jamk

Pilvipohjaisen infrastruktuurin muuntaminen koodipohjaiseksi

Robin Adolfsson

Opinnäytetyö, AMK
Heinäkuu 2025
Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Adolfsson, Robin

Pilvipohjaisen infrastruktuurin muuntaminen koodipohjaiseksi

Jyväskylä: Jyväskylän ammattikorkeakoulu. Heinäkuu 2025, 40 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Jatkuvasti kehittyvässä pilvipalveluiden maailmassa, on siihen helppoa luoda infrastruktuuri ajattelematta tulevaisuuden käyttötarkoituksia. Käyttöasteen kasvaessa, tulee kuitenkin usein vastaan ongelmia, jos arkkitehtuuri on luotu ainoastaan alkuperäistä käyttötarkoitusta varten ja siihen on tarkoitus lisätä uusia komponentteja ja kasvattaa olemassa olevia resursseja vastatakseen lisääntyneeseen kuormaan.

Opinnäytetyön toimeksiantajana toimi Nodeon Finland Oy, jonka verkonvalvontaympäristössä todettiin kasvukipuja sen alkuperäisen toteutuksen ollessa toteutettu nopealla aikataululla, ajattelematta tarpeeksi tulevaisuuden tarpeita. Tavoitteena oli selvittää, miten olemassa olevaa pilvipalveluarkkitehtuuria voitiin kehittää skaalautuvammaksi ja ylläpidettävämmäksi, yhdenmukaistaen ja yksinkertaistaen sen hallintaa osana tuotteistettua ylläpitokokonaisuutta.

Opinnäytetyössä kartoitettiin toimeksiantajan nykyinen pilvipalvelupohjainen infrastruktuuri, sekä suunniteltiin havaintojen perusteella kehitystoimenpiteitä kustannustehokkuuden, skaalautuvuuden ja sujuvamman ylläpidon parantamiseksi. Selvitettiin lisäksi, miksi koodipohjainen infrastruktuuri on toimeksiantoon nähden sopiva edistymistapa, jolla voidaan paremmin taata tulevaisuuden kasvu, sekä millä koodipohjaisella infrastruktuurityökalulla on kannatta pyrkiä ongelmaa ratkaisemaan.

Usean alalla laajalti käytetyn työkalun vertailun perusteella selitetään miksi ja miten HashiCorpin tuottamaa Terraformia voidaan käyttää pilvipalvelutarjoaja Microsoft Azuren tuottaman palvelun hallinnoimiseen. Tutkittiin lisäksi, miten Terraformilla voi luoda modulaarisen kokonaisuuden, josta voi sujuvasti kehittää koodipohjaista infrastruktuuria käyttäen mallipohjia. Luotiin myös tavoitteellinen korkean tason arkkitehtuurikuva tulevaisuuden infrastruktuuria miettien, opinnäytetyön testiarkkitehtuurillisen kuvan lisäksi. Testiarkkitehtuuri ajettiin täysin koodipohjaisesti todelliseen pilvipalveluympäristöön.

Avainsanat (asiasanat)

Pilvipalvelut, Infrastruktuuri koodina, IaC, DevOps, IaaS

Muut tiedot (salassa pidettävät liitteet)

EI liitteitä.

jamk

Description

Adolfsson, Robin

Transforming a cloud-based infrastructure into infrastructure as code

Jyväskylä: JAMK University of Applied Sciences, July 2025, 40 pages.

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

In the constantly evolving field of cloud-based infrastructure, it is easy to create an environment without considering the future. As the usage of the platform increases, problems often arise with the architecture, if it was only ever designed for its original purpose and a need arises to add new components or scale existing resources to accommodate demand for increased capacity.

This thesis was commissioned by Nodeon Finland Oy, whose network monitoring environment experienced growing pains due to the original implementation having been created on a short notice, without properly taking into account future needs. The goal of the thesis was to explore how an existing cloud based architecture could be developed into a more scalable and maintainable cohesive package. Through means of simplifying and streamlining its management, creating a product which could be more easily managed.

As part of the thesis, the commisioner's cloud-based infrastructure was explored, the results of which were used to create plans for future development to solve issues of cost effectiveness, scalability as well as to simplify maintenance of the infrastructure. The thesis also examines why Infrastructure as Code is a suitable approach for this use case, supporting future scalability according to business growth, as well as finding out which tool is best suited for tackling the observed issues.

Based on a comparison of a multitude of tools widely used in the field of cloud-based infrastructure, this thesis explains why and how HashiCorp's Terraform is suited for managing services provided by Microsoft Azure. The thesis also investigates how Terraform can be applied to a modular approach that supports infrastructure development through code by way of templates. A high-level target architecture diagram for future infrastructure development goals was also created, in addition to the thesis' test architecture diagram. The test architecture was implemented exlusively using code, using an actual cloud environment.

Keywords/tags (subjects)

Cloud, Infrastructure as Code, IaC, DevOps, IaaS

Miscellaneous (Confidential information)

No confidential information.

Sisältö

Kuviot

Taulukot

1 Johdanto

Pilvipalvelutarjoajan alustaan pohjautuvassa infrastruktuurissa on usein haasteena ympäristön uudelleenkäytettävyys, skaalautuvuus ja yhdenmukaisena pysyvä rakenne, varsinkin sen elinkaaren kasvaessa. Tätä on pyritty ratkomaan luomalla sovelluskehityksen ja sovellusten käyttöönoton vaatiman infrastruktuurin hallinnalle yhtenäinen ajattelumalli, jota kutsutaan DevOps:ksi. Wakarun (n.d.) mukaan, sanasta DevOps ensimmäinen osa Dev (engl. Development) tarkoittaa kehitystä ja Ops (engl. Operations) tuotantoon vientiä.

DevOps-käytäntöihin kuuluu infrastruktuurin hallinnan puoliautomatisointi, joka johti koodipohjaisen infrastruktuurin (engl. Infrastructure as Code, IaC) käsitteeseen. IaC otettiin julkisutshetkestään nopeasti käyttöön alan suurimmissa organisaatioissa ja luokitellaan tänä päivänä teollisuusstandardiksi pilvipalvelupohjaisen infrastruktuurin hallinnassa. (Artač, Borovšak, Di Nitto, Guerriero & Tamburri 2017)

Opinnäytetyön tavoitteena oli kehittää olemassa olevan pilvipalvelupohjaisen tuotteen infrastruktuurista paremmin skaalautuva ja helpommin ylläpidettävä, ottaen huomioon jo käytössä olevien resurssien sujuva yliheitto uuteen arkkitehtuuriin. Koska tuotteen skaalautuvuus oli tärkeänä tavoitteena, otettiin opinnäytetyössä huomioon myös käyttövalmiin mallipohjan muodostaminen, jolloin skaalautuessa tuotteen uudet komponentit ovat helposti ymmärrettäviä, niiden seuratessa yhtä kaavaa, toimien ennalta-arvatusti ja yhdenmukaisesti toisiinsa verrattuna. Tarkoituksena oli myös säästää pilvipalveluiden kustannuksissa, sekä palvelun ylläpidon helpottamista.

Opinnäytetyössä pyrittiin myös ratkaisemaan olemassa olevan tuotteen eliniän aikana koettua ylläpidollista taakkaa käyttöjärjestelmien ja yleisten sovellusten päivityksissä ja konfiguroinnissa. Näiden koettiin olevan epäkäytännöllisiä pitää tietoturvan ja sovellustuen kannalta päivitettyinä kokonaisina virtuaalikoneina käyttöjärjestelmineen, joiden versiot ja taustalla olevat sovellukset saattoivat vaihdella keskenään.

Opinnäytetyön toimeksiannossa haluttiin pysyä yrityksen jo laajalti, sekä olemassa olevan tuotteen käyttämässä Microsoft Azure-pilvipalvelualustassa, käyttäen parhaaksi katsotulla tavalla sen tarjoamia ratkaisuja. Tähän pyrittiin sekä jo karttuneen kokemuksen vuoksi, että koska jo ajossa oleva tuote sijaitsi myös kyseisellä alustalla.

Opinnäytetyön toimeksiantaja, Nodeon Finland Oy kuvailee itseään älykkään infran monialaiseksi teknologia-asiantuntijaksi. Yrityksen ydinosaamisiin kuuluvat ohjelmistokehitys, IoT- ja tietoliikenneratkaisut sekä automaation, sähkön ja valaistuksen suunnittelu. Nodeon työllistää vuonna 2025 noin 50 asiantuntijaa ja sillä on vankka arvopohja, joka näkyy asiantuntijoiden päivittäisessä työssä. Nodeonilla on muutama toimipiste eri puolella Suomea, mutta pääkonttori sijaitsee Jyväskylässä. Yrityksellä on asiakkaina lukuisia kaupunkeja ja valtiollisia toimijoita yksityisten yritysten lisäksi. (Nodeon Finland n.d.)

Nodeonin pääliikealueet ovat älykkäiden kaupunkielinympäristöjen, älykkään liikkumisen ja infrastruktuurin tuottaman datan digitalisaation kehittäminen. Lisäksi yritys tarjoaa monialaisia asiantuntijapalveluita ja tarjoaa asiakkailleen kattavaa data-analytiikkaa ja älykkään infran ylläpitopalveluita tuote-edustusten lisäksi. Opinnäytetyö on suoritettu Nodeonin vakituisena työntekijänä tietoliikennesuunnittelijan roolissa. (Nodeon Finland n.d.)

2 Tutkimusasetelma

2.1 Tutkimuskysymykset

Nykyisessä toteutuksessa on todettu ylläpidollisia ja skaalautuvuuteen, sekä kustannustehokkuuteen liittyviä vaikeuksia. Opinnäytetyölle muodostui keskeisiksi tutkimuskysymyksiksi suoraan parantavien toimenpiteiden tutkimista seuraavilla kysymyksillä:

- Mitkä ovat nykyisen arkkitehtuurin heikkoudet?
- Mitkä ovat nykyisen arkkitehtuurin kehitystarpeet?
- Millä työkaluilla kehitystoimenpiteitä kannattaa toteuttaa?

Kysymyksiin vastaamalla pyritään luomaan selvitys siitä, miten olemassa olevan pilvipalveluarkkitehtuurin voi soveltaa koodipohjaiseksi, jolloin sen hallittavuus ja skaalautuvuus helpottuu, kun infrastruktuurin mallista on luotu uudelleenkäytettäväksi ja versiohallittava koodikanta.

2.2 Tutkimusmenetelmä

Opinnäytetyössä tutkitaan yrityksen olemassa olevan tuotteen digitaalisen infrastruktuurin heikkouksia, sekä ainakin osittain automatisoimaan sen ylläpitoa, jatkuvaa kehitystä ja skaalautumista käyttäen pilvipalvelualustan tarjoamia resursseja. Heikkouksissa viitataan sekä kokemuspohjaisiin puutteisiin, että lähteisiin viitaten tietoturvallisen ympäristön ylläpidollisia vaatimuksia. Selvitetään myös IaC:n tuomia hyötyjä mallipohjaisessa toteutustavassa ja miten olemassa olevat resurssit siirretään mahdollisimman pienellä vaivalla ja käyttökatkolla uuden arkkitehtuurin lomaan.

Opinnäytetyössä sovelletaan monimenetelmäistä tutkimusmenetelmää, jossa korostuu laadullinen tutkimusmenetelmä olemassa olevan tuotekokonaisuuden parantamiseksi vertailemalla eri IaC-vaihtoehtoja keskenään vastatakseen parhaiten toimeksiannosta muodostuneisiin tutkimuskysymyksiin. (Monimenetelmäisyys n.d.)

3 Olemassa oleva pilvipalveluarkkitehtuuri

3.1 Nykyisen toteutuksen käyttämät tuotteet ja palveluntarjoajat

Nodeon-ylläpidon käytössä oleva pilvipalveluinfrastruktuuri on jo useamman vuoden tuotantoympäristössä käynnissä ollut tuotekokonaisuus, joka vastaa satojen tietoliikenne- ja muiden laitteiden valvonnasta ja vianmäärittelystä. Kokonaisuus muodostuu pilvipalvelutarjoajan ympäristössä sijaitsevasta usean palveluntarjoajan tuotepaketista, jonka tehtävänä on tuottaa ylläpidollisesti arvokasta tietoa ja hälyttää aktiivisista vioista asiakasympäristöissä.

Microsoft Azure on yhdysvaltalaisen Microsoft Corporationin vuodesta 2010 tarjoama pilvipalvelualusta, joka tarjoaa kattavalla tuotevalikoimalla maailmanlaajuisesta konesaliverkostosta ja sisäänrakennetulla tietoturvaratkaisulla useaa palvelumallia. Nodeonilla on käytössä Azureen sijoittuva IaaS-mallinen palvelu, jossa laskenta- tallennus ja tietoverkkotekniikka on tarjoamassa tuotteessaan täysin Microsoftin tarjoamassa pilvessä. (What is Azure? n.d.)

