Operativni Sistemi i Racunarstvo U oblaku

II Semestar – 2023/24 – Vježbe

Sedmica 9,10

Handout za Vježbe

Agenda:

- Alati za upravljanje konfiguracijama I generacije
- Alati za upravljanje konfiguracijama II generacije

Kontakt:

Narcisa.hadzajlic@dl.unze.ba

Adin.jahic2019@size.ba

Vernes.Vincevic@unze.ba

IaC - (Infrastruktura kao kod)

Kroz rad smo objasnili sve koncepte koji su potrebni za napredno shvatanje devOps principa. U ovom poglavlju dolazimo do jednog od najvažnijih segmenata rada u kojem će se sve do sada objašnjeno automatizovati kroz unikatan način održavanja čitavih infrastruktura koje na osnovu takozvanih „manifesta“ koji predstavljaju našu tranziciju sa trenutnog stanja u Željenog stanja

Trenutno stanje

Trenutno stanje se definiše kao svi naši trenutni raspoloživi resursi koje posjedujemo u našem okruženju. Ono predstavlja sav potencijal koji posjeduje okruženje, te sve resurse koje možemo upotrijebiti u razvoju našeg orkuženja. Sadržaj trenutnog stanja se može najjednostavnije enkapsulirati sa 3 glavna tipa resursa koje posjeduje, a to su:

- Provajderi (Google Cloud Platform, Amazon Web Service, Microsoft Azure ,OnPrem),
- Infrastruktura (serveri , clusteri, baze, storage),
- Aplikacije (nezavisni softveri koji su integrisani u naše okruženje).

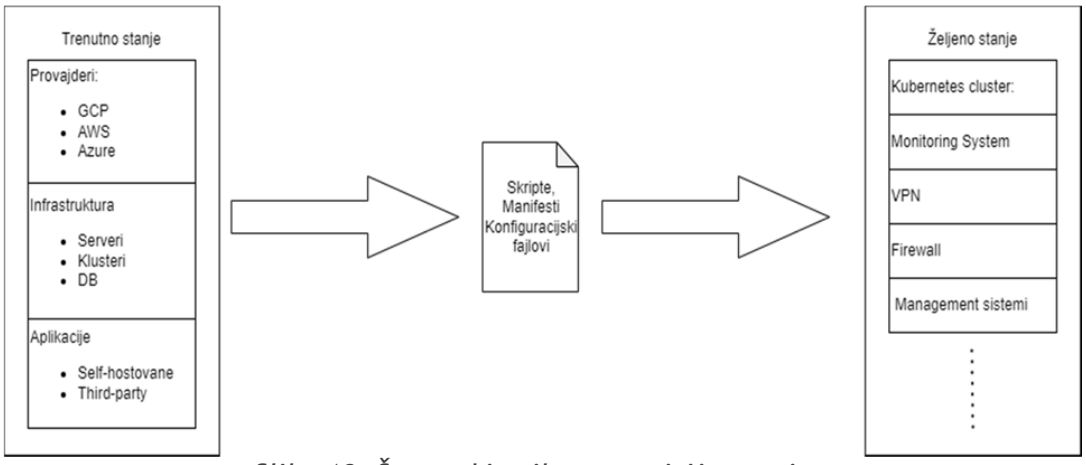
Sve ove stavke zajedno čine naše trenutno stanje. Može se smatrati kao potencijalom koji neko okruženje ispoljava u implementaciji poslovnih potreba, ali da bi ovo stanje imalo ikakvu funkciju mora se definisati željeno stanje.

Željeno stanje

Način na koji se implementira je kroz takovzane manifeste, konfiguracijske fajlove, skripte koje se interpretiraju kao programski jezici od strane računara na kojima se oni izvršavaju. Željenim stanjem implementujemo naše ideje koristeći resurse trenutnog Stanja. Naime ovakvim deklarativnim pisanjem cijele infrastrukture dolazimo do problema kako da garantujemo da će skripta, kod ili manifest koji napišemo raditi na bilo kojem okruženju jer poenta implementacije ovakvih infrastruktura jeste da se izbaci manuelan rad čovjeka prilikom konfiguracije ovakvih okruženja. Moramo biti u stanju da garantujemo da je deklarisana infrastruktura takva da se može pokreniti na svim okruženjima

Ove skripte, manifesti i konfiguracijski fajlovi iz tog razloga znaju biti nevjerovatno kompleksi i dugi. Razlog tome jeste što da bi uspješno deklarisali ovu infrastrukturu moramo automatizovati svaki korak koji čovjek manuelno prođe prilikom manuelne konfiguracije. To uključuje svaku stavku od konfiguracije resursa koji su nam potrebni, provizioniranja virtuelnih mašina, inicijalizacije kontejnera na kojima će se izvršiti aplikacija, definisanja svih programa na samom operativnom sistemu koji su potrebni za rad servisa, održavanje njihovih verzija itd.

