Samuli Ryhänen

Katsaus Infrastructure as Code työkaluihin

Tietotekniikan kandidaatintutkielma

13. tammikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Samuli Ryhänen

Yhteystiedot: smryhane@student.jyu.fi

Ohjaaja: Jonne Itkonen

Työn nimi: Katsaus Infrastructure as Code työkaluihin

Title in English: A Short review of Infrastructure as Code Tool

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 32+0

Tiivistelmä: Tämän kandidaattitutkielman tarkoituksena on selventää pilvipalveluiden yhteydessä käytettävien kehitystyökalujen myötä syntynyttä *Infrastructure as Code*-käytäntöä, esitellä aihealueen tutkimuksia sekä vertailla sitä varten kehiteltyjä työkaluja ja niiden ominaisuuksia. Työkalujen vertailu on rajoitettu, niin että siinä on painettu työkalun kykyä hyödyntää kestäviä, luotettavia, tietoturvallisia sekä muita alan hyviä käytänteitä.

Avainsanat: IAC, Iaas, Azure, AWS, Terrafrom, Pulumi, Openstack Heat, Ansible, Chef, Puppet

Abstract: The purpose of this bachelors degree is to help define the practice called infrastructure as code. This paper will introduce academic journals based on the subject and also perform a comparison of tools and their properties specifically designed to perform cloud computing. The comparison is biased on the tools ability to perform robust, trustworthy, secure operations. Second bias is based on how well the tool can follow the guidelines of good practices.

Keywords: IAC, Iaas, Azure, AWS, Terrafrom, Pulumi, Openstack Heat, Ansible, Chef, Puppet

Termiluettelo

DevOps	Toimintamalli, jossa yhdistetään ohjelmistokehitys sekä sen julkaisu palvelin-ympäristössä
CD	*Continuous Delivery* - Toimintamalli, jossa kaikkien ohjelmistokehittäjien tuottamat muutokset yhdistetään samaan yksittäiseen koodihaaraan
CI	*Continuous Integration* - Toimintamalli, jossa koodia tuotetaan tuotantoon valmiiksi lyhyissä osuuksissa
CI/CD	Toimintamalli, joka yhdistää CI:n sekä CD:n
IaC	*Infrastructure as code* - Palvelin infrastruktuuri koodina
API	*Application programming interface* - Sovellusrajapinta, jonka avulla kaksi ohjelmaa voivat kommunikoida keskenään
Tila	Tässä kontekstissa tilalla tarkoitetaan virtualisoidun ohjelmiston yhtä konfiguroitua versiota
Agentti	Tässä tekstissä sovellus tai entiteetti, joka vuorovaikuttaa virtualisoidussa ympäristössä sille koostetun ohjelmakoodin avulla
Idempotenssi	Operaatio on idempotentti, mikäli se tuottaa saman tuloksen riippumatta sen suorituskertojen lukumäärästä.
Malli	IaC -työkalujen tuotedokumentaatiossa puhutaan malleista (eng. *template*). Tässä tekstissä malleilla tarkoitetaan ensisijaista IaC -lähdekooditiedostoa.
Koodihaju	Lähdekooditiedostosta havaittu ominaisuus, joka on mahdollisesti merkki jostain lähdekooditiedoston ongelmasta.

Kuviot

Taulukot

Sisällys

1 Johdanto

Pilvipalveluiden rooli on yleistynyt huomattavasti IT-yrityksissä viimeisen vuosikymmenen aikana. Termi *infrastruktuuri koodina* (eng. *Infrastructure as Code*, lyh. IaC), tarkoittaa pilvipalvelimien hallinnointia lähdekooditiedostojen avulla. Näiden tiedostojen luomiseen sekä muokkaamiseen on kehitetty erillisiä IaC -kehitystyökaluja. Samalla kun pilvipalvelimia on alettu hyödyntämään entistä enemmän, on myös niihin erikoistettujen kehitystyökalujen kysyntä kasvanut merkittävästi ohjelmistokehittäjien keskuudessa (Kuvio 1).

IaC -kehitystyökalujen idea on automatisoida muutosten tekeminen palvelimelle virtualisoituun sovellukseen. Näiden työkalujen on ennen kaikkea tärkeä tukea valideja, tietoturvallisia sekä kehittäjäystävällisiä ratkaisuja, niin itse lähdekoodissa, kuin myös työkalun käyttämiseen liittyvissä toiminnoissa. Tutkielmassa käydään läpi historiaa ja taustaa näiden työkalujen synnylle, esitellään aiheeseen liittyviä tutkimuksia sekä lopuksi vertaillaan tunnetuimpia IaC -kehitystyökaluja tutkimusten, alan tarpeiden sekä työkalujen ominaisuuksien avulla.

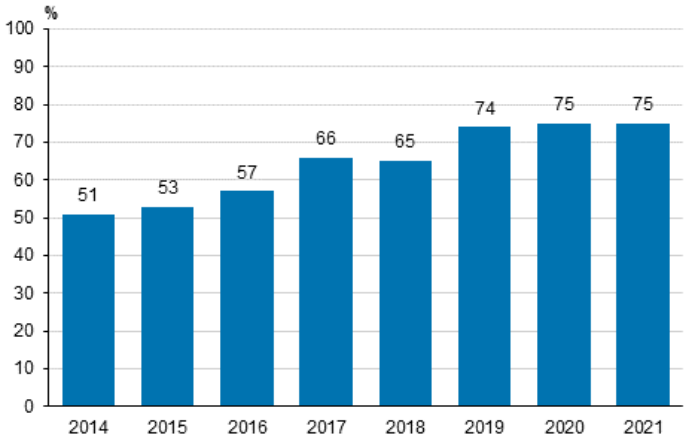
Kappaleessa 2 käydään läpi, että miten IaC:n rooli on kehittynyt alalla viime vuosien aikana. Kappaleessa 3 keskitytään syvällisemmin siihen, että millaisessa roolissa IaC -kehitystyökalut nykypäivänä ovat sovelluskehittäjien keskuudessa. Tähän liittyen kappaleessa käsitellään myös joitain työkaluille suotuisia ominaisuuksia. Kappaleessa 4 käydään läpi työkaluihin liittyviä tutkimuksia. Kappaleen 5 sisällä käydään läpi sitä että millaisia metaominaisuuksia IaC -kehitystyökaluilla on, sekä sitä että millaisia vaikutuksia näillä ominaisuuksilla voi olla työkalun käyttäytymiseen.

Kirja (Brikman 2022) on vaikuttanut suuresti tekstin inspiraation lähteenä. Etenkin kappaleet 5.1, 5.2, 5.4, 5.3 sekä 5.5 ovat saaneet vaikutteita edellä mainitusta kirjasta.

Tutkielmaan valikoitujen työkalujen ominaisuuksien esittely tapahtuu kappaleiden 5.6 ja 5.7 alakappaleissa. Kappale 5.8 on sisällytetty tekstiin antamaan aiheeseen näkökulmaa vaihtoehtoisesta paradigmasta. Kappaleessa 6 vertailu koostetaan vertailun pääkohdat yhteen kokonaisuuteen. Lopuksi 7 kerrotaan lyhyesti vielä yhteenveto.

2 Tausta ja motivaatio tutkimukseen

Viime vuosina pilvipalveluiden suosio on kasvanut tasaisesti suomalaisten yritysten keskuudessa. Kaikkein yleisimmin niitä hyödynnetään informaation ja viestinnän sekä ammatillisen tieteen ja teknisen toiminnan toimialoilla. Pilvipalveluiden suosion kasvua on kuvattuna kuviossa 1.