Azure oli vuoden 2024 lopussa 21 % markkinaosuudella maailman toiseksi käytetyin pilvipalvelualusta, Amazonin vastaavan tuotteen jälkeen. Verratessa kolmanneksi suurimman palveluntarjo-

ajan 12 % markkinaosuuteen, on Azurella kuitenkin valtava osuus alati kasvavista pilvipalvelupohjaisen infrastruktuurin markkinoista. Koska pilvipalveluiden markkinat tuottivat 330 miljardin dollarin edestä liikevaihtoa maailmanlaajuisesti, tekee se alasta hyvin kilpailullisen ja jatkuvasti kehittyvän. (Richter 2025)

Asiakasverkkojen valvontaan käytetään pääasiassa Zabbix-verkonvalvontajärjestelmää. Zabbix on moneen taipuva, pääasiassa ICMP Ping ja SNMP-verkonvalvontaprotokollia käyttävä verkonvalvonta- ja vianseurantatyökalu. Zabbix on Latvialaisen Zabbix LLC:n monen maailmanlaajuisesti tunnetun ja miljardiluokan organisaation käytössä ja sitä on kehitetty jo vuodesta 2001. Zabbixin vahvuuksiin kuuluvat muun muassa avoimen lähdekoodin tuoma toiminnan läpinäkyvyys ja mahdollinen räätälöinti käyttötarkoitusta varten. (About Zabbix LLC n.d.)

Zabbix on saatavilla useassa eri muodossa ja usealle käyttöjärjestelmälle. Opinnäytetyön aiheessa on käytössä pääasiassa komponentit Zabbix Server ja Zabbix Proxy. Kumpikin keskeinen komponentti vaatii tietokannan toimiakseen, joista on valittavissa joko MySQL/MariaDB tai PostgreSQL. Näiden lisäksi Zabbix Proxy voi käyttää kevyempää tietokantaa SQLite3. (Zabbix Requirements n.d.)

Nodeonin Azuressa sijaitsevassa verkkovalvontaan käytetyssä pilvipalveluympäristössä on pääasiallisessa käytössä virtuaalipalvelimia Zabbix-palvelimille, Zabbix-proxyille, sekä niiden käyttämä MySQL-tietokantapalvelin jokaisen Zabbix-instanssin omalla tietokannalla. Jokainen palvelimista vaatii näiden lisäksi tietoliikenteen resursseja toimiakseen, jotka sijaitsevat samassa Azuren ympäristössä.

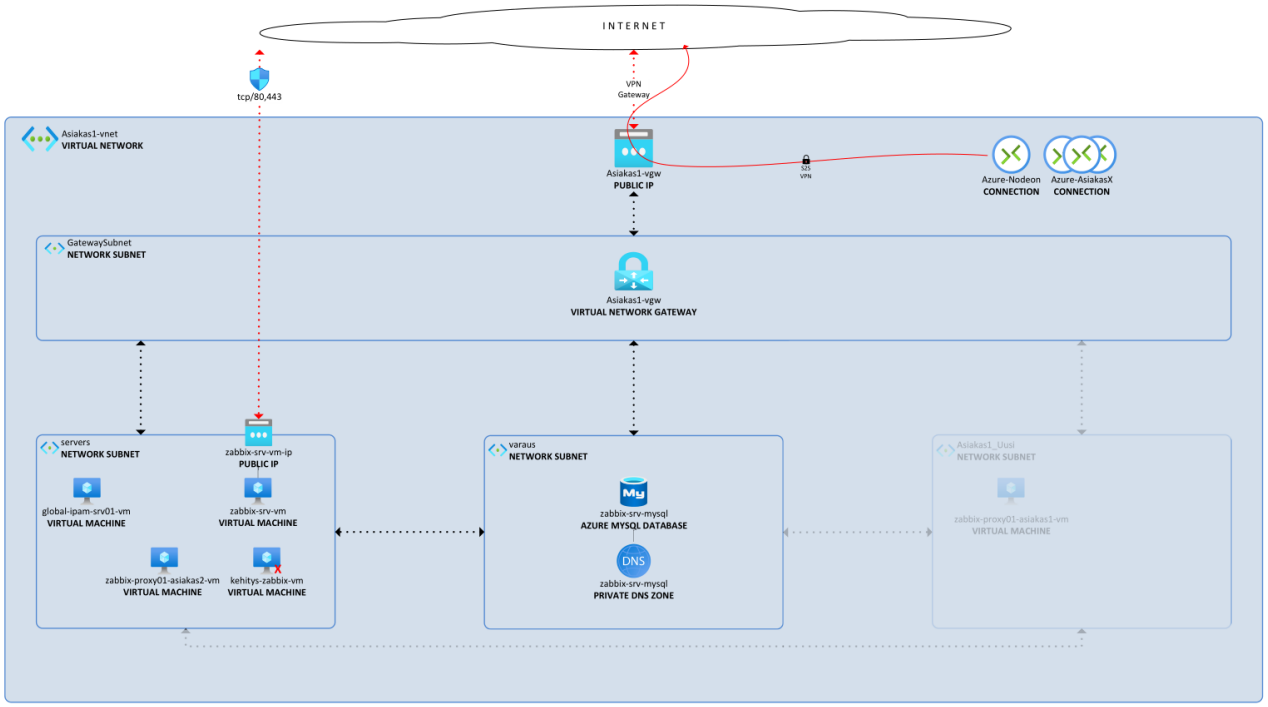
3.2 Nykyisen arkkitehtuurin kartoitus

Toimeksiantoon kuului nykyisen pilvipalveluarkkitehtuurin kartoittamisen lisäksi sen heikkouksien määrittäminen alkukartoituksen perusteella, sekä selvitys vaihtoehdoista sen parantamiseksi. Toiveena oli säilyttää verkonvalvontatyökalun tähän asti tuottama data ja sen käyttäjäoikeudet mahdollisimman pitkälle, sekä tehdä mahdollinen yliheitto uuteen arkkitehtuuriin mahdollisimman pienellä käyttökatkolla.

Nykyinen arkkitehtuuri on alun perin kehitetty lyhyehköllä aikataululla vuonna 2022 asiakasverkon valvontaa varten, eikä sitä alun perin suunniteltu moniasiakasympäristöksi. Alkuperäisen asiakkaan laitteita valvotaan itse pääpalvelimelta, mutta uudempaa asiakkuutta varten perustettiin vuonna 2023 Zabbix Proxy-palvelin jakamaan valvonnan aiheuttamaa kuormaa, sekä eristämään uuden asiakkaan valvontapalvelin muusta ympäristöstä. Proxy-palvelin kuitenkin raportoi tietonsa pääpalvelimelle, joka vuorostaan taltioi sen tietokantaan. Tämän lisäksi alkuperäisen asiakkaan valvontaa varten on perustettu toinen proxy-palvelin, jolle on luotu oma verkkosegmenttinsä erillään muista virtuaalipalvelimista. Uutta proxy-palvelinta ei ole kuitenkaan ole vielä otettu käyttöön, joten sen voi käytännössä jättää huomioimatta.

Arkkitehtuurin kartoitustyössä ja suunnittelussa käytettiin pääasiassa Microsoft Visio-työkalua, joka on varsinkin tietoliikennesuunnittelijoiden yleisessä käytössä oleva Microsoftin laajempaan Office-sovelluspakettiin kuuluva diagrammien ja vuokaavioiden piirto-ohjelma. Koska sekä Azure, että Visio ovat Microsoftin omia tuotteita, on Visioon lisäksi saatavilla Microsoftilta ikonipaketteja ja valmiita suunnittelupohjia Azure-suunnittelua varten. (Compare Visio versions and features n.d.)

Nykyinen arkkitehtuuri kartoitettiin Azuren käyttöliittymästä saatavilla automaattisesti generoituvia topologioita, olemassa olevaa dokumentaatiota, virtuaalipalvelimilta saatuja tietueita ja Visiota käyttäen. Kartoituksen perusteella luotiin kuvio 1, joka kuvastaa nykytilan arkkitehtuuria.



Kuvio 1. Yleistason kuva nykyisestä arkkitehtuurista

3.2.1 Pääsynhallinta ja verkon segmentointi

Arkkitehtuurissa on havaittavissa kaikkien resurssien olevan samassa virtuaalisessa verkossa, eri asiakkuuksien resurssien käyttävän yhteisiä aliverkkoja keskenään ja samantyyppisten resurssien olevan pääasiassa vain nimellä eroteltuja toisistaan. Jokainen tekijöistä hankaloittaa ympäristön ylläpitoa, kun kunnon loogista erottelua ei käytännössä tapahdu.

Kun infrastruktuuri luotiin, sijoitettiin kaikki resurssit yhteen virtuaaliverkkoon, johon luotiin käyttötapauskohtaisesti jokaiselle käyttötarkoitukselle oma verkkosegmentti. Verkkosegmenttien välille ei kuitenkaan asetettu luomishetkellä kunnon suojausta Azuren työkaluilla, vaan esimerkiksi virtuaalipalvelimet olivat käytännössä samassa aliverkossa, jolloin palvelimilla on lähtökohtaisesti vapaa pääsy toisiinsa. Virtuaalipalvelimet suojattiin kuitenkin julkiverkon, sekä toistensa väliseltä kommunikoinnilta palvelimen käyttöjärjestelmään asetetuilla suodatuslistoilla, eivätkä olleet jatkuvan ylläpidon vuoksi haavoittuvassa tilanteessa. Mikäli asiakasympäristöjen IP-osoitteistus olisi muuttunut tai järjestelmään liitettäisiin uusi asiakas, oli syntynyt samalla tarve varmistaa suodatuksista myös muuttuneiden IP-osoitteiden olevan sallittujen IP-osoitteiden sisä- tai ulkopuolella.

Lisäksi olisi ollut huomattavasti hankalampaa estää pääsyä toisen asiakkaan laitteisiin, mikäli kahdella tai useammalla asiakkaalla olisi verkkoja samoilla yksityisillä IP-alueilla.

Pääpalvelimella on käytössään myös IP-osoitteiden maantieteellinen suodatus, jossa käytetään julkiverkon IP-osoitteiden tietokantaa evätäkseen pääsy palvelimelle paitsi sallituista maantieteellisistä alueista, estäen yhdistämisyritykset muualta kuin Suomesta tai Ahvenanmaalta. Tämä aiheuttaa kuitenkin webkäyttöliittymän salauksen sertifikaattien päivityksen kannalta ongelmia, jonka vuoksi SSL-sertifikaatin päivityksiä varten on käytössä varta vasten räätälöity skripti, joka hetkellisesti poistaa maantieteellisen suodatuksen sertifikaatin päivityksen ajaksi. Kyseinen maantieteellinen IP-osoitteiden suodatus on teoriassa hyvä idea, joka parantaa turvallisuutta, mutta ylläpidollisesti se tuo turhia lisähaasteita, varsinkin kun Azurella on omia työkaluja tämän implementoimiseksi.

Koska infrastruktuurikokonaisuus on osa yhtä tai useampaa ylläpitosopimusta, on myös tarve päästä hallinnoimaan valvontajärjestelmän valvomia, eli ylläpitoasiakkaiden laitteita. Tämän vuoksi on virtuaaliseen Azure-yhdyskäytävään luotu VPN-tunneli myös Nodeonin oman verkon yhdyskäytävään, ylläpitoyhteyksien muodostamiseksi asiakasverkoissa sijaitseviin laitteisiin. Käyttäjänhallinta kyseiselle VPN-tunnelille on hajautettu muusta infrastruktuurista, koska sitä suoritetaan Azuren sijasta Nodeonin omalla palomuurilla paikallistunnuksia hyödyntäen, sallien yhteydet vain tarkoitetuilla käyttäjätileillä. Tämä tarkoittaa, että aina kun jokin asiakkaista tekee muutoksia IP-osoiteavaruuteensa, tai valvontaan ja ylläpitoon liitetään uusi asiakas, on tehtävä valvonta- ja ylläpitokokonaisuuteen liittyviä muutoksia useassa kohteessa, kun tätä pystyisi hallitsemaan keskitetysti Azuren valvontajärjestelmästä.

3.2.2 Resurssien mitoitus suhteessa varsinaiseen kuormitukseen

Nykyisen toteutuksen heikkous näkyy myös resurssien valinnoissa, jotka ovat joko ylimitoitettuja yksinkertaisten virtuaalipalvelinten tapauksissa, tai niin huonosti optimoituja, että kustannukset ovat pahimmillaan yli kaksinkertaiset verrattuna siihen, mitä hyvin suunnitellusta käyttötapauksesta voi odottaa.

