Mikael Holm

Sovelluksen käyttöönottaminen pilviympäristössä koodipohjaisesti

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

19.1.2024

Tiivistelmä

Tekijä:	Mikael Holm
Otsikko:	Sovelluksen käyttöönottaminen pilviympäristössä koodipohjaisesti
Sivumäärä:	61 sivua
Aika:	19.1.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaaja:	Tutkijaopettaja Hannu Markkanen

Insinöörityössä tutkittiin sovelluksen käyttöönottamista pilviympäristössä koodipohjaisesti. Työn aikana tutustuttiin siihen, kuinka sekä sovellus että siihen liittyvä infrastruktuuri pystyttiin käyttöönottamaan pilviympäristössä samanaikaisesti mahdollisimman automatisoidusti. Työssä toteutettiin yksinkertainen matemaattinen peli, jossa loppukäyttäjät pystyivät harjoittelemaan yksikkömuunnoksia selainpohjaisesti. Matemaattinen peli ilmensi palvelukokonaisuutta sekä edusta- että taustapalveluiden osalta.

Infrastruktuurin koodipohjaiseen hallintaan käytettiin Terraform-työkalua, ja jatkuvan integraation ja jatkuvan käyttöönoton alustana hyödynnettiin GitHub Actions -palvelua. GitHub Actionsin avulla Google Cloud Platform (GCP) -ympäristöön luotiin sovelluksen vaatima infrastruktuuri Terraformilla ja käyttöönotettiin sovellus. GitHub Actionsin ja Terraformin avulla käyttöönotettiin sekä sovellus että siihen liittyvä infrastruktuuri huomioiden ympäristökohtaiset muuttujat, kuten erilaisten tunnusten ja salasanojen hallinta.

Sovelluksen käyttöönotto pilviympäristöön tehtiin automatisoidusti hyvin korkealla tasolla. Insinöörityössä havaittiin, että Terraform soveltui hyvin infrastruktuurin koodipohjaiseen hallintaan ja sen avulla pystyttiin luomaan infrastruktuuri automatisoidusti GCP-ympäristöön GitHub Actions -työnkululla. Työssä toteutettujen Terraform-moduulien avulla infrastruktuuriin liittyvien konfiguraatiotiedostojen hallinnasta tehtiin sujuvaa sen lisäksi, että infrastruktuurin ohella myös sovellus saatiin käyttöönotettua samalla kerralla hyödyntäen GitHub Actionsia. Infrastruktuuriin tehtävistä muutoksista saatiin annettua myös yhteenveto GitHubin yhdistämispyyntöjen (pull request) yhteyteen omalla GitHub Actions -työnkululla. Tällä tavalla kehittäjät pystyivät näkemään, mitä muutoksia Terraform tekisi infrastruktuuriin, jos kehityshaarassa olevat lähdekoodimuutokset päätettäisiin yhdistää esimerkiksi päähaaraan.

Avainsanat: Google Cloud Platform, GitHub, infrastruktuuri koodina, jatkuva integraatio, jatkuva toimitus, Terraform

Abstract

Author:	Mikael Holm
Title:	Code-based software deployment in cloud environment
Number of Pages:	61 pages
Date:	19 January 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Supervisor:	Hannu Markkanen, Researching Lecturer

In this study, the implementation of an application in the cloud environment was studied based on code. During the work, it was studied how both the application and the related infrastructure could be deployed simultaneously in the cloud environment in the most automated way possible. A simple mathematical game was implemented in in the study where end users could practice unit conversions. The mathematical game was implemented to epitomize services from both frontend services and backend services.

Terraform was used for the code-based management of the infrastructure, and GitHub Actions was used as a platform for continuous integration and continuous deployment. With the help of GitHub Actions, the infrastructure required by the application was created in the Google Cloud Platform (GCP) environment with Terraform and the application was deployed. The application and the related infrastructure were implemented using GitHub Actions and Terraform, considering environment-specific variables, such as the management of different credentials and passwords.

The deployment of the application to the cloud environment was automated at a very high level. During the study, it was found that Terraform was well suited for the code-based management of the infrastructure and it was possible to create the infrastructure automatically in the GCP environment by using the GitHub Actions workflow. With the help of the Terraform modules implemented in the work, the management of the configuration files related to the infrastructure was made fluent; in addition to that, the infrastructure and the application could be deployed simultaneously by using GitHub Actions. A summary of the changes to the infrastructure was also given to GitHub's pull requests by utilizing the GitHub Actions workflow. In this way, the developers were able to see what changes Terraform would perform in the infrastructure if the source code changes in the development branch were decided to be merged with, for example, the main branch.

Keywords: continuous deployment, continuous integration, Google Cloud Platform, GitHub, infrastructure as code, Terraform

Sisällys

Lyhenteet

CI: *Continuous integration*. Jatkuva integraatio, jonka avulla voidaan automaattisesti esimerkiksi kääntää sovelluksen lähdekoodi ja suorittaa lähdekoodiin liittyvät integraatio- ja yksikkötestit.

CD: *Continuous deployment*. Jatkuva käyttöönotto, jonka avulla sovellus voidaan ottaa käyttöön kohdeympäristössä automaattisesti ilman manuaalisia työvaiheita, kuten erilaisia käsin tehtäviä asennuksia.

GCP: *Google Cloud Platform*. Googlen pilviympäristö, jonne luodaan erilaisia Googlen tarjoamia pilvi-infrastruktuuriin liittyviä palveluita, kuten esimerkiksi relaatiotietokantoja, virtuaalikoneita ja tallennustiloja.

JSON: *JavaScript Object Notation*. Tapa esittää tietoa avain-arvopareina. JSON on helposti ihmisen tulkittavissa ja kirjoitettavissa oleva tiedon esittämistapa. Lisäksi koneiden on helppo jäsentää ja luoda tietoa JSON-muodossa.

SSL: *Secure Sockets Layer*. Suojausprotokolla, jota käytetään salatun yhteyden muodostamiseksi palvelimen ja asiakasohjelmiston, kuten verkkoselaimen, välille. Salattua yhteyttä varten tarvitaan varmenne, joka koostuu sekä yksityisestä että julkisesta osuudesta. Varmenteen julkisella osuudella asiakasohjelmiston lähettämät tiedot salataan, kun taas varmenteen yksityisellä osuudella salatut tiedot voidaan purkaa palvelinpäässä.

YAML: *Yet Another Markup Language*. Ihmisen tulkittavissa oleva merkintäkieli, jota käytetään usein konfiguraatiotiedostoissa tai muunlaisissa tiedostoissa tietojen esittämiseksi.

1 Johdanto

Insinöörityössä tutkittiin sovelluksen käyttöönottamista pilviympäristössä koodipohjaisesti. Sovellus ja siihen liittyvä infrastruktuuri haluttiin ottaa käyttöön pilviympäristössä mahdollisimman automatisoidusti hyödyntäen jatkuvaan integraatioon (Continuous Integration, CI) ja jatkuvaan käyttöönottoon (Continuous Deployment, CD) liittyvää alustaa. Insinöörityön aikana toteutettiin yksinkertainen verkkosovellus, joka muodostui sekä edusta- että taustapalveluista. Työn aikana luotiin yksinkertainen matemaattinen peli, jossa sovelluksen loppukäyttäjät pystyvät harjoittelemaan yksikkömuunnoksia. Matemaattinen peli ilmensi palvelukokonaisuutta sekä edusta- että taustapalveluiden osalta.

Työn toimeksiantajana toimi Solita Oy. Työn tarkoituksena oli avartaa ymmärrystä infrastruktuuri koodina -työkalujen hyödyntämisestä pilviympäristössä mahdollisimman automatisoidusti. Lisäksi työ mahdollisti uusien oppien jakamisen sekä työyhteisössä että tietoa tarvitsevien projektien keskuudessa.

Insinöörityön empiirisessä osassa haluttiin selvittää, kuinka valittuun pilviympäristöön voidaan viedä samanaikaisesti sekä sovellus että siihen liittyvä infrastruktuuri. Tutkimustyötä varten valittiin laajasti käytettyjä sovelluskehityksen työkaluja työn toteuttamista varten. Infrastruktuurin koodipohjaiseen hallintaan valittiin Terraform-työkalu ja CI/CD-ratkaisuksi GitHub Actions -palvelu. Terraformin avulla luotiin sovelluksen kannalta olennainen infrastruktuuri kirjoitettujen konfiguraatiotiedostojen pohjalta. GitHub Actionsin avulla automatisoitiin kaikki mahdolliset manuaaliset työvaiheet, kuten infrastruktuurin luominen Terraformilla ja sovelluksen käyttöönotto pilviympäristössä.

Pilviympäristöksi valittiin työssä Google Cloud Platform (GCP). Tämä pilviympäristö ei ollut entuudestaan tuttu insinöörityön toteuttajalle. Työn toimeksiantaja Solita Oy suositteli tutustumaan GCP-ympäristön

hyödyntämiseen osana insinöörityötä ja tarjosi tarvittaessa tukea ongelmatilanteisiin ja vastauksia erilaisiin kysymyksiin.

2 Infrastruktuuri koodina

2.1 Infrastruktuurin luonti koodipohjaisesti

Infrastruktuuri koodina on menetelmä, jolla infrastruktuuria hallitaan koodipohjaisesti. Infrastruktuuri koodina mahdollistaa johdonmukaisen ja uudelleentoistettavan tavan luoda ja muuttaa järjestelmiä sekä niiden määrityksiä esimerkiksi pilviympäristöissä. Koodiin tehtävillä muutoksilla voidaan automatisoida muutosten vienti erilaisiin järjestelmiin infrastruktuurissa. [1, s. 4.]

Infrastruktuurin luonti koodipohjaisesti tuo mukanaan useita erilaisia hyötyjä, kuten esimerkiksi muutosten nopean viennin infrastruktuuriin, mikä luo mahdollisimman paljon arvoa liiketoiminnalle. Infrastruktuurin käyttäjät saavat käyttöönsä tarvittavat resurssit silloin, kun he tarvitsevat niitä. Resursseina voivat toimia esimerkiksi palvelinklusterit ja tarvittavat tietoliikenneyhteydet. Infrastruktuuri koodina mahdollistaa vaivattoman tavan luoda järjestelmiä, jotka ovat luotettavia, turvallisia ja kustannustehokkaita. [1, s. 4.]

Infrastruktuurin luomiseen koodin avulla on nimetty kolme ydinkäytäntöä; kaiken määritteleminen koodina, jatkuva testaaminen ja toimittaminen sekä pienten ja yksinkertaisten osien rakentaminen, joita voidaan muuttaa itsenäisesti. [1, s. 9–10.]

Kun infrastruktuuri määritellään koodina, muutosten tekeminen on sekä nopeaa että luotettavaa. Saman koodin avulla on mahdollista luoda useita esiintymiä infrastruktuurista esimerkiksi jonkun toisen kehittäjän toimesta. Infrastruktuurin osat, jotka on määritelty koodina, luodaan joka kerta samalla tavalla. Näin infrastruktuurista saadaan aina halutunlainen testauksen ja toimituksen helpottuessa samalla. Infrastruktuurin määritteleminen koodina lisää myös läpinäkyvyyttä työssä: kehittäjät näkevät, miten infrastruktuurin eri osat on

rakennettu, ja he voivat katselmoida ja ehdottaa parannuksia lähdekoodiin. Lisäksi kehittäjät voivat oppia lähdekoodin avulla, miten infrastruktuuri tai jokin sen osista toimii. [1, s. 10.]

Infrastruktuuritiimit käyttävät automaatiota järjestelmän jokaisen komponentin testaamiseksi ja käyttöönottamiseksi. Kehittäjien työt integroidaan osaksi kokonaisuutta, mikä mahdollistaa muutosten testaamisen ilman, että keskeneräisen työn tulisi olla valmis. Tarkoituksena on rakentaa laadukkaita tuotoksia sen sijaan, että laatua testattaisiin jälkikäteen. Jatkuvan integraation avulla eri kehittäjien tuotokset on mahdollista yhdistää ja testata koko kehitysprosessin aikana. Jatkuvan käyttöönoton avulla mahdollistetaan nopea tuotantovalmius. [1, s. 10.]

Mitä laajempia järjestelmät ovat, sitä vaikeampaa muutosten tekeminen on ilman, että jokin järjestelmän osista lakkaisi toimimasta muutoksen yhteydessä. Kun järjestelmä määritellään koodina pienissä osissa, kokonaisuutta on helpompi tarkastella selkeän rakenteen vuoksi. Näin kehittäjät voivat helposti muuttaa järjestelmäkokonaisuuden jokaista komponenttia itsessään ja testata tiettyä komponenttia muista komponenteista erillään. [1, s. 11.]

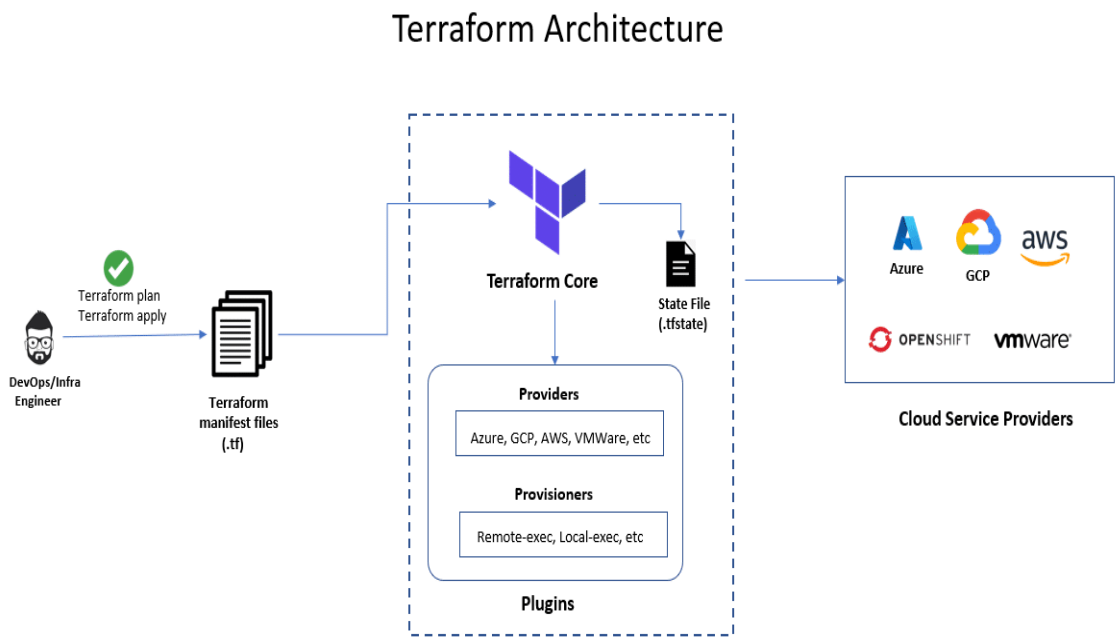
Infrastruktuuri koodina -työkalut mahdollistavat infrastruktuurin hallinnan ilman graafista käyttöliittymää hyödyntäen konfiguraatiotiedostoja. Konfiguraatiotiedostoihin määritellään kulloinkin luotavan infrastruktuurin osalta kaikki tarvittavat resurssit, mikä mahdollistaa muutosten hallinnan sekä turvallisesti että johdonmukaisesti. Tämän lisäksi infrastruktuuriin vietävät muutokset voidaan tehdä niin uudelleenkäytettäviksi kuin myös jaettaviksi. Uudelleenkäytettävyydellä ja jaettavuudella tarkoitetaan sitä, että samanlainen infrastruktuuri voidaan luoda uudelleen alusta alkaen käyttäen samaa lähdekoodia jonkin toisen osapuolen osalta. Muutosten määritteleminen koodina mahdollistaa myös sen, että infrastruktuuriin tehdyt muutokset pystytään versioimaan versionhallintajärjestelmään. [2.]

2.2 Terraform-työkalu

Terraform on HashiCorpin kehittämä työkalu infrastruktuurin hallitsemista varten koodipohjaisesti. Terraform mahdollistaa infrastruktuurin hallinnan hyödyntäen konfiguraatiotiedostoja, jotka ovat ihmisen tulkittavissa. Sen avulla voidaan hallita yhteensä yli 1 000:ta eri pilvipalvelua ja omaa palvelinympäristöä koodipohjaisesti. Esimerkiksi Amazon Web Servicen, Azuren ja Google Cloud Platformin hallinta ovat mahdollisia Terraformilla. [2.]

Terraform on Go-ohjelmointikielellä kirjoitettu työkalu. Se tekee rajapintakutsut kehittäjien puolesta yhteen tai useampaan palveluntuottajaan, joita voivat olla esimerkiksi Azure ja Google Cloud Platform. Terraform päättelee tarvittavien rajapintakutsujen tekemisen määriteltyjen konfiguraatiotiedostojen avulla. Konfiguraatiotiedostot ovat tekstitiedostoja, joihin määritellään tarvittava infrastruktuuri. Konfiguraatiotiedostot toimivat Terraformin koodina ja infrastruktuurin pohjana. [3.]

Terraformin arkkitehtuuri perustuu liitännäisiin. Liitännäiset mahdollistavat Terraformin käytettävyyden laajentamisen osana infrastruktuurin hallitsemista. Liitännäisten avulla mahdollistetaan esimerkiksi kommunikointi tietyn palvelun, kuten esimerkiksi Amazon Web Servicen, kanssa infrastruktuurin hallitsemista varten. Kuvassa 1 esitetään, mistä Terraformin arkkitehtuuri muodostuu. Terraform Core toimii olennaisena osana liitännäisten lataamisessa. [3.]



Kuva 1. Terraformin arkkitehtuuri [4].

Kuvassa 1 infrastruktuurin kehittäjät (DevOps/Infra Engineer) kirjoittavat Terraformin konfiguraatiotiedostot, jotka Terraform Core lataa käynnistyessään. Terraform Core huolehtii muun muassa resurssien tilanhallinnasta, muutoksiin tarvittavien suunnitelmien luomisesta ja kommunikoinnista liitännäisten kanssa. Kuvassa 1 liitännäisiä ovat erilaiset palveluntuottajat (providers) ja palvelunasentajat (provisioners). Terraform Core löytää ja lataa tarvittavat liitännäiset automaattisesti alustusvaiheessa. [3.]