Postoji ogroman broj alata koji radi ovakav tip deklaracije infrastruktura i jako je teško da sa posljedicom razvoja tehnologije bilo ko bude u stanju da bude konstantno upućen u sve ove alate koji su potrebni za dalji rad. Sem u slučaju visoke specijalizacije u nekom određenom sistemu koji je dugotrajan proces. Način na koji moderni inžinjeri DevOps-a implementuju svoja rješenja jeste korištenjem potrebnih funkcionalnosti iz veće palete alata koji nam na kraju daju željeni rezultat. Svi manifesti, skripte i konfiguracijski fajlovi se najčešće čuvaju u git repozitorijima, te se kontroliraju putem git sistema na računarima koji se sam po sebi interpretiran kao program u operativnom sistemu računara što tranzitivnim svojstvom nam govori da i izvršne skripte, manifesti i ostali produkti se interpretiraju i izvršavaju kao niz instrukcija preko komandnog sučelja kao kod.



Slika 10. Šematski prikaz tranzicije stanja

Zašto IaC?

Potreba IaC?

Način na koji se do sada radila implementacija infrastruktura je bio veoma zamoran, težak, nekonzistentan i veoma lagano je moglo doći do pucanja sistema koji su rađeni. Rađeni su na principu da sistemski i mrežni administrator manuelno izvršava konfiguraciju svakog pojedinačnog servera koji se treba provizionirati. To je uključivalo čitav proces od instaliranja operativnog sistema koji na sebi nema ništa prethodno instalirano do momenta kada treba potpuno integrisanu aplikaciju pokrenuti.

Ovo je postao s vremenom jako dug, zamoran i repetitivan proces s obzirom da u modernom dobu gdje skalabilnost je jedan od presudnih faktora za opstanak dovelo za posljedicu da se u malo većim okruženjima zamorni proces konfiguracije ponavlja non-stop.

Vrijeme čekanja na završetak konfiguracije je ekstremno dugo, mjeri se u danima ,što je u poslovnom svijetu ogromno kašnjenje što rezultuje manjom naplatom i obračunavanjem penala. Pored toga kada bi mogli u jednu ruku tolerisati dugotrajnost, potrebno je uspostaviti uniformnost tako da se svi iskonfigurisani serveri mogu složiti o zajedničkom skupu pravila koje moraju da poštuju. Primjer pomenutog je da web server na svim serverima koji idu u produkciju moraju da se konfigurišu u LTS verziji, a ne na posljednjoj verziji softvera. Za jako kratko vrijeme se dostigne sistem kaosa u kojem nakon određenog vremena ni ne znamo šta je na serveru postavljeno i hoće li biti kompatibilno sa novim promjenama koje trebaju da se implementuju.

Popravke na ovakvim serverima su se izvršavale na način da sistemski administrator uradi SSH konekciju na serveri i „popravi“ zadani problem. Takav pristup narušava pojam uniformiteta jer ne postoji garancija da u kaotičnoj infrastrukturi rješenje jednog problema na jedan način biti identično na drugom serveru gdje ima isti problem. Privremena hrabrost u rješavanju ovakvih problema se evidentirala u formiranju dokumentacija ili takozvanih „intranet Wiki“ sistema u kojima se prijavljuje svaka promjena na serverima i postavljaju članci o tome kako treba da se pristupa pri rješavanju problema.

Naime u ovakvom sistemu je potrebno da se jednom preskoči prijava promjena u wiki sistem, nakon čega nastaje nered koji narušava uniformitet koji pokušavamo da ostvarimo na našoj infrastrukturi.

Poslije toga su zaživile implementacije „skripti“. Skripte su u ovom slučaju trebale da zamijene instrukcijski copy-paste sistem sa dokumentacijom te da uvedu redukciju vremena i da uvedu pokoji automatizovan proces u svijet u kojem manuelna konfiguracija vlada.

Ovo je omogućilo privremenu pomoć ali nije uspjelo zato što skripte su u svojoj srži pogodne za instalaciju, a ne za poboljšanje. One djeluju na svježim serverima na kojima se tek treba instalirati prvi put sistem. Razlog iz kojeg ne djeluju zato što sa svakim poboljšanjem se mora prilagoditi skripta na način da pišemo gomilu if-then naredbi koje će uvoditi logiku u skripte da je navodi u to kako treba da se ponaša prilikom izvršavanja na različitim okruženjima na kojima su postavljeni serveri. Međutim prije ili kasnije kompleksnost toliko naraste da je lakše ući na SSH konekciju i odraditi manuelno sve što se treba uraditi na serveru što poništava svrhu ovakvog sistema.[14]

Prva Generacija konfiguracijskih alata

Razvoj alata za upravljanje konfiguracijama kao što je Chef, Puppet; Ansible su bili prvi koji su zauzeli tržište prilikom razvoja. Oni su prvi put pomiješali deklarativne naredbe koje koristimo u programiranju sa problemom koji pokušavamo da riješimo u ovom slučaju , a to je tranzicija izmedju Trenutnog i Željenog stanja.

