LAPIN AMK
Lapland University of Applied Sciences

TIETOJÄRJESTELMÄINFRASTRUKTUURIN LUOMINEN KOODIN AVULLA

Saraniemi Jussi

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2023

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Jussi Saraniemi	**Vuosi**	2023
Ohjaaja	Kenneth Karlsson		
Toimeksiantaja	Lapin AMK Tieto- ja viestintätekniikka		
Työn nimi	Tietojärjestelmäinfrastruktuurin luominen koodin avulla		
Sivu- ja liitesivumäärä	84 + 18		

Opinnäytetyön tavoitteena oli tutustua tietojärjestelmäympäristöjen luomiseen koodin avulla. IaC-tekniikka on varsin kypsää tekniikkaa ja laajasti käytössä muun muassa ohjelmistokehitystyössä. Tavoitteena oli luoda sekä pilvipalveluun että paikalliseen virtualisointiympäristöön kolme palvelinta, joiden väliset verkkoyhteydet olisivat todennettavissa ICMP echo request -viesteillä.

Alkuperäisenä tavoitteena oli käyttää käytännönosuudessa Hashicorpin Terraform-työkalua sekä pilvipalvelussa että paikallisessa virtualisointiympäristössä. Paikallisen virtualisointiympäristön toteuttaminen Terraformilla oli erittäin haastavaa ja koska ongelmat eivät olleet ratkaistavissa opinnäytetyön tekemisen aikarajoissa, päätettiin siirtyä käyttämään toista työkalua paikallisen ympäristön osalta.

Paikallisen ympäristön toteuttamiseksi työkaluksi valittiin Vagrant, joka on Hashicorpin toinen työkalu ja on tarkoitettu paikallisten kehitysympäristöjen luomiseen. Vagrantin käyttö osoittautui erittäin suoraviivaiseksi. Työkalujen osalta tuli selväksi, että on järkevää käyttää tarkoitukseen suunniteltuja työkaluja.

Tietojärjestelmien luominen koodin avulla luo omia tietoturvariskejään ja nämä tulee ottaa huomioon järjestelmiä luotaessa ja järjestelmien luomiseen käytettyjen koodien säilyttämisessä.

Avainsanat automaatio, ohjelmistokehitys, tietokoneohjelmat, virtualisointi

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Degree Programme in Information
and Communication Technology
Bachelor of Engineering

Author Jussi Saraniemi **Year** 2023
Supervisor Kenneth Karlsson
Commissioned by Lapland University of Applied Sciences, Degree Programme in Information and Communication Technology
Title Provisioning Virtual Machines with Infrastructure as Code Tools
Number of pages 84 + 18

The goal of this thesis study was to create three virtual machines to cloud environment and to On Premise environment and there should be a proven network connectivity between the aimed virtual machines.

Amazon Web Services was chosen as cloud environment, because it is one of the market leaders in cloud services. The installations to cloud environment were straight forward. The goals of the thesis goals stated that connectivity should be tested by using ICMP echo request message. To achieve the connectivity goal, a public private key pair had to be made because Amazon Web Services support only key pair authentication. New security group was also needed to Amazon Web Services account that allowed ICMP and SSH traffic. Terraform with VirtualBox in Windows environment was originally chosen to On Premise part of the thesis. The alternative way was Terraform in Linux environment with KVM virtualization. After discussing with the thesis instructor a decision was made to use Different tools for the On Premise part. On Premise was implemented with a tool called Vagrant. The connectivity was also an issue with the On Premise environment. Additional Network Interface Cards were made to support the ICMP echo requests between the different virtual machines.

The conclusion of thesis is that IaC tools are efficient for creating virtual machines. IaC tools save time in different kind of environments. IaC tools can be used in both Cloud and On Premise environments. The user needs to choose a correct IaC tool for different kinds of environments.