Palveluntuottajiin (providers) liittyvät liitännäiset mahdollistavat rajapintakutsujen tekemisen haluttuun palveluun, kuten esimerkiksi Google Cloud Platformiin (GCP). Lisäksi palveluntuottajiin liittyvien liitännäisten avulla autentikoidutaan infrastruktuurin tarjoajassa ja mahdollistetaan siihen liittyvien resurssien määritteleminen konfiguraatiotiedostoissa. [3.]

Palvelunasentajat (provisioners) mahdollistavat erilaisten komentojen ja skriptien suorittamisen Terraformin isäntäkoneessa tai esimerkiksi palvelimella SSH-yhteyden avulla [3].

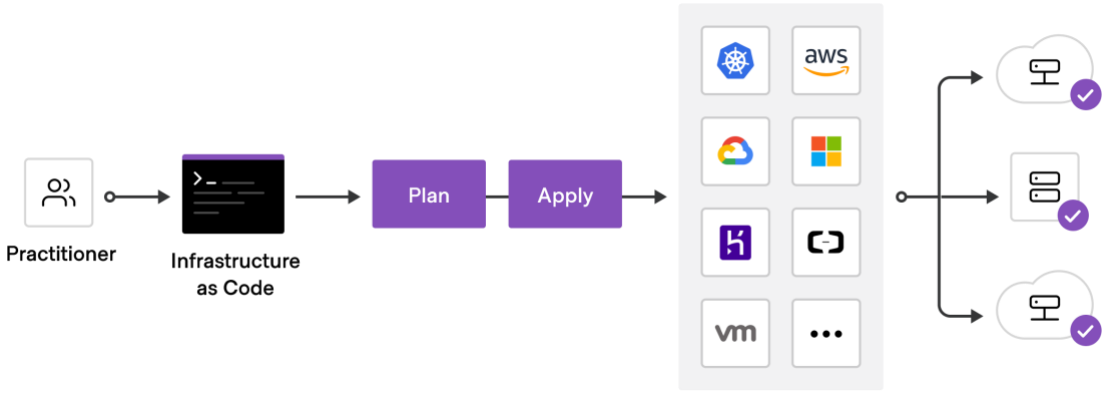
Tilatiedostossa (”.tfstate”-päätteinen tiedosto) on tieto infrastruktuurin ajantasaisesta tilasta. Terraform pystyy päättelemään infrastruktuuriin tarvittavat muutokset vertailemalla konfiguraatiotiedostojen ja tilatiedoston sisältöä keskenään. Terraform tekee tarvittavat muutokset infrastruktuuriin erilaisten rajapintakutsujen avulla.

2.3 Infrastruktuurin käyttöönottaminen Terraformin avulla

Terraformin konfigurointikieli on deklaratiivinen, mikä tarkoittaa, että kirjoitettujen konfiguraatiotiedostojen sisältö kuvaa varsinaista infrastruktuurin tilaa. Terraform päättelee automaattisesti riippuvuudet koodissa määriteltyjen resurssien osalta, jotta niiden luominen ja tuhoaminen voidaan tehdä oikeassa järjestyksessä. [5.]

Korkean tason prosessikaavio infrastruktuurin käyttöönottamisesta Terraformin avulla esitetään kuvassa 2. Infrastruktuurin käyttöönottaminen koostuu viidestä eri vaiheesta:

- suunnittelu: projektissa käytettävän infrastruktuurin hahmottaminen
- toteutus: konfiguraation kirjoittaminen infrastruktuurille
- alustus: tarvittavien liitännäisten asennus Terraformiin infrastruktuurin hallitsemista varten
- käyttöönoton suunnittelu: esikatselu infrastruktuuriin tehtävistä muutoksista
- käyttöönotto: infrastruktuurin käyttöönotto siten, kuin se oli esitetty suunnitelmassa. [6.]



Kuva 2. Prosessikaavio infrastruktuurin käyttöönottamisesta Terraformin avulla [6].

Kuvassa 2 infrastruktuurin kehittämisestä vastaava henkilö (practitioner) kirjoittaa infrastruktuurin koodina konfiguraatiotiedostoihin. Konfiguraatioiden osalta tehdään suunnitelma infrastruktuuriin tehtävistä muutoksista, jotka otetaan käyttöön kohdeympäristössä, kuten esimerkiksi Google Cloud Platformissa (GCP).

Terraform tallentaa kaikki infrastruktuuriin liittyvät tiedot tilatiedostoon. Tilatiedostosta löytyy ajantasaisin tieto infrastruktuurin tilasta, jota Terraform käyttää tarvittavien muutosten arvioimiseksi ja toteuttamiseksi. Tilatiedosto sisältää tietoja vain niiden resurssien osalta, jotka on luotu Terraformilla, eikä esimerkiksi tietoja graafisella käyttöliittymällä tehdyistä muutoksista. Terraform luo JSON-muotoisen tilatiedoston ”terraform.tfstate” automaattisesti, kun infrastruktuuriin tehdään muutoksia. [7; 8.]

2.4 Terraformin asennus

Terraformin asentaminen on mahdollista usealla eri tavalla. HashiCorp jakelee Terraformia valmiiksi käännettyinä binääreinä. Terraform-binäärin asentaminen on mahdollista tunnetuilla paketinhallintajärjestelmillä, kuten Homebrewilla macOS-järjestelmissä ja Chocolateylla Windows-järjestelmissä. Terraform-binääri voidaan asentaa myös manuaalisesti. [9.]

Manuaalisessa asennuksessa Terraformin binääri ladataan työasemalle ja se siirretään sellaiseen sijaintiin, joka soveltuu käyttöjärjestelmässä parhaiten binääreille soveltuvaksi. Binäärien tyypillisen sijainnin voi selvittää esimerkiksi tulostamalla komentokehotteessa ympäristömuuttujan ”PATH” arvon komennolla ”echo $PATH” UNIX- tai Windows-pohjaisilla käyttöjärjestelmillä. Terraformin manuaalinen asennus viimeistellään lisäämällä ”PATH”-ympäristömuuttujaan-Terraform-binäärin polku. [9.]

Terraform-binäärin muodostaminen on myös mahdollista suoraan lähdekoodista kääntämällä se Go-ohjelmointikielen kääntäjällä. Kääntämisen jälkeen binääri tulee sijoittaa soveltuvaan sijaintiin ja lisätä binäärin polku ”PATH”-ympäristömuuttujaan. [9.]

Terraformin asentamisen voi varmistaa käynnistämällä terminaali uudelleen ja suorittamalla esimerkiksi komento ”terraform -help” [9].

2.5 Hakemisto- ja tiedostorakenne

Terraformin osalta suositellaan noudattamaan hakemisto- ja tiedostorakennetta, joka mahdollistaa uudelleenkäytettävien moduulien luomisen. Terraform-projektin juurihakemistossa sijaitsevat ”main.tf”-, ”variables.tf”- ja ”outputs.tf”-tiedostot riippumatta siitä, ovatko ne tyhjiä. Tiedosto ”main.tf” toimii sisääntulopisteenä käytettäessä Terraformia, ja hyvin yksinkertaisen infrastruktuurin osalta se voi sisältää tiedot kaikista infrastruktuurissa tarvittavista resursseista. [10.]

Moduulit sijoitetaan omina hakemistoinaan ”modules”-nimiseen hakemistoon, joka sijaitsee Terraform-projektin juurihakemistossa. Kunkin moduulin hakemistorakenne on samanlainen kuin Terraform-projektin juurihakemistossa, eli siellä sijaitsevat tiedostot ”main.tf”, ”variables.tf” ja ”outputs.tf”. Moduulien avulla infrastruktuuriin luotavat resurssit voidaan kirjoittaa pienemmissä osissa sen lisäksi, että moduuleja voidaan jakaa käyttöön esimerkiksi toiseen Terraform-projektiin, jossa halutaan luoda samankaltaisia resursseja infrastruktuuriin. Terraform-projektin juurihakemistoon voidaan luoda myös

”examples”-hakemisto, joka sisältää esimerkkejä moduulien käyttämisestä ja ”README”-tiedostoja esimerkkien tavoitteista. [10.]

3 Infrastruktuurin luominen Google Cloud Platformiin Terraformin avulla

3.1 Google Cloud Platform -pilvipalvelu

Google Cloud Platform (GCP) tai Google Cloud on Googlen tarjoama pilvipalvelu, joka mahdollistaa lukuisten palveluiden hyödyntämisen osana sovelluskokonaisuutta maailmanlaajuisesti. Google Cloudiin on mahdollista luoda yli 100 erilaista palvelua riippuen kunkin projektin tarpeesta. Projektissa käytettävät palvelut voivat olla esimerkiksi virtuaalikoneita, tietokantoja, konttialustoja ja erilaisia tallennustiloja. Terraformin konfiguraatiotiedostoissa palvelut ja niihin kuuluvat määritykset käsitellään resursseina. [11.]

GCP-ympäristössä on mahdollista isännöidä palveluja useilla maantieteellisillä alueilla, jotka ovat Aasia, Australia, Eurooppa, Pohjois-Amerikka ja Etelä-Amerikka. Jokaisella alueella on vähintään kolme toisistaan erillään olevaa vyöhykettä, jotka mahdollistavat projektissa käytettävien resurssien jakamisen vyöhykkeiden välillä, mikä tekee ympäristöstä sekä vikasietoisen että paremmin loppukäyttäjien saavutettavissa olevan. [12.]

Jokainen GCP-ympäristöön luotu resurssi kuuluu johonkin projektiin. Projektin voidaan ajatella pitävän sisällään tiettyyn sovellukseen tai sovelluskokonaisuuteen liittyvät asiat. Projekti koostuu asetuksista, käyttöoikeuksista sekä metatiedoista, jotka kuvaavat sovellusta. Resurssit voivat kommunikoida kussakin projektissa sisäisen verkon avulla, kun ne sijaitsevat samalla maantieteellisellä alueella. Projektin resurssit eivät voi kommunikoida toisessa projektissa olevien resurssien kanssa, mikäli käytössä ei ole jaettua virtuaaliverkkoa. [13.]

Uutta GCP-ympäristön projektia luotaessa sille syötetään sekä nimi että tunniste. Google luo automaattisesti projektille uniikin numeron. [13.]

3.2 Google Cloud Platformin käyttäminen Terraformilla

Terraformia voidaan käyttää infrastruktuurin hallitsemiseksi GCP-ympäristössä. Jotta Terraform voi hallita infrastruktuuria, tulee ensimmäiseksi luoda GCP-projekti, johon tullaan toteuttamaan kaikki projektissa tarvittavat resurssit. Lisäksi tarvitaan konetili, jota käyttämällä Terraform saa oikeudet tehdä muutoksia GCP-projektissa. [14.]

GCP-ympäristöön liittyvät konetilit ovat erityisiä käyttäjätilejä, joita käyttävät erilaiset sovellukset. Kyseessä ei ole luonnollisen henkilön oma käyttäjätili, vaan konetili, jolle annetaan tarvittavat oikeudet käyttää pilvessä olevia palveluja erilaisten rajapintakutsujen avulla. [15.]

Konetiliä luotaessa sille asetetaan haluttu nimi ja tarvittavat muokkausoikeudet. Konetiliä varten luodaan yksityinen avain. Yksityinen avain on ladattavissa JSON-muodossa, ja sitä käyttämällä Terraform pystyy autentikoitumaan Googleen ja tekemään muutoksia projektin infrastruktuuriin käyttäen konetiliä. JSON-avain on salaisuus, eikä sitä tule jakaa ulkopuolisille. Sitä ei tule esimerkiksi viedä osaksi versionhallintaa. [14; 15.]

3.3 Konfiguraatiotiedostojen syntaksi

Terraform-kieli on suunniteltu helposti luettavaksi ja kirjoitettavaksi. Se määritellään HashiCorpin omaksi kieleksi, jota käytetään myös muissa HashiCorpin tuotteissa Terraformin lisäksi. [16.]

Terraformissa käytettävä kieli koostuu kahdesta peruselementistä: argumenteista ja lohkoista. Argumenteista löytyvät sekä nimi että sitä vastaava arvo. Esimerkkikoodissa 1 argumentille ”image_id” asetetaan arvo ”abc123”. [17.]

```
image_id = ”abc123”
```

Esimerkkikoodi 1. Tunniste ennen yhtäsuuruusmerkkiä tarkoittaa argumentin nimeä ja lauseke yhtäsuuruusmerkin jälkeen ilmaisee argumentin arvoa [17].

Argumenttien osalta on sovittu, miten siinä olevat arvot tulee esittää. Esimerkiksi infrastruktuurin resurssien osalta argumenttien arvot noudattavat tiettyä skeemaa. Argumenttien arvot annetaan tietyissä muodoissa, kuten merkkijonoina, numeroina tai totuusarvoina. [17.]

Lohko toimii sisällön säilönä. Jokaisella lohkolla on tyyppi. Esimerkkikoodissa 2 lohkon tyyppi on resurssi. [17.]

```
resource ”aws_instance” ”example” {
  ami = “abc123”

  network_interface {
    # …
  }
}
```

Esimerkkikoodi 2. Resurssilohko sisältöineen [17].

Lohkolla voi olla tietty määrä nimikkeitä, jotka seuraavat lohkon tyypin jälkeen. Esimerkkikoodi 2:ssa resurssilohkon nimikkeitä ovat ”aws_instance” ja ”example”. Ensimmäisellä nimikkeellä ”aws_instance” ilmaistaan, että Amazon Web Serviceen luodaan virtuaalikone. Toisessa nimikkeessä ”example” määritellään resurssille nimi, jota voidaan käyttää, kun kyseiseen resurssiin viitataan esimerkiksi jossakin toisessa resurssissa. Nimikkeiden lukumäärä riippuu lohkon tyypistä. Esimerkkikoodissa 2 resurssilohkon sisällä olevalla ”network_interface” -lohkolla ei ole lainkaan nimikkeitä. Lohkon tyypin ja mahdollisten nimikkeiden jälkeen seuraavat sekä avautuva että sulkeutuva aaltosulje. Aaltosulkeiden sisälle määritellään argumentit ja mahdolliset sisäkkäiset lohkot. [17.]

3.4 Konfiguraation kirjoittaminen

Infrastruktuurin konfiguraatiotiedostoja varten luodaan kehittäjän valitseman nimen mukainen hakemisto, joka toimii samalla Terraform-projektina. Valittuun hakemistoon luodaan ”main.tf”-tiedosto, joka toimii Terraformin pääsisääntulotiedostona. Terraform lataa työskentelyhakemiston osalta kaikki ”.tf”- tai ”.tf.json”-päätteiset tiedostot käynnistyessään. [18.]

Esimerkkikoodissa 3 tiedostoon ”main.tf” määritellään ”terraform”-, ”provider”- ja ”resource”-lohkot.

```
terraform {
  required_providers {
    google = {
      source = “hashicorp/google”
      version = “4.51.0”
    }
  }
}

provider “google” {
  credentials = file(“<NAME>.json”)

  project = “<PROJECT_ID>”
  region  = “us-central1”
  zone    = “us-central1-c”
}

resource “google_compute_network” “vpc_network” {
  name = “terraform-network”
}
```

Esimerkkikoodi 3. Lohkot ”terraform”, ”provider” ja ”resource” [18].

Esimerkkikoodin 3 ”terraform”-lohko sisältää palveluntuottajiin liittyvien asetusten tiedot. Terraform-lohkossa kerrotaan, mistä Google-palveluntuottaja löytyy (source) ja mikä versio siitä ladataan käyttöön (version). Terraform lataa määriteltyihin palveluntuottajiin liittyvät tiedot infrastruktuurin hallitsemista varten alustuksen yhteydessä. Palveluntuottajiin liittyvät tiedot ladataan oletuksena Terraformin omasta rekisteristä. Esimerkkikoodissa 3 Google-palveluntuottajan tiedot ladataan tarkalleen osoitteesta ”registry.terraform.io/hashicorp/google”. Versioon liittyvä tieto ei ole pakollinen, mutta HashiCorp suosittelee sen asetettavaksi, jotta kirjoitettu konfiguraatio olisi ajantasainen palveluntuottajaan liittyvän version kanssa. Mikäli versiota ei ole määritelty, Terraform lataa automaattisesti uusimman version palveluntuottajan liitännäisestä. [18.]

Esimerkkikoodissa 3 palveluntuottajalohkoon, eli ”provider”-lohkoon, määritetään GCP-ympäristön tiedot. Argumenttiin ”credentials” määritellään sen JSON-tiedoston sijainti, joka sisältää tiedot aikaisemmin luodun GCP-konetilin yksityisestä avaimesta. Lisäksi ”project”-argumenttiin täydennetään projektia vastaava tunniste. Infrastruktuurin oletusarvoinen maantieteellinen sijainti

(region) on kehittäjän päätettävissä. Oletusarvoista vyöhykettä (zone) ei ole pakko määrittää. [18.]

Esimerkkikoodin 3 resurssilohko koostuu kahdesta nimikkeestä ennen aaltosulkeita: luotavan resurssin tyypistä ja resurssin nimestä. Resurssin nimi on konfiguraation kirjoittajan päätettävissä. Esimerkkikoodissa 3 määritellyn resurssin tyyppi on ”google_compute_network”, jonka nimeksi on annettu ”vpc_network”. Resurssin tyyppi ja nimi muodostavat yhdessä uniikin tunnisteen, joka on muotoa ” google_compute_network.vpc_network”. Tunnistetta voidaan käyttää, kun kyseiseen resurssiin viitataan esimerkiksi jossakin toisessa resurssilohkossa. [18.]

Resurssilohkon sisälle määritellään kulloinkin konfiguroitavan infrastruktuuriresurssin asetukset argumentteina. Esimerkkikoodissa 3 luotavan resurssin osalta argumentiksi annetaan virtuaaliverkon nimi (”name”). Resurssityypin osalta määriteltävät pakolliset ja valinnaiset argumentit löytyvät kunkin palveluntuottajan dokumentaatiosta. GCP-palveluntuottajan osalta ajantasainen dokumentaatio löytyy verkkolähteestä <https://registry.terraform.io/providers/hashicorp/google/latest/docs>. Dokumentaatiosta löytyvät esimerkiksi ohjeet ”google_compute_network” -resurssin määrittelemisestä konfiguraatioon. [18.]