Oni omogućavaju da korištenjem serijalizacijskih jezika poput YAML pišemo predloške za konfigurisanje, koji od nule već gotov server iskonfigurišu kako bi manuelno svaki inžinjer iskonfigurisao. Primjer ovakvih skripti

```
---
- hosts: group2
  tasks:
  - name: sshd config file modify port
    lineinfile:
      path: /etc/ssh/sshd_config regexp:
      'Port 28675'
      line: '#Port 22' notify:
        - restart sshd
handlers
    - name: restart sshd service:
      sshd
        name:  sshd
        state: restarted
```
[15]

[14] The DevOps toolkit – Darin Pope

[15] https://www.guru99.com/ansible-tutorial.html

Pojava ovih alata za upravljanje konfiguracijama su bile revolucionarna promjena u IT svijetu jer su smanjile kompleksnost rada i eksponencijalno povećale skalabilnost. Ovakvi snippeti koda se nazivaju „manifesti“. Ujedno služe kao dokumentacija jer koriste deskriptivan jezik koji inžinjeru govori na koji način je server trenutno konfigurisan, a dovoljno deklarativan da može automatizovati proces koji se do sada manuelno radio. Uvođenjem virtualizacije se smatralo da će integracija ovih alata sa konceptima virtualizacije biti idealna, ali nažalost ovi alati pate od jedne ogromne mane.

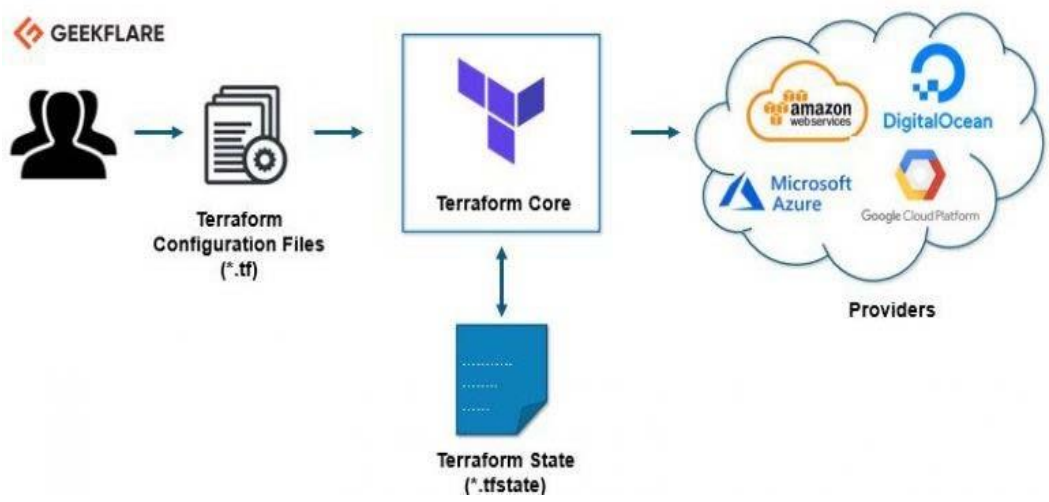
Ta mana se evidentira u činjenici da se sve instrukcije koje definišemo u skripti moraju izvršavati na živim mašinama u „runtime“ fazi. Proviziorniranje VM- ova je još uvijek ostao manuelan proces.

To dovodi to glavnog problema kada treba unaprijediti aplikaciju. Jer još uvijek moramo riješiti problem manuelnog provizioniranja, postavljanja operativnog sistema, održavanja verzija koje on posjeduje. Ovo se riješilo na način da se briše čitava infrastruktura sa starom verzijom softvera i mijenja se postepeno sa infrastrukturom koja ima novu verziju softvera. Ovo se u literaturi naziva **problem imutabilnosti.**

Druga Generacija konfiguracijskih alata

Za potpunu upotrebu IaC potreban nam je softver koji je u stanju da rješava probleme uniformnosti i imutabilnosti, da oduzme sav repetitivan i redundantan posao inžinjeru ne treba da svoje vrijeme troši na repetitivne probleme. Potreban nam je softver koji je u stanju da riješi problem Imutabilnosti koji su imali alati za upravljanje infrastrukturama prve generacije. Trenutno jedini alat na tržištu univerzalno napravljen koji zadovoljava sve ove zahtjeve je **Terraform.**

Slika 11. Prikaz translacije Trenutnog u Željeno stanje koristeći terraform[16]



Na ovom dijagramu se jasno vidi proces i uloga koju ima alat poput terraforma u upravljanju konfiguracijom tj. Translacijom Trenutnog stanja u Željeno stanje naše infrastrukture.

Iako postoje specijalizirana rješenja za svaki od poznatih cloud provajdera nisu ni blizu u tolikoj mjeri rasprostranjeni kao ovaj alat.