Kuvio 1: Pilvipalveluiden käyttö suomalaisissa yrityksissä 2014-2021, (*Suomen virallinen tilasto (SVT): Tietotekniikan käyttö yrityksissä*)

Palvelimen ylläpitäjän näkökulmasta muutosta selittävät seuraavat asiat: Palvelimen ylläpitäjälle koituvat kustannukset fyysisien palvelimien hankinnoista, henkilöstöstä sekä huolto- ja ylläpitotoimista. Kustannuksia koituu myös esimerkiksi sähkön kulutuksesta, sekä fyysisiin tiloihin liittyvistä kustannuksista.

Säädänteistä johtuen fyysisien palvelimien ylläpitäjien vastuulla on huolehtia, että palvelinten turvallisuudesta pidetään huolta määräysten mukaisesti. Tämä tarkoittaa kyberturvallisuuden lisäksi myös muita turvatoimia, kuten fyysisten tilojen turvallisuudesta huolehtimista. Palveluntarjoajan täytyy siis myös varmistaa esimerkiksi, että tiedon säilyttämiseen ja turvallisuuteen liittyviä lakeja noudatetaan. Toisaalta myös palveluntarjojan vastuulla voi olla myös ongelma- sekä katastrofitilanteista toipumisen varmistaminen.

Näistä kuluista sekä velvollisuuksista johtuen, ohjelmistoyritysten onkin usein kannattavampaa ulkoistaa palvelinten ylläpito ulkoiselle toimijalle, kuin ylläpitää itse fyyisisiä palvelimia. Ulkoinen toimija tässä kontekstissa tarkoittaakin pilvialustan tarjoajaa, kuten esimerkiksi *Microsoft Azure*. Ylimääräisistä kustannuksista vapautuminen antaa ohjelmistoyrityksille entistä paremman mahdollisuuden keskittyä juuri kullekin toimijalle olennaiseen liike-

toimintaan.

Tutkimuksessa (Morgan ja Conboy 2013) nostetaan esille pilvipalveluista saatavia moniulotteisia hyötyjä. Taloudellisten etujen lisäksi pilvipalveluiden hyödyntäminen antaa sovelluskehittäjille mahdollisuuden tehdä pilottikokeiluja sekä tutkimuksia riskittömässä palvelin -ympäristössä. Mahdollisuus julkaista tuotantoon sovellus periaatteessa milloin tahansa sovelluksen kehityksen vaihetta virtaviivaistaa sovelluspalveluita tarjoavien yritysten yritystoimintaa. Pilvipalveluiden hyödyntämisen vaikutukset heijastuvat myös positiivisesti ohjelmistokehitystiimien sisäiseen yhteistoimintaan. Tämä johtuu siitä että pilvipalveluiden käyttöönoton jälkeen on huomattu enemmän tiimien välistä yhteistyötä sekä parempaa keskinäistä sitoutuneisuutta.

Infrastruktuuri koodina on etenkin DevOps -periaatteita noudattaville kehitystiimeille hyödyllinen käytäntö. DevOps:n tarkoituksena on lyhentää ohjelmiston julkaisemisen elinkaarta sekä tarjota jatkuvasti uusia sekä entistä laadukkaampia ohjelmistoversioita. DevOps -käytänteen yksi perustukipilareista on prosessien automatisointi. Infrastruktuuri koodina tai IaC eli infrastruktuurin määritelly ohjelmakoodin avulla onkin luontainen jatkumo sille, että kuinka *DevOps*, pilvipalvelut sekä automatisointi yhdistetään toisiinsa.

3 Mitä on infrastruktuuri koodina?

Infrastruktuurilla tässä tekstissä tarkoitetaan pilvipalvelimelle virtualisoitua kokonaisuutta, joka kapseloi sovelluksen toiminnan logiikan, sekä sen käyttämän datan. Infrastruktuurin tilaa pitää muuttaa ensimmäisen kerran sovelluksen julkaisuvaiheessa. Joka kerta kun sovellukseen tehdään korjauksia tai uusia toimintoja, jotka myös sisältävät muutoksia infrastruktuuriin, täytyy tällöin muokata infrastruktuurin tilaa uudelleen.

Infrastruktuuriin tehtävien muutosten asettaminen tapahtuu virtuaalisoituun ympäristöön kohdennettujen muokkauskomentojen avulla. Muokkauskomentoja voi olla muutoksen luonteesta riippuen useita, esimerkiksi yksi komento luo virtuaalikoneen jonka jälkeen toisella komennolla määritellään sille sopivat asetukset. Tämän takia infrastruktuurin muutoksen vaatima kokonaisuus tallennetaan tiedoston, tai useiden tiedostojen muodostamaan kokonaisuuteen, josta käytetään nimitystä IaC (eng. *Infrastructure as Code*). IaC -käsite siis tarkoittaa infrastruktuurin muutosten tuottamisen automatisointia.

IaC -käytännön yleistyttyä ohjelmiston julkaisutahti on kasvanut monessa yrityksessä. Kaikki ei kuitenkaan ole täysin kitkatonta tämänkään teknologian osalta. Silloin kun IaC -käytäntö yhdistetään osaksi muita ohjelmistokehitysprosesseja, huomataankin että jossakin vaiheessa tuotantoketjua automaatio katkeaa. Tämän korjaamiseksi joudutaan turvautumaan manuaalisiin käytäntöihin, tai sisällyttämään IaC:n avuksi ulkoisia työkaluja tai prosesseja.

3.1 Staattinen ja dynaaminen IaC

IaC -työkalujen kirjo rajoittuu toistaiseksi *staattisiin* työkaluihin. *Staattisella* toiminnallisuudella tarkoitetaan sitä, että jokainen infrastruktuuriin toteutettava muutos toimeenpannaan IaC -ohjelman yksittäisellä manuaalisella ajolla.

Kuvitellaan tilanne, jossa infrastruktuuri käyttää ulkopuolelta haettua resurssia staattisesti, jolloin siis kyseinen resurssi pysyy muuttamattomana. Jos kyseisestä ulkopuolisesta resurssista julkaistaan uusi versio, jää infrastruktuuriin silti käyttöön resurssin vanhempi versio. Jotta infrastruktuuri saisi uudemman version resurssista, itse infrastruktuuri pitäisi tuolloin

julkaista uudelleen. Tämän takia *staattiseen* IaC:n yhdistetään lisätyökaluja sekä prosessikäytänteitä, kuten CI/CD -kanava, joka uudelleen käynnistää IaC -ohjelman säännöllisesti.

Tutkimuksessa (Sokolowski 2022) 138 eri IT-alan ammattilaista oli sitä mieltä, että sovellusten välillä ilmenee riippuvuussuhteita, jotka monesti rajoittavat ohjelmistojen päivittämisen keskinäistä järjestystä. Tutkimuksessa argumentoidaan, että *dynaaminen* IaC voisi virtaviivaistaa kehitystiimien työskentelyä. Paperissa esitelty Pulumin 5.6.3 päälle rakennettu viitekehys antaa kyseiselle työkalulle dynaamisen käytöksen.

Toisaalta myös IaC:n erottaminen CI/CD -prosessista selkeyttäisi CI/CD -prosessin hallintaa sekä konfigurointia. Ihmislähtöisestä näkökulmasta ajatellen, jatkuva riippuvuussuhteiden huomiointi sekä tarvittavien muutosten korjaus voi olla ohjelmistokehittäjälle erittäin työlästä.

3.2 Modulaarisuus sekä uudelleenkäytettävyys

Tutkimuksen (Dörbecker, Böhm ja Böhmann 2015) mukaan sovelluksen toteuttaminen modulaarisia periaatteita noudattaen antaa kyvykkyyden vastata tehokkaasti markkinoiden heterogeeniseen kysyntään. Modulaarisuuden tärkeyden rooli korostuu IaC -komponentteja suunnitellessa.