3.5 Konfiguraatioon liittyvä alustus, muotoilu ja validointi

Uusia konfiguraatiotiedostoja käsitellessä työskentelyhakemistossa tulee suorittaa komento ”terraform init” tietojen alustamista varten. Alustuksessa ladataan tarvittavat liitännäiset konfiguraatioon aikaisemmin määriteltyjen ”provider”- ja ”terraform”-lohkojen perusteella. Alustuksen yhteydessä testataan yhteys infrastruktuurin palveluntuottajaan, ja sen yhteydessä voidaan myös määritellä paikka tilatiedoston sijainnille. Onnistuneen alustuksen jälkeen Terraformilla voi tehdä muutoksia kohdeympäristöön, kuten esimerkiksi Google Cloud Platformiin.

Tarvittavien palveluntuottajaliitännäiset asennetaan piilotettuun alihakemistoon ”.terraform”. Tämän lisäksi Terraform luo piilotetun tiedoston ”.terraform.lock.hcl”, jossa sijaitsevat tarkat tiedot palveluntuottajien käytettävistä versioista. Mikäli versioon liittyvä tieto puuttuisi, Terraformilla tehtävät muutokset eivät välttämättä tapahtuisi yhdenmukaisesti suoritusten aikana. [19.]

Infrastruktuuria varten kirjoitettu konfiguraatio voidaan muotoilla ja validoida kahdella komentokehotteella ajettavalla komennolla. Komento ”terraform fmt” tarkistaa konfiguraatiotiedostojen tyylillisen muotoilun ja tekee konfiguraatiotiedostoista keskenään yhtenäisiä muotoiluiltaan. Komento palauttaa niiden tiedostojen nimet, jotka on muotoiltu uudelleen. Jos uudelleenmuotoiltavia tiedostoja ei ole, komento ei palauta mitään. [20.]

Konfiguraatiotiedostojen syntaksin oikeellisuus voidaan tarkistaa komennolla ”terraform validate”. Mikäli konfiguraatio on syntaksiltaan oikein kirjoitettu, komento palauttaa tiedon validoinnin onnistumisesta. [20.]

3.6 Infrastruktuurin luominen GCP-ympäristöön Terraformin avulla

Kun infrastruktuuria varten luotu konfiguraatio on kirjoitettu oikein ja Terraform on alustettu, infrastruktuurin luominen voidaan aloittaa. Ajettaessa komento ”terraform apply” Terraform luo suunnitelman suoritettavista toimenpiteistä ja näyttää ne käyttäjälle. Jos käytettäisiin esimerkkikoodi 3:n mukaista konfiguraatiota, loisi Terraform suunnitelman virtuaaliverkon luomiseksi GCP-ympäristöön olettaen, ettei sellaista vielä ole. Komennon “terraform apply” tulostus on esimerkkikoodin 4 mukainen. [21.]

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_network.vpc_network will be created
  + resource "google_compute_network" "vpc_network" {
      + auto_create_subnetworks         = true
      + delete_default_routes_on_create = false
      + gateway_ipv4                    = (known after apply)
      + id                              = (known after apply)
      + ipv4_range                      = (known after apply)
      + name                            = "terraform-network"
      + project                         = (known after apply)
      + routing_mode                    = (known after apply)
      + self_link                       = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

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

  Enter a value:
```

Esimerkkikoodi 4. Terraform-komennon ”terraform apply” tuloste [21].

Esimerkkikoodista 4 nähdään, että Terraform loisi virtuaaliverkon GCP-ympäristöön. Muutoksen toteuttaminen vahvistetaan vastaamalla komentokehotteessa myönteisesti (”yes”) kysymykseen toimenpiteiden toteuttamisesta. Tulostuksessa olevat plusmerkit osoittavat luotavat resurssit. Maininnat ”known after apply” kertovat, mitkä tiedot selviävät konfiguraation käyttöönottamisen jälkeen. Terraform kertoo käyttäjälle interaktiivisesti infrastruktuurin luomisen tilasta, mikäli muutokset päätetään toteuttaa GCP-ympäristöön. Esimerkkikoodissa 5 Terraform luo konfiguraatiota vastaavan infrastruktuurin ja kehittäjä pystyy seuraamaan muutosten edistymistä reaaliaikaisesti. [21.]

```
  Enter a value: yes

google_compute_network.vpc_network: Creating...
google_compute_network.vpc_network: Still creating... [10s elapsed]
google_compute_network.vpc_network: Still creating... [20s elapsed]
google_compute_network.vpc_network: Still creating... [30s elapsed]
google_compute_network.vpc_network: Creation complete after 38s
[id=projects/testing-project/global/networks/terraform-network]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Esimerkkikoodi 5. Terraform luo infrastruktuurin konfiguraatiotiedostojen pohjalta, kun muutoksen toteuttamiseen vastataan myönteisesti [21].

Terraform kertoo lopuksi muutosten käyttöönoton tilan. Esimerkkikoodin 5 lopussa tuloste kertoo, että muutokset on otettu käyttöön onnistuneesti yhden resurssin lisäämisellä.

3.7 Muutosten tekeminen Terraformin hallitsemaan infrastruktuuriin

Infrastruktuuriin tehdään muutoksia koko sen elinkaaren ajan. Kun infrastruktuurin konfiguraatiota muutetaan, Terraform toteuttaa muutokset ainoastaan tarvittavilta osin, jotta olemassa olevien konfiguraatioiden mukaiseen tilaan päästään. [22.]

Infrastruktuuriin voidaan tehdä muutoksia joko muuttamalla olemassa olevia resursseja tai poistamalla ja lisäämällä niitä. Konfiguraatioon tehtyjen muutosten jälkeen ne saadaan voimaan ajamalla komento ”terraform apply” ja vastaamalla ”yes” muutosten toteuttamiseksi. [23.]

Terraform näyttää lisättävät resurssit plusmerkeillä, muutettavat resurssit tildemerkillä ja poistettavat resurssit miinusmerkeillä. Esimerkkikoodissa 6 virtuaalikoneen tietoihin lisätään tageja. [23.]

```
Terraform will perform the following actions:

  # google_compute_instance.vm_instance will be updated in-place
  ~ resource "google_compute_instance" "vm_instance" {
        id                   = "projects/testing-project/zones/us-
central1-c/instances/terraform-instance"
        name                 = "terraform-instance"
      ~ tags                 = [
          + "dev",
          + "web",
        ]
        # (15 unchanged attributes hidden)



        # (3 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
```

Esimerkkikoodi 6. Konfiguraatioon muutoksia tekevän ”terraform apply” -komennon tulostus [23].

Tietyt muutokset voivat aiheuttaa resurssin tuhoamisen ja uudelleenluomisen. Esimerkki tällaisesta muutoksesta on virtuaalikoneen käyttöjärjestelmän käyttämän näennäistiedoston muuttaminen. Ajamalla komento ”terraform apply” voidaan konsolin tulosteesta päätellä resurssin uudelleenluonti sen edessä olevasta -/+-merkistä. [24.]

Terraformin hallitsema infrastruktuuri voidaan tuhota ajamalla komentokehotteella komento ”terraform destroy”. Tämä komento tuhoaa kaikki Terraformin hallitsemat resurssit infrastruktuurista. Jos esimerkiksi GCP-ympäristöön on tehty Terraformin lisäksi muutoksia käyttöliittymän tai komentokehotteen avulla, Terraform ei näe kyseisiä muutoksia, koska ne eivät ole Terraformin hallitsemia. [25.]

Terraform tuhoaa infrastruktuurin noudattaen tarkkaa järjestystä. Esimerkiksi virtuaalikoneeseen liittyvää VPC-verkkoa ei voida poistaa, ennen kuin virtuaalikone on poistettu. Tämä johtuu siitä, että virtuaalikone on riippuvainen VPC-verkosta. Terraform päättelee aina oikean järjestyksen resursseja luodessaan ja niitä poistaessaan. [25.]

3.8 Terraform-resursseihin liittyvän tiedon hakeminen

Attribuutit ovat Terraformissa arvoja, joita voidaan hakea esimerkiksi infrastruktuuriin luoduista resursseista. Attribuutteja hyödynnetään erityisesti Terraformin tulosteissa ja resursseissa, joissa on tarve viitata toisessa resurssissa muodostuneeseen arvoon, joka on haettavissa attribuuttina. Esimerkiksi GCP-ympäristön eri resurssien muodostamat attribuutit voi löytää palveluntuottajakohtaisesta dokumentaatiosta HashiCorpin verkkosivuilta.

Terraform tallentaa tiedon olemassa olevien resurssien attribuuttien arvoista. Attribuuttina voi toimia esimerkiksi virtuaalikoneen IP-osoite sen luomisen jälkeen. Attribuutteja voi olla infrastruktuurin koon mukaan sadoista jopa tuhansiin. Yleensä kehittäjät ovat kiinnostuneita vain tiettyjen attribuuttien arvoista. Työskentelyhakemiston ”outputs.tf”-tiedostoon määritellään ne attribuutit, joiden arvoista ollaan kiinnostuneita. Kehittäjää kiinnostavien attribuuttien arvot saadaan selville ajamalla komento ”terraform output”. Ehtona on kuitenkin se, että Terraformilla on luotu aikaisemmin infrastruktuuri ”terraform apply” -komennolla. Tämä komento tulostaa myös lopuksi ”outputs.tf”-tiedostoon määriteltyjen attribuuttien arvot. Esimerkkikoodissa 7 määritellään ”output”-lohko, jonka nimikkeeksi on asetettu ”ip”. Argumentissa ”value” viitataan konfiguraatiossa olevaan virtuaalikoneresurssiin, josta voidaan hakea virtuaalikoneen verkkosovittimen IP-osoite attribuutin ”network_interface.0.network_ip” avulla. [26; 27.]

```
output "ip" {
  value =
google_compute_instance.vm_instance.network_interface.0.network_ip
}
```

Esimerkkikoodi 7. Terraform-resurssien attribuuttien valinta tulostusta varten ”outputs.tf”-tiedostossa [27].

Kunkin resurssin tulostettavissa olevat attribuutit löytyvät resurssiin liittyvästä dokumentaatiosta, joka on aina palveluntuottajakohtaista. Esimerkkikoodissa 8 esitetään ”terraform output” -komennon tuloste, kun ”outputs.tf”-tiedosto on määritelty esimerkkikoodin 7 mukaisesti. Lohkoon ”output” määritelty nimike ”ip” kertoo, mistä tulostuksesta on kyse. [27; 28.]

```
ip = "10.128.0.3"
```

Esimerkkikoodi 8. ”terraform output” -komennon tulostus [28].

Esimerkkikoodin 8 mukainen tuloste saadaan aikaiseksi ajamalla ”terraform apply”- tai ”terraform output” -komento. Komennolla ”terraform output” tiedostoon ”outputs.tf” määritellyt tulosteet saadaan näkyviin sen jälkeen, kun infrastruktuuriresurssit ovat luotu ”terraform apply” -komennolla.

3.9 Tilatietojen tallentaminen Cloud Storageen

Terraformiin liittyvät tilatiedot tallentuvat lähtökohtaisesti paikallisesti Terraformin työskentelyhakemistossa olevaan ”terraform.tfstate”-tiedostoon. Eri kehittäjien on haastavaa tehdä muutoksia infrastruktuuriin, mikäli tilatiedostoa ei ole synkronoitu ulkoiseen tallennuspaikkaan. Tilatiedostossa sijaitsee ajantasainen tieto Terraformilla hallitusta infrastruktuurista. GCP-ympäristössä ongelma voidaan ratkaista ottamalla käyttöön tilatietojen etätallennus. Tämän ansiosta infrastruktuuria voidaan muuttaa useiden kehittäjien toimesta niin, että ajantasaisin tieto infrastruktuurin tilasta on aina saatavilla. [29.]

Tilatietojen etätallennus saadaan käyttöön luomalla Google Cloud Storage -palveluun lokero (bucket) globaalisti ainutkertaisella nimellä. Versioinnin käyttöönottaminen Cloud Storage -lokeron luomisen yhteydessä mahdollistaa eri Terraform-käyttöönottojen aikaisten tilatietojen tallentamisen ja estää inhimillisten virheiden tapahtumisen, kuten tilatiedoston poistamisen vahingossa. Kun Cloud Storage -lokero on luotu, se voidaan ottaa käyttöön Terraformin tilatietojen tallennuspaikkana lisäämällä Cloud Storage -lokeroon liittyvät tiedot esimerkiksi uudessa ”backend.tf”-tiedostossa. Tämä tiedosto sijaitsee Terraform-projektin juurihakemistossa. Esimerkkikoodissa 9 määritellään Google Cloud Storage Terraformin tilatietojen tallennuspaikaksi. Lohkossa ”terraform” sijaitsee ”gcs”-nimikkeellä oleva ”backend”-lohko. Argumenttiin ”bucket” määritellään aikaisemmin luodun Cloud Storage -lokeron nimi ja ”prefix”-argumenttiin hakemisto, jonne tilatiedosto sijoitetaan lokerossa. Argumentti ”prefix” on vapaavalintainen. [29.]

```
terraform {
 backend "gcs" {
   bucket  = "BUCKET_NAME"
   prefix  = "terraform/state"
 }
}
```

Esimerkkikoodi 9. ”backend.tf”-tiedoston sisältö [29].

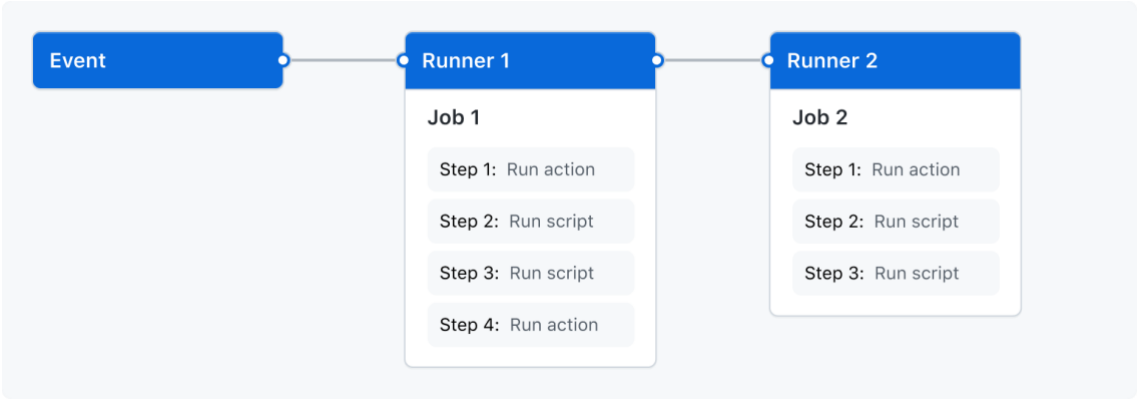
Cloud Storage -rajapinta saadaan käyttöön ajamalla komento ”gcloud services enable storage.googleapis.com” Cloud Shell -terminaalissa tai ottamalla se käyttöön internetelaimella Google-projektin konsolissa. Terraform ei pysty lukemaan tai kirjoittamaan muutoksia Cloud Storageen ilman tätä rajapintaa. Tilatiedon etätallennuksen käyttöönotto viimeistellään ajamalla työskentelyhakemistossa komento ”terraform init”. Terraform pyytää paikallisen tilatiedoston kopioimista uuteen Cloud Storage -lokeroon, mikäli muutoksia on tehty infrastruktuuriin aikaisemmin paikallisesti. [29.]

4 Infrastruktuuriin tehtävien muutosten automatisointi

4.1 GitHub Actions -alusta

GitHub Actions on jatkuvan integraation ja jatkuvan toimituksen alusta, joka tarjoaa mahdollisuudet sovelluksen kääntämiselle, testaamiselle ja automaattiselle käyttöönottamiselle. GitHub Actionsin avulla on mahdollista luoda työnkulkuja, jotka esimerkiksi kääntävät ja testaavat sovelluksen lähdekoodin tapahtuman jälkeen. Tapahtumana voi toimia esimerkiksi pyyntö kehityshaaran yhdistämisestä päähaaraan. Myös uudet muutokset tietyssä kehityshaarassa voivat määritysten mukaisesti aiheuttaa tapahtuman toteutumisen ja työnkulun käynnistymisen. GitHub Actionsin avulla on mahdollista ajaa työnkulkuja GitHubin isännöimien Linux-, Windows- ja macOS-virtuaalikoneiden avulla. Työnkulkuja voi ajaa myös omilla palvelimilla, jotka on isännöity joko omissa liiketiloissa tai pilviympäristöissä. [30.]

GitHub Actions koostuu viidestä komponentista, jotka ovat työnkulut, tapahtumat, työt, työvaiheet ja suorittajat. Kuvassa 3 esitetään korkealla tasolla eräs GitHub Actions -työnkulku, josta käyvät ilmi komponentit. [31.]



Kuva 3. Esimerkkikuvaus GitHub Actionsin työnkulusta [31].

Kuvassa 3 kuvattu työnkulku käynnistyy jonkin tapahtuman pohjalta. Työnkulussa on kaksi työtä. Ensimmäisessä työssä (Job 1) on neljä eri työvaihetta, kun taas toisessa työssä (Job 2) on kolme eri työvaihetta. Työt ajetaan eri suorittajissa.

Työnkulut määritellään omiin YAML-tiedostoihin, jotka sijaitsevat hakemistossa ”.github/workflows”. Työnkulkuja voi olla yksi tai useampia. Työnkulku on määriteltävissä oleva prosessi, jossa suoritetaan yksi tai useampi työ. Työnkulku on mahdollista suorittaa tietyn tapahtuman seurauksena, manuaalisesti tai ajoitetusti. Tapahtumat voivat toimivat herätteinä työnkulkujen käynnistämiselle. Tapahtuma voi olla lähtöisin esimerkiksi siitä, että joku kehittäjistä avaa kehityshaaran yhdistämispyynnön (pull request), luo ongelman tai vie uusia muutoksia GitHubissa olevaan lähdekoodien kuvauskantaan. Työnkulku voidaan myös käynnistää kutsumalla REST-rajapintaa POST-tyyppisellä kutsulla. [31.]