Yksinkertaisille valvonnan proxyille on luotu lähes yhtä tehokkaat virtuaalipalvelimet, kuin itse pääpalvelimelle, joka kuitenkin vastaa valvonnan lisäksi myös web-käyttöliittymästä, sekä tietojen

taltioinnista ja hausta tietokantapalvelimelta. Tässä ei sinänsä ole vikaa, sillä Zabbixen (Zabbix Requirements n.d.) mukaan pienikin instanssi vaatii 2 suoritinydintä ja 8 gigatavua välimuistia. Käytännössä tämä on kuitenkin ylimitoitettu, sillä yli 900 verkkolaitetta valvovalla proxylla on käyttänyt edellä mainituista resursseista ainoastaan noin puolet suorituskapasiteetistaan ja alle kolmanneksen sille allokoidusta välimuistista, kuten kuvion 2 esittämästä graafista voi päätellä.

Kuvio 2. Tuotannossa olevan palvelimen resurssikulutus 30 päivän ajalta

Virtuaalipalvelinten lisäksi tietokantapalvelin, joka kullakin palvelulla on käytössään, on jouduttu rankasti ylimitoittamaan, koska Zabbixen ja MySQL-tietokannan välistä keskustelua ei ole lainkaan optimoitu. Tietokantapalvelimelle suuntautuvien hyvin luku- tai kirjoitusintensiivisten toimintojen aikana, palvelin oli historiallisesti jopa jumittunut, joten sen laskentakapasiteettia nostettiin ennaltaehkäisevästi. Palvelinta olisi kuitenkin voinut Cioen (2021) mukaan optimoida monella tapaa Zabbixen kanssa paremmin toimivaksi ennen resurssien kasvattamista.

3.2.3 Dokumentoinnin taso

Tämänhetkinen arkkitehtuuri on sekava ja vaikeasti ylläpidettävä, sillä kaikki resurssit on perustettu yhteen ainoaan resurssiryhmään, heikoilla Azuren sisäisillä dokumentaatioilla ja usein puuttuvilla leimoilla (tags). Uusien resurssien lisääminen on näin epämääräisessä ympäristössä luontevinta tehdä verkkoselaimen käyttöliittymää käyttäen, koska olemassa olevia resursseja ei ole luotu yhtenäistä kaavaa seuraamalla, eikä niitä ole kategorisoitu yhtenevästi. Azuren verkkoselainpohjainen graafinen käyttöliittymä on kuitenkin huonotyökalu silloin, kun tahdotaan luoda yhteneväisillä nimillä tai muilla tiedoilla resursseja, sillä ne pitää käytännössä kopioida ja liittää yksi kohta kerrallaan tekstikenttiin, muistaen muuttaa toisistaan poikkeavat kentät, kuten osa nimestä tai IP-osoite. Tämä melkein liialliseksikin luokiteltava vapaus tuo usein myös uusille resursseille pieniä henkilökohtaisia eroavaisuuksia riippuen siitä, kuka ne on tehnyt ja mitä ajatusmallia sinä hetkenä käyttäen. Nämä poikkeamat eivät ole toivottuja laajassa ympäristössä, jossa tahdotaan yhteneväisesti hallinnoida ja katsastaa resursseja kokonaisuutena.

Moni yhteiskäytössä oleva resurssi on nimetty alkuperäisen käyttötarkoituksensa mukaan, mikä on käytännössä sidottu ensimmäiseen asiakkuuteen. Tämä ei kuvasta tarpeeksi hyvin esimerkiksi virtuaalisen yhdyskäytävän – joka myös toimii VPN-yhdyspisteenä jokaisen valvottavan asiakkaan verkkoon – roolia keskeisenä resurssina jokaiselle asiakkuudelle.

Järjestelmän arkkitehtuurista löytyi vanhentunut, tavoitteellinen toiminnallisuuden kuvituskuva, joka ei kuitenkaan tavoitteellisuutensa vuoksi pitänyt tosiasiassa paikkaansa. Kuva osoitti silti, että tarkoitus on jo ennestään ollut erotella resursseja paremmin toisistaan, sekä evätä Azuren puolella virtuaalipalvelinten pääsyt niille tarkoittamattomiin verkkoihin nykyisen suodatuslistan sijaan.

4 Arkkitehtuurin suunnittelu ja työkalun valinta

4.1 Uuden arkkitehtuurin suunnittelussa käytetyt työkalut

Visual Studio Code (VSC) on Microsoftin kehittämä, avoimeen lähdekoodiin perustuva monipuolinen ohjelmointiin tarkoitettu koodieditori. VSC tukee useita ensimmäisen ja kolmansien osapuolten laatimia lisäosia, sekä sisäänrakennetun komentotulkin, jolloin ohjelmoinnin ja testauksen pystyy suorittamaan samasta sovelluksesta, joka mahdollistaa koodista nopeasti vikojen löytämisen.

Opinnäytetyön kaikki ohjelmointi on tehty VSC:tä ja siihen saatavia lisäosia hyödyntäen, nopeuttaen koodin kirjoittamista ja virheenkorjausta.

Aiemmin ainoastaan Microsoft Windowsille saatava, nyt avoimeen lähdekoodiin ja useampaan käyttöjärjestelmään saatavilla oleva PowerShell on komentotulkki, joka perustuu moniin muihin komentotulkkeihin olio-ohjelmoinnista lainattuun syntaksiin. Vaikka kaikki opinnäytetyössä tutkittavat komentotulkkiin perustuvat teknologiat toimivat muillakin komentotulkeilla, on PowerShell helpoiten integroitavissa VSC:n käyttöliittymään, sekä Microsoftin tuotteena saa kattavaa ensimmäisen osapuolen tukea VSC:n lisäksi Azuren kanssa toimimiseen. (PowerShell 2025)

4.2 Uuden arkkitehtuurisuunnitelman lähtökohdat

Kun vanhan arkkitehtuurin heikkoudet oli tunnistettu, uutta pilvipalvelupohjaista arkkitehtuuria alettiin suunnittelemaan ratkaisukohtaisesti niiden perusteella. Uudessa arkkitehtuurimallissa pyrittiin ratkomaan pääasiassa kustannustehokkuuden ongelmat, siirtämään tietoturva-asetukset pois palvelimilta Azuren natiiveihin ratkaisuihin läpinäkyvyyden edistämiseksi hallintapaneelista, sekä luomaan helposti luettavia ja päivitettäviä, kuitenkin vielä tässä vaiheessa tavoitteellisia korkean tason arkkitehtuurillisia kuvia järjestelmän tulevasta toimintamallista. Tarpeellista oli myös tehdä järjestelmästä mahdollisimman helposti ylläpidettävä useamman tahon toimesta sellaisella tavalla, että muutoksista jää helposti tunnistettava lokimerkintä haavoittuvuuksien ja mahdollisten väärinkäyttöjen tunnistamiseksi.

Dokumentoinnin tueksi siirryttiin jo suunnitteluvaiheessa käyttämään osittain Microsoftin ehdottamaa formaattia palveluiden nimeämisessä mahdollisimman hyvin, joka toisi läpinäkyvyyttä resurssien käyttötarkoituksesta ja soveltamisalasta, josta olisi valtavasti hyötyä varsinkin järjestelmän kasvaessa. (Define your naming convention 2025)

4.3 Arkkitehtuurisuunnitelma

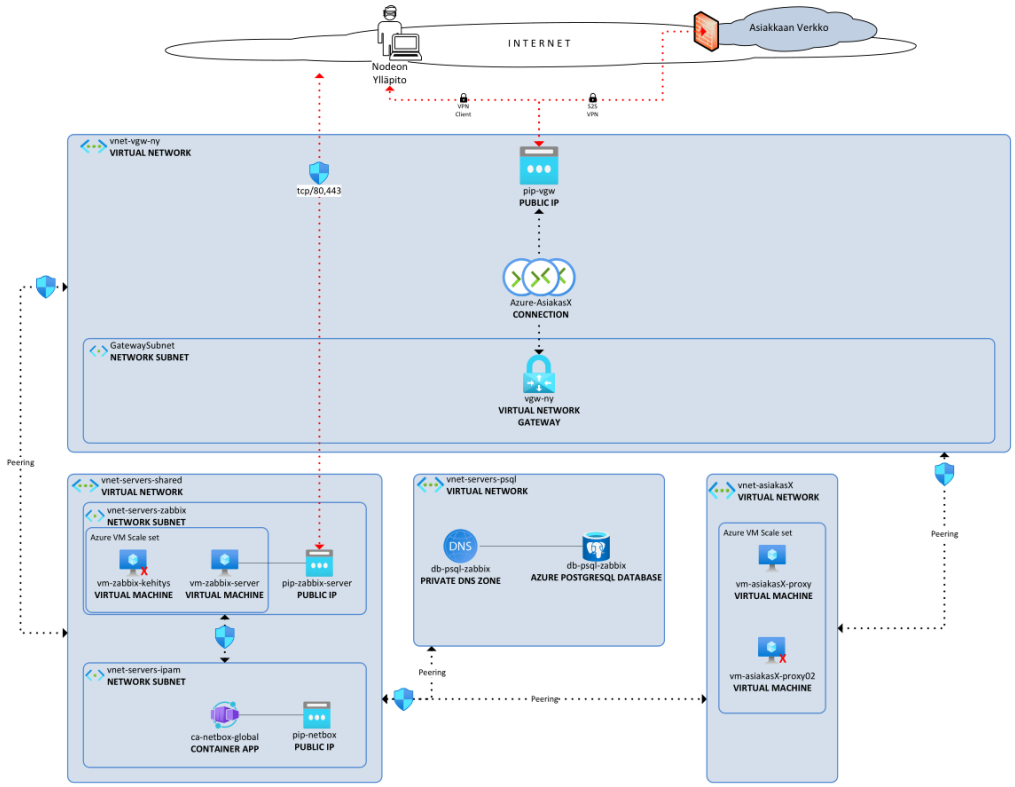
Suunnitteluvaiheessa pyrittiin ongelma kerallaan ratkaisemaan nykyisen suunnitelman puutteet, jotka havaittiin arkkitehtuurisuunnittelun esivaiheessa. Sekä pääsynhallinnan siirto palvelimien suodatuslistoilta Azuren natiiveihin toimintoihin, että siihen liittyvän ylläpidon helpottamisen pys-

tyi luontevasti tekemään hyödyntäen Azuren Network Security Group (NSG)-toimintoa, kun eri asiakkuuksien käyttämän resurssit eroteltiin omiin virtuaaliverkkoihinsa. NSG:n implementointi onnistuu parhaiten siten, että erotetaan laitteet eri verkkoihin pelkkien verkkosegmenttien sijaan ja verkkojen välille asetetaan suodatukset, joissa sallitaan vain tarpeellinen liikenne verkkojen välillä. NSG:t toimivat käytännössä sekä palomuurin, että nykyisten suodatuslistojen tavoin, jossa lisätään sekä lähteville IP-osoitteille, että niiden päämäärille listaukset, käytössä olevan tietoliikenneprotokollan luokituksen ja porttinumero lisäksi. Suodatuslistaan tulee esimerkiksi MySQL-tietokantapalvelinliikenteelle virtuaalipalvelimen IP-osoite lähtevän liikenteen osoitteeksi, MySQL-tietokantapalvelimen IP-osoite päämääräksi ja protokollaksi TCP/3306. (Overview of Azure network security 2025)

Tässä vaiheessa otettiin myös huomioon IP-osoitteiden hallintasovelluksen lisäämistä Azure Container App-menetelmää hyödyntäen kontitettuna (engl. containerised) sovelluksena, joka on Azuren tarjoama alusta kontissa ajettavien sovellusten automaattiseen skaalaukseen käyttöasteen perusteella. (Azure Container Apps overview 2025)

Kontitettujen sovellusten lisäksi suunniteltiin nykyisille virtuaalipalvelimille Azuren virtuaalipalvelinten skaalautuvuus (engl. Virtual Machine Scale Set, VMSS), joka on joukko automaattisesti tai manuaalisesti skaalautuvia virtuaalipalvelimia, jolla pystyy sekä varmistamaan palvelimille korkeamman saavutettavuuden vikatilanteissa, että suorittamaan niin sanottuja rullaavia päivityksiä (engl. rolling updates), jotka takaavat lähes täyden saavutettavuuden päivitystilanteissa, kun päivitykset tehdään ensin varalla olevalle virtuaalipalvelimelle, joka päivitysten jälkeen muutetaan primääriksi. Tämä mahdollistaa päivitysten suorittamisen hyvin pienellä käyttökatkolla tuotantoympäristöön, kun toiminnalliset testaukset tehdään vararoolissa olevaan virtuaalipalvelimeen. Vaihdon jälkeen päivitetään aiemmin primäärinä ollut virtuaalipalvelin ja jätetään se passiiviseen rooliin, toimien seuraavalla kerralla päivitysten ensisijaisena kohteena. (Configure rolling upgrades on Virtual Machine Scale Sets 2025)

Kehitystoimenpiteet huomioon ottaen, luotiin Visiolla kuviossa 3 ilmenevä korkean tason arkkitehtuurikuva suunnitellusta toteutuksesta, jota päivitetään tarpeen mukaan aina kun tehdään muutoksia, jotka vaikuttavat kokonaisarkkitehtuuriin, eli käytännössä jokaisen ylläpitoasiakkaan käyttämiin resursseihin.