Infrastruktuurin muutokset ovat helpompia toteuttaa, mitä isompia kokonaisuuksia voidaan vaihtaa kerralla. Toisaalta myös infrastruktuurien rakentaminen on nopeampaa ja helpompaa mikäli apuna voidaan hyödyntää valmiita modulaarisia komponentteja. Taulukossa 1 on esillä komponentin modulaarisuutta määrittäviä tekijöitä.

Vaikutus	Määritelmä
Yhdisteltävyys	Jonkin tuoteversion toteuttamisessa tarvittava teoreettinen komponenttien minimimäärän suhde komponenttien todelliseen lukumäärän
Yleisyys	Komponentin saman ohjelmaversion käyttäminen useissa eri tuotteissa
Räätälöitävyys	Yrityksen asiakkaalle tarjoaminen ominaisuuksien lukumäärän suhde toimintojen ja ominaisuuksien enimmäismäärään, joista todellisuudessa voidaan valita
Uudelleenkonfigurointi	Mikäli kahden eri tuotteen funktio on identtinen, pitäisi niistä pystyä muokkaamaan sama tuote suhteessa yhtä pienillä muutoksilla
Uudelleensuunnittelu	Riippuvuussuhteiden rooli pitäisi olla mahdollisimman pieni, jotta uudelleensuunnitelulla olisi minimaaliset vaikutukset tuotteen muihin komponentteihin
Standardisointi	Eri tuotemalleilla voi olla funktioltaan identtisiä toimintoja. Nämä toiminnot voidaan toteuttaa samanlaisilla tai jopa identtisillä rakenteilla eli moduuleilla. Näitä moduuleja voidaan standardoida ja valmistaa suurempina eräkokoina

Taulukko 1: Modulaarisuuden tunnuspiirteitä; Mukailtu tutkimuksesta (Dörbecker, Böhm ja Böhmann 2015)

4 Katsaus alalla tehtyihin tutkimuksiin

Vuonna 2019 tehdyssä systemaattisessa mittauksessa alan tieteellisistä julkaisusta (Rahman, Mahdavi-Hezaveh ja Williams 2019) kerrottiin että IaC -aihepiiriä käsittelevien tutkimusten polttopisteet ovat keskittyneet työkalujen ja erilaisten viitekehysten kehittämiseen, työkalun käyttöönottamiseen, empiirisiin tutkimuksiin sekä työkalujen testaamiseen. Tutkimuksessa ehdotetaan, että jatkossa tutkimusta kohdennettaisiin etenkin epäjohdonmukaisten kaavojen havaitsemiseen, häiriötekijöiden analyysiin, turvallisuuteen, koulutukseen sekä hyvien käytänteiden kehittämiseen. Seuraavissa alakappaleissa käydään tarkemmin läpi löydettyjä julkaisuja.

4.1 Vikatilat ja tietoturva

IaC -työkalujen alkuaikoina eräs huolenaihe oli se, että työkalut eivät aina tuottaneet idempotenttia lopputulosta. Esimerkiksi vuonna 2013 Hummer (Hummer ym. 2013) paperissaan esittää, että *Chef* -työkalulla 5.7.1 tuotetuista järjestelmistä noin kolmasosa olivat jonkin muun kuin idempotentin funktion aikaansaannos. Näistä ajoista on tultu kehityksessä eteenpäin ja *IaC* -työkalujen funktionaalisuus on lähes aina idempotenttia. Nykyään häiriöiden lähde onkin tutkimusten kautta havaittu olevan lähtöisin työkalun käyttäjästä.

Vikatila (eng. *defect*) eli *jonkun prosessin tarkoitukseton toiminta* lähdekoodissa voi aiheuttaa hyvinkin mittavia tuhoja yritykselle. Virheellinen IaC -tiedosto saattaa vakavassa tapauksessa aiheuttaa koko palvelimen kaatumisen.

Mikäli IaC sisältää useita manuaalisesti määriteltäviä riippuvuussuhteita, tai sen toteutus on jollain muulla tavalla huono, niin katastrofi- ja vikatilanteissa ongelman lähdettä ole aina helppo paikantaa. Huono toteutus voi pitää sisällään esimerkiksi paljon vaikealukuisia tiedostoja.

Epätarkoituksenmukaisen toiminnallisuus voi osaltaan aiheuttaa lisäksi tietoturvariskin. Tutkimuksessa (Rahman ym. 2020) analysoitu että *defektin* aiheuttaja voi olla esimerkiksi väärin kirjoitettu TCP/DHCP -portti, tai vaikkapa väärin kirjoitettu tietokannan käyttäjätunnus

tai salasana. Tutkimuksen (Rahman ja Williams 2019) mukaan vikatilan tuottavien komentokäskytiedostojen kanssa korreloivat kovakoodatut merkkijonot.

Tutkimuksessa (Rahman, Farhana ja Williams 2020) on selvitetty kelvottomien tiedostojen sekä koodihajujen syntyä. Niitä aiheuttavat seuraavanlaiset tilanteet:

- Mikäli tiedostoa ei ole katselmoitu tarpeeksi pätevän henkilön toimesta.
- Liian monen eri kehittäjän tekemät muokkaukset yhtä tiedostoa kohti.
- Kehittäjän muutokset tiedostoon, joka on vähintään 95-prosenttisesti muiden kehittäjien kirjoittama.
- Hajautettu sekä epäorganisoitu kehitystiimien välinen toiminta.
- Ensisijaisen tiedoston muokkaamisen ohella myös jonkin toisen tiedoston muokkaaminen.

Tutkimuksessa (Rahman, Parnin ja Williams 2019) analysoitiin avoimen lähdekoodin *IaC* -tiedostoja, joiden alkuperänä olivat *Github*, *Mozilla*, *Openstack* sekä *Wikipedia*. Samassa järjestyksessä luetellen tiedostoista 29,3%, 17,9%, 32,9% ja 26,7% sisälsi jonkun seuraavista tietoturvaan liittyvistä koodihajuista:

- Pääkäyttäjän oikeudet oletuksena.
- Tyhjä salasana.
- Kovakoodattu salaisuus.
- Virheellinen IP-osoite.
- Epäilyttävä kommentti tiedostossa.
- HTTP-protokollan soveltaminen ilman TLS-protokollaa.
- Heikon kryptograafisen algoritmin käyttö.

Tietoturvahajujen yleisyydestä kerrotaan julkaisussa (Rahman ym. 2021). Tutkimuksessa analysoitiin 50323 kappaletta avoimen lähdekoodin *Chef*5.7.1, *Puppet*5.7.2 ja *Ansible*5.7.3 tiedostoja. Näiden tiedostojen joukosta 46600 kappaleesta löytyi jonkinasteinen tietoturvahaju. Esimerkiksi kovakoodattuja salasanoja löytyi 7849 kappaletta.

IaC -tiedostojen luotettavuuden sekä turvallisuuden takaamiseksi monissa tutkimuksissa onkin ehdotettu sisällyttää osaksi ohjelmiston kehitysprosessia automaattisia koodin staattisesti

analysoivia työkaluja sekä automatisoituja testejä. Toisaalta näissä tutkimuksissa on myöskin painotettu vertaisarvioinnin tärkeyttä.

5 IaC -työkalujen vertailu

Nykyisissä IaC -työkaluissa on tiettyjä metatason ominaisuuksia sekä toimintoja, joiden avulla niiden keskinäisiä eroja voidaan vertailla yleiseltä tasolta.