[16] https://www.devopsschool.com/blog/terraform-architecture-and-components-through-diagram/

YAML

Pošto pisanje deklarativnih skripti i manifesta koji trebaju da automatizuju postavljanje različitih konfiguracija na raznim okruženjima, jedna od stvari sa kojom se moramo upoznat a dijelom smo je dotakli u prošlom poglavlju je rad sa serijalizacijskim jezicima.

Primjer takvog jezika je XML i JSON koji se intenzivno koriste u razvoju aplikacija baziranih na webu i uopšteno u developerskom svijetu. Serijalizacijski jezik koji se najčešće koristi u Oblasti IT operacija pa indirektno u oblasti DevOpsa je YAML (engl. YAML A'int Markup Language).

Serijalizacijski jezici imaju specijalni karakter iz razloga što predstavljaju način na koji možemo da komuniciramo i razmjenjujemo podatke na uniforman način. Na osnovu njih možemo da razmjenjujemo podatke između aplikacija koje su napisane u potpuno različitim tehnologijama.

YAML je jezik koristi indentacije (poput jezika kao što je python). Predstavlja najrazumniji od tri serijalizacijska jezika sa najlakšom sintaksom i zato je glavni izbor prilikom pisanja skripti za automatizaciju različitih konfiguracija.

YAML se intenzivno koristi najviše u pisanju docker-compose skripti, definisanju K8 klastera, pravljenju infrastruktura prve generacije (Ansible). Ekstenzija koja ga definiše jeste .yaml ili .yml.

Snippeti

Slika 12. Prikaz Sintakse kroz docker-compose YAML skriptu

Iz slike 12. Možemo izdvojiti par osnovnih dedukcija o sintaksi pisanja ovog jezika. Najvažnije su:

- Indentacija predstavlja opseg (engl. Scope),
- CAPS sensitive,
- Jednostruki ili dvostruki navodnici ili bez navodnika u pisanju stringova,
- Varijable se označavaju prefiksom &,
- Liste se označavaju sa crticom –,
- Nema zareza u nabrajanjima.

Usporedba sa JSON i XML

```
{
    "version": "2.3",
    "x-common": {
        "database": {
            "MYSQL_PASSWORD": "secret",
            "MYSQL_ROOT_PASSWORD": "secret"
        },
        "panel": {
            "APP_URL": "https://games.adinjahic.com",
            "APP_TIMEZONE": "UTC",
            "APP_SERVICE_AUTHOR": "adin.jahic@gmail.com",
            "TRUSTED_PROXIES": "*"
        },
        "mail": {
            "MAIL_FROM": "adin.jahic@gmail.com",
            "MAIL_DRIVER": "smtp",
            "MAIL_HOST": "mail",
            "MAIL_PORT": "1025",
            "MAIL_USERNAME": "secret",
            "MAIL_PASSWORD": "secret",
            "MAIL_ENCRYPTION": "true"
        }
    }
}
```