Kuvio 3. Korkean tason kaavio suunnitellusta arkkitehtuurista

4.4 Koodipohjainen infrastruktuuri

Koodipohjaisen infrastruktuurin, (engl. Infrastructure as Code, IaC) tärkein käyttö on toimia lähdekoodin lailla takaamaan infrastruktuurin ennalta-arvattavuus, replikointi ja versionhallinta. Koodipohjaisen infrastruktuurin filosofia on samantyyppinen, kuin avoimen sovelluksen lähdekoodilla ja se toimii pohjapiirustusten tai mallipohjan lailla, luoden joka ajolla saman lopputuloksen. Erona on se, että koodipohjainen infrastruktuuri ajetaan yleensä pilvipalveluntarjoajan rajapintaan, jolloin se luo tai muokkaa valittuja resursseja virtuaalipalvelimista palomuureihin ja laajaan valikoimaan muitakin resursseja. Tämä ajattelumalli luotiin, kun tahdottiin yhtenäistää infrastruktuurin hallinta sovellusten kehittäjien ja operatiivisten tekijöiden käyttämään, niin sanottuun DevOps-ajattelumalliin. (Artač, Borovšak, Di Nitto, Guerriero & Tamburri 2017)

Infrastruktuurin kääntämisestä koodipohjaiseksi on valtavasti hyötyä myös muille, kuin sovellusten kehittäjille pohjapiirustuksen tai sabluunan roolissa toimimisen vuoksi. Ajattelumallia hyödyntäen voi luoda mille tahansa sovellukselle, tai sovelluskombinaatiolle koodipohjaista infrastruktuuria

käyttäen määrittämänsä infrastruktuurin joko testi- tai tuotantoajoa varten. Koodipohjainen infrastruktuuri onkin Quattrocchin ja Tamburrin mukaan (2022) jo vuonna 2022 kypsynyt osaksi alati kehittyvää ohjelmistotuotannon alaa, sekä merkittävä osa lähitulevaisuuden kehitystä.

Valmiin tuotteen, joka voi käytännössä sisältää mitä tahansa ohjelmistoa, elinkaaren pitkittämisen kannalta osalta parhaisiin käytäntöihin kuuluu myös taustalla olevan infrastruktuurin dokumentointi. Koodipohjaisella infrastruktuurilla tämä tapahtuu melkein luonnostaan, kun infrastruktuurin jokainen komponentti on määritetty koodipohjassa ja koodin uudelleenkäytettävyys modulaarisuuden kautta tapahtuu lähes itsestään, kun koodipohjaisen infrastruktuurin kokonaisuus muodostuu koodipätkistä, joista jokainen kuvastaa eri resurssia. Koodin ajohetkellä kommunikoidaan joko suoraan pilvipalvelutarjoajan rajapintaan, hallinnoiden resursseja tai sillä luodaan mallitiedosto, jota myöhemmin voi samantyyppisesti ajaa. Kun infrastruktuuria hallinnoidaan aina koodipohjaisesti, saadaan myös koodin määrittelemä lopputulos, jolloin koodimuuttujia ja iterointia käyttäen saadaan luotua useampia samankaltaisia resursseja yhdellä komennolla. Täten luodaan massana useita resursseja, jotka poikkeavat toisistaan vain tarvittavan määrän, kuten IP-osoitteissa ja tunnistetiedoissa, mutta pääasiallisesti vältytään inhimillisiltä virheiltä kuten olennaisilta kirjoitusvirheiltä. Lisäksi on usein helpompaa oikolukea vain muutaman kriittisen koodimuuttujan arvo, kuin verkkokäyttöliittymän kautta kyseisen resurssin hallintapaneeli kokonaisuudessaan.

4.4.1 Imperatiivinen ja deklaratiivinen lähestymistapa

Jokaisen malli- tai koodipohjaisen infrastruktuurin hallintamenetelmän voi kategorisoida joko imperatiiviseksi tai deklaratiiviseksi malliksi. Nämä saavat nimensä ohjelmoinnin aatemaailmasta, jossa puhutaan imperatiivisista- ja deklaratiivisista ohjelmointiparadigmoista. Brzychczyn, Szpyrkan, Korskin ja Nalepan (2023) mukaan, imperatiivinen ohjelmointi tarkoittaa käytännössä sitä ohjelmoinnin edistymistapaa, jossa kerrotaan vaihe vaiheelta koko menetelmä, jolla lopputulokseen aiotaan päästä manipuloimalla järjestelmää vaihe kerrallaan. Deklaratiivisissa ohjelmointikielissä ei tarvitse kertoa koko prosessia, vaan kun koodi ajetaan, imperatiiviset välivaiheet lisätään sitä ajavan taustajärjestelmän toimesta syntyneeseen lopputulokseen. Näin deklaratiivisen kielen toimintona on vain kertoa mihin lopputulokseen koodilla on tarkoitus päästä, ei välttämättä täysin miten järjestelmän on tämän saavutettava. Tämä tuo paitsi helppokäyttöisyyttä, myös turvallisuutta koodin ajoon, kun välivaiheet luodaan tarkkaa järjestystä ja säännöstöä seuraten.

4.5 Koodipohjaisen infrastruktuurin vaihtoehdot

IaC-maailmassa, imperatiivinen metodi on käytännössä suora Azuren konfiguraatio käyttäen joko selainpohjaista käyttöliittymää, komentokehotetta tai komentokehotteen automaatiota PowerShell-skriptauksen kautta. Imperatiivinen lähestymistapa on yksityiskohtaisemmin räätälöitävissä, koska jokaisen parametrin päättää käyttäjä tai käyttäjän luoma skripti. Käytännössä myös Azuren web-käyttöliittymä, jonka kautta nykyinen arkkitehtuuri on luotu, on imperatiivinen. Tämä on osasyy siihen, että nykyinen toteutus on sekava, kun se on rakennettu vaihe kerrallaan seuraamatta yhtenäistä kaavaa.

Deklaratiiviset metodit eroavat siten, että niissä käytetään työkaluja, joille kerrotaan mihin lopputulokseen tahdotaan päästä käyttäen parametreja. Työkalu joko luo resurssin tämän pohjalta, tai linkittää jo luotuja resursseja yhteen käyttäen sille annettuja olemassa olevien resurssien parametreja. Deklaratiiviset työkalut käyttävät Azuren tapauksessa poikkeuksetta jonkin sortin mallitiedostoa, jonka perusteella työkalu hallinnoi resursseja, yleensä silti Azuren rajapinnan kautta. Tämä tarkoittaa sitä, että jokainen työkalu on poikkeuksetta Azuren rajapinnassa kiinni, mutta rajapintayhteyden kompleksisuutta voidaan päättää työkalun valinnalla.

Koska imperatiivisille työkaluille ei ainakaan resurssienhallinnassa ole Nodeonin käyttötapauksessa varteenotettavaa tarvetta niiden tuomien lisäkompleksisuuksien ja ylläpidollisten haasteiden vuoksi, verrataan tässä opinnäytetyössä pääosin deklaratiivisten työkalujen soveltuvuutta Nodeonin arkkitehtuurin hallintaan.

Sen lisäksi, että opinnäytetyössä mainitut varteenotettavat vaihtoehdot ovat deklaratiivisia, ovat useimmat niistä myös inkrementaalisia tai ainakin konfiguroitavissa siksi. Inkrementaalisuus tarkoittaa tässä asiayhteydessä sitä, että tehdään ainoastaan muutoksia sitä vaativille resursseille, koskematta muutoksen kannalta epäolennaisiin resursseihin. Ellei konfiguraatiotiedostoissa ole tehty olemassa olevalle ja työkalun tiedossa olevalle resurssille muutoksia, ei niitä myöskään yritetä ajaa. Tämän lisäksi useimmat työkalut osaavat etukäteen kertoa, mitä muutoksia ne aikovat tehdä, pyytäen käyttäjän hyväksyntää ennen itse komentojen ajoa, tuoden turvallisuutta varsinkin tuotannossa olevan ympäristön hallinointiin.

4.5.1 Microsoftin ARM-mallitiedostot

Azuren vanhin mallipohjainen ratkaisu on ARM-mallitiedosto, jonka tarkoituksena on toimia sabluunana uusien resurssien luontiin ja hallintaan. ARM-mallitiedostot käyttävät JSON-muotoa, joka on suhteellisen helposti luettavissa sekä ihmisille, että ohjelmistoille. JSON ei kuitenkaan tue kommentteja ja paisuu muutenkin helposti vaikealukuiseksi ihmiselle monimutkaisuutensa vuoksi. ARM-mallitiedoston JSON-tiedostoja voi kuitenkin modularisoida, luoden jokaiselle resurssille oman, helpompilukuisemman tiedoston pienemmän yksilöllisen kokonsa vuoksi. ARM-mallitiedostoja ei voi käyttää kuin Azuressa ja mallitiedostoa on pakko aina päivittää sen odotetun rakenteen mahdollisesti muuttuessa Microsoftin suorittamien päivitysten myötä Azure-pilvipalvelualustassaan. (What are ARM templates? 2025)

4.5.2 Pulumi

Pulumi on työkalu, jonka ylpeydenaiheena on toimia juuri sillä koodikielellä, mitä käyttäjä on tottunut jo käyttämään. Pulumin tuettuja kieliä on muun muassa Python, TypeScript, C# ja Pulumi YAML. Koska Pulumi tukee imperatiivisia koodikieliä, on se myös pohjimmiltaan imperatiivinen menetelmä hallinnoida pilvipalveluresursseja. Muiden vaihtoehtojen tapaan, Pulumille ilmoitetaan jokaisella tukemallaan koodikielellä pääasiassa, minkälaisen arkkitehtuurin tahtoo saavuttaa ja Pulumi toteuttaa sen automaattisesti, lisäten toivotut parametrit muuttujien perusteella. Pulumi toimii laajalti imperatiivisesti silloin, kun sitä komennetaan imperatiivista koodikieltä käyttäen. Pulumia voi kuitenkin käskeä deklaratiiviseen tapaan käyttäen merkintäkieltä, kuten heidän omaa vahvasti nimensäkin perusteella YAML-pohjaista Pulumi YAML:a. (How Pulumi Works n.d.)

Pulumin sekä voimakkuus että heikkous – riippuen käyttäjästä – onkin koodipohjainen edistymistapa pilviresurssien hallinnassa. Koska Nodeonilla on pilvipalvelun tarkoitus olla ylläpidettävissä muidenkin, kuin ohjelmistosuunnittelijoiden tahosta, tuo tämä turhaa lisäkompleksisuutta käyttötarkoitukseen verrattuna. Pulumi vaatii myös täyden toiminnallisuuden saavuttamiseksi heidän pilvessään sijaitsevan, maksullisen palvelun toimiakseen. Yhteys käyttäjän, Pulumin palvelinten ja Azuren välillä on salattu, mutta tämä toisi lisäkustannuksia kuukausimaksujen muodossa, sekä asettaisi sekä Nodeonin omia, että asiakkaidensa salaisuuksia kolmannen osapuolen salauksen varaan. Pulumilla on laajan tukensa vuoksi mahdollisuus konfiguroida muitakin, kuin Azuren resurs-

seja, tarkoittaen sen soveltuvan myös pidempiaikaiseen käyttöön, mikäli päätetäänkin siirtyä muuhun pilvipalvelutarjoajaan tai jos pilvipalveluiden hallintaa tahdotaan tulevaisuudessa kaupallistaa omana tuotteenaan. Koodinatiivisuutensa ansiosta voi Pulumista myös tehdä niin modulaarisen ja tyypitetyn kuin haluaa, tehden siitä ohjelmointia osaavalle luontevan metodin pilvipalveluissa sijaitsevan infrastruktuurin hallintaan.

