrukturom. Ovaj pristup omogućuje precizno i programibilno upravljanje infrastrukturom, što je posebno korisno za DevOps timove i druge stručnjake za infrastrukturu. Međutim, nedostatak grafičkog sučelja može biti zastrašujući za neke korisnike koji nisu navikli na korištenje naredbene linije ili za one kojima je potreban interaktivni pristup za upravljanje infrastrukturom.

2.5. Usporedba Ansiblea i Terraforma

Uspoređeni su Ansible i Terraform alati međusobno te se, također, osvrnulo i na ostale popularne IaC alate. Alati su uspoređeni po različitim kriterijima:

- Alati za upravljanje konfiguracijom naspram alata za kreiranje resursa
- Mutabilna infrastruktura naspram imutabilne infrastrukture
- Proceduralni jezik naspram deklarativnog jezika
- Utjecaj velike zajednice naspram utjecaja manje zajednice
- Ustanovljeni programi naspram novih programa na tržištu
- Korištenje više alata zajedno.

Alati za upravljanje konfiguracijom naspram alata za kreiranje resursa

Chef, Puppet, Ansible i SaltStack su alati za upravljanje konfiguracijom dok su CloudFormation, Terraform i OpenStack Heat alati za kreiranje resursa. Iako razlika nije u potpunosti jasna budući da alati za upravljanje konfiguracijom obično mogu imati jedan dio alata za kreiranje resursa (npr. S Ansibleom se može kreirati poslužitelja), a alati za kreiranje resursa obično mogu imati dio alata za upravljanje konfiguracijom (npr. na svakom poslužitelju s Terraformom mogu se pokrenuti konfiguracijske skripte), obično se želi odabrati alat koji je najpogodniji za konkretan slučaj. (Brikman, 2019, p. 40)

Mutabilna infrastruktura naspram imutabilne infrastrukture

Alati za upravljanje konfiguracijom poput Chefa, Puppeta, Ansiblea i SaltStacka obično se temelje na zadanim postavkama paradigme promjenjive infrastrukture. Na primjer, ako se Chefu naredi da instalira novu verziju OpenSSLa, on će pokrenuti ažuriranje softvera na postojećim poslužiteljima i promjene će se dogoditi na tim poslužiteljima. Tijekom vremena, kako se primijeni više ažuriranja, svaki poslužitelj stvara jedinstvenu povijest promjena. Kao rezultat, svaki poslužitelj postaje nešto drugačiji od svih drugih, što dovodi do suptilnih grešaka u konfiguraciji koje je teško dijagnosticirati. Čak i s automatiziranim testovima te je greške teško uhvatiti; promjena u konfiguraciji upravljanja može dobro funkcionirati na testnom poslužitelju, ali ista ta promjena može drugačije djelovati na proizvodnom poslužitelju zbog toga što proizvodni poslužitelj ima pohranjene promjene koje su se događale mjesecima, a koje nisu prikazane u testnom okruženju. (Brikman, 2019, p. 40)

Proceduralni jezik naspram deklarativnog jezika

Chef i Ansible potiču proceduralni stil u kojem se piše kôd koji određuje korak po korak kako postići željeno konačno stanje. Terraform, CloudFormation, SaltStack, Puppet i OpenStack Heat potiču deklarativni stil u kojem se piše kôd koji određuje korisnikovo željeno konačno stanje, a IaC alat je odgovoran za pronalaženje načina da se to stanje postigne. (Brikman, 2019, p. 41)

Utjecaj velike zajednice naspram utjecaja manje zajednice

Kad god se odabere tehnologija, odabire se i njezina zajednica. U mnogim slučajevima, ekosustav oko projekta može imati veći utjecaj na korisnikovo iskustvo, nego sama kvaliteta tehnologije. Zajednica određuje koliko ljudi sudjeluje u projektu, koliko će dodataka, integracija i proširenja biti dostupno, koliko je lako pronaći pomoć online (npr. blogovi i pitanja na StackOverflowu) i koliko je lako zaposliti nekoga da korisniku pomogne (npr. zaposlenika, savjetnika ili podršku tvrtke). Teško je napraviti točnu usporedbu između zajednica, ali se mogu primijetiti neki trendovi pretraživanjem na internetu. (Brikman, 2019, p. 47)

Tablica 1 Komparacija IaC zajednica (Brikman, 2019, p. 48)

	Izvorni kôd	**Platforma za računarstvo u oblaku**	**Razvijatelji**	**Priručnici**	**StackOverflow**
Chef	Otvoren	Sve	562	3,832	5,982
Puppet	Otvoren	Sve	515	6,110	3,585
Ansible	Otvoren	Sve	4,386	20,677	11,746
SaltStack	Otvoren	Sve	2,237	318	1,062
CloudFormation	Zatvoren	AWS	?	377	3,315
Heat	Otvoren	Sve	361	0	88
Terraform	Otvoren	Sve	1,261	1,462	2,730

Ustanovljeni program naspram programa novih na tržištu

Ovo je još jedan čimbenik koji treba uzeti u obzir pri odabiru bilo koje tehnologije. To nije usporedba jednakih stvari, jer različiti alati imaju različite sheme izrade verzija, ali neki trendovi su jasni. Terraform je, bez sumnje, najmlađi IaC alat u ovoj usporedbi. (Brikman, 2019, p. 49)

Tablica 2 Komparacija verzija IaC alata (Brikman, 2019, p. 49)

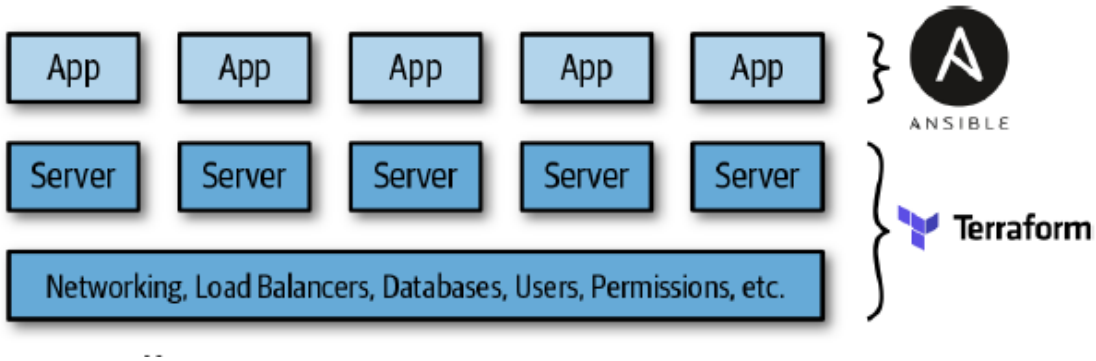
	Prvo izdanje	**Trenutna verzija**
Puppet	2005	6.0.9
Chef	2009	12.19.31
CloudFormation	2011	???
SaltStack	2011	2019.2.0
Ansible	2012	2.5.5
Heat	2012	12.0.0
Terraform	2014	0.12.0