Slika 13. JSON verzija prethodne skripte

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <version>2.3</version>
    <x-common>
        <database>
            <MYSQL_PASSWORD>secret</MYSQL_PASSWORD>
            <MYSQL_ROOT_PASSWORD>secret</MYSQL_ROOT_PASSWORD>
        </database>
        <panel>
            <APP_URL>https://games.adinjahic.com</APP_URL>
            <APP_TIMEZONE>UTC</APP_TIMEZONE>
            <APP_SERVICE_AUTHOR>adin.jahic@gmail.com</APP_SERVICE_AUTHOR>
            <TRUSTED_PROXIES>*</TRUSTED_PROXIES>
        </panel>
        <mail>
            <MAIL_FROM>adin.jahic@gmail.com</MAIL_FROM>
            <MAIL_DRIVER>smtp</MAIL_DRIVER>
            <MAIL_HOST>mail</MAIL_HOST>
            <MAIL_PORT>1025</MAIL_PORT>
            <MAIL_USERNAME>secret</MAIL_USERNAME>
            <MAIL_PASSWORD>secret</MAIL_PASSWORD>
            <MAIL_ENCRYPTION>true</MAIL_ENCRYPTION>
        </mail>
    </x-common>
</root>
```

Slika 14. XAML verzija prethodne skripte

Vidimo da su ostali serijalizacijski jezici prilikom implementacija puno kompleksniji, zahtjevaju puno veću preciznost u pisanju u obliku tagova u slučaju XAML-a i u obliku vitičastih zagrada koje primjećujemo u JSON serijalizacijskom jeziku.

Podjela i primjena IaC alata

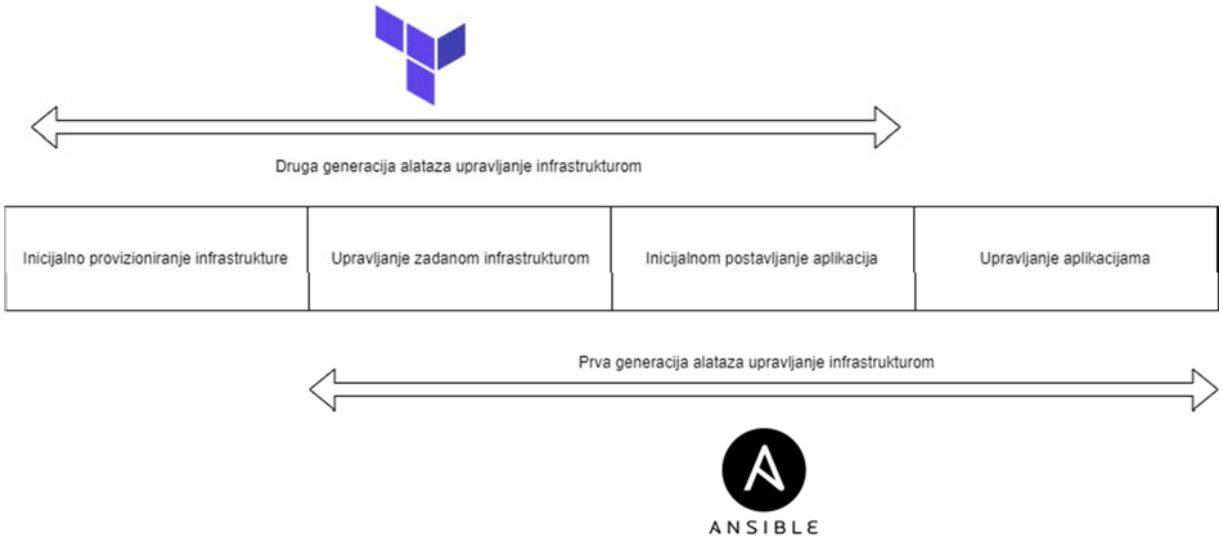
Da bi bolje izvršili podjelu alata za upotrebu infrastrukture, definisat ćemo četiri glavna zahtjeva koja su potrebna u upravljanju infrastrukturama:

- Inicijalno provizioniranje infrastrukture,
- Upravljanje zadanom infrastrukturom,
- Inicijalnom postavljanje aplikacija,
- Upravljanje postavljenim aplikacijama.

Ovo su osnovi zahtjevi koje alati za upravljanje infrastrukturom trebaju da izvrše.

Naime trenutno ni jedan alat na tržištu ne ispunjava sva četiri uslova koja smo naveli. Stoga inžinjeri u praksi koriste kombinaciju više alata da bi vršili upravljanje infrastrukturom. U ovom slučaju glavni razlog zašto su potrebna 2 alata je u tome što ne postoji alat koji može istovremeno da provizionira infrastrukturu i upravlja aplikacijama u toj strukturi. Što je i logično ako bolje pogledamo jer taj slučaj to bi značilo da postoji univerzalni tip softvera koji je kompatibilan sa svakom vrstom API-a od Cloud provajdera i svakom vrstom API-a od programa što je ogroman poduhvat. Trenutno postoji slično rješenje koje objedinjuje oba ova koncepta, ali to izlazi iz opsega IaC, a ulazi u oblast pravljenja IDP-a o kojima će biti govora u narednim poglavljima.

Slika 15. Prikaz podjele jezika za upravljanje infrastrukturom



Cloud specific

Navedeni alati nisu jedini ove kategorije za upotrebu. Naime svaki provajder da bi unaprijedio svoje poslovanje pokušava da napravi svoje alate koji najbolje rade uz njegov softver. To ne znači da drugi alati ne rade dobro uz taj softver, nego jednostavno daju malu prednost kupcima koji odluče da svoju lojalnost produže koristeći neke njihove specifične alate.

Takvi alati se nazivaju cloud specific alati i oni predstavljaju tip softvera koji vrši transformaciju Željenog u Trenutno stanje gdje su Trenutno stanje i alat za tu transformaciju od iste kompanije. Primjer toga je AWS, koji ima svoj IaC alat koji radi najbolje za njihove resurse, a naziva se CloudFormation alat. Google Cloud provider ima svoj Google Cloud Deployment Manager, dok Microsoftova verzija je Azure resource manager. Ovi alati su striktno namijenjeni za tip infrastrukture koji upravljaju.

Ovo je dobar izbor za kompanije koje su bazirane na jednom provideru i koji rade takav posao da trebaju da izvuku svaki atom snage iz zadanog sistema po najvećoj brzini sistema koju posjeduje.

Cloud agnostic

IaC alati poput Terraforma ili Crossplane-a su Cloud agnostic. Ovo znači da su kompatibilni sa svim provajderima koji postoje. Jedan te isti kod se može pisati za sve provajdere. Ovakav koncept povećava kompleksnost ali nam daje puno veću moć u kontroli resursa.

Ovaj izbor je trenutno u vodstvu zato što sve više kompanije uzima resurse od različitih provajdera i samim time im treba fleksibilan alat kojim mogu da izvršavaju svoje poslovne zahtjeve.

Glavni princip rada ovog tipa alata je da postoji „jezgro“ koje naše zahtjeve koje smo definisali kroz željeno stanje prevode u trenutno na način da vrši transformaciju našeg manifesta u tom alatu i na osnovu njega izvrši niz API-a poziva datom provajderu koji to provizionira na način koji je zapisano u manifestu.

Terraform kroz implementaciju jednostavne infrastrukture

Kada su u pitanju resursi na cloud-u znamo da postoje tri načina da vršimo provizioniranje tih resursa.

- Panelom (GUI) koji dobijemo od svakog provajdera,
- Konzolom (LUI) gdje preko komandi specifičnih za provajdera provizioniramo resurse,
- IaC – u kojem pišemo manifeste koji nevezano za provajdera vrše provizioniranje našim resursa tj. vrše translaciju između Željenog stanja u Trenutno stanje.

Da bi razumjeli kako tačno ta translacija se dešava moramo uvesti par pomoćnih koncepata, ali prvo da se složimo za potrebne zahtjeve koje mora da operativni sistem ima prilikom korištenja ovakvih tehnologija.

Potrebno je prvo da se na OS-u sa kojeg će se pisati IaC bude instaliran IaC alat (u ovom slučaju terrafrom). Instalacija je trivijalna na svakom od poznatih OS-a.

- MacOS – preko package managera poput Homebrew,
- Linux - instaliranjem dependencia,
- Windows – Manuelna instalacija GUI-om.