Eloisa yhteisö (kappale 5.1) tuottaa kehittäjälle tärkeää informaatiota, jonka avulla työkalua voi käyttää tehokkaasti. Työkaluissa kirjoitettavien mallien (kappaleet 5.2 ja 5.3) ilmaisutyyli vaikuttaa suoraan siihen, että miten helppo työkalulla on konfiguroida infrastruktuuria. Näiden lisäksi sovelluksen käyttötapaus (5.4), tai muista työkaluista poikkeavat protokollat (kappale 5.5) voivat vaikuttaa työkalun valintaan.

5.1 Yhteisöjen rooli

Nimi	Yhteisössä aktiivisia jäseniä	Tähdet	Kirjastot	Maturiteetti	Osallisten lkm muutos	Tähtien lkm muutos	Kirjastojen lkm muutos
Chef	640	6910	3695	Korkea	+34%	+56%	+21%
Puppet	571	6581	6871	Korkea	+32%	+58%	+55%
Ansible	5328	53,479	31,329	Keskitaso	+258%	+183%	+289%
Pulumi	1402	12,723	15	Matala	-	-	-
Openstack Heat	395	379	0	Matala	+40%	+34%	0
Terraform	1621	33,019	9641	Keskitaso	+148%	+476%	+24003%

Taulukko 2: Kirjastot: Yhteisöjen valmiit mallit, Tähdet: Tähdet *Githubissa*. Muutosprosentit ovat laskettu aikaväliltä 2016-2022. Mukaelma kirjasta (Brikman 2022)

Osana työkalun käyttökokemusta kuuluu vuorovaikuttaminen sitä ympäröivässä yhteisössä. Yhteisöt tuottavat havaintoja työkalun toiminnallisuudesta. Ne toimivat kehittäjän omana tukiverkostona, jolta voi kysyä tarvittaessa apua ongelmatilanteissa. Ne myös vaikuttavat

merkittävästi työkalun kehityskaareen, sillä työkalun kehittäjät tekevät valintoja ja ratkaisuja osittain yhteisön antaman palautteen pohjalta. Yhteisön rooli on siis hyvä huomioida työkaluja vertailtaessa. Luonnollisesti eloisan ja aktiivisen yhteisön kanssa on helppo vuorovaikuttaa. Kuviossa 2 on esitettynä eri IaC -työkaluista koostettua dataa yhteisöjen näkökulmasta.

5.2 Mallit

Yksinkertaisimmillaan IaC -lähdekooditiedosto voi olla komentokäskytiedosto, johon on tallennettuna sarja kuoren komentokäskyjä. Näitä tiedostoja suorittamalla koneympäristössä saadaan aikaiseksi infrastruktuurin muutokset. Komentokäskytiedostot antavat kehittäjälle vapauden hyödyntää kaikkein tunnetuimpia ohjelmointikieliä, kuten esimerkiksi *Bash* tai *Python*.

Kuitenkin infrastruktuurin konfigurointiin liittyviä komentokäskyjä on niin paljon, pelkästään komentokäskyjen avulla infrastruktuurin konfigurointi ei onnistu. Monessa IaC -työkalussa onkin käytössä *malli*, johon kirjoitetaan joko *deklaratiivisesti* tai *proseduraalisesti* 5.3 haluttu konfiguraatio.

```
"parameters": {
  "<parameter-name>" : {
    "type" : "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter>",
    "allowedValues": [ "<array-of-allowed-values>" ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array-parameters>,
    "metadata": {
      "description": "<description-of-the parameter>"
    }
  }
}
```

Kuvio 2: Esimerkki JSON -mallista

```
parameters:
- name: myString
  type: string
  default: a string
- name: myMultiString
  type: string
  default: default
  values:
  - default
  - ubuntu
- name: myNumber
  type: number
  default: 2
  values:
  - 1
  - 2
```

Kuvio 3: Esimerkki YAML -mallista

Malli voidaan kirjoittaa esimerkiksi JSON -muotoiseksi (Kuvio 2 [1]) tai sitä helppolukuisemmalla YAML -kielellä (Kuvio 3 [2]).

[1] Osoitteessa: https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/syntax.

[2] Osoitteessa: https://learn.microsoft.com/en-us/azure/devops/pipelines/process/templates. Viitattu 10.12.2022.

Mainittakoon tähän huomiona, että kappaleessa 4 havaitut ongelmat IaC -tiedostoista löytyivät työkaluista, jotka käyttävät YAML- tai Ruby -pohjaisia malleja.

Ohjelmistokoodin julkaisemistahtiin vaikuttavat osaltaan koodin luettavuus sekä se että kuinka helppo sitä on yksittäisen kehittäjän tai kehitystiimin muokata. Tämän tekstin IaC -työkalujen mallit eroavat toisistaan luettavuudessa sekä muokattavuudessa. Esimerkkejä IaC -työkaluilla kirjoitettavista malleista on nähtävillä kuvissa 4, 5, 6 ja 7.

5.3 Proseduraalisen ja deklaratiivisen ilmaisutavan ero

IaC -työkalut voidaan myös jaotella ohjelmointityylin perusteella *proseduraalisesti* ja *deklaratiivisesti* ilmaistaviin työkaluihin.

Proseduraalinen ohjelmointityyli tarkoittaa, että käyttäjä kertoo askel askeleelta miten tiettyyn *tilaan* päästään. Tämä antaa vapauden käyttää ohjelmointikielten perusrakenteita, kuten vaikkapa silmukoita. Infrastruktuuria konfiguroidessa käyttäjän siis täytyy koko ajan olla tietoinen siitä millaisia tapahtumia tiedosto pitää sisältää, jotta tarvittavat muutokset tapahtuvat oikeassa järjestyksessä.

Mahdollisuus ohjelmoida käyttäen proseduraalisia ilmaisuja tekee infrastruktuurin konfiguroinnista ketterämpää, sillä proseduraaliset komennot antavat kyvykkyyden määritellä että *kuinka* yksittäiset muutokset tehdään. Deklaratiivisissa työkaluissa voidaan ainoastaan määritellä infrastruktuurin lopputila. Kehittäjätiimit voivat tehdä näiden työkalujen avulla muutoksia räätälöidysti, jolloin säännönmukaisuuksien noudattaminen on helpompaa ja voidaan välttyä ennalta tiedostetuilta riskeiltä. Yksi esimerkki ennalta tiedostetusta riskistä olisi olisi tilanne jossa ei haluta että infrastruktuurin muuttuessa palvelin lopettaa hetkellisesti toimintansa (tunnetaan eng. terminä *Zero Downtime Deployment*).

Deklaratiivisessa ilmaisutavassa käyttäjä ainoastaan kuvailee halutun *tilan*. Tämä tarkoittaa sitä, että palveluntarjojan vastuulla on tehdä tarvittavat toimenpiteet halutun *tilan* saavuttamiseksi. Proseduraalisiin ilmaisuihin verrattuna, deklaratiivisesti ilmaistun tiedoston sisältämä kokonaisuus on helpompi ymmärtää. Toisaalta deklaratiivisten työkalujen heikkous on proseduraalisten toimintojen puute. Tästä syystä jotkin deklaratiiviset työkalut, esimerkiksi

Terraform 5.6.1, ovat sisällyttäneet mahdollisuuden käyttää joitain proseduraalisia komentoja.

Ohjelmistokehittäjän näkökulmasta tällä voi olla paljonkin merkitystä. *Proseduraalisesti* ilmaistut mallit saavuttavat ajon aikana jokaisen komennon jälkeen ovat jonkin *tilan*. *Deklaratiivisesti* ilmaistuissa malleissa käyttäjän toimesta voidaan vaikuttaa ainoastaan mallissa kuvattuun lopputilaan.