Korištenje više alata zajedno

Ovo su tri najčešće kombinacije koje dobro funkcioniraju i puno tvrtki ih koristi:

- Alati za kreiranje resursa plus alati za upravljanje konfiguracijom

Primjer: Terraform i Ansible. Terraform se koristi za kreiranje svih temeljnih infrastruktura, uključujući topologiju mreže (npr. virtualne privatne mreže, podmreže, tablice ruta), pohrane podataka (npr. MySQL, Redis), balansere opterećenja i poslužitelje. Zatim se Ansible koristi za kreiranje aplikacija na tim poslužiteljima, kako je prikazano na Slika 2.6. (Brikman, 2019, p. 50)

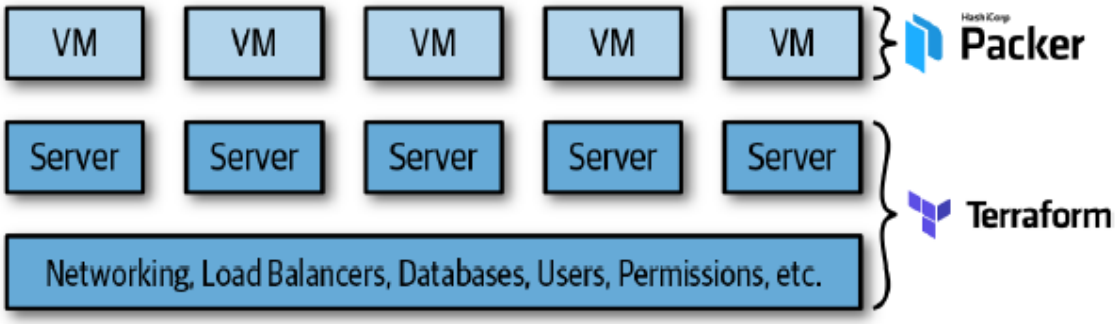


Slika 2.6 Korištenje Terraforma i Ansible zajedno (Brikman, 2019, p. 50)

Ovo je jednostavan pristup za početak jer nema dodatne infrastrukture za pokretanje (Terraform i Ansible su samo klijentske aplikacije (engl. client-only applications)) i postoji mnogo načina da se usklade u radu. Glavni nedostatak je da je za korištenje Ansiblea obično potrebno pisati puno proceduralnog kôda, s mutabilnim poslužiteljima, tako da kako se baza kôda, infrastruktura i tim razvijaju, održavanje postaje teže. (Brikman, 2019, p. 50)

- Alati za kreiranje resursa plus alati za izradu predložaka poslužitelja

Primjer: Terraform i Packer. Packer se koristi za pakiranje korisnikovih aplikacija kao preslika VM-a. Zatim se Terraform koristi za kreiranje (a) poslužitelja s tim preslikama VM-a i (b) ostatka infrastrukture, uključujući topologiju mreže (npr. podmreže, tablice ruta), pohrane podataka (npr. MySQL, Redis) i balansere opterećenja, kako je prikazano na Slika 2.7. (Brikman, 2019, p. 50)

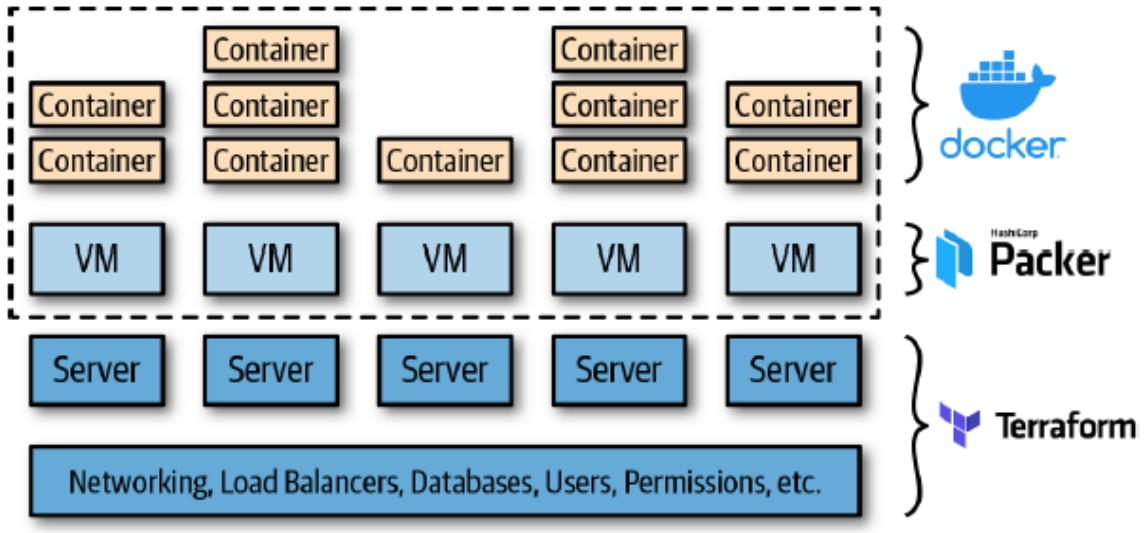


Slika 2.7 Korištenje Terraforma i Packera zajedno (Brikman, 2019, p. 51)

Ovo je, također, jednostavan pristup za početak jer nema dodatne infrastrukture za pokretanje (Terraform i Packer su samo klijentske aplikacije (engl. client-only applications)). Ovo je imutabilna infrastruktura što olakšava održavanje. Međutim, postoje dva glavna nedostatka. Prvo, kreiranje VM-ova može dugo trajati, što usporava brzinu iteracije. Drugo, strategije objavljivanja koje se mogu implementirati s Terraformom su ograničene (npr. ne može se implementirati nativni *blue-green deployment*), tako da na kraju ili se mora pisati puno složenih skripti za kreiranje ili se moraju koristiti alati za orkestraciju. (Brikman, 2019, p. 51)

- Alati za kreiranje resursa plus alati za izradu predložaka poslužitelja plus orkestracijski alati

Primjer: Terraform, Packer, Docker i Kubernetes. Koristi se Packer da se stvori preslika VM-a koja sadrži instaliran Docker i Kubernetes. Zatim, koristi se Terraform da se kreira (a) klaster poslužitelja, unutar kojeg svaki izvodi presliku VM-a, i (b) ostatak infrastrukture, uključujući topologiju mreže (npr. podmreže, tablice ruta), pohrane podataka (npr. MySQL, Redis) i balansere opterećenja. Na kraju, kada se klaster servera pokrene, on formira klastere Kubernetesa koji se koristi da se pokreće i upravlja aplikacijama u Dockeru, kako je prikazano na Slika 2.8. (Brikman, 2019, p. 51)