Nakon što je terraform instaliran na OS-u potrebno je izvršiti autentifikaciju sa željenim cloud provajderom kako bi terraform dobio pristup da u naše ime provizionira resurse.

Ovaj proces se vrši dodavanjem security policy-a gdje će kao rezultat generisanja novog policy-a cloud provajder nam dati pristup javnom i privatnom ključu za autentifikaciju sa našim profilom.

Naš primjer će vezati terraform i AWS tako da 2 varijable koje će nam izgenerisati su

- AWS_ACCESS_KEY_ID,
- AWS_ SECRET_ACCESS_KEY.

Ove dvije varijable vezuju našu terraform instancu i cloud provajder. Drugim riječima dajemo na ovaj način pristup terraformu da u naše ime provizionira resurse na našem profilu.

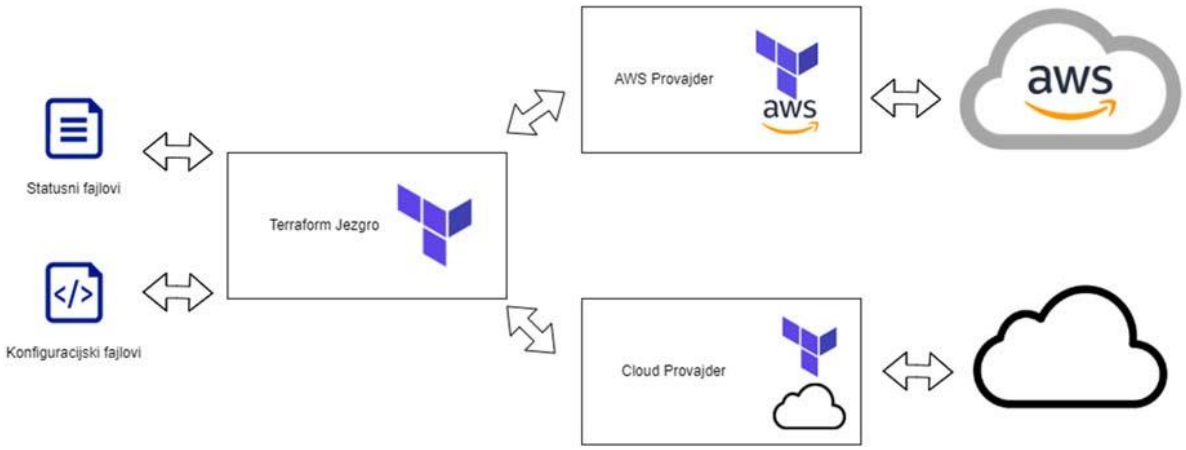
Arhitektura Terraforma

Iz Slike 11. vidimo opću sliku kako alati poput Terraforma funkcionišu, ali postoji par stvari koje moramo definisati prije nego se upustimo u primjer pravljenja infrastrukture. U ovom poglavlju ćemo detaljnije pogledati kako radi Terraform i zašto je napravljen na način kako jeste.

Već znamo da Željeno stanje definišemo takozvanim manifestima ili konfiguracijskim fajlovima. Oni su označeni u ovom slučaju sa ekstenzijom .tf. Pored toga što kroz manifeste definišemo Željeno stanje, kroz manifeste definišemo i Trenutno stanje.

Terraform radi upravljanje Trenutnim stanjem koristeći drugi tip konfiguracijskih fajlova koji se zovu statusni fajlovi. Ovi fajlovi služe za sinhronizaciju izmedju Željenog stanja i Trenutnog stanja na cloud-u. U njima su definisani svi trenutni resursi koje posjedujemo. Sa svakim pokretanjem skripte prvi korak koji se radi je usporedba konfiguracijskog fajla sa statusnim fajlom i onog momenta kada dođe do dijela kada nisu isti, Željeno stanje se prepiše u Trenutno stanje.

Sinhronizaciju željenog i Trenutnog stanja vrši terraform jezgro koje je napisano u hashicorp programskom jeziku kojem nažalost nemamo mi pristup. Sljedeća faza koja slijedi je pisanje upita na API-je. Pošto poenta korištenja ovih alata jeste da eliminiše manuelno provizioniranje resursa. Terraform mora da shvati kako statusni fajl koji trenutno imamo da pročita i formulira kompleksan upit na API tog zadanog cloud provajdera koji nam kao povratnu vrijednost daje resurse na našem panelu. Statusni fajlovi se isto tako u dokumentaciji referiraju kao „backendi“.



Slika 16. Detaljna Arhitektura Terraforma

Arhiviranje Statusnih fajlova

Statusni fajlovi su jako važni jer nam prikazuju trenutno stanje naših resursa na cloud

provajderima. Potrebno je uvijek držati statusni fajl sinhroniziran sa trenutnim stanjem da se ne desi da jedan inžinjer provizionira promjene na svom lokalnom statusnom fajlu koje će narušiti sinhronizaciju svih ostalih korisnika koji koriste te resurse, samim time njihove verzije statusnih fajlova će postati zastarjele i svaka naredna promjena koju ostali korisnici naprave kroz svoj statusni fajl će poremetiti strukturu resursa i projekata koji se nalaze na njima. Rješenje za ovu situaciju je da se ne sprema statusni fajl na lokalnim računarima svakog korisnika nego da se drži na nekom remote području kojem će pristup imati svi članovi kojima je potrebno da izvršavaju proviziorniranje. Ovakav sistem se naziva **Remote Backend** i koristeći ovaj sistem svaki korisnik koristi jedan statusni fajl za resurse koje treba da provizionira. I time smo postigli sinhronizaciju izmedju više remote korisnika koji koriste alate za upravljanje infrastrukturom. Napomena je da po defaultu terraform ako se ne naglasi drukčije statusni fajl snima na lokalno okruženje na kojem se nalazi instaliran terraform.

Da bi se spriječilo da više korisnika istovremeno pokušaju urediti fajl najbolje je koristiti storage sa sistemom zaključavanja koji sprečava formiranja „stanja trke“ primjer ovoga sistema je Amazon DynamoDB.

Napomena je da nije bitno na kojoj lokaciji se nalazi arhiva statusnih fajlova bitno je da su samo na odvojenoj lokaciji kojoj svi inžinjeri imaju pristup tokom rada na projektu.

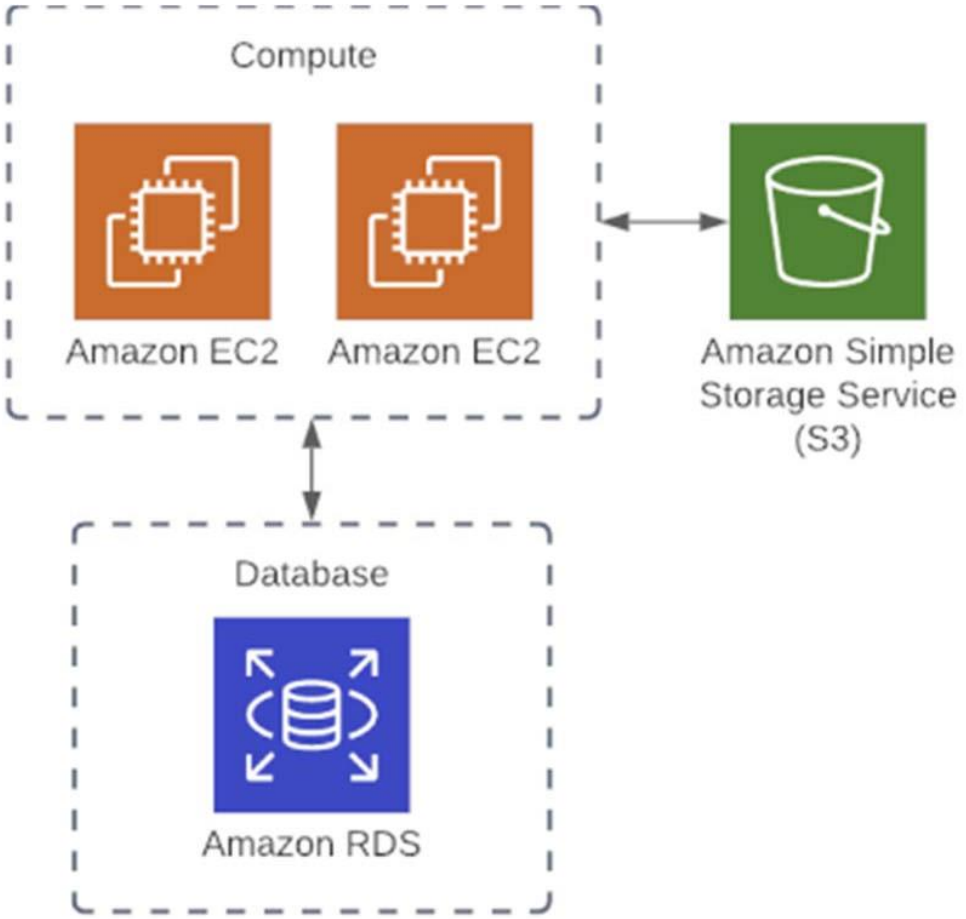
U slučaju ako se desi gubitak sinhronizacije između statusnih fajlova kojima se provizioniraju resursi. Rezultati mogu biti katastrofalni jer svaka nečije promjene mogu prepisati se preko promjena koje je drugi inžinjer napravio i napraviti ogromnu pometnju koju je jako teško ispraviti.

Implementacija infrastrukture

Kao primjer za demonstraciju kako se vrši provizioniranje resursa koristeći alate za upravljanje infrastrukturom. Napravit ćemo konfiguracijsku skriptu u terrafromu koja koja će provizionirati 2 Virtuelne mašine sa amazonovog EC2 servisa, 1 storage spremnik sa S3 servisa, 1 bazu podataka na amazonovom RDS servisu.

Ovo je neka osnovna arhitektura prilikom dizajniranja aplikacija za jedan ili dva stepena kompleksnija od normalne ali još u relativno jednostavnim kategorijama.

Dijagram koji ćemo slijediti izgleda ovako:



Slika 17. Dijagram za implementaciju[17]

[17] https://devopsdirective.com/

Skripta će koristiti remote backend sistem koji se nalazi na S3 bucketu. Napomena je da kod iz ovog primjera se neće moći kompajlirati jer će resursi biti nepostojeći ali analognim načinom rada i prilagođavanju svojih resursa će dati rezultat koji je tražen. Poenta primjera je u demonstraciji koncepta.

Prvi korak je definisati remote backend i registrovati cloud provajder koji je u ovom slučaju AWS. To postižemo osnovnim *terraform* i *provider* blokom koji definišemo na sljedeći način.