5.4 Provisioivien ja konfiguroivien työkalujen ero

IaC -työkalut ovat jaoteltu tässä tekstissä kahteen erityyppiseen: konfiguraatiota hallinnoiviin (eng. *configuration management*) työkaluihin sekä provisiointi -työkaluihin (eng. *provisioning*).

Konfiguraation hallinnointiin suunnitellut työkalut ovat ensisijaisesti tarkoitettu toteuttamaan muutoksia jo valmiiksi olemassa oleviin infrastruktuureihin. Niitä voidaan käyttää kuitenkin myös uusien infrastruktuurien rakentamiseen.

Provisiovinti -työkalut puolestaan ovat ensisijaisesti suunniteltuja infrastruktuurin rakentamiseen, mutta toisaalta vastaavasti niitä voidaan käyttää myös infrastruktuurin konfigurointiin sen pystyttämisen jälkeen.

5.5 Isäntä -palvelimet ja agenttiohjelmat

Jotkin IaC -työkalut hyödyntävät isäntä -palvelinta (isäntä eli eng. *master*) ja *agenttiohjelmistoja*. Tässä tekstissä esiteltävistä työkaluista *Chef* 5.7.1 ja *Puppet* 5.7.2 käyttävät kumpikin isäntä -palvelinta sekä agenttiohjelmistoja. Molempia työkaluja voi kuitenkin käyttää ilman isäntä -palvelinta. Tällöin kuitenkin työkalun toiminta on rajoittuneempaa ks. (Brikman 2022, kappale *Master Versus Masterless*).

Isäntäpalvelimella tarkoitetaan keskitettyä palvelinta, joka toteuttaa varsinaiset infrastruktuurin muutokset ylläpidettäviin palvelimiin. Käytännössä siis isäntä -palvelimeen muodostetaan yhteys, jonka jälkeen sinne ladataan konfiguraation muutokset.

Hallinnoitaville solmuille, eli palvelimille johon infrastruktuuri on virtualisoitu, asennetut agenttiohjelmistot keräävät tietoa ja faktoja palvelimista, jotka ne lähettävät takaisin isäntä -palvelimelle. Mikäli isäntä -palvelimen agenttiohjelma havaitsee eron vastaanotettujen tietojen ja kuvatun infrastruktuurin välillä, suorittaa tällöin isäntä -palvelin tarvittavat komennot, joiden vaikutuksesta hallinnoitaville palvelimille konfiguroituu oikea tila.

Tätä prosessia kutsutaan *agentti-palvelin* -protokollaksi. Agentin vastuulla on päivitysten asennus, sekä uusien tilojen päivittäminen hallinnoitaville solmuille. Palvelimilla pyöriviä agenttiohjelmistoja täytyy monitoroida, jotta niiden julkaisuversiot täsmäävät kaikilla. Agenttiohjelmistojen monitorointi ja asentaminen ovat osaltaan yksi ylimääräinen työvaihe lisää, verrattuna työkaluihin jotka toimivat ilman tätä protokollaa.

5.6 Provisiointi -työkalut

Työkalut *Pulumi*5.6.3, *Openstack Heat*5.6.2 ja *Terraform*5.6.1 ovat provisiointi -työkaluja. Taulukosta 2 on nähtävissä, että ne ovat maturiteetiltaan alhaisia, mutta siitä huolimatta etenkin Terraform:n suosio on kasvanut räjähdysmäisesti ohjelmistokehittäjien keskuudessa. Työkalut *Pulumi* ja *Openstack Heat* ovat maturiteetiltaan alhaisempia kuin *Terraform*, mikä osaltaan voi selittää kuilua työkalujen suosion kasvussa.

5.6.1 Terraform

Taulukosta 2 nähdään että eniten suosiotaan on kasvattanut *Go* -ohjelmointikielen päälle rakennettu *Terraform*. Työkalun malleja kirjoitetaan Terraformin omalla HCL -ohjelmointikielellä (HCL eli *Hashicorp Configuration Language*).

Työkalua varten kehitetyn ohjelmointikielen käytössä hyvä puoli on se, että usein ne ovat yksinkertaistettuja sekä suppeampia versiota korkean tason ohjelmointikielistä. Tämän ansiosta niiden käyttöönottaminen on nopeampaa sekä helpompaa kuin monet korkean tason ohjelmointikielet. Työkalua varten kehitetyn ohjelmointikielen käyttämisessä haittapuolena on se, että kehittäjän täytyy opetella uusi ohjelmointikieli.

Terraform perustuu deklaratiivisesti ilmaistaviin malleihin (Kuvio 4), mutta niihin voi kir-

joittaa myös joitain proseduraalisia komentoja. Sellaiset järjestelmät, joihin ei ole asennettu HCL -kirjastoa, *Terraform* muuttaa HCL -ohjelmakoodin JSON -muotoon, joka on kohdejärjestelmille ymmärrettävä formaatti. Terraform -mallit käännetään tiedostosta suoraan konekieliseksi binääriksi, jonka muoto riippuu kohdepalvelimen käyttöjärjestelmästä.

Terraform -mallit tukevat modulaarisia komponentetteja. Halutut muutokset tehdään *Terrafrom Configuration* -tiedostoon 4, joka voidaan validoida, testata sekä ladata versionhallintaan.

Terraform -työkalussa on käytössä erillinen resurssitiedosto, johon on tallennettuna infrastruktuurin edelliset *tilat*. Siitä voi siis halutessaan tarkastella infrastruktuurin muutoksia, mikä on kehittäjän näkökulmasta hyödyllinen ominaisuus. Resurssitiedosto sisältää kovakoodattua dataa palvelimista. Tämä tarkoittaa siis sitä, että se sisältää kovakoodattuja salaisuuksia. Näitä resurssitiedostoja ei siis kuuluisi tallentaa versionhallintaan.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}

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

resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

Kuvio 4: Esimerkki valmiista Terraform konfiguraatiosta [3]

Terraform tekee infrastruktuurin muutokset niin että se ensin vertaa resurssitiedoston vanhaa tilaa uuteen määriteltyyn tilaan. Vertailun jälkeen Terraform tekee muutosten perusteella kullekin pilvitarjoajille ominaiset API -kutsut, jotka puolestaan tuottavat infrastruktuurin tilamuutokset. Resurssien luominen, päivittäminen sekä tuhoaminen ovat kaikki Terraformin vastuulla. Terraform myöskin ymmärtää näiden resurssien keskinäiset riippuvuussuhteet. Riippuvuussuhteiden automatisointi virtaviivastaa huomattavasti infrastruktuurin automatisointia, sekä ennaltaehkäisee vikatilojen 4.1 muodostumista.

[3] Osoitteessa: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build. Viitattu 4.12.2022.

5.6.2 Openstack Heat

Openstack on infrastruktuuri -työkalu, joka sisältää useita eri rajapintoja. Se on siis alusta, jossa ollaan vuorovaikutuksessa yksittäisen pilviresursseja ohjaavan solmukohdan kanssa, joka puolestaan on vuorovaikutuksessa muiden laskentaa tekevien solmukohtien kanssa palvelin -instanssien luomiseksi.

Varsinainen työkalu, joka vastaa tämän tekstin kontekstissa IaC -työkalua, on nimeltään *Openstack Heat*.

Heat käyttää YAML -pohjaisia malleja (Kuvio 5). Mallit voidaan validoida, testata sekä vertaisarvioida. YAML -pohjaiset mallit voidaan yhdistää muihin YAML -pohjaisiin malleihin kanssa, kuten esimerkiksi Terraform 5.6.1, Pulumi 5.6.3 ja Ansible 5.7.3. Yhteensopivuus muiden työkalujen kanssa on siinä mielessä hyödyllinen ominaisuus, että se antaa mahdollisuuden hyödyntää moduuleita muiden IaC -työkalujen yhteisökirjaistoista.