Suorittajat ovat palvelimia, jotka ajavat työnkulkuja tapahtumien pohjalta. GitHub tarjoaa käyttäjille virtuaalisia Ubuntu Linux-, Microsoft Windows- ja macOS-suorittajia. Työnkulut ajetaan alustetuissa virtuaalikoneissa, mikä mahdollistaa sen, että työnkulku ajetaan jokaisella suorituskerralla samalla tavalla. GitHub tarjoaa käyttöön myös tehokkaampia virtuaalikoneita ja mahdollistaa omien palvelinten käyttämisen suorittajina, mikäli laitteistokohtaiset asetukset vaativat sitä. [31.]

Työnkulussa on yksi tai useampi työ, jotka suoritetaan oletuksena rinnakkain. Mikäli työ on riippuvainen jostakin toisesta työstä, se voidaan merkitä osaksi työnkulkua. Tämän avulla voidaan varmistaa, ettei työtä suoriteta, ennen kuin riippuvainen työ on suoritettu onnistuneesti loppuun. Esimerkki riippuvaisesta työstä voisi olla esimerkiksi paketointi. Sovelluksen arkkitehtuurisesti erilaiset komponentit voidaan kääntää rinnakkain, mutta ennen kuin ne voidaan paketoida, tulee kaikkien komponenttien olla käännettyinä. [31.]

Työ koostuu erilaisista työvaiheista, joissa määritellään suoritettavat komennot tai toimenpiteet. Jokaisen työn työvaihe ajetaan samassa suorittajassa. Tämä mahdollistaa sen, että työn eri vaiheissa syntyvät tiedot ovat käytettävissä myös seuraavissa työvaiheissa. Työn eri vaiheissa voidaan esimerkiksi hakea lähdekoodit GitHubin kuvauskannasta, asentaa Node.js ja suorittaa testit. [31.]

Toimenpiteellä tarkoitetaan GitHub Actions -alustalle tarkoitettua mukautettua sovellusta, joka suorittaa sekä monimutkaisen että usein toistuvan tehtävän. Toimenpiteiden avulla voidaan yksinkertaistaa työnkulkuun tarkoitettua YAML-tiedostoa vähentämällä työnkulun määrittelyssä käytettävää toistoa. Toimenpiteinä voivat toimia esimerkiksi lähdekoodien hakeminen GitHubin kuvauskannasta ja Terraformin asentaminen. Valmiita toimenpiteitä löytyy GitHub Marketplacesta, ja niitä voi luoda myös itse. [31.]

4.2 Työnkulut GitHub Actionsissa

Työnkulkujen määrittämiseen käytetään YAML-syntaksia. Työnkulut luodaan projektin juurihakemiston kansioon ”.github/workflows”. Tässä kansiossa voi olla yksi tai useampi työnkulku. [29.]

Kansioon voidaan luoda esimerkiksi uusi tiedosto nimeltä ” learn-github-actions.yml”, joka on sisällöltään esimerkkikoodin 10 mukainen.

```
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

Esimerkkikoodi 10. GitHub Actionsin työnkulku määriteltynä YAML-tiedostoon [29].

Tarkastellaan esimerkkikoodia 10, johon on määritelty GitHub Actionsin työnkulku. Tiedoston alussa on määritelty työnkulun nimi. Tiedoston toiselta riviltä löytyy vapaavalintainen suoritusaikainen nimi, jota käytetään ajon yhteydessä. Kolmannelta riviltä löytyvät tapahtumat, jotka aiheuttavat työnkulun käynnistymisen. Tässä esimerkissä työnkulku käynnistetään aina, kun GitHub kuvauskantaan viedään lähdekoodimuutoksia. Neljännestä rivistä lähtien löytyy työnkulun töihin liittyvää tietoa. Esimerkkikoodiin 10 on määritelty yksi työ, jossa on neljä vaihetta. [30.]

Työn alussa on määritelty työn nimi ”check-bats-version”. Työn eri vaiheet suoritetaan Ubuntu Linux -virtuaalikoneessa. Työn ensimmäisessä vaiheessa GitHubin kuvauskannasta haetaan ajantasaiset lähdekoodit hyödyntäen toimenpidettä ”actions/checkout@v3”. Toisessa vaiheessa käytetään toimenpidettä ”actions/setup-node@v3”, jolla asennetaan Node.js-ajoympäristön versio 14. Kolmannessa työvaiheessa asennetaan Noden paketinhallintajärjestelmä NPM:llä globaaliriippuvuus ”bats”. Viimeisessä työvaiheessa tulostetaan ”bats”-paketin versio. [30.]

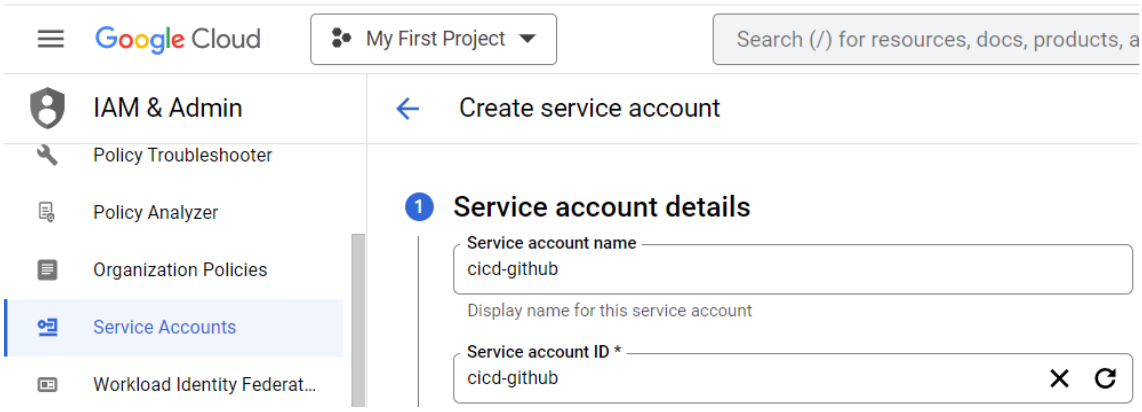
Työnkulun tarkasteleminen onnistuu GitHubin välilehdellä ”Actions”. Välilehdeltä löytyvät kaikki projektiin määritellyt työnkulut. Työnkulkua on mahdollista seurata valitsemalla tarkasteltava työnkulku sivupalkista. Sivulta löytyvät parhaillaan käynnissä olevat työnkulut ja aikaisemmin suoritetut työnkulut. Sivulta löytyy muun muassa tieto siitä, onko työnkulku suoritettu onnistuneesti

vai epäonnistuneesti. Yksityiskohtaiset tiedot tietyn työnkulun suoriutumisesta löytyvät valitsemalla tarkasteltava ajokerta. [31.]

4.3 Terraformin automatisointi GitHub Actionsilla

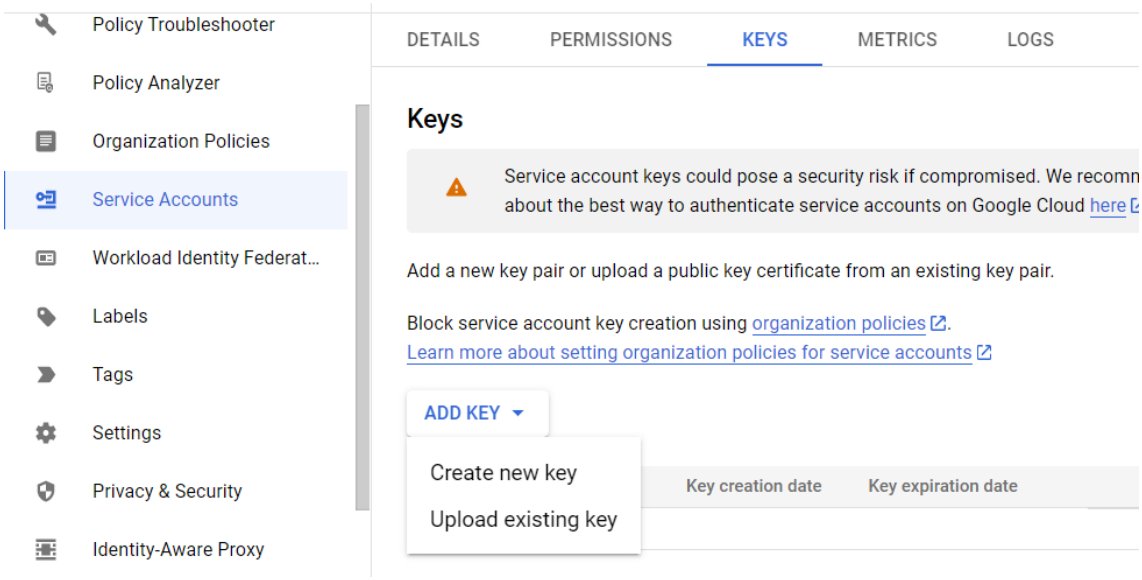
GitHub Actionsin avulla on mahdollista automatisoida sovelluksen jatkuva integraatio ja jatkuva käyttöönotto. Sovellus voidaan kääntää, testata ja ottaa käyttöön valittuun ympäristöön, kuten Google Cloud Platformiin (GCP). GitHub Actionsin avulla voidaan myös automatisoida infrastruktuurin käyttöönotto Terraformilla. [32.]

Tarkastellaan autentikoitumista GCP-ympäristöön Googlen konetilillä. Jotta Terraform voi luoda GCP-ympäristöön infrastruktuuriin, sitä varten tulee luoda konetili GCP:n identiteetin- ja pääsynhallinta -näkymässä. Kuvassa 4 esitellään, miltä GCP:n selainkonsolissa oleva konetilin luontinäkymä näyttää identiteetin- ja pääsynhallinta -osiossa. [32.]



Kuva 4. Konetilin luominen identiteetin- ja pääsynhallinta -näkymässä [32].

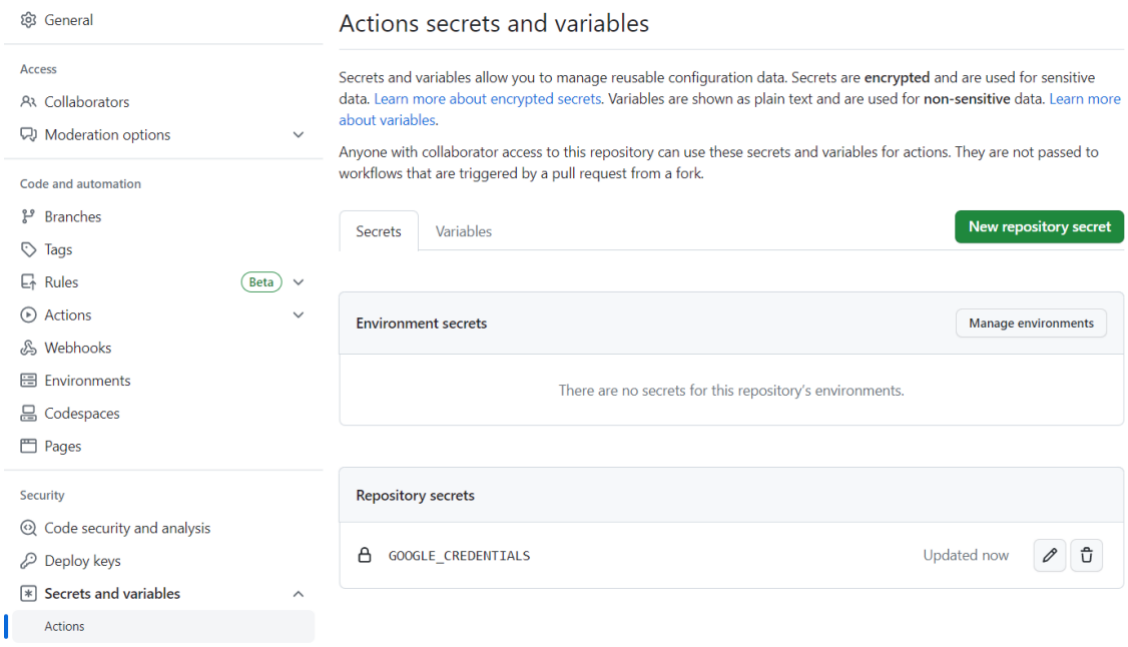
Konetilille annetaan käyttäjän valinnan mukainen nimi. Kun konetili on luotu, sen jälkeen muodostetaan ja ladataan kyseistä tiliä vastaava JSON-muotoinen avain, jota Terraform käyttää autentikoituessaan GCP-ympäristöön. Kuvassa 5 aikaisemmin luodulle konetilille luodaan avain ”keys”-osiossa. [33.]



Kuva 5. Konetilin avaimen luonti GCP:n selainkonsolissa [33].

Lopuksi konetilille annetaan muokkausoikeudet, jotta se voi tehdä muutoksia infrastruktuuriin. Terraform-projektin juurihakemiston johonkin konfiguraatiotiedostoon, kuten ”main.tf”-tiedostoon, lisätään ”provider”-lohko ”google”-nimikkeellä, ja siihen määritellään argumentti ”credentials”, jossa kerrotaan JSON-avaintiedoston sijainti. [33.]

Luotu JSON-avain tulee viedä osaksi GitHub Actionsin työnkulkua niin, ettei se vuoda ulkopuolisille. GitHubin lähdekoodien kuvauskannan asetuksissa luodaan uusi salaisuus, johon asetetaan arvoksi JSON-avaimen sisältö. Salaisuuden nimeksi voidaan asettaa esimerkiksi ”GOOGLE_CREDENTIALS”. Kuvassa 6 on GitHubin lähdekoodien kuvauskannan asetuksien näkymä salaisuuksien ja muuttujien kohdalta. Kuvauskantaan on luotu uusi salaisuus ”GOOGLE_CREDENTIALS”, jonka sisältönä on Google-konetilin JSON-avaimen sisältö.



Kuva 6. GitHubin kuvauskantaan on luotu salaisuus nimeltään ”GOOGLE_CREDENTIALS” [33].

Google Cloud -projektiin liittyvät tiedot on myös hyvä viedä osaksi työnkulkua salaisuuksina. Esimerkiksi Terraform-projektin juurihakemiston ”main.tf”-tiedoston ”provider”-lohkoon voidaan kirjoittaa ”project”- ja ”region”-argumenttien arvot siten, että ne luetaan muuttujista. Tätä tilannetta kuvataan esimerkkikoodissa 11.

```
provider "google" {
  project = var.project
  region = var.region
}
```

Esimerkkikoodi 11. ”provider”-lohkon sisällä oleviin argumentteihin ”project” ja ”region” haetaan arvot muuttujista [33].

Muuttujat on määritelty samassa työskentelyhakemistossa olevaan ”variables.tf”-tiedostoon. Muuttujien määritelmät on esitetty esimerkkikoodissa 12.

```
variable "project" {
    type= string
    description = "ID Google project"
}