4.5.3 HashiCorpin Terraform

Terraform on deklaratiivisista työkaluista vanhin, mutta edelleen käytetyin. HashiCorp, Terraformin luojat, ovat tehneet sille oman deklaratiivisen kielen: Hashicorp Configuration Language, eli HCL. HCL:n tarkoitus on tehdä muun muassa ARM-mallitiedostojen yhteydessä mainitusta JSON-formaatista paljon ihmisluettavampaa. Lisäksi se on varta vasten luotu pilvipalveluiden resurssienhallintaan deklaratiivisella tavalla, tehden siitä hyvin räätälöidyn ja olemassa olevan ongelman ratkaisevan kokonaisuuden. (HCL 2020)

Pulumin tapaan, Terraform vaatii keskitetyn hallintapisteen, jonka tarkoituksena ei kuitenkaan ole toimia rajapintana resurssienhallinnan toimintojen ajoa varten, vaan ainoastaan niin sanotun tilatiedoston (State) ylläpitämiseen. Tämän tilatiedoston voi tallentaa joko HashiCorpin tarjoamaan HashiCorp Cloudiin (HCP), paikalliseksi tiedostoksi, tai esimerkiksi Azureen tai muuhun pilvipalvelutallennustilaan. HashiCorpin omassa pilvessä tulee vastaan samat ongelmat, kuin Pulumilla, eli kolmannen osapuolen salaukseen riippuminen ja kuukausittaiset lisämaksut. Paikallisella koneella tämän tallettaessa, on sen avulla ryhmätyöskentely ja laajempi ylläpito huomattavasti hankalampaa, kuin esimerkiksi sen Azuressa säilyttäminen, jossa mahdollisia salaisuuksia säilytetään joka tapauksessa. (Andre Lopes 2024)

Terraformin konfigurointi on luonnostaan modulaarista ja tarkoituksella viety ohjelmointiajattelutavasta ihmisläheisempään suuntaan, tarkoituksena tehdä sen käytöstä sekä helpompaa, että soveltuvampaa useampaan pilvipalvelun toimittajan ympäristöön, vaatien vain pieniä muutoksia mallipohjaan toimiakseen eri pilvipalvelualustoilla.

Terraform on pitkäikäisyytensä ja suosionsa vuoksi muodostunut alan standardiksi IaC-käyttötarkoituksissa. Tämä asema tuo sen edun, että pilvipalveluiden tarjoajat itse tuottavat virallisesti tuettuja komponentteja HashiCorpin kanssa pystyäkseen kilpailemaan muiden palveluntarjoajien

kanssa, kun ohjelmistokehittäjät ja pilvipalveluarkkitehdit ovat jo tottuneet käyttämään Terraformia. Tämä pätee Microsoft Azuren lisäksi muihinkin markkinoiden johtaviin pilvipalveluiden tarjoajiin, kuten Amazonin AWS ja Googlen GCP. (Providers n.d.)

4.5.4 Microsoftin Bicep

Bicep on ARM-mallitiedostojen tapaan Microsoftin oma tuote. Bicep luotiin, kun käyttäjät alkoivat kääntyä vaihtoehtoisiin työkaluihin, kuten Pulumi ja Terraform. ARM-mallitiedostojen hylkiminen johtui sen kompleksisuuden ja vaikeaksi todetun ihmisluettavuuden vuoksi. Suurimmaksi inspiraatioksi Bicepille voikin lukea Terraformin, josta Bicep on saanut useita vaikutteita. Bicep toimii ilman Pulumin ja Terraformin vaatimia keskitettyjä tilanhallintatietoja, sillä se on käytännössä vain ARM-mallitiedostojen korkeampi taso, joka luo ja ajaa infrastruktuurin mallipohjan, käyttäen ARM-mallitiedostoa yksinkertaisempaa ja ihmisluettavampaa kieltä. Koska ARM-mallitiedostot luodaan Bicep:n kautta, on ne helpompi pitää ajan tasalla Microsoftin päivitysten myötä, kuin jos niitä kirjoitetaan alusta asti käsin Bicep-versiopäivitysten ja uudelleenajon myötä. Tämä tuo hyötynä paitsi tilatiedon tallennuspaikan tarpeettomuuden, myös hyvin kohdistetun lähestymistavan juuri Azuren resurssienhallintaan. Bicep on vaihtoehdoista helpoin ottaa käyttöön, mutta tukee ainoastaan Azurea, eikä ole mukautettavuudessaan tai inkrementaalisuudessaan yhtä tehokas, kuin Pulumi tai Terraform. Bicep ei oletuksena tarjoa muutosten esikatselua ennen todellista ajoa, vaan komentoon pitää muistaa lisätä lippu ”what-if”, jolloin Bicep kertoo tehtävistä muutoksista, kuitenkaan suorittamatta niitä. (What is Bicep? 2025)

4.6 Työkalun valinta

Verrattaessa kaikkia vaihtoehtoja, ARM-mallitiedostot muuttuvat helposti vaikealuettaviksi ja edustavat Microsoftin vanhempaa edistymistapaa Azuressa sijaitsevien resurssien konfigurointiin. Työkaluista Pulumi tosin osoittautuu usean koodikieleen taipuvuudestaan huolimatta usein liian vaikeakäyttöiseksi muille, kuin ohjelmointia syvällisemmin osaaville laajalti imperatiivisen ajattelumallinsa, sekä tilatiedoston säilyttämisestä kolmannen osapuolen palvelimilla johtuen, asettaen arkaluontoisen tiedon heidän salauksensa varaan. Sekä ARM-mallipohjien, että Pulumin huonous kulminoituu siten, että toimeksiantoon kuuluu käytettävyys ja ylläpidettävyys myös ei-ohjelmoijien toimesta. Tämän vuoksi aiemmin verratuista työkaluista järkevimmiksi Nodeonin kannalta ovat

Bicep ja Terraform, joista valitaan ainoastaan toinen opinnäytetyössä suoritettavan syvällisen tutkinnan kohteeksi.

Bicep:n ja Terraformin erot Azuressa ovat isossa kaavassa verrattain pienet, mutta työkaluja vertaamalla voidaan päästä ratkaisevaan lopputulokseen siitä, mihin työkaluun kannattaa toteutuksessa painottua. Valitusta tuotteesta tahdotaan sekä helppokäyttöisyyttä, että mahdollisimman paljon uudelleenkäytettävyyttä ja mahdollisesti soveltuvuutta useamman pilvipalvelutarjoajan ympäristöihin. Keskittyen ainoastaan eroavaisuuksiin, voimme muodostaa taulukkoon 1 viitaten perustellusti mielipiteen siitä, mitä työkalua Nodeonin kannattaa käyttää pilvipalveluidensa hallinnassa.

Taulukko 1. Terraformin ja Bicepin olennaiset ominaisuudet

Hallintatyökalu	**Terraform**	**Bicep**
Tilatiedon hallinta	Kyllä, paikallisesti tai pilvessä	Bicepin luoma ARM-mallitiedosto ei tue tilatietoja
Konfigurointisyntaksi	Oma, HCL	Oma, JSON-pohjainen Bicep
Tuetut alustat	Kaikki suuret pilvipalveluntarjoajat	Vain Azure
Versionhallinta	Git, State-tilatieto	Git, mallitiedoston arkistointi
Esikatseluominaisuus	Oletuksena	Valinnainen ”what-if”-lippu
Modulaarisuus	Kyllä	Kyllä

Ensimmäisen osapuolen tuki	AzureRM-rajapinta HashiCorpin ja Microsoftin yhteistuote	Täysin Microsoftin oma tuote
Kolmansien osapuolten tuki	Usean osapuolen ylläpitämiä rajapintoja ja dokumentaatiota	Käytännössä vain Microsoftin oma rajapinta ja dokumentaatio

Taulukossa 1 työkaluja verrattaessa, on Bicep yksinkertaisempi ja suoraviivaisempi työkalu, jolla pääsisi samaan lopputulokseen Nodeonin tarpeet täyttävässä arkkitehtuurissa. Bicep ei kuitenkaan tuo kaikkia Terraformin hyötyjä versionhallinnassa ja työkalun ylläpidon sujuvuudessa, kun sen toimintamalliin kuuluu generoida ARM-mallitiedostot uudelleen jokaisen ajon välillä, joita työkalu käyttää toteuttaakseen muutokset. Lisäksi Bicep on luotu tukemaan ainoastaan Azurea, mikä ei mahdollista yhtä laajaa pilvipalveluhallinnan tuotteistamista tulevaisuudessa, ottaen huomioon mahdolliset kaupalliset hyödyt useamman pilvipalvelutarjoajan konfigurointia kykenevänä toimijana. Tämän vuoksi Terraform on lisäkompleksisuudestaan huolimatta parempi työkalu Nodeonin ylläpidossa olevien järjestelmien taustalla olevalle pilvipalveluarkkitehtuurin kehittämiselle.

5 Terraformin käyttö Azuren konfiguroinnissa

5.1 Ympäristön alustaminen

Terraformin käyttöönoton ja ensimmäisen ajon voi kiteyttää muutamaan olennaiseen askeleeseen, kun käytettävissä on jo aktiivinen Azure-tilaus, sekä mahdollisuus käyttää komentotulkin Command Line Interface (CLI) sovelluksia:

- Azure CLI:n ja Terraform CLI:n asennus käyttöohjeita seuraten
- Azure CLI:hin sisäänkirjautuminen ja tarvittavien käyttöoikeuksien varmistaminen
- Terraformin alustus ja esiasetukset konfiguraatiota varten

5.2 Azuren ja Terraformin asennus komentokehotteeseen

Azure CLI on kolmelle suurelle tietokoneiden jakelualustalle, eli Windowsille, macOS:lle ja Linuxille saatavilla oleva työkalu, jolla hallinnoidaan Azuressa sijaitsevia resursseja rajapinnan kautta. Azure CLI on vaatimuksena useimmille IaC-metodille, koska infrastruktuuria hallitseva koodi kommunikoi pilveen sitä käyttäen. Muilla suurilla pilvipalvelutarjoajilla on vastaavanlaiset CLI-sovellukset. Terraform CLI on vastaavasti Terraformin terminaalityökalu, jonka tehtävänä on suorittaa deklaratiivisen tiedoston parsiminen, mahdollinen vianetsintä ja vertaus Azuressa sijaitseviin resursseihin.

Opinnäytetyössä on valittu Microsoftin PowerShell-komentotulkki, johon Azure CLI:n saa asennettua Microsoftin (2025) ajankohtaisia ohjeita seuraten. Asennuksen voi todentaa kuvion 4 mukaisesti listaamalla asennetun version, sekä vertaamalla sitä tarjolla oleviin versioihin.

```
PS C:\Users\RobinAdolfsson> az -v
azure-cli                         2.74.0

core                              2.74.0
telemetry                          1.1.0

Dependencies:
msal                              1.32.3
azure-mgmt-resource               23.3.0
```

Kuvio 4. Azure CLI:n versiolistaus

Kun Azure CLI on asennettu onnistuneesti, voidaan Terraform asentaa seuraten HashiCorpin (n.d.) ohjeita, valiten huolellisesti Windowsin sovellus oikealle suoritinarkkitehtuurille ja seuraten sen asennusohjeita.

5.3 Käyttöoikeuksien määrittäminen Azuressa

Jotta Azuressa voi luoda, muokata tai poistaa resursseja, vaaditaan käyttäjälle tarvittavat oikeudet. Tämän voi asettaa käytössä olevan tilauksen asetuksissa, Azuren web-käyttöliittymän kautta käyttäjän toimesta, jolla on oikeudet muokata muiden käyttäjien oikeuksia. Käyttäjäoikeudet on pa-

rasta asettaa aina pienimmän tarpeellisen käyttöoikeuden perusteella, suoden ainoastaan oikeudet esimerkiksi yhteen resurssiryhmään tai yksittäisille resursseille. Tämä toimii viimeisenä suojakerroksena varsinkin IaC-menetelmällä luodessa, muokatessa ja poistaessa resursseja – tuhoisilta ja mahdollisesti kalliilta vahingoilta. Koska opinnäytetyössä oli tarkoitus uudistaa tai muokata kokonaista järjestelmää, sekä järjestelmähallinnan vastuu oli jo muutenkin allekirjoittaneella, ei tätä ollut tarpeen tehdä Nodeonin tapauksessa. (Assign Azure roles using the Azure portal 2025)

Parhaan käytännön mallin mukaan, kuuluisi tilaukseen osallistuvilla henkilöillä aina olla pienimmät mahdolliset oikeudet, mitä vaaditaan kunkin työtehtävän suorittamista varten. Tämän vuoksi voi olla suotavaa joko luoda uusi rooli, jossa sallitaan vain tarvittavien resurssien muokkausoikeudet, tai antaa Contributor-rooli resurssityypeittäin tai laiteryhmittäin. Hienojakoisen käyttäjäoikeusjaottelun olisi hyvä tehdä viimeistään siinä vaiheessa, kun arkkitehtuuri siirtyy tuotantoon. Tämänkin pystyy tekemään Terraformin kautta, mutta koska riittävät käyttäjäoikeudet ovat perusvaatimus Terraformin käytölle, on ne varmistettava ensimmäistä käyttöä varten muulla tavalla.