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: F18-x86_64-cfntools
      flavor: m1.small
```

Kuvio 5: Esimerkki Openstack Heat -työkalun mallista[4]

Infrastruktuurin konfigurointi tapahtuu niin että Heat -malliin kirjotetaan uusi tila, jonka jälkeen kutsutaan *Openstack Heat* -rajapintaa. Kutsun saatuaan *Openstack Heat* suorittaa sopivat API-kutsut muihin Openstack:n rajapintoihin. Openstack:n muut rajapinnat hoitavat lopulta infrastruktuuriin tehtävät muutokset, kullekin rajapinnalle ominaisella protokollalla. Malleissa kuvattavien resurssien riippuvuussuhteiden *erikoistaminen* on mahdollista. Tuolloin Heat kutsuu muita Openstack:n rajapintoja, jolloin muutokset tapahtuvat halutussa järjestyksessä.

Openstack eroaa muista tekstin IaC -työkaluista siten, että se on suunniteltu mahdollisimman joustavaksi sekä yhteensopivaksi muiden pilvipalveluiden tarjoajien kanssa. Yleisesti IaC -työkalut ovat suunniteltuja yleensä yhden organisaation sisällä, eivätkä transparentiksi muiden IaC -toimijoiden kanssa. Openstack on nimensä mukaisesti (suom. *avoin kasaantuma*)

[4]Osoitteessa: https://wiki.openstack.org/wiki/Heat. Viitattu 11.12.2022.

suunniteltu avoimeksi alustaksi, jonka kanssa on mahdollista hyödyntää eri organisaatioiden pilvipalveluita.

5.6.3 Pulumi

Tämän tekstin vertailtavista työkaluista Pulumi eroaa muista mm. siten, että se on suunniteltu käytettäväksi mahdollisimman usealla eri ohjelmointikielellä. Tällä hetkellä Pulumi tukee seuraavia korkean tason ohjelmointikieliä: *Javascript*, *Typescript*, *Python*, *Go*, *.NET* -viitekehyksen kielet, *Java* sekä *YAML*. Ohjelmointikieleksi voi siis valita ennalta tutun kielen, tai sen voi valita käyttötapauksen mahdollisia erityispiirteitä ajatellen.

Pulumi -malleja voi siis kirjoittaa ohjelmointikielen oman kehitysympäristö -ohjelman avulla. Kehitysympäristö -ohjelman hyödyntämisessä on se etu, että tuolloin IaC -tiedostoja analysoidaan automaattisesti staattisen korjaustyökalun avulla (eng. *linting* eli syntaksin automaattiset korjausehdotukset). Toisaalta, mikäli käytössä on esikäännettävä ohjelmointikieli, ei tuolloin kelvotonta IaC -tiedostoa voida kääntää konekieliseksi. Mallien testaaminen sekä validointi voidaan tehdä ohjelmointikielille standardisoituneiden testauskirjastojen avulla. Pulumi tarjoaa siis kehittäjälle vapauden valita sopivan ohjelmointikielen, kehitysympäristön sekä muita oleellisia työkaluje kuten esimerkiksi sopivan testauskirjaston.

Infrastruktuurin hallinnointi tapahtuu *Pulumi Servicen* avulla, joka salaa automaattisesti salaisuudet sekä muun arkaluontoisen datan silloin, kun informaatiota lähetetään tai kun se varastoidaan. Tällöin vältytään arkaluontoisen tiedon tallentamiselta versionhallintaan (ks. kappale 4.1).

Varsinainen infrastruktuuri kapseloidaan *Pulumi Package*-nimiseksi projektikokonaisuudeksi, joka voidaan jakaa ja uudelleenkäyttää muiden kehittäjien keskuudessa. Työkalulla on myös mahdollista luoda sekä tallentaa omia räätälöityjä komponentteja. Pulumissa infrastruktuurin muutosprosessi perustuu tilan vertailuun (tilojen vertailusta 5.6.1). Prosessi eroaa kuitenkin siinä, ettei Pulumi tarjoa erillistä tiedostoa, johon on tallennettuna edeltävät tilat.

Ohjelmistokehityspaketti *Pulumi Automatisation -API* mahdollistaa työkalun komentorivi -toiminnallisuuden vahvasti tyypitetyssä ympäristössä, mikä tekee komentorivityöskentelystä

turvallisempaa. Infrastruktuurin julkaiseminen voidaan tehdä komentoriviltä, *Pulumi Automatisation -API:n* avulla tai sisällyttää suoraan osaksi valmista CI/CD -prosessia.

Infrastruktuurin hallinnoinnin turvatoimia varten Pulum:ssa on saatavilla käyttäjäkohtaiset oikeudet (eng. *role-based permission*) sekä yksivaiheinen kirjautuminen (eng. *single sign-on*).

Työkalu tarjoaa kehittäjälle myös muita hyödyllisiä toimintoja. Näistä yhtenä esimerkkinä on *Pulumi Crossguard*, jonka avulla voidaan tarkkailla mm. määräystenmukaisuutta ja pilvipalveluiden käytöstä aiheutuvia kustannuksia.

5.7 Konfiguraation hallinnointi työkalut

Työkalut *Chef* 5.7.1, *Puppet* 5.7.2 ja *Ansible* 5.7.3 lukeutuvat konfiguraation hallinointi työkaluihin. Konfiguraation hallinointi työkaluihin liittyviä ongelmia on nostettu esille kappaleessa 4.1. Ongelmista huolimatta, näidenkin työkalujen suosio on ollut viime vuosina kasvussa (Taulukko 2).

5.7.1 Chef

Konfiguraation hallinnointia varten kehitetty työkalu *Chef* julkaistiin jo ensimmäisen kerran vuonna 2009. Chef -malleja kirjoitetaan *deklaratiiviseen* tapaan Ruby -ohjelmointikielellä. Chef tarjoaa *InSpec* -nimisen testikirjaston, jolla voidaan testata säännösten noudattamista sekä turvallisuutta. Testausta ja validointia voidaan tehdä myöskin Rubylle standardisoituneilla testikirjastoilla.

Chef -mallit ovat tarkoitettu kirjoitettavan pieniksi koodilohkoiksi. Jokainen malli toteuttaa yksittäisen tehtävän (Kuvio 6). Malleista muodostetaan kokonaisuuksia, joita kutsutaan resepteiksi. Reseptien tehtävänä puolestaan on määritellä systeemin yhden osan konfigurointi, esimerkiksi tietokannan käyttöönottaminen. Lopulta reseptit ja konfiguraatiodata kapseloidaan keittokirjoihin, eli siis *Cookbook* -nimisiin kokonaisuuksiin.

Kuvio 6: Esimerkki Chef -mallista[5]

Chef hyödyntää *Infra Client* -nimistä agenttia ks.5.5. *Infra Server* on solmukohta, jossa on säilössä keittokirjat, infrastruktuuriin liittyvät käytänteet (eng. *policies*) sekä metadata jokaisesta hallinoitavasta palvelimesta. Metadata on siis tietoa palvelimista, kuten käytettävä käyttöjärjestelmä ja IP-osoite. Agentti muodostaa säännöllisesti yhteyden *Infra Serveriin*, johon on tallennettuna ajankohtaiset keittokirjat. Mikäli agentin antamat tiedot palvelimen infrastruktuurista eroavat *Infra Serveriltä* saaduista tiedoista, suorittaa tuolloin agentti tarvittavat muutokset, jotta haluttu *tila* saavutetaan.

5.7.2 Puppet

Chef:n kanssa hyvin samankaltainen IaC -työkalu *Puppet* julkaistiin ensimmäisen kerran vuonna 2005. Puppet -malleja kirjoitetaan sen omalla *Puppet Language* -ohjelmointikielellä.

Mallit kirjoitetaan yksittäisiksi tehtäviksi, joita käytetään modulaarisesti suuremmissa kokonaisuuksissa. Moduuleista koostetut kokonaisuudet hoitavat infrastruktuurin konfiguraation jonkin osion, esimerkiksi tietokannan pystyttämisen. Moduulit sisältävät mallit, konfiguraatioon vaadittavan datan sekä *manifestit*. *Manifestit* sisältävät metatiedot hallinnoitavista palvelimista. Moduulien data voidaan erottaa muusta toiminnallisuudesta keskitettyyn tietokantaan *Puppet Hiera* -työkalun avulla. Tämä mahdollistaa koodin kokonaisvaltaisen testaamisen, sekä testaamisen parametrien erilaisilla variaatiolla.

Puppet käyttää *agentti-palvelin* -protokollaa (ks. 5.5). Edellä mainitut *manifestit* ovat siis agenttiohjelmiston (*Puppet Facter*) tuottamia tiedostoja.

[5]Osoitteessa: https://docs.chef.io/resources/template. Viitattu 10.12.2022.

Puppet tarjoaa *Puppet DSL intellisense* -laajennoksen *Visual Studio Code* -työkalulle. Laajennoksen ansiosta tiedostojen kirjoittamisen yhteyteen on saatavilla syntaksin automaattiset korjausehdotukset.

Puppet tarjoaa myös *Litmus* -työkalun, jolla voidaan testata moduulien yhteensopivuus kohdejärjestelmälle. Tämä vähentää riskiä infrastruktuurin tilan muuttamisen yhteydessä syntyvistä ongelmista.

Ohjelmistokehittäjän näkökulmasta on oleellista tietää myös, että Puppet kerää kaiken palvelimista generoidun datan, esimerkiksi IP-osoitteen, *PuppetDB* -nimiseen tietokantaan.

5.7.3 Ansible

Ansible on konfiguraation hallinnointi työkaluista maturiteetiltaan muita alhaisempi, vaikka se on kasvattanut eniten suosiotaan niiden keskuudessa.

Ensisijaisesti Ansible -malleja kirjoitetaan YAML -kielellä tiedostoon, josta käytetään nimeä *Playbook* (Kuvio 7). Ansiblen toiminallisuuden voi kuitenkin toteuttaa ohjelmallisesti millä kielellä tahansa, joka kykenee tuottamaan JSON -tiedoston. Malleihin kirjoitetut tehtävät suoritetaan oletuksena niiden annetussa järjestyksessä.