Slika 2.8 Korištenje Terraforma, Packera, Dockera i Kubernetesa zajedno (Brikman, 2019, p. 52)

Prednost ovog pristupa je da se Docker preslike brzo izrađuju, mogu se pokrenuti i testirati na lokalnom računalu i mogu se iskoristiti sve ugrađene funkcionalnosti Kubernetesa, uključujući razne strategije kreiranja, automatsko obnavljanje (engl. auto healing), automatsko skaliranje i slično. Nedostatak je dodatna složenost u pogledu dodatne infrastrukture za pokretanje (kreiranje i upravljanje klastera Kubernetesa je komplicirano i skupo, međutim većina glavnih pružatelja usluga u oblaku sada pruža upravljane Kubernetes usluge, što može olakšati dio posla), kao i u pogledu nekoliko dodatnih slojeva apstrakcije (Kubernetes, Docker, Packer) za učenje, upravljanje i ispravljanje pogreškama. (Brikman, 2019, p. 52)

3. Dizajn testne okoline

U današnjem svijetu u kojem se tehnologija brzo razvija i poduzeća stalno traže načine za povećanje učinkovitosti i automatizaciju infrastrukture, konfiguracije sustava postaju sve važnije. Ansible i Terraform su dva popularna alata otvorenog kôda koji se često koriste u modernim DevOps okruženjima kako bi se automatizirali procesi upravljanja infrastrukturom i konfiguracije sustava. Cilj ovog projekta je izvršiti komparativnu analizu ova dva alata te predočiti način na koji funkcioniraju.

3.1. Opis okoline i zahtjeva

Korištena je VMware Workstation platforma i AWS usluga. Unutar VMwarea kreirana su tri virtualna računala, dva za Ansible i jedan za Terraform. Za Ansible je napravljeno jedno virtualno računalo koje je bilo kontrolno, a s drugim virtualnim računalom je upravljano pomoću Ansiblea. Oba virtualna računala su koristila isti operacijski sustav (Ubuntu 20.04) i bila su konfigurirana sa stvarnim IP adresama kako bi se Ansible mogao povezati i upravljati ciljnim strojem.

Za Terraform je napravljeno jedno virtualno računalo koje je bilo kontrolno, a pomoću njega je kreirano drugo virtualno računalo unutar AWS platforme. Oba virtualna računala su, također, koristila isti operacijski sustav (Ubuntu 20.04).

Kako bi testirali okruženje, instalirali smo LAMP stack. LAMP stack je skup tehnologija koji se sastoji od Linux operativnog sustava, Apache web poslužitelja, MySQL baze podataka i PHP (ili drugog jezika za programiranje slične namjene) za izradu dinamičkih web stranica i web aplikacija. Linux operativni sustav pruža sigurno i stabilno okruženje za razvoj i izradu web aplikacija. Apache je najčešće korišteni web poslužitelj na svijetu, a koristi se za obradu zahtjeva posjetitelja web stranica i isporuku web sadržaja. MySQL je vrlo popularna baza podataka koja se često koristi s LAMP stackom, ali se mogu koristiti i druge baze podataka, kao što su PostgreSQL, MariaDB i druge. PHP (ili drugi jezici za programiranje slične namjene) se koristi za izradu dinamičkog sadržaja web stranica, poput, formi za unos podataka, interakcije s bazom podataka i dr.

LAMP stack se često koristi za izradu dinamičkih web stranica i web aplikacija, a prednosti uključuju brzinu razvoja, skalabilnost i prilagodljivost. Postoji velika zajednica razvijatelja koji rade s LAMP stackom, što znači da postoji mnogo resursa, alata i biblioteka koji su dostupni za uporabu.

3.2. Dizajn metoda testiranja

Prvi dio praktičnog rada uključio je postavljanje čiste infrastrukture bez nepotrebnih opterećenja s čistom implementacijom Ubuntu operativnog sustava. Ubuntu je popularan i besplatan operativni sustav baziran na Linuxu. Osnovao ga je Mark Shuttleworth 2004. godine s ciljem da bude jednostavan za korištenje i dostupan svima. Ubuntu je danas jedan od najpopularnijih operativnih sustava na svijetu, s velikom zajednicom korisnika i razvijatelja koji pružaju podršku i rade na razvoju projekta. Jedna od glavnih prednosti Ubuntu operativnog sustava je jednostavnost korištenja i instalacije, što ga čini popularnim izborom za početnike u svijetu Linuxa. Također, Ubuntu se može koristiti na različitim vrstama uređaja, uključujući osobna računala, servere i mobilne uređaje. Ubuntu dolazi s velikim brojem aplikacija koje se lako mogu instalirati putem softverskog centra.

U drugoj fazi je postavljena konfiguracija na kontrolnoj strani Ansiblea i Terraforma. Postavljena je konfiguracija na kontrolnoj VM Ansiblea kako bi se omogućila upotreba Ansiblea za konfiguriranje ciljnih sustava. Kontrolna VM mora imati instaliran Ansible, a inventar ciljnih sustava i autentikacija moraju biti konfigurirani prije nego što se mogu definirati Ansible playbookovi. Playbookovi su datoteke koje opisuju zadatke koji će se izvršiti na ciljnim sustavima, uključujući instalaciju aplikacija, konfiguraciju usluga i resursa. Konfiguracija na kontrolnoj VM Ansiblea omogućuje brzo i jednostavno konfiguriranje ciljnih sustava, što može biti ključno za automatizaciju IT infrastrukture i poboljšanje produktivnosti. Terraform VM mora imati instaliran Terraform i vezu na internet.