```
terraform {
  backend "s3" {
    bucket         = "Terrafrom-primjer"
    key            = "Terraform-primjer.tfstate"
    region         = "us-east-1"
    dynamodb_table = "terraform-state-locking"
    encrypt        = true
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}
```

Slika 18. Inicijalna postavka terraforma

Važno je naglasiti da je iskorištena dynamoDB tabela koja ima sistem zaključavanja koji sprječava stanje trke.

Provizioniranje bilo kakvih resursa se izvršava na način da se koristi *resource* koji zahtjeva (kao na GUI načinu) da se osnovni parametri za provizioniranje popune kao tip AMI-a, tipa instance, koja sigurnosna grupa pripada.

Ovdje je na kraju napisana obična bash skripta koja ispisuje na portu 8080 rečenicu :“ Dobrodošli na VM“.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "VM1" {
  ami             = "ami-011899242bb902164" # Ubuntu 20.04 LTS // us-east-1
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instances.name]
  user_data       = <<-EOF
              #!/bin/bash
              echo "Dobrodosli na VM1" > index.html
              python3 -m http.server 8080 &
              EOF
}

resource "aws_instance" "VM2" {
  ami             = "ami-011899242bb902164" # Ubuntu 20.04 LTS // us-east-1
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instances.name]
  user_data       = <<-EOF
              #!/bin/bash
              echo "Dobrodosli na VM2" > index.html
              python3 -m http.server 8080 &
              EOF
}
```