Keywords: Automation, Computer Programs, Infrastructure as Code, Virtualization

SISÄLLYS

KÄYTETYT LYHENTEET JA TERMIT

API	Application Programming Interface (Amazon Web Services 2023b)
CLI	Command Line Interface
EC2	Elastic Compute Cloud
IaC	Infrastructure as Code
ICMP	Internet Control Message Protocol
Kernel	käyttöjärjestelmän ydin, joka toimii linkkinä laitteen fyysisten resurssien ja käyttöjärjestelmän välillä (GeeksforGeeks 2022)
KVM	Kernel-based Virtual Machine
Libvirt	ohjelmointirajapinta virtualisointialustojen hallintaan
NIST	National Institute of Standards and Technology
On Premise	paikallinen tietojärjestelmäympäristö
ping	ICMP echo request -sanoma
Private Cloud	yksityinen pilvipalvelu yhden organisaation käyttöön
RSA	julkisen avaimen salausalgoritmi
Terraform	IaC-työkalu
Thin Client	yksinkertainen pienitehoinen tietokone, jolla esitetään virtualisoitua työpöytää (Fortinet 2023)
Vagrant	IaC-työkalu
VM	Virtual Machine
VMM	Virtual Machine Management

1 JOHDANTO

Opinnäytetyössä on tarkoitus tutustua tietojärjestelmäympäristöjen luomiseen koodin avulla. Tietojärjestelmäympäristöjen luominen koodin avulla tunnetaan yleisesti englanninkielisellä termillä Infrastructure as Code. Yleisesti puhekielessä käytetään lyhennettä IaC.

Laajojen tietojärjestelmäympäristöjen luominen käsin graafisen käyttöliittymän kautta on hidasta ja virheiden mahdollisuus on suuri. Virheet järjestelmien luomisvaiheessa tuottavat edelleen ongelmia, kun luotuihin koneisiin asennetaan ohjelmistoja. Tietojärjestelmäympäristöjen ylläpitäminen manuaalisesti vaarantaa ympäristön konfiguraation muuttumiseen siten, että samantyyppisillä palvelimilla on todellisuudessa erilaisia konfiguraatioita kuin toisella vastaavalla. Konfiguraatioiden muuttuminen lumihiutalemaisiksi johtuu inhimillisistä virheistä ylläpidon yhteydessä. (Red Hat 2017.)

Tietojärjestelmäympäristöjen luominen ja ylläpitäminen koodin avulla vähentää virheiden määrää. IaC-työkalulla luotavat koneet sekä ylläpidettävät koneet pysyvät yhdenmukaisina (VMware 2023a). Tietojärjestelmäympäristöjen luominen koodin avulla juontaa juurensa vuosituhannen alkuun. IaC-tekniikan käyttö luo kysymyksiä järjestelmien tietoturvallisuudesta. Tietojärjestelmien luomiseen käytettävien koodien tietoturvallisuutta ei ole tutkittu tieteellisesti vielä riittävästi.

IaC-tekniikan käyttö on yleistä sovelluskehitystoiminnassa. IaC-tekniikan käyttö sovelluskehityksessä lyhentää ohjelmistojen testausaikaa, koska itse ympäristöjen luominen nopeutuu ja työssä voidaan keskittyä varsinaiseen ohjelmistokehitykseen. Nopeuden lisäksi IaC-tekniikkaa hyödyntämällä saadaan aina samanlainen ympäristö, eikä inhimillisistä virheistä johtuvia variaatiota pääse syntymään. Tarpeen tullen ympäristöä on helppo muokata muuttamalla koodia halutttuun suuntaan.

IaC-tekniikkaa voidaan soveltaa myös muuhun kuin ohjelmistokehitykseen. Suurien tietojärjestelmäympäristöjen päivittäminen käsin on erittäin aikaa vievää ja vaatii paljon työaikaa. Automatisoimalla tietojärjestelmien ylläpito saadaan muutostyöt tehtyä pienemmällä työmäärällä, mikä voidaan yrityksen johdossa mitata suoraan rahassa ja säästettynä aikana.

Opinnäytetyön tarkoituksena on tutkia Terraform-työkalun soveltuvuutta sekä pilvipalveluympäristöihin että paikallisiin virtualisointiympäristöihin. Tarkoituksena on luoda IaC-tekniikan avulla kolme virtuaalikonetta, joiden välillä voidaan lähettää onnistuneet ICMP echo request -viestit sekä pilvipalveluun että paikalliseen ympäristöön.