variable "region" {
    type= string
    description = "Region Google project"
}
```

Esimerkkikoodi 12. ”variables.tf”-tiedoston sisältöä [33].

Esimerkkikoodissa 12 muuttujaan ”project” on mahdollista asettaa merkkijono. Kuvauksen mukaisesti muuttujaan säilötään Google Cloud -projektin tunnus. Vastaavasti ”region”-muuttujaan säilötään haluttu maantieteellinen alue merkkijonona. Muuttujille ”project” ja ”region” ei ole asetettu oletusarvoja esimerkkikoodissa 12. Arvot asetettaisiin lisäämällä ”variable”-lohkojen sisälle ”default”-argumentit, joihin asetetaan muuttujia kuvaavat arvot. Koska ”project”- ja ”region”-muuttujat halutaan asettaa salaisuuksina, niitä varten luodaan uusi ”.tfvars”-päätteinen tiedosto GitHub Actions -työnkulun työvaiheessa ennen Terraformin alustamista. GitHubin kuvauskantaan luodaan salaisuuksina Google Cloud -projektin tunnus ja maantieteellinen alue. Tämän jälkeen työnkulun työvaiheessa luodaan ”.tfvars”-päätteinen tiedosto, jossa Terraformissa käytettävien ”project”- ja ”region”-muuttujien arvot asetetaan viittaamalla GitHubissa asetettuihin salaisuuksiin. Tiedot voisivat olla työvaiheessa myös ympäristömuuttujina, mutta silloin tieto olisi lähtökohtaisesti avointa. Esimerkkikoodissa 13 kuvataan GitHub Actions -työnkulun työvaihe, jossa luodaan ”.tfvars”-tiedosto salaisuuksilla. [33.]

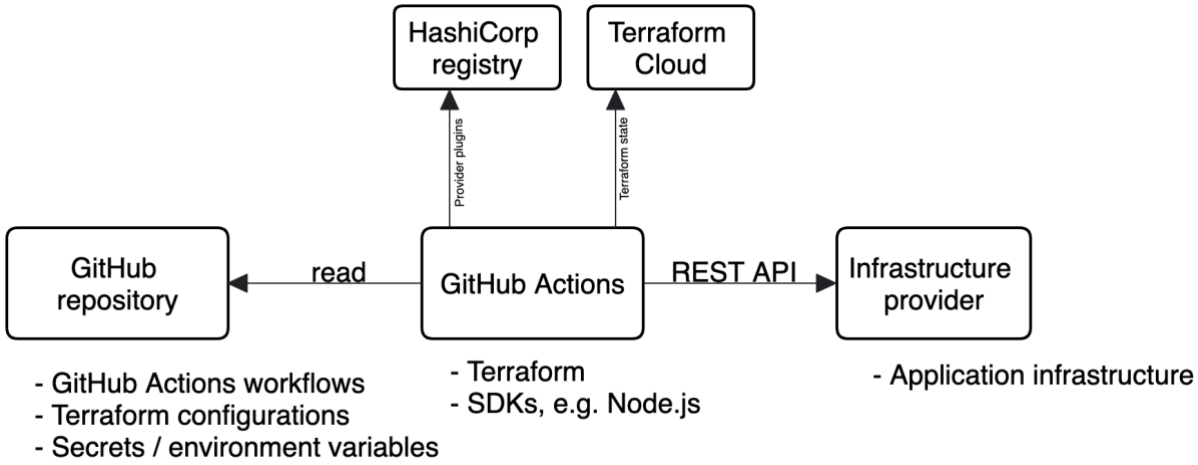
```
- name: Setup terraform variables
      id: vars
      run: |-
        cat > pipeline.auto.tfvars <<EOF
        region="${{ secrets.GCP_REGION }}"
        project="${{ secrets.GCP_PROJECT }}"
        EOF
```

Esimerkkikoodi 13. GitHub Actions -työnkulun työvaiheessa luodaan ”pipeline.auto.tfvars”-tiedosto, jossa määritellään muuttujien ”project” ja ”region” arvot siten, että ne tulevat GitHubissa luoduista salaisuuksista [33].

Terraform huolehtii ”.tfvars”-tiedostojen lataamisesta automaattisesti käynnistyessään, mikäli tiedostojen nimien loppuosa on muotoa ”.auto.tfvars”.

Näihin tiedostoihin asetetut muuttujien arvot otetaan käyttöön Terraformin käynnistymisen yhteydessä. [33.]

Esimerkkikoodissa 14 tarkastellaan yhtä mahdollista GitHub Actions -työnkulkua, jossa hyödynnetään Terraformia. Kuvassa 7 esitellään työnkulun arkkitehtuurikuva ennen varsinaista työnkulun konfiguraatiota sen seuraamisen helpottamiseksi.



Kuva 7. Esimerkkikoodin 14 mukaisen GitHub Actions -työnkulun arkkitehtuuri.

Kuvassa 7 GitHubin kuvauskannassa (repository) sijaitsevat Terraformin konfiguraatiotiedostot ja GitHub Actions -työnkulkujen konfiguraatiotiedostot. GitHub Actions -palvelu lukee työnkulkujen konfiguraatiot kuvauskannasta ja päättelee konfiguraatioihin määriteltyjen tapahtumien pohjalta työnkulkujen käynnistymisen. Esimerkkikoodissa 14 työnkulku käynnistyy, kun ”main”-haaraan viedään lähdekoodimuutoksia tai GitHubiin avataan yhdistämispyyntö (pull request) ”main”-haaraan.

GitHub Actions lukee kuvassa 7 kuvauskannasta työnkulkujen lisäksi kuvauskantaan määritellyt salaisuudet ja ympäristömuuttujat. Työnkulun konfiguraatiossa olevan toimenpiteen ”actions/checkout@v3” avulla kuvauskannassa olevat lähdekoodit, kuten Terraformin konfiguraatiotiedostot, saadaan haettua osaksi työnkulun työtä. Terraform saadaan asennettua ja määriteltyä omalla toimenpiteellään ”hashicorp/setup-terraform@v2”. Terraform

lataa työnkulkua suorittavassa virtuaalipalvelimessa palveluntuottajiin liittyvät liitännäiset HashiCorpin rekisteristä, sen lisäksi että Terraformiin liittyvät tilatiedot luetaan Terraform Cloud -palvelusta ja tallennetaan sinne. Terraform tekee konfiguraatiotiedostojen pohjalta rajapintakutsuja infrastruktuurin palveluntuottajan palveluun muutosten arvioimiseksi ja toteuttamiseksi.

```
name: "Terraform"

on:
  push:
    branches:
      - main
  pull_request:

jobs:
  terraform:
    name: "Terraform"
    runs-on: ubuntu-latest
    permissions:
      pull-requests: write
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v2
        with:
          # terraform_version: 1.4.2
          cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}

      - name: Terraform Format
        id: fmt
        run: terraform fmt -check

      - name: Terraform Init
        id: init
        run: terraform init

      - name: Terraform Validate
        id: validate
        run: terraform validate -no-color

      - name: Terraform Plan
        id: plan
        if: github.event_name == 'pull_request'
        run: terraform plan -no-color -input=false
        continue-on-error: true

      - name: Update Pull Request
        uses: actions/github-script@v6
        if: github.event_name == 'pull_request'
        env:
          PLAN: ${{ steps.plan.outputs.stdout }}
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const output = `#### Terraform Format and Style 🖌\`${{
steps.fmt.outcome }}\`
            #### Terraform Initialization ⚙\`${{ steps.init.outcome
}}\`
            #### Terraform Validation 🤖\`${{ steps.validate.outcome
}}\`
            #### Terraform Plan 📖\`${{ steps.plan.outcome }}\`
            <details><summary>Show Plan</summary>
            \`\`\`terraform\n
            ${process.env.PLAN}
            \`\`\`
```

```
            </details>
            *Pushed by: @${{ github.actor }}, Action: \`${{
github.event_name }}\`*`;
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: output
            })
      - name: Terraform Plan Status
        if: steps.plan.outcome == 'failure'
        run: exit 1

      - name: Terraform Apply
        if: github.ref == 'refs/heads/main' && github.event_name ==
'push'
        run: terraform apply -auto-approve -input=false
```

Esimerkkikoodi 14. GitHub Actions -työnkulku, jossa luodaan suunnitelma tehtävistä muutoksista infrastruktuuriin yhdistämispyynnön yhteydessä. Muutokset astuvat voimaan, kun ne viedään päähaaraan ”main”. [34.]

Esimerkkikoodin 14 mukaisessa GitHub Actions -työnkulussa määritellään aluksi, minkä tapahtuman yhteydessä työnkulku käynnistetään. Se käynnistetään, kun GitHubiin luodaan yhdistämispyyntö (pull request) tai uusia lähdekoodimuutoksia viedään osaksi ”main”-haaraa (push). Työnkulussa on yksi työ, joka sisältää yhteensä yhdeksän eri työvaihetta. Työ suoritetaan Ubuntu-virtuaalikoneessa.

Ensimmäisessä työvaiheessa (”Checkout”) haetaan lähdekoodit GitHubin kuvauskannasta hyödyntäen GitHub Actions -toimenpidettä ”actions/checkout@v3”.

Toisessa työvaiheessa (”Setup Terraform”) hyödynnetään GitHub Actions -toimenpidettä ”hashicorp/setup-terraform@v2”, joka asentaa Terraformin työnkulun työn käytettäväksi. Kun Terraform on asennettu kertaalleen tietyssä työvaiheessa, Terraform ja siihen liittyvät komennot ovat käytettävissä kaikissa seuraavissa samaan työhön liittyvissä työvaiheissa. Esimerkkikoodissa 14 hyödynnetään Terraform Cloudia, joten tilatiedot ja konetiliin liittyvät tiedot löytyvät HashiCorpin ympäristöstä.

Kolmannessa työvaiheessa (”Terraform Format”) tarkistetaan Terraformiin liittyvien konfiguraatiotiedostojen tyylittely muokkaamatta niitä. Neljännessä

vaiheessa (”Terraform Init”) alustetaan Terraform lataamalla tarjoajien tiedot liittyen ympäristöön, jossa tehdään muutoksia infrastruktuuriin. Palveluntuottajat on määritelty esimerkiksi Terraform-konfiguraatiotiedostojen juurihakemiston ”main.tf”-tiedostoon. Palveluntuottajat määritellään ”terraform”-lohkon sisäkkäiseen lohkoon ”required_providers” ja omina ”provider”-lohkoina. Alustuksen yhteydessä tarkistetaan myös Terraformin tilaan liittyvät sijainnit.

Viidennessä työvaiheessa (”Terraform Validate”) tarkistetaan olemassa olevien konfiguraatioiden syntaksellinen oikeellisuus tarjoajien tietojen pohjalta. Komentoon ”terraform validate” annettava ”-no-color”-valitsin tarkoittaa, että konsoliin tulostamisessa ei käytetä värejä.

Kuudennessa työvaiheessa (”Terraform Plan”) tehdään suunnitelma infrastruktuuriin tehtävistä muutoksista. Suunnitelmaa luodessaan Terraform tarkistaa infrastruktuurin tilan ja vertaa sitä nykyiseen konfiguraatioon samalla tavalla kuin ”terraform apply” -komento, mutta muutoksia ei voi tällä komennolla tehdä infrastruktuuriin. Tarvittaessa suunnitelman lopputulos voidaan tallentaa esimerkiksi ”tfplan”-tiedostoon ja toteuttaa muutokset myöhemmin ”terraform apply tfplan” -komennolla. Suunnitelman nimi on kehittäjän päätettävissä. Kuudenteen työvaiheeseen on annettu ehto ”if: github.event_name == 'pull_request'”, jonka perusteella työvaihe suoritetaan vain siinä tapauksessa, että kyseessä on yhdistämispyyntö-tapahtuma. Seuraavaan työvaiheeseen jatketaan, vaikka ”Terraform Plan” -työvaihe epäonnistuisi. Tämä voidaan todentaa koodirivistä ”continue-on-error: true”.

Seitsemännessä työvaiheessa (”Update Pull Request”) hyödynnetään GitHub Actions -toimenpidettä ”actions/github-script@v6”. Tällä toimenpiteen avulla yhdistämispyyntöön (pull request) julkaistaan kommentti, joka sisältää tiedot edellisten työvaiheiden, ”Terraform Format”, ”Terraform Init”, ”Terraform Validate” ja ”Terraform Plan”, suoriutumisesta. Viittaus muuttujaan ”steps.id.outcome”, jossa ”id” on työvaiheen tunniste, sisältää joko merkkijonon ”success”, jos työvaihe on onnistunut, tai ”failure”, jos työvaihe on epäonnistunut. Työvaiheessa ”Update Pull Request” luodaan ympäristömuuttuja

”PLAN”, johon tallennetaan kaikki työvaiheessa konsoliin tulostettavat tiedot. Työvaihe suoritetaan vain, jos kyseessä on yhdistämispyyntö.

Kahdeksas työvaihe (”Terraform Plan Status”) ajetaan vain, jos edellinen työvaihe ”Terraform Plan” on epäonnistunut. Tässä tilanteessa työnkulun suorittaminen lopetetaan välittömästi komennolla ”exit 1”.

Yhdeksäs eli viimeinen työvaihe (”Terraform Apply”) suoritetaan vain, jos koodimuutoksia viedään ”main”-haaraan. Viimeisessä työvaiheessa infrastruktuuriin tehdään muutokset todellisesti.

5 Sovelluksen koodipohjaisen käyttöönoton toteutus

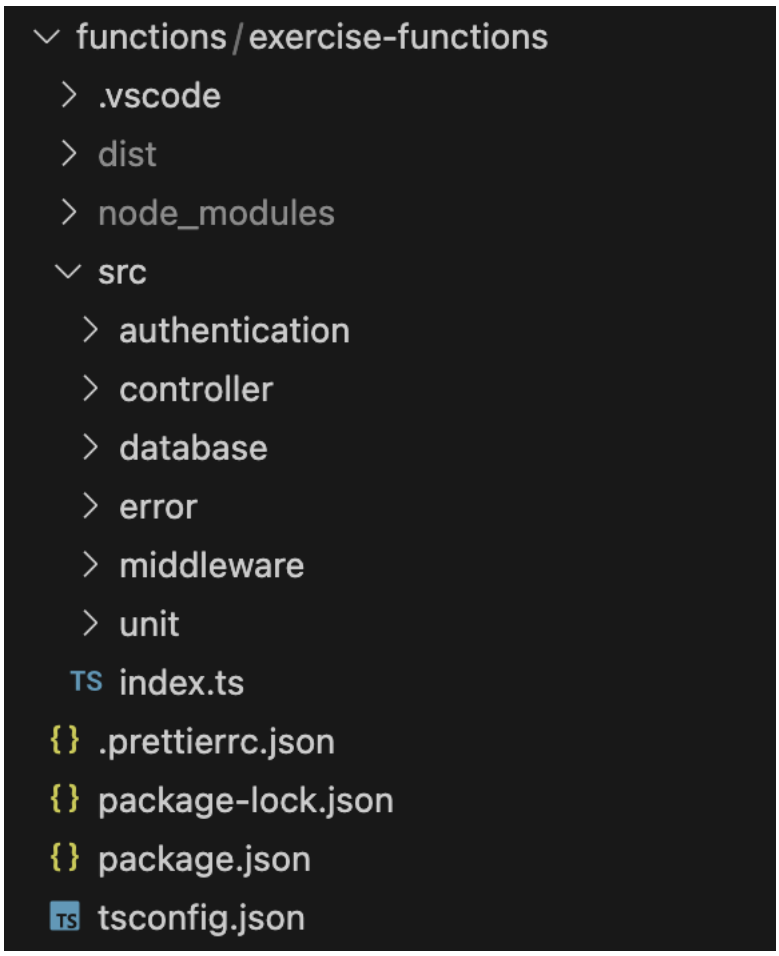
5.1 Sovellus

Insinöörityössä luotiin Google Cloud Platform (GCP) -ympäristöön Terraformia ja GitHub Actionsia hyödyntäen sovelluskokonaisuus, joka muodostui sekä edusta- että taustapalveluista. Sovellus ja siihen liittyvä infrastruktuuri otettiin käyttöön GCP-ympäristössä mahdollisimman automatisoidusti ja koodipohjaisesti. Tässä työssä sovelluskokonaisuus muodosti yksinkertaisen matemaattisen oppimispelin, jossa pelaajat pystyivät harjoittelemaan yksikkömuunnoksia pituuksien, pinta-alojen ja tilavuuksien osalta. Pelaajien syötteet tarkistettiin ja tallennettiin taustapalveluissa. Matemaattinen peli ilmensi insinöörityössä sovelluskokonaisuutta.

Edustapalveluihin kuuluvat sellaiset sovelluskokonaisuuden osat, joiden kanssa loppukäyttäjä pystyy olemaan vuorovaikutuksessa. Insinöörityössä toteutettiin loppukäyttäjien saavutettavissa oleva käyttöliittymä TypeScriptillä hyödyntäen React-sovelluskehystä. Käyttöliittymän TypeScript-lähdekoodit käännettiin JavaScript-koodeiksi, ja lopuksi JavaScript-koodit yhdistettiin ja optimoitiin mahdollisimman pieneen muotoon tuotantokäyttöä varten. Valmiista käyttöliittymästä tehtiin web-palvelin-kontti hyödyntäen Dockeria. Docker-kontissa käyttöliittymää isännöitiin nginx-palvelinohjelmistolla, jota ajettiin GCP-

ympäristön Cloud Run -palvelussa. Käyttöliittymä oli saavutettavissa ainoastaan ulkoisen kuormantasaajan kautta.

Taustapalveluihin kuuluvat kaikki sellaiset sovelluskokonaisuuden osat, jotka eivät ole varsinaisesti loppukäyttäjien nähtävissä. Taustapalveluissa suoritetaan sovelluksen toiminnallisuuteen liittyviä toimintoja, kuten tietojen lukemista ja kirjoittamista. Tässä työssä taustapalvelut toteutettiin GCP:n toisen sukupolven Cloud Functions -palvelulla. Sovelluksen pilvifunktiot huolehtivat esimerkiksi käyttäjien sisään kirjaamisesta ja peliin liittyvien tietojen tallentamisesta tietokantaan. Kaikki pilvifunktiot toteutettiin Node.js:n versiolla 20 hyödyntäen TypeScriptiä. Pilvifunktiot vietiin osaksi GCP-ympäristöä muun infrastruktuurin luonnin yhteydessä hyödyntäen Terraformia. HTTPS-kutsut toimivat oletusarvoisesti herätteinä pilvifunktioiden suoriutumiselle. Pilvifunktio suoritetaan, kun funktion uniikkia HTTPS-osoitetta kutsutaan. Ulkoisen kuormantasaajan avulla kaikki ”/api”-alkuiset verkkokutsut ohjattiin oikeille pilvifunktioille. Kuvassa 8 esitetään yhteinen hakemistorakenne kaikille Google Cloud -funktioille.



Kuva 8. Kaikki Google Cloud -funktiot on toteutettu samaan hakemistoon.

Jokainen pilvifunktio voitaisiin toteuttaa omaan hakemistoonsa, mutta yhteinen hakemisto kaikille pilvifunktioille oli tässä tapauksessa perusteltua, koska jokainen pilvifunktio hyödynsi samoja toiminnallisuuksia, kuten esimerkiksi käyttäjän auktorisoimisen, HTTP-kutsun kelvollisuuden tarkistamisen ja erilaisia yhteisiä tietokantakyselyitä. Tiedostossa ”index.ts” määriteltiin jokaiselle pilvifunktiolle oikea sisääntulo. Kuvassa 9 esitellään tiedoston sisältöä.