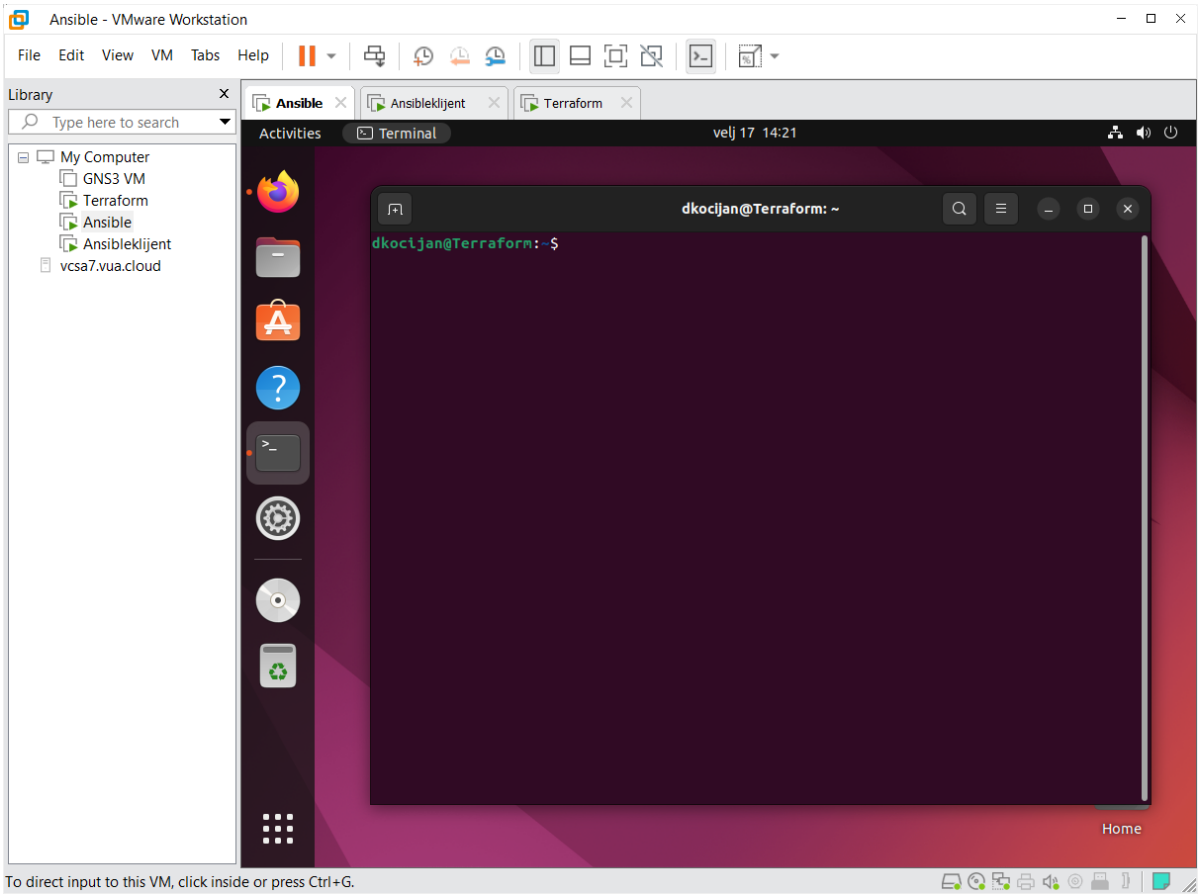
Treća faza projekta odnosi se na instalaciju LAMP stacka na klijentsku stranu infrastrukture. LAMP stack je popularna tehnologija koja se često koristi za razvoj web aplikacija, a sastoji se od Linux operacijskog sustava, Apache web poslužitelja, MySQL baze podataka i PHP dinamičkog programskog jezika.

4. Implementacija rješenja i testiranje

U ovom poglavlju opisano je kako je kreirana i instalirana potrebna infrastruktura.

4.1. Opis instalacije okoline

Unutar VMware platforme kreirana su potrebna virtualna računala. Za početak je sa službene Ubuntu internet stranice preuzeta ISO preslika operacijskog sustava verzije Ubuntu 22.04.1. Nakon uspješne instalacije su podešene osnovne postavke po osobnim preferencama te su dodijeljeni nazivi virtualnim računalima. Sva tri virtualna računala su podešena jednako osim naziva koji su sljedeći: Ansible, Ansibleklijent i Terraform. Na računalima je postavljen korisnik dkocijan kojem su dodijeljena sudo prava.



Slika 4.1 Prikaz VMware workstation platforme s kreiranim virtualnim poslužiteljima

4.2. Opis alata i metoda za testiranje

Za testiranje je instaliran LAMP stack pomoću Ansible i Terraform alata. LAMP stack se sastoji od Apache web servera, MySQL baze podataka i PHP programskog jezika. Testiranje ovog stacka pomoću Ansible i Terraform alata omogućuje usporedbu ova dva alata za upravljanje infrastrukturom. Prvo je instaliran LAMP stack na virtualno računalo pomoću Ansible alata. Na kontrolnoj VM je instaliran Ansible, a zatim je s njim instaliran LAMP stack na drugi virtualni stroj. Za testiranje Terraform instalacije napravljen je novi AWS EC2 poslužitelj s LAMP stackom. Prvo je konfiguriran AWS CLI, a zatim je kreirana Terraform datoteka koja opisuje AWS resurse. U datoteci su specificirani AWS regija, tip instance, OS preslika i drugi detalji. Nakon toga, Terraform je automatski stvorio poslužitelja s potrebnim softverom i postavkama.

4.3. Provedba testiranja

4.3.1. Ansible

Kako bi se mogao instalirati LAMP stack pomoću Ansiblea, potrebno je prvo kreirati playbook u kojem je navedeno što se želi instalirati. Playbook koji je korišten može se vidjeti u primjeru Kôd 4.1 [5]:

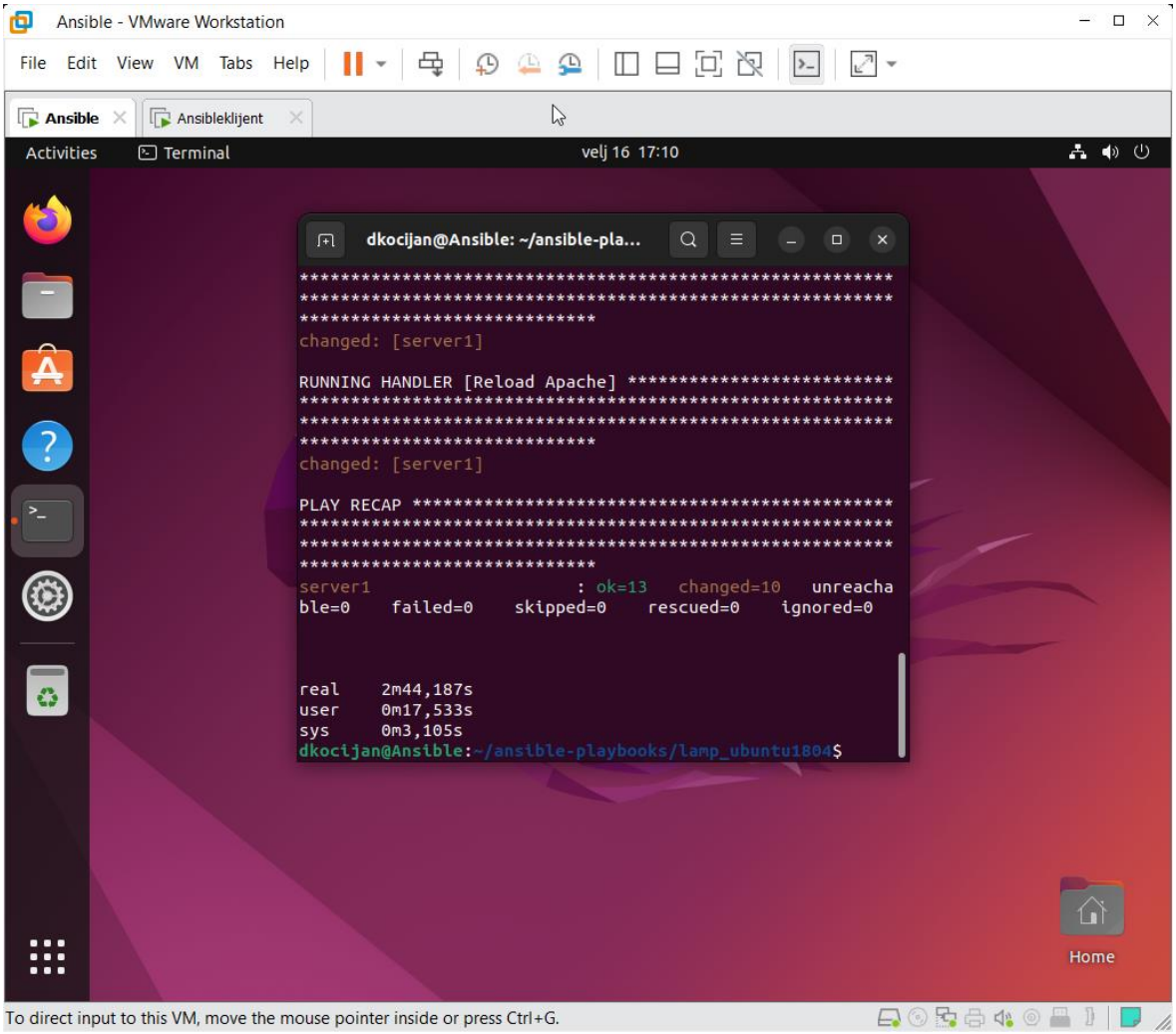
```
---
- hosts: servers
  become: true
  vars_files:
    - vars/default.yml

  tasks:
    - name: Install prerequisites
      apt: name={{ item }} update_cache=yes state=latest
force_apt_get=yes
      loop: [ 'aptitude' ]

#Apache Configuration
    - name: Install LAMP Packages
      apt: name={{ item }} update_cache=yes state=latest
      loop: [ 'apache2', 'mysql-server', 'python3-pymysql',
'php', 'php-mysql', 'libap>
```

```
    - name: Create document root
      file:
        path: "/var/www/{{ http_host }}"
        state: directory
        owner: "{{ app_user }}"
        mode: '0755'

    - name: Set up Apache virtualhost
      template:
        src: "files/apache.conf.j2"
        dest: "/etc/apache2/sites-available/{{ http_conf }}"
      notify: Reload Apache

    - name: Enable new site
      shell: /usr/sbin/a2ensite {{ http_conf }}
      notify: Reload Apache

    - name: Disable default Apache site
      shell: /usr/sbin/a2dissite 000-default.conf
      when: disable_default
      notify: Reload Apache

#MySQL Configuration
    - name: Sets the root password
      mysql_user:
        name: root
        password: "{{ mysql_root_password }}"
        login_unix_socket: /var/run/mysqld/mysqld.sock

    - name: Removes all anonymous user accounts
      mysql_user:
        name: ''
        host_all: yes
        state: absent
        login_user: root
        login_password: "{{ mysql_root_password }}"

    - name: Removes the MySQL test database
      mysql_db:
        name: test
```

```
        state: absent
        login_user: root