Kun käyttäjällä on tarvittavat oikeudet, vaaditaan vielä sisäänkirjautuminen Azureen komentotulkilla, että Terraformilla on pääsy muokkaamaan tietoja sitä käyttäen. Tämä tapahtuu yleisesti ottaen suoraan Azure CLI:tä käyttäen, joka ohjeistaa tunnistautumaan joko suoraan käyttöjärjestelmän käyttäjähallinnan, tai Microsoftin sisäänkirjautumisportaalin kautta, tukien monivaiheista tunnistautumista. (Authenticate to Azure using Azure CLI 2025)

5.4 Terraformin alustus ja esiasetukset konfiguraatiota varten

Ennen testiajoa tai minkäänlaista konfiguraatiota, on Terraformille annettava pakolliset tiedot, jotta se osaa kommunikoida Azuren rajapinnan kanssa. Liitteenä 1 on pakolliset kentät sisältävä Terraform-tiedosto, jonka perusteella voidaan jo alustaa ympäristö Terraformin varsinaista käyttämistä varten. Terraform tukee paitsi omia *.tf*-päätteisiä tiedostoja myös JSON-tiedostoja paikallisille muuttujille. Tätä hyödyntäen, ilmenee liitteessä 1 myös *locals*-lohko, joka viittaa erilliseen JSON-tiedostoon, johon on päätetty sijoittaa arkaluonteiset tiedot. Tästä voi olla hyötyä esimerkiksi lisäämällä arkaluonteiset tiedot sisältävä tiedosto versiohallintatyökalun poikkeuksiin.