```
import * as functions from '@google-cloud/functions-framework'
import {
  createExerciseController,
  checkExercise,
} from './controller/exerciseControllers'
import {
  getUserDetailsController,
  updateNickController,
} from './controller/userControllers'
import {
  tokenController,
  refreshController,
} from './controller/authControllers'

// Exercise controllers
functions.http('create-exercise', createExerciseController)
functions.http('check-exercise', checkExercise)
// User controller
functions.http('user', getUserDetailsController)
functions.http('update-nick', updateNickController)
// Auth controllers
functions.http('token', tokenController)
functions.http('refresh', refreshController)
```

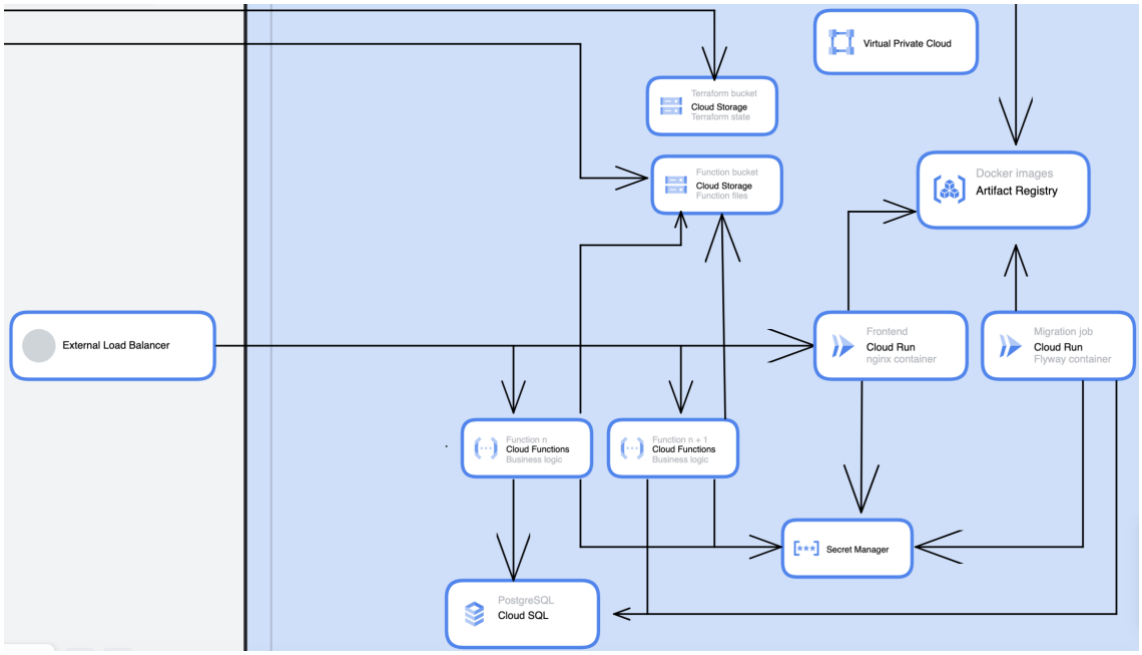
Kuva 9. Tiedoston ”index.ts” sisältöä.

Mikäli GCP-ympäristöön halutaan luoda esimerkiksi pilvifunktio, jolla voitaisiin hakea sisään kirjautuneen käyttäjän tiedot, pilvifunktion määrittelyssä sisääntulotietona käytettäisiin arvoa ”user”. Näin määriteltyyn pilvifunktioon tehty kutsu ohjautuu oikeaan käsittelijään (kuvassa 9 ”getUserDetailsController”-funktio).

5.2 Arkkitehtuuri

Insinöörityössä luotiin infrastruktuuri GCP-ympäristöön koodipohjaisesti hyödyntäen Terraformia ja GitHub Actionsia. Kaikki sovelluksen lähdekoodit isännöitiin GitHubissa. Kuvassa 10 esitellään infrastruktuurin arkkitehtuuri yleisellä tasolla. Nuolista käyvät ilmi eri palveluiden riippuvuudet. Kuvan 10

ulkopuolelle jäänyt arkkitehtuurin osa on GitHub, joka kommunikoi erityisesti Cloud Storagen ja Artifact Registryn kanssa.



Kuva 10. GCP-ympäristöön luotu infrastruktuuri.

Infrastruktuuriin kuuluvat palvelut isännöitiin Googlen Haminan-konesaleissa. Terraformin konfiguraatioissa tätä maantieteellistä sijaintia (region) vastaa arvo ”europe-north-1” tai ”eu-north-1”. Infrastruktuuriin luotiin kaksi Cloud Storage -lokeroa (bucket), joista kuvassa 10 ylemmässä sijaitsevat Terraformin tilatiedot ja alemmassa pilvifunktioiden sovellustiedot. Terraformin tilatietoihin liittyvä lokero luotiin manuaalisesti Google-konsolin kautta, jotta Terraform voitiin alustaa alusta alkaen kirjoittamaan ja lukemaan tilatietoja keskitetystä tallennuspaikasta. Terraformin tilatietoihin liittyvä lokero ei ole Terraformin hallitsema. Pilvifunktioiden sovellustietoihin liittyvä lokero luotiin automaattisesti Terraformilla. Lisäksi pilvifunktioiden sovellustietojen pakkaaminen ZIP-muotoon ja lataaminen Cloud Storage -lokeroon toteutettiin myös Terraformilla.

Artifact Registry luotiin Docker-näennäistietojen tallentamista varten. Docker-näennäistiedot luotiin ja vietiin automaattisesti Artifact Registryyn osana GitHub Actions -työnkulkua. Cloud Run -palvelu mahdollistaa Docker-konttien ajamisen ilman palvelimia (serverless). Sovelluksen käyttöliittymään liittyvä Docker-kontti

oli ajossa Cloud Run -palvelussa. Tämän lisäksi Cloud Run -palvelua hyödynnettiin tietokantamuutosten automaattiseen tekemiseen osana GitHub Actions -työnkulkua suoritettavana Cloud Run -työnä. Cloud Run määriteltiin hakemaan Docker-näennäistiedostot Artifact Registrystä.

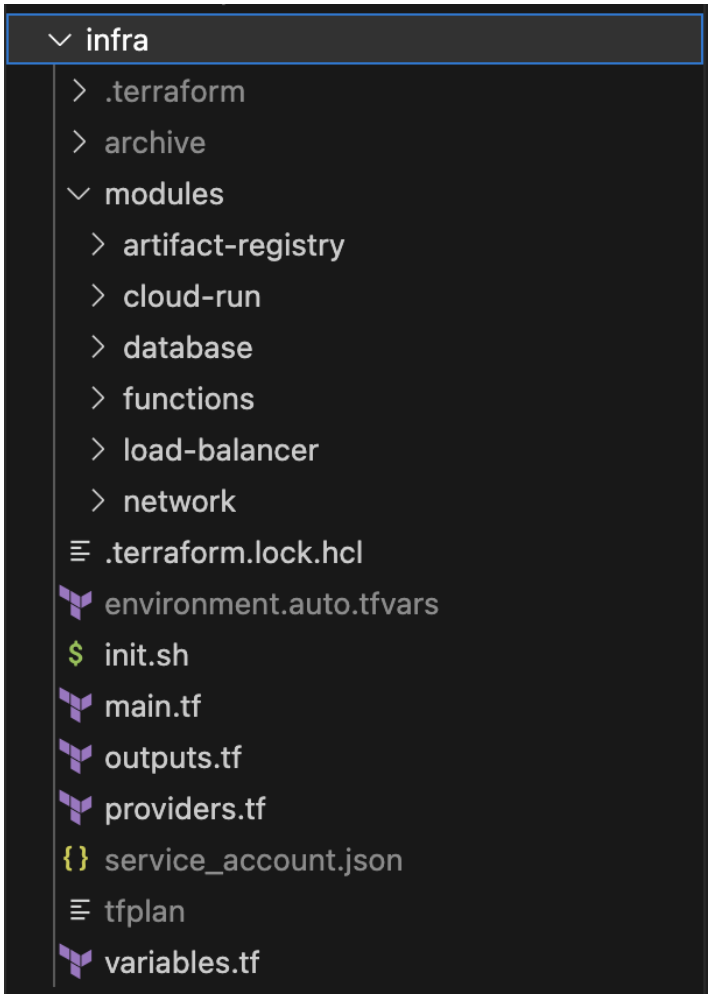
Kuvassa 10 ulkoinen kuormantasaaja huolehtii sekä verkkokutsujen ohjaamisesta oikeaan palveluun että SSL-terminoinnista. SSL-terminoinnissa salatut HTTPS-kutsut puretaan ja liikenne jatkaa kulkuaan kohdepalveluun HTTP-kutsuna. Kaikki kutsut ohjattiin oletusarvoisesti sovelluksen käyttöliittymään. Vastaavasti kutsut, joissa on jokin ”/api”-alkuinen polku, ohjattiin pilvifunktioiden käsiteltäviksi. Esimerkiksi polku ”/api/user” ohjasi kutsun kuormantasaajalta pilvifunktioon, joka vastasi käyttäjän tietojen palauttamisesta.

Cloud SQL on Google Cloudin tarjoama palvelu relaatiotietokannoille. Insinöörityössä Terraformilla luotiin PostgreSQL-tietokanta, jonne tallennettiin esimerkiksi käyttäjäkohtaisia tietoja ja matemaattisen pelin tehtäviin liittyviä tietoja. Tietokantaan pystyi luomaan yhteyden ainoastaan Google-projektiin kuuluvasta Virtual Private Cloud (VPC) -verkosta sisäisesti. Kaikki pilvifunktiot ja Cloud Run -työ, joka huolehti tietokantaan tehtävistä muutoksista, kommunikoivat tietokannan kanssa. Tietokantamuutokset tehtiin Flyway-ohjelmistolla, ja muutokset ajettiin osana GitHub Actions -työnkulkua tietokantaan. Tietokantamuutokset voivat olla esimerkiksi uusien tietokantataulujen luomista sekä tietojen lisäämistä, poistamista ja päivittämistä. Flyway tallentaa tehtyjen muutosten osalta tiedot omaan ”schema_history” -tauluun.

Secret Managerissa hallinnoitiin sovelluskokonaisuuteen liittyviä salaisuuksia. Secret Managerin avulla GCP-ympäristön eri palvelut voivat hakea turvallisesti salaisuuksina pidettäviä tietoja osaksi niiden toimintaa esimerkiksi ympäristömuuttujien muodossa. Salaisuuksia ovat esimerkiksi tietokantakäyttäjien salasanat ja erilaiset API-avaimet rajapintakutsujen tekemisessä.

5.3 Terraformin konfiguraatiot

Insinöörityössä luotiin Terraformin konfiguraatiotiedostoja varten oma ”infra”-niminen hakemisto. Hakemistossa sijaitsivat kaikki tiedostot infrastruktuurin hallitsemista varten. Kuvassa 11 esitellään tämän hakemiston rakennetta.



Kuva 11. Hakemiston ”infra” rakenne.

Kuvassa 11 harmaalla merkityt hakemistot ja tiedostot eivät ole osa versionhallintaa. Esimerkiksi ”service_account.json”-tiedosto sisältää salaisuutena Google-konetilin tiedot, joita Terraform käyttää muutosten tekemiseksi infrastruktuuriin.

Terraformin moduulit luotiin ”modules”-nimiseen hakemistoon. Moduulien avulla GCP-ympäristöön käyttöönotettiin kontekstiltaan samankaltaiset resurssit, jotta

infrastruktuurin ja siihen liittyvien konfiguraatiotiedostojen hallitseminen olisi mahdollisimman helppoa. Esimerkiksi tietokantamoduuli (kuvassa 11 ”modules”-hakemiston alla sijaitseva ”database”-hakemisto”) loi GCP-ympäristöön tietokantaesiintymän, tietokannan ja sovelluskäyttäjän SQL-kyselyiden suorittamiseksi sovelluksessa. Kaikki moduulit otettiin käyttöön ”infra”-hakemiston juuressa sijaitsevassa ”main.tf”-tiedostossa.

Tiedostossa ”providers.tf”-määriteltiin kaikki tarvittavat palveluntuottajaliitännäiset. Lisäksi tiedostoon määriteltiin keskitetty paikka, jonne Terraformin tilatiedot tallennettiin. Kuvassa 12 on ”providers.tf”-tiedoston sisältö.

```
1   terraform {
2     required_providers {
3       google = {
4         version = "4.78.0"
5       }
6     }
7     # Backend configuration is given during initilization,
8     backend "gcs" {}
9   }
10
11  provider "google" {
12    credentials = "service_account.json"
13    project     = var.gcp_project.project
14    region      = var.gcp_project.region
15  }
```

Kuva 12. Tiedosto ”providers.tf”.

Kuvan 12 Terraform-lohkossa määritellään Google-palveluntuottajaliitännäisen versio, joka ladataan Terraformin alustuksen yhteydessä. Terraform-lohkoon on myös määritelty ulkoinen tallennuspaikka Terraformin tilatiedoille (kuvassa 12 ”backend”-lohko). Alustuksessa (komento ”terraform init”) valitsimien avulla kerrottiin, mihin Google Cloud Storage -lokeroon tilatiedot tallennetaan. Alustusta varten luotiin apuskripti ”init.sh”, jolla tieto käytettävästä Cloud

Storage -lokerosta asetettiin. Apuskriptin ansiosta Terraform-ympäristön alustus voitiin tehdä helposti huomioiden ympäristökohtainen valmius, vaikka työssä olikin käytössä vain testiympäristö. Kuvan 12 ”provider”-lohkossa määritellään Google-palveluntuottajaan liittyviä asetuksia. Argumentilla ”credentials” kerrotaan konetilin yksityisen avaimen sijainti. Argumentilla ”project” asetettiin Google-projektin yksilöivä tunniste ja argumentin ”region” avulla oletusarvoinen maantieteellinen sijainti. Insinöörityön projektissa nämä kaksi arvoa luettiin oliotyppisestä muuttujasta ”gcp_project”. Tähän muuttujaan asetettiin arvo Terraformin käynnistyksen yhteydessä tiedostosta ”environment.auto.tfvars”, jonka Terraform latasi automaattisesti käynnistyessään. Tätä tiedostoa pidettiin salaisuutena, sillä esimerkiksi Google-projektia yksilöivä tieto voidaan katsoa arkaluonteiseksi. Lisäksi tiedostossa määriteltiin muun muassa OAuth2-avaimia, jotka eivät myöskään sovellu avoimeksi tiedoksi. Projektissa OAuth2-avaimia hyödynnettiin sovellukseen liittyvässä Google-kirjautumisintegraatiossa. GitHub Actions -työnkulussa luotiin tiedosto ”environment.auto.tfvars” sisältöineen ennen Terraformin alustamista.

5.4 Terraformin moduulit

Hakemistoon ”modules” sijoitettiin kaikki sellaiset infrastruktuurin osat, jotka muodostivat selvästi omia kokonaisuuksia, kuten esimerkiksi kuormantasaaja ja tietokanta. Luvussa 2.1 käsiteltiin kolmea ydinkäytäntöä infrastruktuurin hallitsemiseksi koodipohjaisesti, ja yksi näistä käytännöistä oli infrastruktuurin määritteleminen selkeinä ja tarpeeksi pieninä osina, jotka muodostavat itsenäisiä komponentteja. Mikäli laajempia kokonaisuuksia ei olisi luonut Terraform-moduuleina, infrastruktuuriin liittyvän koodin tulkitseminen ja infrastruktuurin hallinta olisivat olleet haasteellisia. Jokaiselle moduulille luotiin oma hakemisto ”modules”-nimisen hakemiston alle, ja kunkin moduulin hakemistossa oli lähtökohtaisesti kolme tiedostoa: ”main.tf”, ”outputs.tf” ja ”variables.tf”. Moduulit otettiin käyttöön ”infra”-hakemiston juuressa olevassa ”main.tf”-tiedostossa.

Tutustutaan tarkemmin siihen, kuinka esimerkiksi tietokantaan liittyvä moduuli toteutettiin. Kuvassa 13 esitellään, miten tietokantamoduuli otettiin käyttöön ”infra”-hakemiston ”main.tf”-tiedostossa.

Kuva 13. Tietokantamoduulin määritelmä ”infra”-hakemiston ”main.tf”-tiedostossa.

Terraformissa moduulit otetaan käyttöön kirjoittamalla ”module”-tyyppinen lohko ja antamalla sille moduulia kuvaava nimi ensimmäisellä nimikkeellä. Kuvassa 13 moduulin nimi on ”database”. Lohkon sisälle määritellään argumenttina lähde (source), josta hakemistosta kyseinen moduuli löytyy. Muut mahdolliset argumentit määritellään sen perusteella, mitkä ovat pakollisia muuttujia kohdemoduulissa. Kuvan 13 tietokantamoduulissa määriteltiin ”vpc_network_id”-argumentti, jolla asetettiin tieto käytettävästä Virtual Private Cloud (VPC) -verkosta. Argumentissa viitattiin olemassa olevan ”network”-moduulin tulosteeseen (output) ”vpc_network_id”.

Tarkastellaan seuraavaksi, miten tietokantamoduuli on toteutettu ”database”-hakemistoon. Kuvassa 14 esitellään tietokantamoduulin Terraform-koodia tiedostossa ”main.tf”.

```
resource "random_id" "database_instance_suffix" {
  byte_length = 4
}

resource "google_sql_database_instance" "database_instance" {
  name             = "database-instance-${random_id.database_instance_suffix.hex}"
  database_version = "POSTGRES_15"

  settings {
    tier = "db-f1-micro"

    ip_configuration {
      require_ssl     = false
      ipv4_enabled    = false
      private_network = var.vpc_network_id
    }

    maintenance_window {
      day          = 7
      hour         = 3
      update_track = "stable"
    }

    backup_configuration {
      enabled                        = true
      start_time                     = "04:00"
      point_in_time_recovery_enabled = true
      transaction_log_retention_days = 7
    }
  }

  deletion_protection = false
}

resource "google_sql_database" "database" {
  name     = "math"
  instance = google_sql_database_instance.database_instance.name
}

resource "random_password" "application_user_password" {
  length = 16
}

resource "google_sql_user" "application_user" {
  name     = "application"
  instance = google_sql_database_instance.database_instance.name
  password = random_password.application_user_password.result
  type     = "BUILT_IN"
}
```

Kuva 14. Tietokantamoduulin koodia tiedostossa ”main.tf”.

Kuvassa 14 koodiriveillä 1–3 määriteltiin resurssi, jolla luotiin satunnainen merkkijono tietokantaesiintymän nimen päätteeksi. Koodiriveillä 5–30 määriteltiin varsinainen tietokantaesiintymä. Resurssissa viitataan esimerkiksi muuttujaan ”vpc_network_id” koodirivillä 15 argumentissa ”private_network”. Koodiriveillä 35–38 luotiin tietokanta osaksi tietokantaesiintymää. Tietokantaesiintymään viitataan koodirivillä 37 ”instance”-argumentissa käyttäen attribuuttia ”google_sql_database_instance.database_instance.name” aiemman tietokantaesiintymäresurssin osalta. Lopuksi koodiriveillä 40–49 tietokantaan luotiin sovelluskäyttäjä satunnaisella salasanalla.

Kuvassa 15 esitellään tietokantamoduulin ”variables.tf”-tiedosto. Tietokantamoduulin ainoa pakollinen muuttuja oli ”vpc_network_id”, jolla määriteltiin, mitä verkkoa tietokantaesiintymä käyttää.