```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

Kuvio 7: Esimerkki Ansiblen YAML -tiedostosta[6]

Muutokset infrastruktuuriin tehdään Ansible:lla niin, että ensin muodostetaan hallinnoituihin solmuihin tietoliikenneyhteys SSH:n tai *Windows Remote Managementin* avulla. *Windows Remote Managementin* käyttäminen voi olla mieleistä silloin jos haluaa mahdollistaa *Windows Powershell:n* käytön. Tämän yhteyden yli hallinnoitavalle palvelimelle asennetaan suoraan väliaikainen agentti, josta käytetään nimitystä *Ansible module*. Nämä agenttiohjelmat kapseloivat sisälleen malleihin kuvatun *tilan*. Useimmat

[6]Osoitteessa: https://docs.ansible.com/ansible-core/devel/playbook_guide/playbooks_intro. Viitattu 2.12.2022.

Ansible:n agenttiohjelmat tarkistavat vielä lopuksi, että palvelin on saavuttanut kuvatun *tilan*. Agenttiohjelmien toiminta kannattaa tarkistaa hiekkalaatikko -ympäristössä ennen niiden suorittamista kohdepalvelimilla, sillä kaikki Ansible:n agenttiohjelmat eivät suorita lopputarkastusta. Tämä lisää jonkin verran epävarmuutta työkalun käyttämiseen, sillä ohjelmistokehittäjän täytyy jokaisella kerralla tarkistaa, että suorittaako agenttiohjelmisto lopputarkastuksen. Tästä toiminnallisuudessa on varoitettu Ansiblen tuotedokumentaatiossa. Suorituksen lopuksi agenttiohjelma poistaa itsensä automaattisesti hallinnoitavalta palvelimelta.

Ansiblen dokumentaatiossa on kerrottu, että työkalusta on pyritty tekemään mahdollisimman minimalistinen ja helppokäyttöinen. Tämä osaltaan pitää paikkansa, sillä työkalu ei vaadi mitään muuta kuin asennuksen päätelaitteelle sekä tekstieditorin. Kaikki muu toiminnallisuus tapahtuu komentorivin kautta. Komentorivillä työskennellessä on omat riskinsä, siksi Ansible tarjoaa komentorivilaajennoksen *Ansible Lint*. Komentorivilaajennus analysoi staattisesti komentoriville kirjoitettavia käskyjä ennen niiden suorittamista, joka tekee komentorivillä työskentelystä turvallisempaa.

5.8 KOTLESS: Kohti käyttäjäystävällistä IaC -käytäntöä

Tutkimuksessa (Tankov ym. 2021) esitetään toisenlaista ratkaisua, jossa infrastruktuuri rakennettaisiin varsinaisen ohjelman lähdekoodista metatietojen avulla koostamalla. Vastaavia IiC -työkaluja (eng. *Infrastructure in Code*) on jo ennalta olemassa *AWS Chalice*, *Zappa* sekä *Osiris*. Näitä työkaluja käytetään kuitenkin *serverless* -tyyppisessä infrastruktuurissa, joka on vain yksi osa-alue koko IaC -termin alaisuudessa. IiC -lähestymistapa vapauttaisi kehittäjän infrastruktuurin suunnittelusta, ylläpidosta sekä konfiguroinnista.

Pilvialustojen tarjonta on heterogeenistä, minkä takia tietyissä tapauksissa jotkin pilvipalveluiden tarjoajat voivat vastata kysyntään tehokkaammin, kuin toiset. Eroja voi ilmetä esimerkiksi yksittäisten palveluiden hinnoitteluissa. Tämän takia IaC -käytänteen avulla voidaan optimoida kustannuksia valitsemalla tarpeeseen sopivia pilvipalveluita.

IaC -käytäntö myöskin mahdollistaa protokollien suunnittelun ja kehittämisen vika- sekä katastrofitilanteita varten. Tällöin IaC:n avulla toimivien ohjelmien toiminta on ketterämpää sekä joustavampaa vikatilanteissa.

Näihin tarpeisiin voidaan vastata juuri sen takia, koska IaC mahdollistaa infrastruktuurin suunnittelun ja testaamisen. Herääkin kysymys, että kuinka IiC voisi vastata näihin tarpeisiin, mikäli infrastruktuurin testaus ja suunnittelu tehtäisiin automaattisesti?

6 Pohdinta

IaC -työkalujen keskuudessa ei ole mitään yksittäistä työkalua, jonka toiminnallisuus olisi ylivertaista muihin verrattuna.

Terraform:ssa on selkeästi suurin ja aktiivisin yhteisö, sekä sen maturiteetti on korkeampi kuin muilla provisionti -työkaluilla (Taulukko 1).

Pulumi taas toisaalta on huomioinut paremmin turvallisuuteen liittyviä ongelmia, sekä työkalun selkeä etu on että sitä voi kirjoittaa korkean tason ohjelmointikielillä.