Slika 19. Provizioniranje virtuelnih mašina na EC2 servisu koristeći terrafrom

Konfiguracija RDS servisa je veoma jednostavna jer zahtjeva popunjavanje osnovnih parametara koji čine connection string (tip, engine, verzija,username,password,naziv baze)

Pripadajući blok analogno izgleda ovako:

```
resource "aws_db_instance" "Instanca RDS-a" {
  allocated_storage   = 20
  storage_type        = "standard"
  engine              = "postgres"
  engine_version      = "12.5"
  instance_class      = "db.t2.micro"
  name                = "Primjer Baze"
  username            = "secret"
  password            = "secret"
  skip_final_snapshot = true
}
```

Slika 20. Provizioniranje baze podataka na RDS servisu koristeći terraform

Za kraj ostaje klasično formiranje S3 bucketa koji predstavlja storage sistem koji koristimo u našoj arhitekturi.

```
resource "aws_s3_bucket" "S3 primjer kroz terraform" {
  bucket        = "primjer-S3-kroz-terrafrom"
  force_destroy = true
  versioning {
    enabled = true
  }

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
```

Slika 21. Provizioniranje S3 bucketa kroz terraform

Primjetit ćemo da su pored osnovnih parametara dodani i sigurnosni parametri poput enkripcije na serverskoj strani koja nam daje sigurnost naših podataka u slučaju neovlaštenog pristupa.

Za sve eventualne primjedbe, komentare, sugestije obratiti se na mail