```
variable "vpc_network_id" {
  type        = string
  description = "VPC network that is associated with database"
}
```

Kuva 15. Tietokantamoduulin ”variables.tf”-tiedosto.

Tietokantamoduuli sisälsi myös tulosteita erilaisten attribuuttien osalta, joita tarvittiin muiden moduulien käyttöönotossa. Kuvassa 16 esitellään kaikki tietokantamoduulin määritellyt tulosteet.

```
output "database_user" {
  value       = google_sql_user.application_user.name
  description = "Application user's name"
}

output "database_password" {
  value       = google_secret_manager_secret.database_application_password.id
  sensitive   = true
  description = "Application user's password in Secret Manager"
}

output "database_password_secret_manager_id" {
  value       = google_secret_manager_secret.database_application_password.secret_id
  sensitive   = true
  description = "Application user's password Secret Manager ID"
}

output "database_host" {
  value       = google_sql_database_instance.database_instance.private_ip_address
  description = "Database host"
}

output "database_name" {
  value       = google_sql_database.database.name
  description = "Database name"
}
```

Kuva 16. Tietokantamoduulin ”outputs.tf”-tiedosto.

Tietokantamoduulin tulosteita hyödynnettiin esimerkiksi osana pilvifunktioiden moduulia ”functions” ja Cloud Run -moduulia ”cloud_run”. Pilvifunktioissa asetettiin esimerkiksi tietokannan osoite ja tietokannan nimi ympäristömuuttujina. Lisäksi pilvifunktiot hakivat tietokantakäyttäjän salasanan ympäristömuuttujaksi Google Cloudin Secrets Manager -palvelusta.

5.5 GitHub Actions -työnkulut

Insinöörityössä sekä infrastruktuuri että sovellus otettiin käyttöön täysin automatisoidusti hyödyntäen GitHub Actions -palvelua. GitHub Actions -työnkulkuja varten luotiin hakemisto ”.github”, jonne luotiin alihakemisto ”workflows”. Projektiin luotiin kaksi eri työnkulkua, joista toinen huolehti

infrastruktuurin ja sovelluksen käyttöönotosta, kun uutta koodia vietiin osaksi GitHubin ”main”-haaraa, ja toinen Terraformin konfiguraatiotiedostojen tarkistamisesta ja suunnitelman tekemisestä yhdistämispyynnön (pull request) yhteydessä. Tutustutaan ensimmäiseksi GitHub Actions -työnkulkuun, jonka herätteenä toimi yhdistämispyyntö (esimerkkikoodi 15).

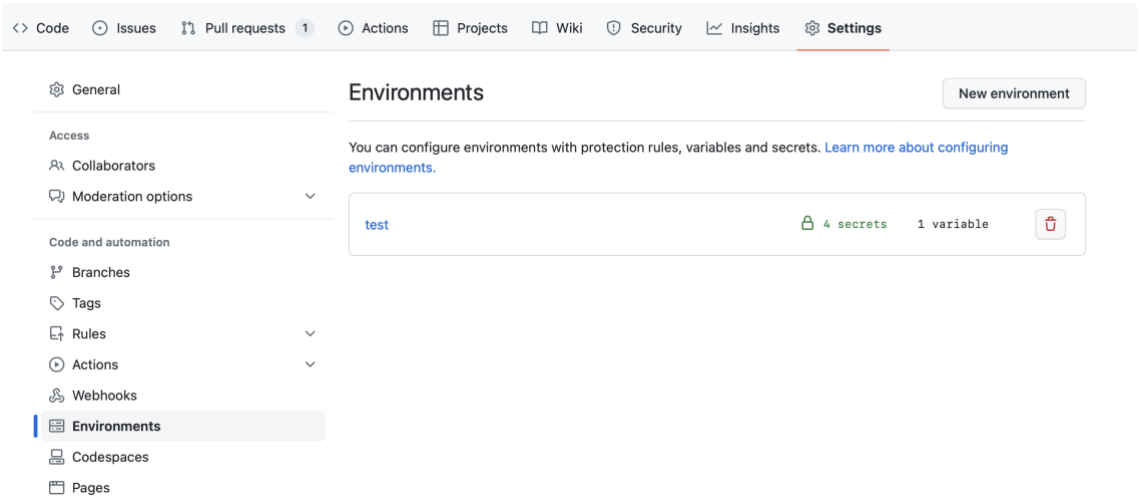
```
name: Pull request workflow
on:
  pull_request:
    branches:
      - '*'

jobs:
  pull_request:
    environment: test
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Google Authentication
        uses: 'google-github-actions/auth@v1'
        with:
          credentials_json: ${{ secrets.TERRAFORM_SERVICE_ACCOUNT }}
      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v2
      - name: Terraform format
        working-directory: ./infra
        run: terraform fmt -recursive -check
      - name: Initialize Terraform
        env:
          GCS_BUCKET: ${{ secrets.TERRAFORM_STATE_BUCKET }}
          TERRAFORM_SERVICE_ACCOUNT: ${{
secrets.TERRAFORM_SERVICE_ACCOUNT }}
          TERRAFORM_VARIABLES: ${{ secrets.TERRAFORM_VARIABLES }}
        working-directory: ./infra
        run: |
          echo "$TERRAFORM_SERVICE_ACCOUNT" > service_account.json
          echo "$TERRAFORM_VARIABLES" > environment.auto.tfvars
          ./init.sh "$GCS_BUCKET"
      - name: Terraform validate
        working-directory: ./infra
        run: terraform validate -no-color
      - name: Terraform plan
        id: terraform_plan
        working-directory: ./infra
        continue-on-error: true
        run: terraform plan -lock=true -no-color
      - name: Post Terraform plan result as a comment
        env:
          PLAN_RESULT: ${{ steps.terraform_plan.outputs.stdout }}
        uses: actions/github-script@v6
        with:
          retries: 3
          script: |
            const { PLAN_RESULT } = process.env

            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: `${PLAN_RESULT}`
            })
```

Esimerkkikoodi 15. GitHub Actions -työnkulku yhdistämispyynnöille.

Esimerkkikoodin 15 alussa määritellään työnkulun nimi ja heräte, jolla työnkulku suoritetaan. Työnkulku suoritetaan, kun mistä tahansa kehityshaarasta tehdään yhdistämispyyntö (pull request). Työnkulussa on yksi työ, jossa on kahdeksan eri työvaihetta. Työ suoritetaan GitHubin isännöimällä Ubuntu-virtuaalikoneella. Ympäristöksi on määritelty testi. Ympäristökohtaisella tiedolla GitHub Actions -työnkulun työssä voidaan käyttää tiettyyn ympäristöön liittyviä ympäristömuuttujia ja salaisuuksia. Kuvassa 17 esitetään, miltä insinöörityön ympäristöt-näkymä näytti GitHubissa.



Kuva 17. GitHub-kuvauskannan ympäristöt-näkymä.

Ympäristöjä pystyy luomaan GitHub-kuvauskannan asetuksissa ympäristöt-osiossa. Kuvassa 17 on yksi ympäristö, jonka nimi on ”test”. Tässä ympäristössä on neljä salaisuutta ja yksi ympäristömuuttuja.

Esimerkkikoodin 15 toisessa työvaiheessa (”Google Authentication”) Google Cloudiin autentikoidutaan käyttäen aiemmin luotua Terraform-konetilin yksityistä avainta. Yksityisen avaimen sisältö haetaan GitHub-kuvauskannan salaisuuksista.

Neljännessä työvaiheessa (”Terraform format”) konfiguraatiotiedostojen koodinmuotoilu tarkistetaan rekursiivisesti. Rekursiivisessa tarkistuksessa konfiguraatiotiedostojen muotoilu tarkistetaan myös alihakemistojen osalta. Insinöörityössä rekursiivisen tarkistuksen avulla voitiin tarkistaa myös

moduuleihin liittyvien konfiguraatiotiedostojen muotoilu. Työnkulku epäonnistuisi, mikäli konfiguraatiotiedostojen muotoilu olisi vääränlainen.

Viidennessä työvaiheessa (”Initialize Terraform”) alustetaan Terraform. Työvaiheen alussa ympäristömuuttujiin asetetaan salaisuuksia. Ympäristömuuttujia käytettiin konetilin JSON-muotoisen avaimen luomiseen (”service_account.json”) ja Terraformin muuttujatiedoston luontiin (”environment.auto.tfvars”). Muuttujaan ”GCS_BUCKET” tallennetaan tieto Google Cloud Storage -lokerosta, jota käytetään Terraformin tilatietojen tallennuspaikkana. Lopuksi työvaiheessa alustetaan Terraform hyödyntäen skriptiä ”init.sh”. Skriptin sisältö on kuvassa 18.

```
1  #!/usr/bin/env bash
2
3  BUCKET="$1"
4  PREFIX="terraform/state"
5
6  if [[ -z "$BUCKET" ]]
7  then
8      echo "Please provide a bucket name as an argument"
9      exit 1
10 fi
11
12 terraform init \
13     -no-color \
14     -backend-config="bucket=$BUCKET" \
15     -backend-config="prefix=$PREFIX"
```

Kuva 18. Skriptin ”init.sh” sisältö.

Kuvassa 18 komennolle ”terraform init” annetaan ”backend-config”-valitsimien avulla Terraformin tilatietojen tallentamiseen liittyviä asetuksia. Aiemmin kuvassa 12 esitellyssä ”providers.tf”-tiedostossa ei määritelty ”backend”-lohkoon esimerkiksi lokeroa (bucket). Lohko ”backend” jätettiin tyhjäksi, jotta Terraform voitaisiin alustaa aina ympäristökohtaisesti GitHub Actions -työnkuluissa. Esimerkiksi tuotantoympäristön ja testiympäristön tilatiedot ovat erilaisia keskenään, eikä niitä voi sekoittaa.

Seitsemännessä työvaiheessa (”Terraform plan”) luodaan suunnitelma infrastruktuuriin tehtävistä muutoksista. Komennon ”terraform plan” avulla

Terraform ei tee muutoksia infrastruktuuriin, vaan vertailee infrastruktuuria ja olemassa olevaa konfiguraatiota keskenään ja tekee suunnitelman toteutettavista muutoksista. Muutostarpeet tulostetaan konsoliin. Työvaiheesta siirrytään seuraavaan työvaiheiseen, vaikka työvaihe epäonnistuisi. Viimeisessä työvaiheessa (”Post Terraform plan result as a comment”) käytetään Terraformin suunnitteluvaiheesta syntyneitä tietoja, jotka julkaistaan kommenttina GitHubin yhdistämispyyntöön (pull request).

Lopuksi tutustutaan vielä insinöörityössä luotuun työnkulkuun, jolla toteutettiin varsinaiset muutokset infrastruktuuriin ja käyttöönotettiin sovellus. Esimerkkikoodin 16 työnkulun osalta käydään läpi vain sellaiset työvaiheet, jotka ovat erilaisia esimerkkikoodiin 15 nähden.

```
name: Main workflow
on:
  push:
    branches:
      - main

jobs:
  build_and_deploy:
    environment: test
    runs-on: ubuntu-latest
    env:
      REGION: ${{ vars.REGION }}
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Build frontend and its Docker image
        working-directory: ./client
        env:
          FRONTEND_CONFIG: ${{ secrets.FRONTEND_CONFIG }}
        run: |
          echo "$FRONTEND_CONFIG" > .env
          npm install
          npm run build
          docker build -t math-platform-frontend .
      - name: Build Flyway Docker image
        working-directory: ./flyway
        run: docker build -t flyway .
      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v2
      - name: Terraform format
        working-directory: ./infra
        run: terraform fmt -recursive -check
      - name: Google Authentication
        uses: 'google-github-actions/auth@v1'
        with:
          credentials_json: ${{ secrets.TERRAFORM_SERVICE_ACCOUNT }}
      - name: Set up Google Cloud SDK
        uses: google-github-actions/setup-gcloud@v1
      - name: Initialize Terraform
        env:
          GCS_BUCKET: ${{ secrets.TERRAFORM_STATE_BUCKET }}
          TERRAFORM_SERVICE_ACCOUNT: ${{
secrets.TERRAFORM_SERVICE_ACCOUNT }}
          TERRAFORM_VARIABLES: ${{ secrets.TERRAFORM_VARIABLES }}
        working-directory: ./infra
        run: |
          echo "$TERRAFORM_SERVICE_ACCOUNT" > service_account.json
          echo "$TERRAFORM_VARIABLES" > environment.auto.tfvars
          ./init.sh "$GCS_BUCKET"
      - name: Terraform validate
        working-directory: ./infra
        run: terraform validate -no-color
      - name: Enable necessary Google APIs
        working-directory: ./infra
        run: |
          terraform plan -lock=true -
target=google_project_service.gcp_services -out=tfplan -no-color
          terraform apply -lock=true -no-color "tfplan"
      - name: Check Artifact Registry existence
        id: artifact_registry
        working-directory: ./infra
```