Liitteen 1 tiedoston voi Terraformin sisäänrakennetun modulaarisuuden vuoksi erotella varsinaisesta konfiguraatiosta, jolloin varsinaisessa konfiguraatiotiedostossa voidaan keskittyä infrastruktuurin hallintaan rajapinta-asetusten sijaan. *Locals*-lohkossa mainittu tiedosto laitetaan samaan kansioon, ja siihen täytetään muuttujille samoja nimiä, kuin Terraform-tiedostoissa niihin viitataan. Esimerkiksi liitteessä 1 on viite muuttujaan *local.locals.subscription_id*, jolloin myös *locals.json*-tiedostossa käytetään objektin nimenä *subscription_id*. Kun JSON-tiedostoon on täytetty pakolliset tiedot, voidaan Terraform alustaa *terraform init*-komennolla. Komennon onnistuneen ajon myötä saadaan kuviossa 5 esiintyvä ilmoitus, joka kertoo ympäristön olevan valmis varsinaisten mallipohjien kirjoittamista varten.

```
PS C:\Users\RobinAdolfsson\Documents\rimps\Terraform-test\tf-oppari> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/azurerm...
- Installing hashicorp/azurerm v4.34.0...
- Installed hashicorp/azurerm v4.34.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Kuvio 5. Terraformin alustus komentokehotteella

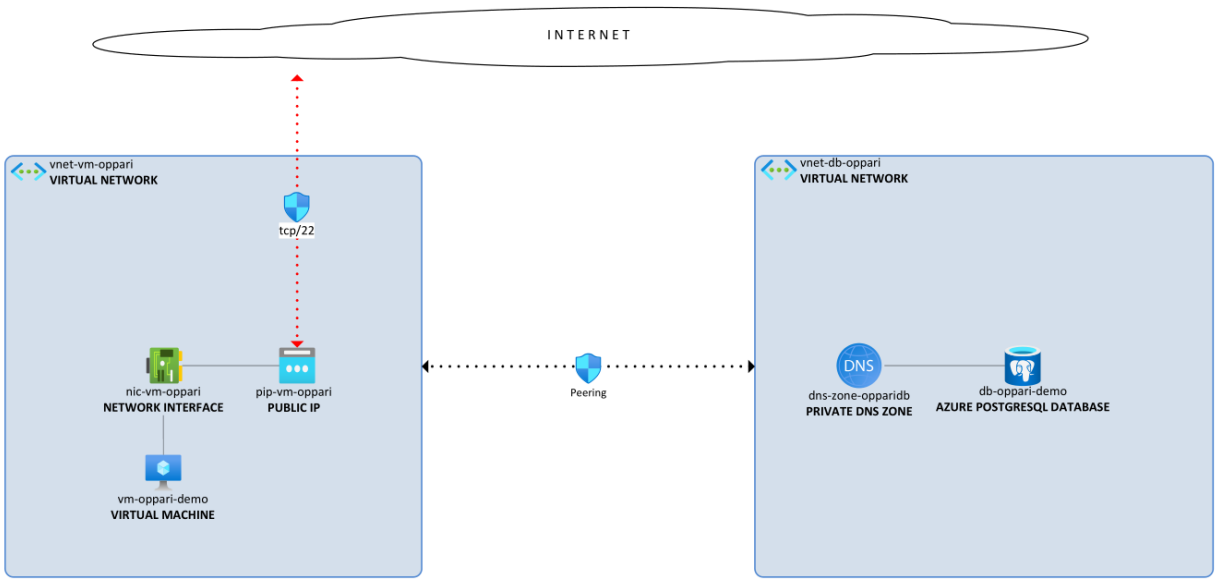
5.5 Konfiguraatiotiedoston luonti

Alustettuun Terraform-ympäristöön voi alkaa kirjoittamaan konfiguraatiotiedostoja. Terraformin modulaarisuutta tukee se, että kun Terraform alustetaan käyttämään yhtä kansiota, olettaa se kaikkien kansiossa sijaitsevien *.tf*-päättyvien tiedostojen olevan relevantteja sen käyttötarkoitukseen, eikä niitä tarvitse missään erikseen mainita. Tämä mahdollistaa alustuksen ja erillisten paikallismuuttujien lisäksi myös sen, että esimerkiksi tietoliikennekonfiguraatiolle voi luoda oman tiedostonsa, virtuaalipalvelimille omansa, ja niin edelleen. Liitteen 2 esimerkkikonfiguraatiossa kaikki

on kuitenkin sijoitettu yhteen tiedostoon, mutta tiedoston voisi pilkkoa mainitun esimerkin mukaisesti niin halutessa.

Koska Terraform tukee konfiguraatiossa asetettujen arvojen uudelleenkäyttämistä myös sisäisesti, on hyvä asettaa oleellinen arvo vain kerran ja tämän jälkeen viitata siihen käyttäen muuttujan nimeä, jotta mahdollisissa muutostarpeissa ei samaa arvoa tarvitse muuttaa kuin alkuperäisessä käyttökohteessa. Laajalti uudelleenkäytettyjä arvoja on esimerkiksi resurssiryhmän nimi, joka ilmenee *azurerm_resource_group*-lohkossa ja siihen viitataan moneen otteeseen lisäämällä objektien arvoksi *azurerm_resource_group.opparirg.name*. Tällöin Terraform osaa kopioida resurssiryhmän arvoksi saman, joka alkuperäisellä deklaraatiolla on jo kerran ilmoitettu tekstinä. Tällä menetelmällä vältytään laajalti kirjoitusvirheistä ja tehdään konfiguraatiopohjasta helpommin ylläpidettävän.

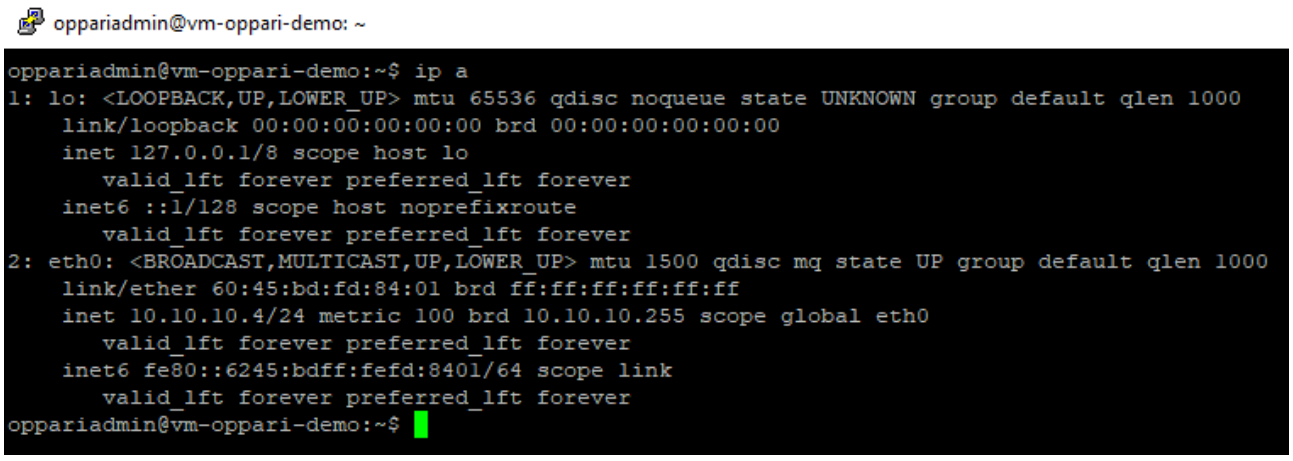
Liitteen 2 konfiguraatiotiedoston ajamalla pystyy luomaan kuvion 6 testiarkkitehtuurin yhdellä komennolla, mukaan lukien peruslaatuiset tietoturvakovennukset yksinkertaisilla palomuurisäännöillä eri virtuaaliverkkojen välille.



Kuvio 6. Terraform-testiajon arkkitehtuuri

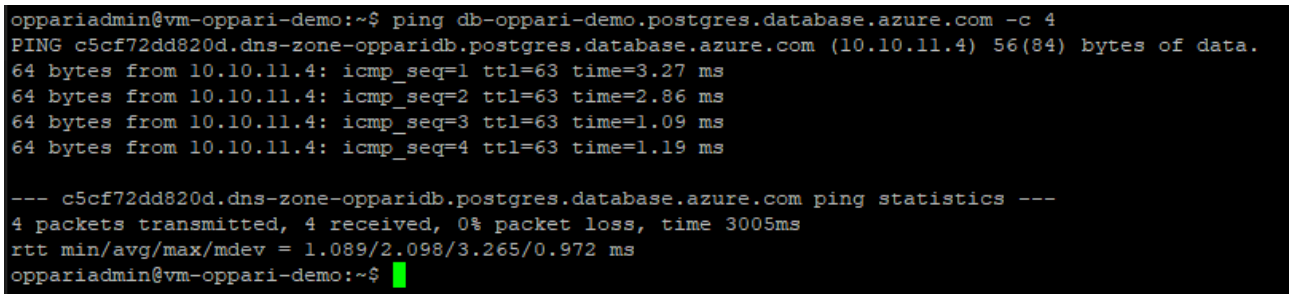
5.6 Terraformilla luodun arkkitehtuurin toimivuus

Kun liitteen 2 tiedosto on ajettu Terraformilla, voi yhteyden ottaa käyttäen *locals*-tiedostossa kerrotulla *admin_key*-avaimella yhteyttä palvelimeen sen julkiverkon osoitteeseen. Julkiverkon osoite näkyy Azuren web-hallintapaneelista ja käyttäjänimi pääasiallisen Terraform-mallipohjaan kirjoitettuna *oppariadmin*. Kuviosta 7 käy ilmi yhteydenoton onnistuneen, sekä paikallisen aliverkotuksen IP-osoitetiedot.



Kuvio 7. Virtuaalipalvelimen IP-osoitetiedot

Voidaan myös todeta virtuaalipalvelimen saavan yhteyden tietokantapalvelimelle nimen perusteella, jolla todistetaan yksityisen nimiosoitepalvelimen olevan oikein konfiguroitu molemmalle virtuaaliselle aliverkolle konfiguraation mukaisesti kuviossa 8 esiintyvän verkkotunnukseen kohdistuvan pingin perusteella.



Kuvio 8. Virtuaalipalvelimelta lähtevä ping-pyyntö tietokantapalvelimen verkkotunnukselle

Alkuperäisen ajon jälkeen, lisättiin vielä tietokantapalvelimelle erillinen tietokanta *db-oppari-demo-database* esittääkseen sitä, että uuden ajon yhteydessä suoritetaan inkrementaalisesti ainoastaan yksi muutos, koskematta muuhun infrastruktuuriin. Tämä on liitteessä 2 kuvattuna lopullisessa tiedostossa, mutta lisättiin siihen vasta alkuperäisen ajon jälkeen. Kuviossa 9 ilmenee *terraform apply*-komennon ilmoitus siitä, että sen aikomuksena on tehdä ainoastaan yksi muutos infrastruktuuriin koodiin lisätyn lohkon perusteella.

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

Terraform will perform the following actions:

  # azurerm_postgresql_flexible_server_database.opparidbdatabase will be created
  + resource "azurerm_postgresql_flexible_server_database" "opparidbdatabase" {
      + charset   = "UTF8"
      + collation = "en_US.utf8"
      + id        = (known after apply)
      + name      = "db-oppari-demo-database"
      + server_id = "/subscriptions/                                  /resourceGroups/rg-robin-oppari/providers/Microsoft.DBforPostgreSQL/flexibleServers/db-oppari-demo"
    }

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

Kuvio 9. Tietokannan lisäys tietokantapalvelimelle tekemättä muita muutoksia

5.7 Jatkokehityksen mahdollisuudet

Todetun IaC-pohjaisen ratkaisun lisäksi, on mahdollista lisätä olemassa oleva infrastruktuuri osaksi Terraform-mallipohjaa HashiCorpin (n.d.) ohjeita seuraten. Tällöin tarvitsee luoda uudet lohkot Terraform-konfiguraatioon, joihin lisää tarvittavat ja paikkansapitävät parametrit mallipohjaan, ajaen kohdistetusti *terraform import*-komennon jokaiselle Terraformiin tuotavalle resurssille. On huomioitavaa, että jos resurssi ei ole oikein tuotu Terraformin konfiguraatioon, on suuri riski kyseisen resurssin poistamiselle, kun seuraavan kerran ajaa muutokset infrastruktuuriin. Tämä korostaa tarvetta luoda huolellisesti Terraformin aikomukset tehdä muutoksia infrastruktuurille.

Terraformin käyttämän tilatiedoston olisi suotava varsinkin monikäyttäjäympäristössä siirtää pilvessä sijaitsevaan palveluun, jolloin se on jokaisen ajon välissä päivitetty ja tallennuspaikkansa myötä turvattu esimerkiksi laiterikoilta, eikä tuotannossa olevaa tilatiedostoa tarvitse edellä mainitulla *import*-komennolla tuoda työläästi uudelleen tai pitää usean käyttäjän toimesta aina päivitettynä esimerkiksi toimintaa tukevaa skriptiä käyttäen. Tilatiedoston ulkoista, pilvipalvelussa sijaitsevaa tallennusta varten on olemassa sisäänrakennettu toiminta, jota hyödyntämällä ohittaa useat tässä kappaleessa todetut ongelmat. (Andre Lopes 2024)

Terraformin mallipohjatiedostot tulisi tuotantokäytössä säilyttää keskitetyssä versionhallintajärjestelmässä, kuten GitHubissa tai sitä vastaavassa tuotteessa. Tällä ja henkilöstön ohjeistamisella vältytään versioiden eroavaisuuksista, varsinkin jos käytössä on pilvipalvelussa sijaitseva tilatiedosto, josta joko puuttuu tai johon on lisätty resursseja, joita ei ole tarve käyttää. Kuten monessa muussakin ohjelmointiprojektissa, myös Terraformin kannalta löytyy arkaluonteisia tiedostoja, kuten opinnäytetyössä käytetyn *locals.json*-tiedoston sisältämät tiedot, jotka voidaan kuitenkin poissulkea versionhallinnasta *gitignore*-tiedostolla, joka poistaa versionhallintatyökalun seuraamista tiedostoista määritellyt tiedostot. (gitignore - Specifies intentionally untracked files to ignore n.d.)

Terraform tukee myös useanlaista rekursiivista ohjelmointimenetelmää monen ohjelmointikielen lailla esimerkiksi for-luupeilla. Näitä hyödyntäen, voidaan vielä vahvemmin yhdenmukaistaa esimerkiksi resurssien nimitykset, jos paikallismuuttujien tiedostoon lisätään pelkästään käyttökohteen nimet, mutta muuten seurataan opinnäytetyössä aiemmin mainittua nimeämiskäytäntöä. Ihmisvirheiltä voidaan säästyä vielä paremmin, mitä vähemmän niitä vaaditaan mallipohjan muutoksissa. Luuppeja hyödyntäen, voidaan luoda lähes identtisiä resursseja liitteen 2 siistimpää mallipohjaa käyttäen, joissa lisätään paikalliseen JSON-tiedostoon ainoastaan muuttuvat parametrit, mutta varsinaisessa mallipohjassa on asetettu kaikki muu tieto valmiiksi modulaarisuuden edistämiseksi. (Dinu & Roper 2025)

6 Tulokset

Opinnäytetyössä vastattiin toimeksiannosta syntyneisiin tutkimuskysymyksiin, joiden perusteella muodostettiin jatkuvalle kehitykselle suuntaa näyttävä kokonaisuus, minkä perusteella nykyistä infrastruktuuria voidaan kehittää entistä ylläpidettävämpi ja skaalautuvampi, mahdollistaen sen kasvava käyttö kaupallisena tuotteena.

Nykyisen pilvipalveluntarjoaja Azuren ympäristössä sijaitsevan arkkitehtuurin heikkouksiksi muodostuivat pääsynhallinnan toteutus virtuaalipalvelimella Azuren omien työkalujen sijaan, resurssien ylimitoitus suhteessa todelliseen käyttöasteeseen, dokumentaation puutteellisuus ja skaalautuvuuden vaikeus.

Viitaten arkkitehtuurin heikkouksiin, esitettiin opinnäytetyössä suunta jatkokehityksen toimenpiteille, jolla nykyiseen arkkitehtuuriin verratessa, voidaan luoda uusi, kestävämpi pilvipalvelumalli.

Nimeämiskäytäntöä on selkeytettävä, virtuaalipalvelimia on pienennettävä ja tietokantapalvelinta on optimoitava ja tämän jälkeen mahdollisuuksien mukaan kutistettava käyttämään vähempiä resursseja. Parantamalla käytössä olevien resurssien kokoa, on saavutettavissa huomattavia kustannussäästöjä vuositasolla Nodeonin käyttämässä kokonaisuudessa. Lisäksi on siirryttävä helpommin yhdenmukaisena pidettävään toimintamalliin ja parannettava dokumentaatiota entuudestaan, jotta kokonaisuus pysyy käyttöasteen kasvaessa selkeänä.

Vertailun perusteella todettiin, että paras menetelmä havaittujen ongelmien korjaamiseksi pilvipalvelutarjoajan infrastruktuurin puolella on IaC-menetelmään perustuva Terraform. Todettiin myös, että Terraform on soveltuva Nodeonin käyttötarkoitukselle, jos koodikantaa kehitetään kattamaan kaikki nykyisen ja tulevan infrastruktuurin tarpeet, jolloin resursseja hallitaan keskitetysti koodipohjaisesti ja versiohallitusti. Luotiin Azuren resurssienhallintaan esimerkkipohja, jota pystyy helposti skaalaamaan kopioimalla siinä käytettyjä lohkoja, muuttaen ainoastaan pieniä osia muuttujista, kuten IP-osoitteita ja nimeämisiä. Lisäksi uusien resurssityyppien lisääminen IaC-mallilla on tehty helpoksi, kun niille on luotu opinnäytetyön liitteessä 2 esiintyvään pohjaan tyylimalli ja nimeämiskäytäntö.

Opinnäytetyössä luotiin myös arkkitehtuurilliset periaatekuvat, joita käytetään Nodeonin käyttämän arkkitehtuurin kehitysehdotuksena, joiden perusteella palvelua voidaan parantaa kestävästi ja yhteneväisesti, jos ne huomioidaan IaC-kehitystä suorittaessa.

7 Pohdinta

Opinnäytetyössä vastattiin toimeksiantajan toiveeseen luoda pilvipalvelupohjaiselle arkkitehtuurille kehittyneempi toimintamalli, joka on helpommin ja yhdenmukaisemmin päivitettävissä ja ylläpidettävissä, kuin nopeammalla aikataululla luotu nykyinen ympäristö. Löydettiin yrityksen tarpeisiin paras työkalu IaC-maailmaan astumiselle, poissulkien varsinaiseen käyttötarkoitukseen soveltumattomia työkaluja. Laajalla IaC-työkalujen vertaamisella saatiin kuitenkin lähtökohta, johon viitata, mikäli jokin niistä muovautuu alati kehittyvällä alalla tulevaisuudessa varteenotettava työkalu tulevaisuuden tarpeita varten.

Toimeksiannosta jätettiin ulkopuolelle joitakin asioita, kuten AVSS- ja Azure Container App-testaukset, sekä mahdollisen uuden asiakasympäristön lisääminen Terraformilla toteutettuun infrastruktuuriin. Koska olemassa olevaa infrastruktuuria ei vielä ole lisätty Terraformiin, ei myöskään ehditty selvittämään sen yliheittoa uuteen arkkitehtuuriin. Näitä lisätestejä suositellaan jatkotehtäväksi käyttäen opinnäytetyötä perusteena.

Todettiin tietoturvallisen versionhallinnan käytettävyys IaC-maailmassa olevan mahdollista ja useamman työntekijän käytettävissä, kun lisätään pilvipalvelussa sijaitseva tilatiedosto osaksi käytäntöä, mahdollistaen ylläpidettävyyden pitkäksi ajaksi.

Opinnäytetyöstä saatiin tutkimukseen pohjautuva perusta jatkokehitykselle, jonka perusteella Nodeon pystyy kehittämään pilvipalveluinfrastruktuuriaan yhdenmukaisempaan suuntaan, käyttäen Visiolla luotuja kaavioita pohjana korkean tason suunnittelussa tulevaisuuden kehitystoimenpiteisiin.

Opinnäytetyön tulosten lisäksi tuotiin esiin jatkokehitysajatuksia, kuten versionhallintaa ja tilatiedon keskittämistä pilvipalvelualustalle, jossa monitoimijaympäristössä voi useampi henkilö työstää samaa projektia, tehden muutokset aina samaan tilatietoon muiden kanssa.

Lähteet

Abbreviation recommendations for Azure resources. 2025. Microsoftin ehdottamat lyhenteet Azuren resursseille. Viitattu 22.6.2025. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-abbreviations.

About Zabbix LLC. n.d. Zabbix-verkonvalvontajärjestelmää tarjoavan organisaation esittely omilla verkkosivuillaan. Viitattu 19.6.2025. https://www.zabbix.com/about.

Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M., Tamburri, D. A. 2017. DevOps: Introducing Infrastructure-as-Code. International Conference on Software Engineering Companion, 497–498. doi: 10.1109/ICSE-C.2017.162. Viitattu 20.6.2025. https://janet.finna.fi/, IEEE.

Assign Azure roles using the Azure portal. 2025. Microsoftin dokumentaatio roolipohjaisista käyttäjäoikeuksista Azuressa. Viitattu 26.6.2025. https://docs.azure.cn/en-us/role-based-access-control/role-assignments-portal.

Authenticate to Azure using Azure CLI. 2025. Microsoftin dokumentaatio sisäänkirjautumisesta Azuren komentotulkkirajapintaan. Viitattu 26.6.2025. https://learn.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest.

Azure Container Apps overview. 2025. Microsoft Azuren dokumentaatio Azure Container App-palvelunsa toiminnasta. Viitattu 25.6.2025. https://learn.microsoft.com/en-us/azure/container-apps/overview.

Brzychczy, E., Szpyrka, M., Korski, J., Nalepa, G. J. 2023. Imperative vs. Declarative Modeling of Industrial Process. The Case Study of the Longwall Shearer Operation, 54497. doi: 10.1109/ACCESS.2023.3281304. Viitattu 10.7.2025. https://janet.finna.fi/, IEEE.

Cioe, V. 2021. MySQL performance tuning 101 for Zabbix. Viitattu 20.6.2025. https://blog.zabbix.com/mysql-performance-tuning-101-for-zabbix/13899/.

Compare Visio versions and features. n.d. Microsoft Vision ominaisuuslistaus omalla tuotesivulla. Viitattu 10.7.2025. https://support.microsoft.com/en-us/office/compare-visio-versions-and-features-c659cbc1-34c7-42d8-92e6-80c87fb572a7.

Configure rolling upgrades on Virtual Machine Scale Sets. 2025. Microsoftin dokumentaatio Virtual Machine Scale Sets-toiminnon käyttämisestä rullaaville päivityksille. Viitattu 10.7.2025. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-configure-rolling-upgrades.

Define your naming convention. 2025. Microsoftin ehdottama nimeämiskäytäntö Azuren resursseissa soveltamisalan määrittämiseksi. Viitattu 22.6.2025. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-naming.

Dinu, F., Roper, J. 2024. Terraform For Loop – Expression Overview with Examples. Viitattu 10.7.2025. https://spacelift.io/blog/terraform-for-loop.

gitignore - Specifies intentionally untracked files to ignore. n.d. git-versionhallintajärjestelmän virallinen dokumentaatio versionhallinnan poissulkemien tiedostojen konfiguroinnista. Viitattu 10.7.2025. https://git-scm.com/docs/gitignore.

HCL. 2020. HashiCorpin GitHub-sivustoilla oleva README-tiedosto, selittäen HCL:n keskeiset tavoitteet ja käyttökohteet. Viitattu 25.6.2025. https://github.com/hashicorp/hcl/blob/main/README.md.

How Pulumi Works. n.d. Pulumin oman dokumentaation selitys deklaratiivisuudesta ja imperatiivisuudesta. Viitattu 25.6.2025. https://www.pulumi.com/docs/iac/concepts/how-pulumi-works/#declarative-and-imperative-approach.

Import. n.d. Terraformin virallinen dokumentaatio olemassa olevan infrastruktuurin tuontia varten osaksi mallipohjaa HashiCorpin omilla sivuilla. Viitattu 10.7.2025. https://developer.hashicorp.com/terraform/language/import.

Install Azure CLI on Windows. 2025. Microsoftin käyttöohjeet Azure CLI:n asentamiseksi Windows-jakelualustalla. Viitattu 26.6.2025. https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest&pivots=winget.

Install Terraform. n.d. Ohjeet Terraformin asennukseen eri julkaisualustoille HashiCorpin nettisivuilla. Viitattu 26.6.2025. https://developer.hashicorp.com/terraform/install.

Ketterä toimintamalli digipalveluiden tuottamiseen. n.d. Tietoa DevOps-ajattelumallista ohjelmistokehityksessä Watarun kotisivuilla. Viitattu 21.7.2025. https://www.wakaru.fi/devops/.

Lopes, A. 2024. Terraform Backends – Local and Remote Explained. Spacelift 8.7.2024. Viitattu 25.6.2025. https://spacelift.io/blog/terraform-backends.

Monimenetelmäisyys. n.d. Artikkeli monimenetelmäisyyden tutkimusstrategiasta Jyväskylän yliopiston sivustolla. Viitattu 15.7.2025. https://sites.app.jyu.fi/mehu/fi/menetelmapolku/tutkimusstrategiat/monimenetelmaisyys.

Nodeon Referenssit. n.d. Nodeon Finlandin asiakasreferenssit sen omilla verkkosivuilla. Viitattu 19.6.2025. https://www.nodeon.com/referenssit/.

Nodeon Yritysesittely. n.d. Toimeksiantajan yritysesittely sen omilla verkkosivuilla. Viitattu 19.6.2025. https://www.nodeon.com/yritys/.

Overview of Azure network security. 2025. Microsoft Azuren dokumentaatio tietoliikenneverkon suojauksen menetelmistä. Viitattu 25.6.2025. https://learn.microsoft.com/en-us/azure/security/fundamentals/network-overview.

PowerShell. 2025. Esittelytiedosto Microsoftin ylläpitämästä PowerShell-lähdekoodikannasta GitHub-sivulla. Viitattu 10.7.2025. https://github.com/PowerShell/PowerShell/blob/master/README.md.

Providers. n.d. Terraformin virallisesta rekisteristä löytyä listaus Terraformin virallisesti ylläpidettävistä yhteensopivuusrajapinnoista. Viitattu 25.6.2025. https://registry.terraform.io/browse/providers?tier=official.

Quattrocchi, G., Tamburri D. A. 2022. Infrastructure as Code. IEEE Software, 40, 37–40. doi: 10.1109/MS.2022.3212034. Viitattu 20.6.2025. https://janet.finna.fi/, IEEE.

Richter, F. 2025. Amazon and Microsoft Stay Ahead in Global Cloud Market. Viitattu 10.7.2025. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

What are ARM templates? 2025. Microsoftin dokumentaatio Azure Resource Manager-mallitiedostojen käyttötarkoituksesta. Viitattu 25.6.2025. https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview.

What is Azure? n.d. Microsoftin palveluesittely Azure-pilvipalvelualustastaan. Viitattu 19.6.2025. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/.

What is Bicep? 2025. Microsoftin oma dokumentaatio Bicep:n ominaisuuksista ja toiminnasta. Viitattu 25.6.2025. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview.

Zabbix Requirements. n.d. Zabbix-verkonvaltontajärjestelmän version 7.0 laitteisto-, käyttöjärjestelmä- ja sovellusvaatimukset. Viitattu 19.6.2025. https://www.zabbix.com/documentation/7.0/en/manual/installation/requirements.

Liitteet

Liite 1. backend.tf-tiedoston sisältö

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id = local.locals.subscription_id
}

locals {
  locals = jsondecode(file("./locals.json"))
}
```