Openstack:n etu puolestaan on se, että siitä on pyritty kehittämään mahdollisimman yhteensopiva muiden IaC -työkalujen kanssa.

Toisaalta kuitenkin ne työkalut, joiden infrastruktuurin tilan muuttaminen tapahtuu tilavertailulla, sisältävät valuvian. Mikäli konfiguraation määrittävä malli on vanhentunutja sen kuvaamaan tilaan tarvittavat muutokset suoritetaan, muodostuu palvelimelle tiedostamatta virheelinen *tila*. Tällainen virhe voi ilmentyä tilanteessa, jossa versionhallinnasta on unohtanut ladata viimeisimmät muutokset, tai mikäli sinne on unohtunut lisätä uudet muutokset. Virhe on ihmislähtöinen, mutta se johtuu puuttellisesta suunnittelusta. Ehdottaisin tämän virheen ehkäisemiseksi automaattisen tarkastuksen kehittämistä, jotta infrastruktuurin tilaa ei voisi muuttaa vanhentuneeseen versioon.

Konfiguraation hallinnointi työkalu Ansible on selkeästi paras valinta silloin jos halutaan työkalu mahdollisimman yksinkertaisella toiminnallisuudella. Toisaalta myös se ei käytä *agentti-palvelin* -protokollaa.

Chef:n sekä Puppet:n IaC -projektit ovat laajempia kokonaisuuksia, joihin kuuluu esimerkiksi konfiguraatioon tarvittava data, mallit sekä metatiedot hallinnoitavista palvelimista. Ansiblessa käytännössä kaikki toiminnallisuus on yhden moduulin sisällä. Mikäli halutaan laajempia infrastruktuurin muutoksia kerralla, Ansiblessa täytyy vain luoda uusi moduuli. Kun taas toisaalta Chef- ja Puppet -työkaluissa muutosten tuottaminen tarkoittaa usean tiedoston muokkaamista ja testausta.

Kappaleessa 4.1 polttopiste on etenkin konfiguraation hallinnointi työkaluissa. Tietoturvaongelmat eivät kuitenkaan rajoitu ainoastaan näihin työkaluihin, sillä käyttämässäni tutkimuksissa lähdeaineisto sisälsi lähinnä niiden IaC -työkalujen tiedostoja, joiden ongelmia kappaleessa 4.1 on nostettu esille.

Provisionti -työkalut ovat huomattavasti uudempia alalla, mikä osaltaan voi selittää tutkimusaineistoissa niiden vähäisen käytön.

Tietoturvaongelmat ja koodihajut eivät pelkästään johdu työkalusta. Kuten kappaleessa 4.1 on mainittu, kyseessä on sekä kehittäjän itsensä, että kehitystyökalun luoma summa, jonka lopputuloksena saattaa ilmetä epätarkoituksenmukaista toiminnallisuutta tai tietoturvaongelmia.

7 Yhteenveto

Tämän tutkimuksen tarkoituksena oli esitellä lyhyesti IaC -käsitteen taustaa, sekä vertailla tämänhetkisiä IaC -työkaluja nykyisten tutkimusten sekä alan tarpeiden näkökulmasta. Tekstissä on tuotu työkalujen perustoimintojen ja faktojen lisäksi myös syvällisempää tietoa siitä, että miten nämä perustoiminnot vaikuttavat työkalun käytettävyyteen. Tekstissä on nostettu myös esille IaC -käytänteeseen liittyviä koodihajuja ja tietoturvaongelmia sekä selvitetty niiden syy-seuraus -suhdetta IaC -työkalujen yhteydessä.

Työkalujen toiminnallisuutta on esitelty niin, että niiden kaikkein oleellisimmat piirteet ovat nostettuina esille. Suuresta informaatiomäärästä johtuen kaikkia mahdollisia näkökulmia ole pystytty tuomaan esille tässä tekstissä, jotta asiasisältö pysyisi selkeänä ja helposti ymmärrettävänä.

Suurin osa aiheeseen löytämistäni tutkimuksista liittyivät nimenomaan konfiguraation hallinnointi työkaluihin, joten ehdottaisin että jatkossa tutkimusta painotettaisiin enemmän provisionti -työkaluihin. Etenkin sen takia, että niiden suosio on kasvanut huomattavasti viime vuosien aikana. Tutkimusta on myös aiheellista lisätä etenkin hyvien koodikäytänteiden osalta. Selkeä yhtenäinen linja IaC -käytänteen yhteydessä virtaviivaistaisi työkalua kehittävien toimijoiden sekä yksittäisten kehittäjien työtä. Tämä myöskin vähentäisi versionhallintaan sekä yhteisökirjastoihin ladattavia virheellisiä tai ongelmallisia kooditiedostoja.

Lähteet

Brikman, Yevgeniy. 2022. *Terraform: Up and Running.* "O'Reilly Media, Inc.".

Dörbecker, Regine, Daniela Böhm ja Tilo Böhmann. 2015. "Measuring Modularity and Related Effects for Services, Products, Networks, and Software – A Comparative Literature Review and a Research Agenda for Service Modularity". Teoksessa *2015 48th Hawaii International Conference on System Sciences,* 1360–1369. https://doi.org/10.1109/HICSS.2015.167.

Hummer, Waldemar, Florian Rosenberg, Fábio Oliveira ja Tamar Eilam. 2013. "Testing idempotence for infrastructure as code". Teoksessa *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing,* 368–388. Springer.

Morgan, Lorraine, ja Kieran Conboy. 2013. "Key Factors Impacting Cloud Computing Adoption". *Computer* 46 (10): 97–99. https://doi.org/10.1109/MC.2013.362.

Rahman, Akond, Effat Farhana, Chris Parnin ja Laurie Williams. 2020. "Gang of eight: A defect taxonomy for infrastructure as code scripts". Teoksessa *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE),* 752–764. IEEE.

Rahman, Akond, Effat Farhana ja Laurie Williams. 2020. "The 'as code'activities: development anti-patterns for infrastructure as code". *Empirical Software Engineering* 25 (5): 3430–3467.

Rahman, Akond, Rezvan Mahdavi-Hezaveh ja Laurie Williams. 2019. "A systematic mapping study of infrastructure as code research". *Information and Software Technology* 108:65–77.

Rahman, Akond, Chris Parnin ja Laurie Williams. 2019. "The Seven Sins: Security Smells in Infrastructure as Code Scripts". Teoksessa *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE),* 164–175. https://doi.org/10.1109/ICSE.2019.00033.

Rahman, Akond, Md Rayhanur Rahman, Chris Parnin ja Laurie Williams. 2021. “Security smells in ansible and chef scripts: A replication study”. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30 (1): 1–31.

Rahman, Akond, ja Laurie Williams. 2019. “Source code properties of defective infrastructure as code scripts”. *Information and Software Technology* 112:148–163. ISSN: 0950-5849. https://doi.org/https://doi.org/10.1016/j.infsof.2019.04.013. https://www.sciencedirect.com/science/article/pii/S0950584919300965.

Sokolowski, Daniel. 2022. “Infrastructure as Code for Dynamic Deployments”.

Tankov, Vladislav, Dmitriy Valchuk, Yaroslav Golubev ja Timofey Bryksin. 2021. “Infrastructure in Code: Towards Developer-Friendly Cloud Applications”. Teoksessa *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 1166–1170. https://doi.org/10.1109/ASE51524.2021.9678943.

Suomen virallinen tilasto (SVT): Tietotekniikan käyttö yrityksissä. http://www.stat.fi/til/icte/2021/icte_2021_2021-12-03_kat_003_fi.html.