```
        run: terraform plan -lock=true -
target=module.artifact_registry -out=tfplan -detailed-exitcode -no-
color
      - name: Create Artifact Registry
        working-directory: ./infra
        if: ${{ steps.artifact_registry.outputs.exitcode != '0' }}
        run: terraform apply -lock=true -no-color "tfplan"
      - name: Push Docker images
        working-directory: ./infra
        run: |
          gcloud auth configure-docker europe-north1-docker.pkg.dev
          terraform output -raw docker_service_account_key | docker
login -u _json_key_base64 --password-stdin https://eu-north1-
docker.pkg.dev
          docker tag flyway europe-north1-docker.pkg.dev/mikael-holm-
sandbox-v2/docker-repository/flyway
          docker tag math-platform-frontend europe-north1-
docker.pkg.dev/mikael-holm-sandbox-v2/docker-repository/math-platform-
frontend
          docker push europe-north1-docker.pkg.dev/mikael-holm-
sandbox-v2/docker-repository/flyway
          docker push europe-north1-docker.pkg.dev/mikael-holm-
sandbox-v2/docker-repository/math-platform-frontend
      - name: Terraform plan
        id: terraform_plan
        working-directory: ./infra
        run: terraform plan -lock=true -out=tfplan -detailed-exitcode
-no-color
      - name: Terraform apply
        working-directory: ./infra
        if: ${{ steps.terraform_plan.outputs.exitcode != '0' }}
        run: terraform apply -lock=true -no-color "tfplan"
      - name: Run Flyway migrations
        run: gcloud run jobs execute flyway-migrations --
region="$REGION" –wait
```

Esimerkkikoodi 16. GitHub Actions -työnkulku ”Main workflow”.

Esimerkkikoodin 16 mukainen GitHub Actions -työnkulku toteutettiin ”main.yml”-tiedostoon. Työnkulussa on yksi työ (”build_and_deploy”), jossa on 16 työvaihetta. Työnkulku suoritetaan, kun kehityshaaraan ”main” viedään uusia lähdekoodimuutoksia.

Työvaiheissa ”Build frontend and its Docker image” ja ”Build Flyway Docker image” luodaan Docker-näennäistiedostot sovelluksen käyttöliittymästä ja Flyway-sovelluksesta. Docker-näennäistiedostoja käytettiin GCP-ympäristön Cloud Run -palvelussa.

Työvaiheessa ”Enable necessary Google APIs” otetaan käyttöön Terraformin avulla kaikki tarvittavat rajapinnat, joita Terraform tarvitsee Google-

infrastruktuurin hallitsemista varten. Tämän työvaiheen ”run”-osiossa luodaan Terraform-suunnitelma (”terraform plan”), jonka kohteena on ainoastaan konfiguraatiotiedostossa oleva resurssi, jolla rajapinnat saadaan käyttöön. Seuraavassa komennossa (”terraform apply”) suunnitelman mukaiset muutokset otetaan käyttöön infrastruktuurissa. Vaihtoehtoisesti rajapinnat voidaan ottaa käyttöön Google-konsolissa tai ”gcloud”-komentojen avulla, mutta kaikkien tarvittavien rajapintojen muistaminen ja käyttöönottaminen manuaalisesti on hidasta.

Työvaiheessa ”Check Artifact Registry existence” tarkistetaan, onko Artifact Registry -palvelua luotu GCP-ympäristöön. Artifact Registryn luominen oli edellytys ennen Cloud Run -palvelun käyttöönottamista, sillä Cloud Run haki tarvittavat Docker-näennäistiedostot Artifact Registrystä. Seuraavassa työvaiheessa ”Create Artifact Registry” luodaan Artifact Registry vain, jos sitä ei ole olemassa. Aiemmin luodut Docker-näennäistiedostot viedään Artifact Registryyn työvaiheessa ”Push Docker images”. Artifact Registryn käyttöönoton yhteydessä luotiin oma konetili Dockeria varten, jolla voitiin viedä Docker-näennäistiedostot kyseisen Artifact Registryn kuvauskantaan. Konetilin yksityistä avainta käytettiin Docker-sisäänkirjautumisen yhteydessä, ja se saatiin haettua ”terraform output” -komennon avulla. Yksityisen avaimen tuloste saatiin haettua ”terraform output” -komennon ”raw”-valitsimella. Ilman ”raw”-valitsinta ei ole mahdollista tulostaa arvoja, jotka on merkitty ”output”-lohkoissa salaisiksi. Kun Docker-näennäistiedostot oli viety Artifact Registryyn, Terraformilla voitiin luoda kaikki puuttuvat infrastruktuurin resurssit GCP-ympäristöön työvaiheiden ”Terraform plan” ja ”Terraform apply” avulla.

Viimeisessä työvaiheessa ”Run Flyway migrations” tehdään tietokantamuutokset, kuten tarvittavien tietokantataulujen luominen suoritettavana Cloud Run -työnä. Työvaiheen ”Set up Google Cloud SDK” aikana työn kaikkien työvaiheiden käytettäväksi asennettiin Google Cloud -terminaali, jota käytettiin viimeisessä työvaiheessa Cloud Run -työn ajamiseksi.

5.6 Työn tuloksien arviointi

GitHub Actions ja Terraform muodostivat yhdessä toimivan kokonaisuuden sovelluksen käyttöönottamiseksi Google Cloud Platformissa (GCP). Infrastruktuuriin liittyvien Terraform-konfiguraatioiden kirjoittaminen oli helppoa kattavan Google-palveluntuottajaliitännäiseen perustuvan dokumentaation avulla. Dokumentaatio oli haettavissa HashiCorpin verkkosivuilta. Infrastruktuurin Terraform-konfiguraatiot kirjoitettiin Microsoftin Visual Studio Code -kehitinohjelmalla. Tähän ohjelmaan on mahdollista asentaa useita erilaisia laajennuksia helpottamaan sovelluskehittäjän työtä. Terraformin konfiguraatiotiedostojen kirjoittamisessa hyödynnettiin HashiCorpin Terraform-laajennusta, joka auttoi konfiguraatioiden kirjoittamisessa korostamalla Terraformin syntaksia ja ennakoimalla tekstiä sen automaattiseksi täydentämiseksi.

Konfiguraatioiden validoinnissa käytettävä komento ”terraform validate” ei toiminut täysin odotusten mukaisesti. Terraformin validointi osasi puuttua konfiguraatioihin liittyviin puutteisiin, mutta sen avulla ei kuitenkaan saatu kiinni kaikkia konfiguraatioihin liittyviä puutteita. Konfiguraatiossa ollut puute saattoi olla muun muassa jonkin argumentin puuttuminen resurssilohkon sisältä. Tietyt puutteet kävivät ilmi vasta Terraformin suunnittelu- (plan) tai käyttöönottovaiheissa (apply). Konfiguraatiotiedostojen muotoilussa ja muotoilun tarkistuksessa käytettävä komento ”terraform fmt” toimi sen sijaan hyvin.

Terraform tekee muutoksia GCP-ympäristön infrastruktuuriin useiden eri rajapintakutsujen avulla. GCP-ympäristön palvelukohtaiset rajapinnat tuli ottaa käyttöön ennen muutosten tekemistä infrastruktuuriin. Mikäli tarvittavia rajapintoja ei ollut otettu käyttöön GCP-ympäristössä esimerkiksi Cloud Functions- tai Cloud Run -palveluiden osalta, infrastruktuurin käyttöönotto Terraformilla epäonnistui. Rajapinnat tuli ottaa käyttöön manuaalisesti Google-projektin konsolissa selaimella tai Google Cloud -terminaalin avulla. Terraformilla automatisoitiin rajapintojen käyttöönotto kirjaamalla kaikki sovelluskokonaisuuden osalta tarvittavat rajapinnat Terraformin muuttujaan

listana ja ottamalla ne käyttöön omalla ”google_project_service”-resurssilla. GCP-ympäristön rajapintojen palauttamat virheilmoitukset olivat helposti tulkittavissa, ja niiden avulla onnistuttiin korjaamaan konfiguraatioihin liittyvät virheet helposti.

Terraformiin liittyvien moduulien avulla onnistuttiin luomaan sekä selkeitä että tarpeeksi pieniä osia infrastruktuurin eri osista. Moduulit mahdollistivat infrastruktuurin eri osien käyttöönottamisen testaamisen omina kokonaisuuksina ilman, että koko infrastruktuuria olisi pitänyt luoda kerralla. Lisäksi moduulien olemassaolo oli tärkeää esimerkiksi silloin, kun Artifact Registry piti luoda ennen muun infrastruktuurin käyttöönottamista GCP-ympäristössä. Artifact Registryn olemassaolo ennen muun infrastruktuurin käyttöönottamista oli tärkeää, sillä Cloud Run -resurssien käyttöönottaminen vaati Docker-näennäistiedostojen löytymistä Artifact Registrystä. Docker-näennäistiedostot olisi halutessaan voinut säilöä myös muuhun rekisteriin, kuten Docker Hubiin. Komennossa ”terraform plan” pystytään määrittelemään valitsimen ”target” avulla, mistä resursseista Terraform tekee suunnitelman niiden käyttöönottamiseksi. Artifact Registry -moduulin osalta muutokset pystyttiin ottamaan käyttöön asettamalla ”terraform plan”-komennon ”target”-valitsimeen kohteeksi ”module.artifact_registry” ja ottamalla muutokset käyttöön vain siinä tapauksessa, jos Artifact Registryä ei olisi ollut tai jokin olisi muuttunut siihen liittyvässä konfiguraatiossa. Valitsinta ”target” käytettiin myös tarvittavien GCP-rajapintojen käyttöönottamiseksi, sillä GCP-ympäristöön ei pysty tekemään muutoksia, ennen kuin tarvittavat rajapinnat on käyttöönotettu.

GitHub Actions -työnkulkujen konfiguroiminen YAML-tiedostoihin tapahtui myös luontevasti GitHubin dokumentaation avulla, jossa oli konkreettisia esimerkkejä konfiguraatioiden luomisesta. Toimenpiteiden (actions) hyödyntäminen osana työnkulkuja pienensi konfiguraatiotiedostojen kokoa, vähensi niissä käytettävää toistoa ja nopeutti konfiguraatioiden kirjoittamista huomattavasti. GitHubilta ei löytynyt omaa ratkaisua työnkulkujen suorittamiseksi paikallisessa kehitysympäristössä, joten työnkulkuja päädyttiin testaamaan viemällä uudet lähdekoodimuutokset insinöörityön GitHub-kuvauskantaan jokaisen muutoskerran jälkeen.

Salaisuuksien hallinta oli joissakin määrin haasteellista sovelluskokonaisuuden kannalta, sillä salaisuuksien hallinnassa piti ottaa huomioon ympäristökohtainen valmius. Esimerkiksi tuotantoympäristössä ja testiympäristössä olevat tunnukset ja salasanat ovat aina erilaisia. Lisäksi GitHub Actions -työnkulkujen suorittamisessa oli erityisen tärkeää varmistaa se, että salaisuudet eivät näkyneet missään vaiheessa tulosteissa. Insinöörityön salaisuuksia olivat esimerkiksi sovelluksen Google-kirjautumisintegraatiossa käytettävät OAuth2-avaimet ja Terraformin konetilin yksityinen avain. Insinöörityössä oli käytössä vain testiympäristö, mutta myös muiden ympäristöjen luominen olisi ollut mahdollista. Toimivasta työnkulusta pystyy tekemään kopion, ja kopioon pystyy määrittelemään eri ympäristön (environment) ja kehityshaaran (branch), mikä toimii herätteenä työnkulun käynnistämiselle, kun kehityshaaraan viedään lähdekoodimuutoksia. Tällä tavalla muutokset pystyttäisiin ottamaan käyttöön myös muissa mahdollisissa ympäristöissä olettaen, että toiseen GCP-projektiin ja GitHubin ympäristökohtaisin asetuksiin olisi tehty tarvittavat esivalmistelut.

6 Yhteenveto

Insinöörityössä perehdyttiin siihen, kuinka sovellus voidaan käyttöönottaa pilviympäristössä koodipohjaisesti. Työssä tarkasteltiin, miten sekä sovelluksen että siihen liittyvän infrastruktuuriin käyttöönotto pilviympäristöön voitiin tehdä mahdollisimman automatisoidusti hyödyntäen olemassa olevia sovelluskehityksen työkaluja. Terraform-työkalua käytettiin infrastruktuurin hallintaan lähes kokonaan, ja GitHub Actions -palvelua hyödynnettiin jatkuvan integraation (CI) ja jatkuvan toimituksen (CD) alustana. GitHub Actionsin ansiosta sekä sovellus että siihen liittyvä infrastruktuuri onnistuttiin käyttöönottamaan pilviympäristössä automatisoidusti. GitHub Actions oli helposti integroitavissa insinöörityön projektiin, sillä kaikki lähdekoodit isännöitiin GitHubissa. Tässä työssä infrastruktuuri luotiin Google Cloud Platform (GCP) -ympäristöön.

Työn manuaalisiin vaiheisiin kuuluivat GCP-ympäristön esivalmistelu ja GitHub Actions -työnkulkujen konfiguroiminen. GCP-ympäristöön luotiin Cloud Storage -lokero (bucket), jota käytettiin Terraformin tilatietojen tallennuspaikkana.

Tämän lisäksi muutosten tekemiseksi Terraformilla GCP-ympäristöön luotiin oma konetili manuaalisesti tarvittavilla oikeuksilla.

GitHub Actions -palvelun hyödyntäminen jatkuvan integraation ja jatkuvan toimituksen alustana toimi hyvin tässä työssä. Työnkulkujen kirjoittaminen oli suoraviivaista, sillä GitHub Actionsiin liittyvä syntaksellinen dokumentaatio oli ajan tasalla, sen lisäksi että dokumentaation esimerkkikoodit antoivat hyviä viitteitä työnkulkujen kirjoittamiseen. GitHub Actions -työnkulkujen testaaminen ja viimeistely olivat joissakin määrin haasteellisia, sillä työnkulkujen suorittamiseksi niiden konfiguraatiot piti päivittää GitHubin lähdekoodien kuvauskantaan jokaisen muutoskerran jälkeen. Työnkulkujen suorittamiseksi paikallisessa kehitysympäristössä löytyi avoimia työkaluja, kuten GitHubista löytyvä ”nektos/act”, mutta tässä työssä niiden käyttämiseen ei tutustuttu tarkemmin.

Terraform soveltui hyvin infrastruktuurin hallintaan, ja sen avulla pystyttiin käyttöönottamaan kaikki tarvittavat resurssit pilviympäristössä. Moduulien avulla infrastruktuurin konfiguraatioista saatiin aikaiseksi sekä pieniä että selkeitä kokonaisuuksia, ja niiden avulla infrastruktuuri pystyttiin käyttöönottamaan GCP-ympäristöön pienissä osissa tiettyjen kokonaisuuksien osalta. Infrastruktuuriin tehtäviä muutoksia oli helppo testata paikallisessa kehitysympäristössä luomalla yksi kokonaisuus kerrallaan GCP-ympäristöön kirjoitettujen Terraform-moduulien avulla. Samalla saatiin selvyys siitä, mitkä GCP-ympäristön rajapinnat piti ottaa käyttöön ennen varsinaisen infrastruktuurin luomista.

Terraform-työkalu ja GitHub Actions -palvelu toimivat keskenään hyvin, ja valmiiden GitHub Actions -toimenpiteiden (actions) avulla tarvittavien ohjelmistojen käyttöönotto onnistui vaivattomasti. Toimenpiteiden avulla työnkulkujen työvaiheiden käyttöön saatiin tarvittavat ohjelmistot, kuten Terraform ja Google Cloud -komentokehote.

Jatkuvan integraation (CI) ja jatkuvan toimituksen (CD) alustat ovat pääpirteiltään samanlaisia keskenään, mutta niiden konfiguroiminen eroaa

riippuen käytettävästä tuotteesta. Tässä työssä sekä sovellus että siihen liittyvä infrastruktuuri otettiin käyttöön automatisoidusti hyödyntäen GitHub Actionsia, mutta samaan lopputulokseen päästäisiin haluttaessa toisellakin tuotteella, kuten Jenkinsillä tai GitLabilla.

Insinöörityössä otettiin huomioon ympäristökohtainen valmius sovelluskokonaisuuden salaisuuksien osalta. Ympäristökohtaisten salaisuuksien huomioiminen oli tärkeää, sillä salaisuudet ovat lähtökohtaisesti aina erilaisia esimerkiksi testi- ja tuotantoympäristöissä. Salaisuuksia olivat esimerkiksi Google-projektia yksilöivä tunniste, Terraformin konetilin yksityisen avaimen sisältö ja Google-kirjautumisintegraatiossa käytettävät OAuth2-avaimet. Insinöörityössä oli käytössä ainoastaan testiympäristö. Lähdekoodit sisältävään GitHub-kuvauskantaan määriteltiin testiympäristö (test), jonne ympäristöä vastaavat salaisuudet tallennettiin. GitHub Actions -työnkuluissa ympäristökohtaisia salaisuuksia pystyttiin lukemaan testiympäristön osalta, kun työnkulun työn ympäristöksi (environment) määriteltiin arvo ”test” työnkulun konfiguraatioon.

Insinöörityössä keskityttiin pääasiassa GCP-ympäristöön ja GitHub Actions -palveluun sovelluksen koodipohjaisessa käyttöönotossa, kun infrastruktuuri koodina -työkaluna hyödynnettiin Terraformia. Tutkimusta sovelluksen koodipohjaisesta käyttöönotosta voisi jatkaa eri teknologioiden avulla. Jatkotutkimuksessa voisi tarkastella esimerkiksi sitä, soveltuisiko jokin muu pilviympäristö kuin GCP paremmin sovelluksen automatisoidulle käyttöönotolle tai olisiko jokin toinen jatkuvan integraation ja jatkuvan käyttöönoton alusta yksinkertaisempi sovelluskokonaisuuden käyttöönottamisessa. Erilaisten infrastruktuuri koodina -työkalujen hyödyntämistä infrastruktuurin luomisessa voisi myös tarkastella. Jatkotutkimuksessa voisi esimerkiksi vertailla sitä, olisiko infrastruktuurin hallinta helpompaa Puppetin avulla kuin Terraformilla.

Lähteet

1 Morris, Kief. 2020. Infrastructure as Code. E-kirja. O'Reilly Media.

2 Manage any infrastructure. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/infrastructure-as-code#manage-any-infrastructure>. Luettu 3.4.2023.

3 Achar, Sandesh. 2021. Enterprise SaaS Workloads on New-Generation Infrastructure-as-Code (IaC) on Multi-Cloud Platforms. Global Disclosure of Economics and Business. Volume 10, s. 65–66.

4 Rapaka, Vivekanad. 2023. Terraform Architecture Overview – Structure and Workflow. Verkkoaineisto. Spacelift. <https://spacelift.io/blog/terraform-architecture>. Luettu 20.9.2023.

5 What is Infrastructure as Code with Terraform. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/infrastructure-as-code#infrastructure-as-code>. Luettu 3.4.2023.

6 Standardize your deployment workflow. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/infrastructure-as-code#standardize-your-deployment-workflow>. Luettu 3.4.2023.

7 Track your infrastructure. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/infrastructure-as-code#track-your-infrastructure>. Luettu 4.4.2023.

8 State. Verkkoaineisto. HashiCorp. < https://developer.hashicorp.com/terraform/language/state>. Luettu 4.5.2023.

9 Install Terraform. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/install-cli#install-cli>. Luettu 4.4.2023.

10 Standard Module Structure. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/language/modules/develop/structure>. Luettu 4.4.2023.

11 Google Cloud resources. Verkkoaineisto. Google. <https://cloud.google.com/docs/overview#gcp_resources>. Luettu 5.4.2023.

12 Regions and zones. Verkkoaineisto. Google. <https://cloud.google.com/docs/geography-and-regions>. Luettu 6.4.2023.

13 Projects. Verkkoaineisto. Google. <https://cloud.google.com/docs/overview#projects>. Luettu 6.4.2023.

14 Set up GCP. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build#set-up-gcp>. Luettu 10.4.2023.

15 Service accounts overview. Verkkoaineisto. Google. <https://cloud.google.com/iam/docs/service-account-overview>. Luettu 6.4.2023.

16 Configuration Syntax. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/language/syntax/configuration#configuration-syntax>. Luettu 5.5.2023.

17 Arguments and blocks. Verkkoaineisto. HashiCorpl. <https://developer.hashicorp.com/terraform/language/syntax/configuration#arguments-and-blocks>. Luettu 5.5.2023.

18 Write configuration. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build#write-configuration>. Luettu 10.4.2023.

19 Initialize the directory. Verkkoaineisto. HashiCorp. < https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build#initialize-the-directory>. Luettu 12.4.2023.

20 Format and validate the configuration. Verkkoaineisto. HashiCorp. < https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build#format-and-validate-the-configuration>. Luettu 14.4.2023.

21 Create infrastructure. Verkkoaineisto. HashiCorp. < https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build#create-infrastructure>. Luettu 14.4.2023.

22 Change infrastructure. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-change#google-cloud-platform-change>. Luettu 17.4.2023.

23 Modify configuration. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-

started/google-cloud-platform-change#modify-configuration>. Luettu 17.4.2023.

24 Introduce destructive changes. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-change#introduce-destructive-changes>. Luettu 18.4.2023.

25 Destroy Infrastructure. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-destroy>. Luettu 18.4.2023.

26 Query Data with Output Variables. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-outputs#google-cloud-platform-outputs>. Luettu 18.4.2023.

27 Define outputs. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-outputs#define-outputs>. Luettu 18.4.2023.

28 Inspect outputs. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-outputs#inspect-outputs>. Luettu 18.4.2023.

29 Store Terraform state in a Cloud Storage bucket. Verkkoaineisto. Google. <https://cloud.google.com/docs/terraform/resource-management/store-state>. Luettu 8.5.2023.

30 Overview. Verkkoaineisto. GitHub. <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>. Luettu 27.4.2023.

31 The components of GitHub Actions. Verkkoaineisto. GitHub. <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#the-components-of-github-actions>. Luettu 1.5.2023.

32 Create an example workflow. Verkkoaineisto. GitHub. <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#create-an-example-workflow>. Luettu 1.5.2023.

33 Understanding the workflow file. Verkkoaineisto. GitHub. <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#understanding-the-workflow-file>. Luettu 1.5.2023.

34 Viewing the activity for a workflow run. Verkkoaineisto. GitHub. <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#viewing-the-activity-for-a-workflow-run>. Luettu 2.5.2023.

35 Automate Terraform with GitHub Actions. Verkkoaineisto. HashiCorp. <https://developer.hashicorp.com/terraform/tutorials/automation/github-actions#set-up-terraform-cloud>. Luettu 2.5.2023.

36 Back, Jozimar. 2023. Using GitHub Actions with Terraform on GCP. Verkkoaineisto. Medium. <https://jozimarback.medium.com/using-github-actions-with-terraform-on-gcp-d473a37ddbd6>. Luettu 2.5.2023.

37 Näyte lähdekoodista. Verkkoaineisto. GitHub. <https://github.com/hashicorp-education/learn-terraform-github-actions/blob/8fd256e51e302958c2d00bb09ba7f556461c0201/.github/workflows/terraform.yml>. 22.3.2023. Luettu 18.5.2023.