Liite 2. main.tf-tiedoston sisältö

```
resource "azurerm_resource_group" "opparirg" {
  location = "swedencentral"
  name     = "rg-robin-oppari"
}

resource "azurerm_linux_virtual_machine" "oppariubuntu" {
  admin_username        = "oppariadmin"
  location              = "swedencentral"
  name                  = "vm-oppari-demo"
  network_interface_ids = [ azurerm_network_interface.opparivmnic.id ]
  resource_group_name   = azurerm_resource_group.opparirg.name
  size                  = "Standard_B1ls"
  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "StandardSSD_LRS"
  }
  admin_ssh_key {
    public_key = local.locals.admin_key
    username   = "oppariadmin"
  }
  source_image_reference {
    publisher = "Canonical"
    offer     = "ubuntu-24_04-lts"
    sku       = "server"
    version   = "latest"
  }
  depends_on = [ azurerm_network_interface.opparivmnic ]
}

resource "azurerm_network_interface" "opparivmnic" {
  location            = "swedencentral"
  name                = "nic-vm-oppari"
  resource_group_name = azurerm_resource_group.opparirg.name
  ip_configuration {
    name                          = "ipconfig1"
    private_ip_address_allocation = "Static"
    private_ip_address            = "10.10.10.4"
    public_ip_address_id          = azurerm_public_ip.opparipip.id
    subnet_id                     = azurerm_subnet.opparivmsnet.id
  }
  depends_on = [
    azurerm_resource_group.opparirg,
    azurerm_subnet.opparivmsnet,
    azurerm_public_ip.opparipip
  ]
}

resource "azurerm_public_ip" "opparipip" {
```

```
  location            = azurerm_resource_group.opparirg.location
  name                = "pip-vm-oppari"
  resource_group_name = azurerm_resource_group.opparirg.name
  allocation_method   = "Static"
  sku                 = "Standard"
  depends_on = [
    azurerm_resource_group.opparirg,
  ]
}

output "vm-oppari-public-ip" {
  value = azurerm_public_ip.opparipip.ip_address
}

resource "azurerm_postgresql_flexible_server" "opparidb" {
  name                          = "db-oppari-demo"
  resource_group_name           = azurerm_resource_group.opparirg.name
  location                      = azurerm_resource_group.opparirg.location
  version                       = "16"
  delegated_subnet_id           = azurerm_subnet.opparidbsnet.id
  private_dns_zone_id           = azurerm_private_dns_zone.opparidbdns.id
  public_network_access_enabled = false
  administrator_login           = "psqladmin"
  administrator_password        = local.locals.db_password
  zone                          = "1"

  storage_mb   = 32768
  storage_tier = "P4"

  sku_name   = "B_Standard_B1ms"
  depends_on = [
    azurerm_subnet.opparidbsnet
  ]
}

resource "azurerm_postgresql_flexible_server_database" "opparidbdatabase" {
  name      = "db-oppari-demo-database"
  server_id = azurerm_postgresql_flexible_server.opparidb.id
  collation = "en_US.utf8"
  charset   = "UTF8"

  depends_on = [
    azurerm_postgresql_flexible_server.opparidb
   ]
}

resource "azurerm_virtual_network" "opparivmvnet" {
  name                = "vnet-vm-oppari"
  location            = azurerm_resource_group.opparirg.location
  resource_group_name = azurerm_resource_group.opparirg.name
```

```
  address_space       = ["10.10.10.0/24"]
}

resource "azurerm_subnet" "opparivmsnet" {
  name                 = "snet-vm-oppari"
  resource_group_name  = azurerm_resource_group.opparirg.name
  virtual_network_name = azurerm_virtual_network.opparivmvnet.name
  address_prefixes     = ["10.10.10.0/24"]

  depends_on = [
    azurerm_virtual_network.opparivmvnet
  ]
}

resource "azurerm_virtual_network" "opparidbvnet" {
  name                = "vnet-db-oppari"
  location            = azurerm_resource_group.opparirg.location
  resource_group_name = azurerm_resource_group.opparirg.name
  address_space       = ["10.10.11.0/24"]
}

resource "azurerm_subnet" "opparidbsnet" {
  name                 = "snet-db-oppari"
  resource_group_name  = azurerm_resource_group.opparirg.name
  virtual_network_name = azurerm_virtual_network.opparidbvnet.name
  address_prefixes     = ["10.10.11.0/24"]

  service_endpoints = [
    "Microsoft.Storage"
  ]

  delegation {
    name = "snet-db-psql-oppari-delegation"
    service_delegation {
      actions = [
        "Microsoft.Network/virtualNetworks/subnets/join/action"
      ]
      name = "Microsoft.DBforPostgreSQL/flexibleServers"
    }
  }

  depends_on = [
    azurerm_virtual_network.opparidbvnet
  ]
}

resource "azurerm_private_dns_zone" "opparidbdns" {
  name                = "dns-zone-opparidb.postgres.database.azure.com"
  resource_group_name = azurerm_resource_group.opparirg.name
}
```

```
resource "azurerm_private_dns_zone_virtual_network_link" "opparivmdnslink" {
  name                  = "dnslink-vm-oppari"
  private_dns_zone_name = azurerm_private_dns_zone.opparidbdns.name
  virtual_network_id    = azurerm_virtual_network.opparivmvnet.id
  resource_group_name   = azurerm_resource_group.opparirg.name
  depends_on            = [
    azurerm_virtual_network.opparivmvnet,
    azurerm_private_dns_zone.opparidbdns
  ]
}

resource "azurerm_private_dns_zone_virtual_network_link" "opparidbdnslink" {
  name                  = "dnslink-db-oppari"
  private_dns_zone_name = azurerm_private_dns_zone.opparidbdns.name
  virtual_network_id    = azurerm_virtual_network.opparidbvnet.id
  resource_group_name   = azurerm_resource_group.opparirg.name
  depends_on            = [
    azurerm_virtual_network.opparidbvnet,
    azurerm_private_dns_zone.opparidbdns
  ]
}

resource "azurerm_virtual_network_peering" "opparivmtodb" {
  name                      = "peer-vm-to-db"
  resource_group_name       = azurerm_resource_group.opparirg.name
  virtual_network_name      = azurerm_virtual_network.opparivmvnet.name
  remote_virtual_network_id = azurerm_virtual_network.opparidbvnet.id
  depends_on = [
    azurerm_virtual_network.opparivmvnet,
    azurerm_virtual_network.opparidbvnet
  ]
}

resource "azurerm_virtual_network_peering" "opparidbtovm" {
  name                      = "peer-db-to-vm"
  resource_group_name       = azurerm_resource_group.opparirg.name
  virtual_network_name      = azurerm_virtual_network.opparidbvnet.name
  remote_virtual_network_id = azurerm_virtual_network.opparivmvnet.id
  depends_on = [
    azurerm_virtual_network.opparivmvnet,
    azurerm_virtual_network.opparidbvnet
  ]
}

resource "azurerm_network_security_group" "opparivmnsg" {
  name                = "nsg-vnet-vm-oppari"
  location            = azurerm_resource_group.opparirg.location
  resource_group_name = azurerm_resource_group.opparirg.name
```

```
  security_rule {
    name                       = "Allow_SSH-To-VM"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "Internet"
    destination_address_prefix = "*"
  }

  depends_on = [
    azurerm_subnet.opparidbsnet,
    azurerm_subnet.opparivmsnet
  ]
}

resource "azurerm_subnet_network_security_group_association" "opparivmsnetnsg" {
  subnet_id = azurerm_subnet.opparivmsnet.id
  network_security_group_id = azurerm_network_security_group.opparivmnsg.id
  depends_on = [
    azurerm_network_security_group.opparidbnsg,
    azurerm_subnet.opparidbsnet
  ]
}

resource "azurerm_network_security_group" "opparidbnsg" {
  name                = "nsg-vnet-db-oppari"
  location            = azurerm_resource_group.opparirg.location
  resource_group_name = azurerm_resource_group.opparirg.name

  security_rule {
    name                       = "Allow_VM-To-DB_PostgreSQL"
    priority                   = 200
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "5432"
    source_address_prefix      = azurerm_subnet.opparivmsnet.address_prefixes[0]
    destination_address_prefix = "*"
  }
  security_rule {
    name                       = "Deny_VM-To-DB_Rest"
    priority                   = 300
    direction                  = "Inbound"
    access                     = "Deny"
    protocol                   = "Tcp"
```

```
    source_port_range          = "*"
    destination_port_range     = "*"
    source_address_prefix      = azurerm_subnet.opparivmsnet.address_prefixes[0]
    destination_address_prefix = "*"
  }
  depends_on = [
    azurerm_subnet.opparidbsnet,
    azurerm_subnet.opparivmsnet
  ]
}

resource "azurerm_subnet_network_security_group_association" "opparidbnsgassoc" {
  subnet_id = azurerm_subnet.opparidbsnet.id
  network_security_group_id = azurerm_network_security_group.opparidbnsg.id
  depends_on = [
    azurerm_network_security_group.opparidbnsg,
    azurerm_subnet.opparidbsnet
  ]
}
```