        login_password: "{{ mysql_root_password }}"

#UFW Configuration
    - name: "UFW - Allow HTTP on port {{ http_port }}"
      ufw:
        rule: allow
        port: "{{ http_port }}"
        proto: tcp

#PHP Info Page
    - name: Sets Up PHP Info Page
      template:
        src: "files/info.php.j2"
        dest: "/var/www/{{ http_host }}/info.php"

  handlers:
    - name: Reload Apache
      service:
        name: apache2
        state: reloaded

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

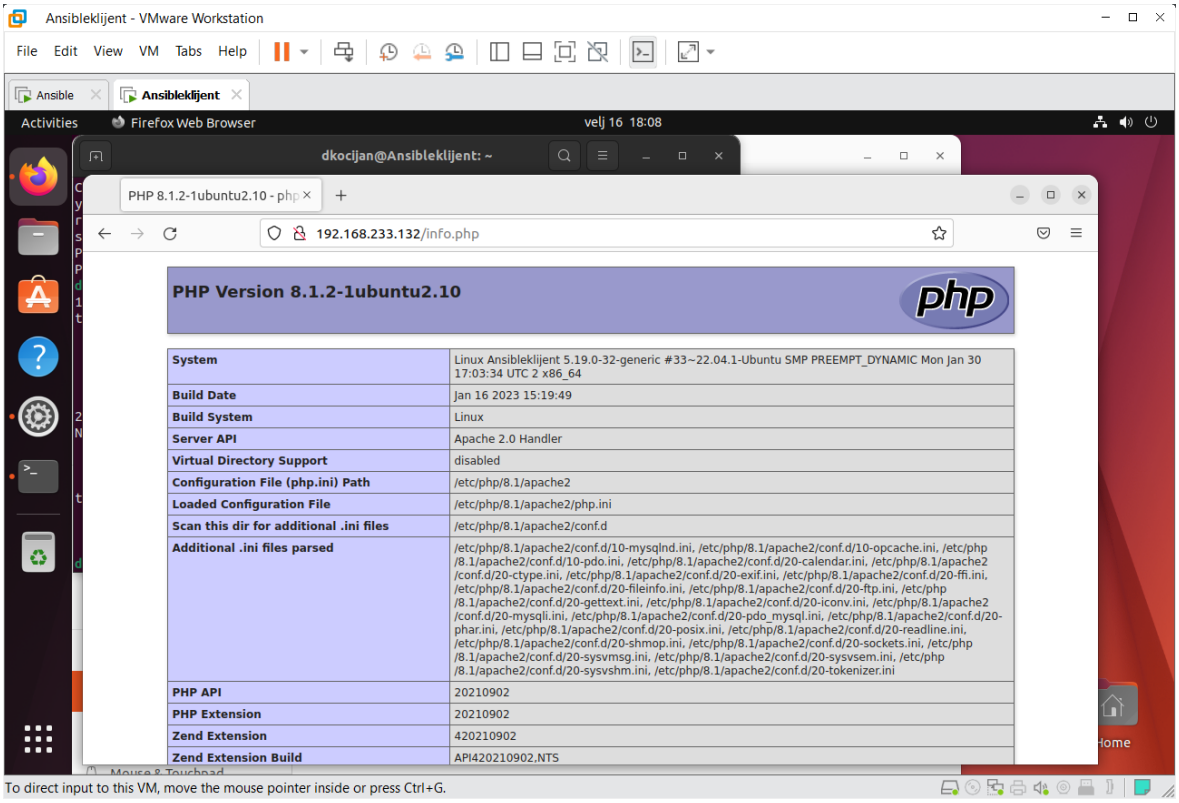
Kôd 4.1 Primjer playbooka za instalaciju LAMP stacka

Da bi se pokrenuo kreirani playbook unosi se naredba `time ansible-playbook playbook.yml`. Segment „time“ na kraju procesa prikazuje koliko je bilo potrebno vremena da se izvrši instalacija što se može vidjeti na Slika 4.2. Na istoj slici može se vidjet da je playbook uspješno primijenjen bez grešaka što znači da je LAMP stack instaliran.



Slika 4.2 Prikaz izvršenog playbooka

Kako bi se provjerilo da je LAMP stack uistinu instaliran, najjednostavniji način je da se na ciljnom virtualnom računalu otvori internet pretraživač i da se pretraži http://server_host_or_IP/info.php što je u ovom slučaju http://192.168.233.132/info.php. Otvaranjem stranice može se vidjet koja verzija PHP-a je instalirana.



Slika 4.3 Prikaz PHP verzije

4.3.2. Terraform

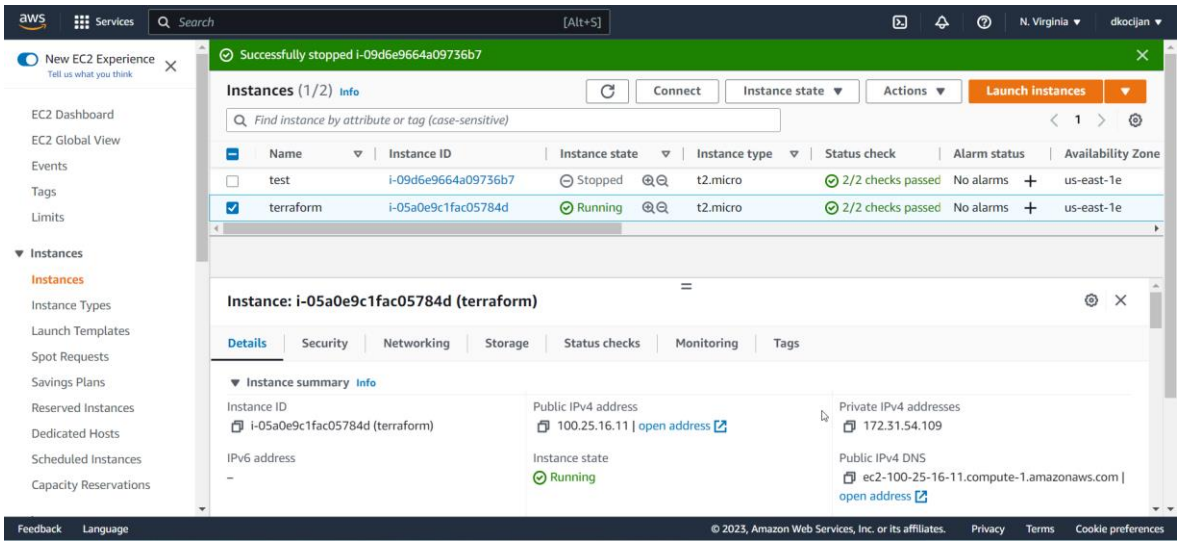
Kako bi se mogao instalirati LAMP stack pomoću Terraforma, potrebno je prvo kreirati datoteku sa specifikacijama AWS resursa u kojoj se navodi sve što se želi instalirati. Datoteka koja je korištena može se vidjeti u primjeru Kôd 4.2 [6]:

```
  provider "aws" {
  access_key = "my_access_key"
  secret_key = "my_secret_key "
  region     = "us-east-1"
}

resource "aws_instance" "terraform" {
  ami           = "ami-0557a15b87f6559cf"
  instance_type = "t2.micro"
  key_name      = "terraformkey"
  vpc_security_group_ids = ["${aws_security_group.web.id}"]
  user_data = <<-EOF
              #!/bin/bash
              sudo apt-get update
              sudo apt-get install apache2 mysql-server php
libapache2-mod-php php-mys>
              EOF

  tags = {
    Name = "web"
  }
}

resource "aws_security_group" "web" {
  name_prefix = "web"
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
```

```
      cidr_blocks = ["0.0.0.0/0"]
    }
}
```

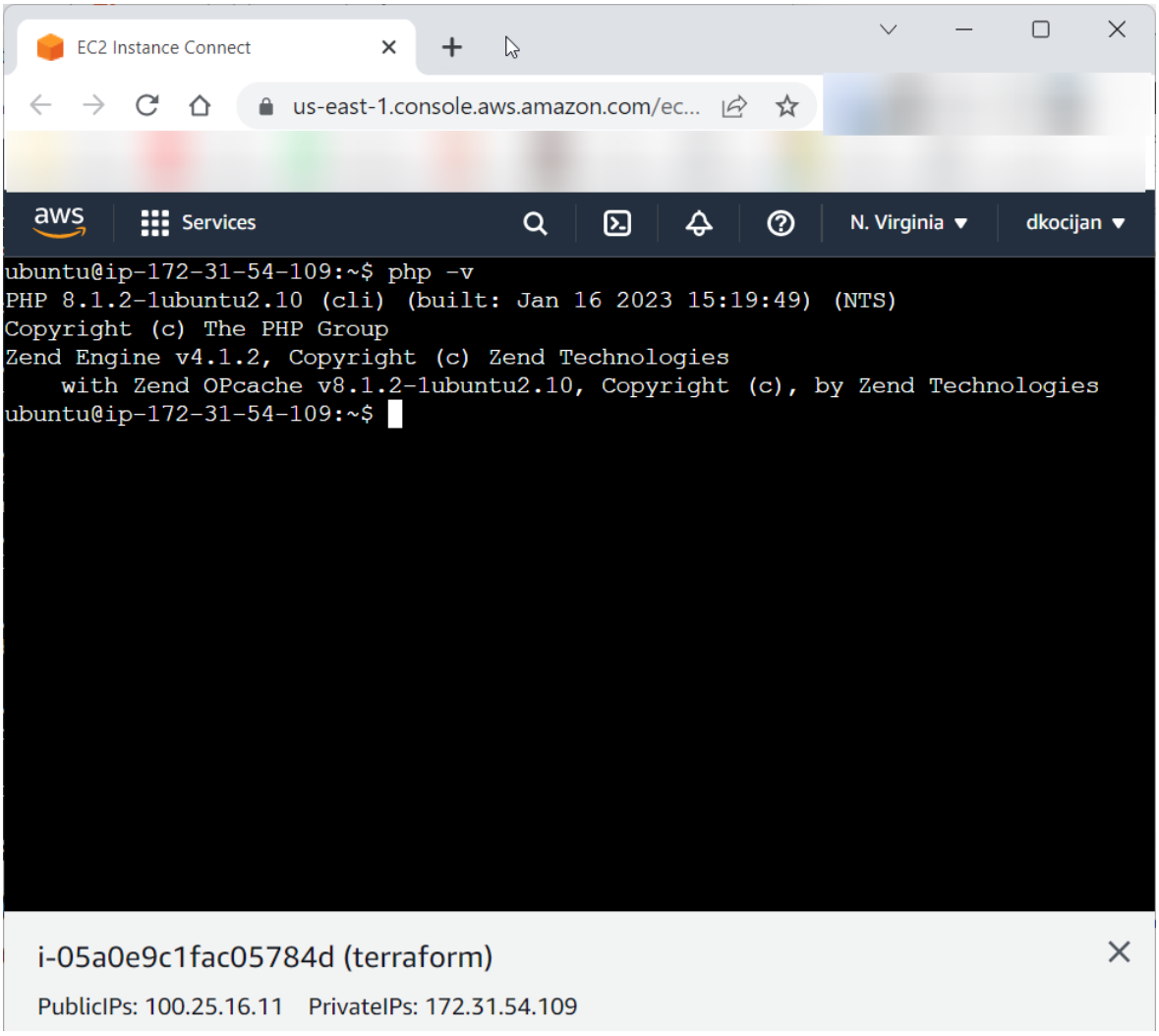
Kôd 4.2 Primjer AWS datoteke sa specifikacijama

Kako bi se izvršila kreirana datoteka, izvršava se naredba `terraform apply`. Nakon uspješnog izvršenja datoteke dolazi zelena poruka „*Apply complete*“. U AWS konzoli, u sekciji EC2, trebala bi se vidjeti nova instanca koja je kreirana pomoću Terraform datoteke, a koja je vidljiva na Slika 4.4.



Slika 4.4 Nova instanca u AWS-u

Kako bi se provjerilo da je LAMP stack uistinu instaliran unutar konzole kreirane instance, izvršava se naredba `php -v` koja daje uvid u verziju instaliranog PHP-a, što je vidljivo na Slika 4.5.



Slika 4.5 Prikaz PHP verzije

Ovime je uspješno završen praktični dio instalacije LAMP stacka te se može krenuti s analizom rezultata.

5. Analiza rezultata

5.1. Analiza upotrebljivosti (jednostavnost jezika)

Ansible i Terraform su alati za automatizaciju infrastrukture, ali se razlikuju u sintaksi i pristupu. Ansible je bezagentni alat koji se temelji na deklarativnom jeziku YAML. Koristi module za upravljanje konfiguracijom sustava i nudi mnoge funkcije kao što su upravljanje paketima, servisima, korisnicima i datotekama. Ansible, također, podržava izvršavanje ad-hoc naredbi na udaljenim sustavima. Primjer sintakse za Ansible:

```
- hosts: servers
  become: true
  vars_files:
    - vars/default.yml

  tasks:
    - name: Install prerequisites
      apt: name={{ item }} update_cache=yes state=latest
force_apt_get=yes
      loop: [ 'aptitude' ]

#Apache Configuration
    - name: Install LAMP Packages
      apt: name={{ item }} update_cache=yes state=latest
```

Kôd 5.1 Dio korištenog Ansible kôda

Terraform, s druge strane, je deklarativni jezik usluga infrastrukture kao kôda (IaC) koji se koristi za upravljanje resursima na platformama za računarstvo u oblaku. Terraform koristi HCL (HashiCorp Configuration Language) jezik za opisivanje infrastrukturnih resursa i modula. Terraform, također, podržava kontrolu verzije i upravljanje resursima na platformama za računarstvo u oblaku.

Primjer sintakse za Terraform:

```
provider "aws" {
  access_key = "my_access_key"
  secret_key = "my_secret_key "
  region     = "us-east-1"
}

resource "aws_instance" "terraform" {
  ami           = "ami-0557a15b87f6559cf"
  instance_type = "t2.micro"
  key_name      = "terraformkey"
  vpc_security_group_ids = ["${aws_security_group.web.id}"]
  user_data = <<-EOF
              #!/bin/bash
              sudo apt-get update
              sudo apt-get install apache2 mysql-server php
```

Kôd 5.2 Dio korištenog Terraform kôda

Na prvi pogled Ansibleov kôd je jednostavniji i lakši za razumijevanje, ali Terraformov kôd također nije teško razumjeti. Ako usporedite duljinu kôda u ovoj situaciji, Terraform je svakako u prednosti jer ima samo 37 linija kôda, dok Ansible ima 83 linije. Uzimajući u obzir da je kraći kôd lakši za razumijevanje, održavanje i skaliranje, Terraform je bolji izbor u nekim slučajevima. Odluka o korištenju Ansiblea ili Terraforma ovisi o konkretnoj situaciji, veličini infrastrukture, potrebama za skaliranjem i drugim specifikacijama okruženja.

5.2. Analiza mogućnosti automatizacije

Terraform i Ansible se mogu koristiti za automatizaciju infrastrukture, ali se razlikuju u načinu na koji to čine.

Terraform je namijenjen za upravljanje infrastrukturnim resursima na različitim platformama za računarstvo u oblaku i omogućava automatizaciju kreiranja, ažuriranja i brisanja resursa. Fleksibilan je jer podržava konfiguraciju i upravljanje resursima u različitim okruženjima kao što su AWS, Azure, Google Cloud i druga. Terraform koristi HCL (HashiCorp Configuration Language) za opisivanje infrastrukturnih resursa i omogućava

automatsko izvršavanje planova i primjena promjena. Kako bi se izbjeglo dupliciranje kôda i olakšalo upravljanje Terraform omogućuje modularizaciju infrastrukture.

Ansible se koristi za automatizaciju konfiguracije sustava. Fleksibilan je jer se može koristiti za upravljanje različitim vrstama resursa kao što su mrežni uređaji, poslužitelji, baze podataka i drugi. Ansible koristi deklarativni YAML jezik za definiranje stanja sustava i automatsko upravljanje konfiguracijom.

U slučaju ovog rada, Terraform se pokazao fleksibilnijim jer smo ga koristili za izradu poslužitelja i instaliranje LAMP stacka, dok smo s Ansibleom trebali sami stvoriti poslužitelj i zatim na njega instalirati LAMP stack. U slučaju proširenja infrastrukture u ovom radu s Terraformom ćemo brže proširiti infrastrukturu jer će nam on kreirati nove poslužitelje i instalirati željene aplikacije na istima. Međutim odluka o korištenju Ansiblea ili Terraforma ovisi o konkretnoj situaciji, veličini infrastrukture, potrebama za skaliranjem i drugim specifikacijama okruženja.

5.3. Analiza upravljanje resursima

Terraform i Ansible se mogu koristiti za upravljanje resursima, ali se razlikuju u načinu na koji to čine.

Terraform je dizajniran za upravljanje infrastrukturnim resursima na različitim platformama za računarstvo u oblaku, kao što su AWS, Azure, Google Cloud i druge. Koristi deklarativni jezik koji omogućava korisnicima da opišu svoje željene resurse, a Terraform će se pobrinuti za kreiranje, ažuriranje i brisanje resursa u skladu s tim opisom. Terraform koristi tzv. datoteku stanja, u kojoj se nalaze informacije o trenutnom stanju infrastrukture. Terraform osigurava da se infrastruktura kreira u skladu s definiranim planom i osigurava konzistentnost između stvarne infrastrukture i stanja opisanog u datoteci stanja (engl. state file).

Ansible se može koristiti za upravljanje resursima kao što su mrežni uređaji, poslužitelji, baze podataka i drugi. Ansible koristi ad-hoc ili playbook naredbe za definiranje konfiguracije sustava. Ansible se ne oslanja na datoteku stanja, već svaki put kada se playbook pokreće, Ansible izvršava naredbe koje su definirane u playbooku kako bi postigao željeno stanje sustava. Ansible ne provjerava trenutno stanje sustava, već se oslanja na definiciju u playbooku za uspostavu željenog stanja.

U ovom rada smo kreirali jednostavne skripte instalacije i pokretanja LAMP stacka nema razlike u korištenju Ansiblea ili Terraforma. Ansible ne provjerava stanje resursa na kojem izvršava skriptu nego samo izvrši što je zapisano u playbooku. S druge strane Terraform provjerava stanje i ukoliko unutar kôda naiđe na nešto što već postoji (npr. korisnik s istim imenom) taj dio ne izvršava. Ako bi dvaput ponovili istu skriptu na Ansible sustavu bi mogli na kraju dobiti dva korisnika istog imena dok to na Terraformu nije moguće.

5.4. Analiza skalabilnosti

Kada je riječ o skalabilnosti, Terraform i Ansible nude različite značajke i pristupe.

Terraform je dizajniran za upravljanje tisućama resursa u skladu s definiranim planom. Ima mogućnost korištenja modula, što omogućava organiziranje i ponovnu upotrebu konfiguracije resursa. Moduli su mali dijelovi kôda koji definiraju skup resursa i mogu se upotrebljavati u različitim konfiguracijama, što olakšava upravljanje velikim brojem resursa.

S druge strane, Ansible može imati poteškoće u skaliranju za upravljanje velikim infrastrukturama. Playbookovi nisu dizajnirani za rad sa velikim brojem poslužitelja, a ako se koriste za upravljanje velikim brojem poslužitelja, mogu se pojaviti poteškoće s performansama.

Kada se radi o skaliranju i upravljanju velikim infrastrukturama, Terraform je obično bolji izbor zbog svojih značajki i modula. Terraform omogućuje organiziranje i upravljanje velikim brojem resursa, a Ansible može biti bolji izbor za manje infrastrukture i za složene scenarije automatizacije, kao što su poslužiteljske konfiguracije ili distribuirani sustavi.

Zaključak

U ovom radu uspoređena su dva popularna alata za automatizaciju i upravljanje infrastrukturom - Ansible i Terraform. Iako oba alata nude slične značajke i funkcionalnosti, postoje razlike u načinu na koji se koriste i rješavaju probleme.

Kada je riječ o sintaksi, Terraform i Ansible se razlikuju u načinu na koji definiraju konfiguraciju resursa, pri čemu Terraform koristi HCL, a Ansible YAML jezik. Terraform, također, ima više fokus na infrastrukturu kao kôd, a Ansible je više orijentiran na automatizaciju konfiguracije sustava.

Kada se radi o fleksibilnosti konfiguracije, U slučaju ovog rada, Terraform se pokazao fleksibilnijim jer smo ga koristili za izradu poslužitelja i instaliranje LAMP stacka, dok smo s Ansibleom trebali sami stvoriti poslužitelj i zatim na njega instalirati LAMP stack.

U smislu upravljanja resursima, u ovom radu nema razlike u korištenju Ansiblea ili Terraforma. Ansible ne provjerava stanje resursa na kojem izvršava skriptu nego samo izvrši što je zapisano u playbooku. S druge strane Terraform provjerava stanje i ukoliko unutar kôda naiđe na nešto što već postoji (npr. korisnik s istim imenom) taj dio ne izvršava. Ako bi dvaput ponovili istu skriptu na Ansible sustavu bi mogli na kraju dobiti dva korisnika istog imena dok to na Terraformu nije moguće.

Kada se radi o skalabilnosti, Terraform je obično bolji izbor zbog svojih značajki i modula. Terraform omogućuje organiziranje i upravljanje velikim brojem resursa, a Ansible može biti bolji izbor za manje infrastrukture i za složene scenarije automatizacije, kao što su poslužiteljske konfiguracije ili distribuirani sustavi.

Konačni zaključak je da ovisno o potrebama, Ansible i Terraform nude različite prednosti i mane. Ansible je dobar izbor za automatizaciju konfiguracije pojedinačnih resursa i manjih infrastruktura dok je Terraform bolji izbor za automatizaciju infrastrukture kao kôda i upravljanje većim infrastrukturama. Oba alata se mogu koristiti u kombinaciji kako bi se postigao najbolji mogući rezultat u automatizaciji i upravljanju infrastrukturom.

Popis kratica

IaC	*Infrastructure as code*	Infrastruktura kao kôd
API	*Application programming interface*	Sučelje za programiranje
VM	*Virtual machine*	Virtualno stroj
SSL	*Secure Sockets Layer*	Internetski sigurnosni protokol
AWS	*Amazon Web Services*	Amazon web usluge
IP	*Internet Protocol address*	Adresa internetskog protokola
DNS	*Domain Name System*	Sustav domenskih imena
IT	*information technology*	Informatička tehnologija
HCL	*HashiCorp Configuration Language*	HashiCorp konfiguracijski jezik
SSH	*Secure Shell*	Mrežni komunikacijski protokol
AWX	*Web-based user interface*	Web-bazirano korisničko sučelje
JSON	*JavaScript Object Notation*	JavaScript objektna notacija
XML	*Extensible Markup Language*	Extensible Markup jezik
SQL	*Structured Query Language*	Structured Query jezik
OS	*Operating system*	Operacijski sustav
GCP	*Google Cloud Platform*	Google platforma za računarstvo u oblaku
YAML	*Yet another markup language*	Yet another markup jezik
LAMP	*Linux, Apache, MySQL, PHP*	Linux, Apache, MySQL, PHP
CLI	*Command-line interface*	Sučelje naredbenog retka

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] LORIN HOCHSTEIN, Ansible Up & Running, First Edition, 2015.

[2] LUCA BERTON, Ansible for VMware by Examples, First Edition, 2022; 11/2022 https://doi.org/10.1007/978-1-4842-8879-5

[3] YEVGENIY BRIKMAN, Terraform: Up & Running, Second edition, 2019.

[4] SCOTT WINKLER, Terraform in Action, 2021

[5] https://linuxopsys.com/topics/ansible-playbook-to-install-apache?utm_content=cmp-true; 02/2023

[6] https://www.middlewareinventory.com/blog/terraform-aws-example-ec2/ ; 02/2023