Opinnäytetyössä keskitytään siihen, miten saadaan luotua kolme virtuaalikonetta Amazon Web Services -pilvipalveluympäristöön. Opinnäytetyön toinen toiminnallinen tehtävä on luoda kolme virtuaalikonetta paikalliseen virtualisointiympäristöön käyttäen joko Windows-ympäristöä ja VirtualBox-virtualisointiohjelmaa tai vaihtoehtoisesti Linux-ympäristössä KVM-virtualisointia ja Libvirt-ohjelmointirajapintaa. Opinnäytetyön tuloksena syntyy seikkaperäinen ohje, jonka avulla saadaan luotua kolme virtuaalikonetta todennettavien keskinäisten verkkoyhteyksien kanssa sekä Amazon Web Servicessä että paikallisessa virtualisointiympäristössä.

Kiinnostukseni IaC-tekniikkaan johtuu työtehtävistäni virtualisoitujen tietojärjestelmäympäristöjen ylläpidon kanssa. Käsin tehtäville ylläpitotoimille on olemassa jo vaihtoehto, varsin laajasti käytetty tekniikka nimeltään IaC.

2 VIRTUALISOINTI

Virtualisointi on Andrew Herbertin mukaan käyttöjärjestelmän ominaisuus, joka luo illuusion, että useita itsenäisiä tietokoneita voidaan ajaa yhtaikaa yhdellä tietokoneella. Käyttöjärjestelmän ominaisuus antaa käyttäjälle tunteen, että hän käyttää oikeaa erillistä tietokonetta. (Herbert 2015)

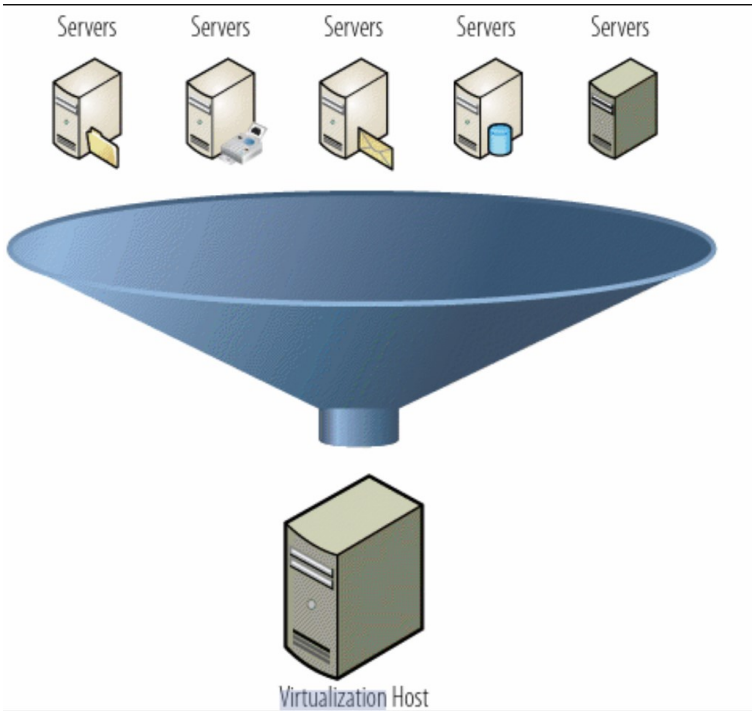
William Von Hagen kuvailee virtualisoinnin olevan loogisesti fyysisestä resurssista erotettu palvelupyyntö, joka kuitenkin todellisuudessa tuotetaan fyysisessä resurssissa. Virtualisointi mahdollistaa palvelujen suorittamisen loogisesti erotetussa järjestelmässä, joka toimii itsenäisesti irrallaan isäntäkoneesta, vaikka todellisuudessa käyttääkin isäntäkoneen resursseja. (Von Hagen 2008, 2.)

2.1 Hypervisor

Hypervisor tai toiselta nimetään Virtual Machine Manager (VMM) toimii tulkkina fyysisen ja virtuaalisen tietokoneen välillä. Hypervisor mahdollistaa tietokoneen tarjolla olevien resurssien tarkemman käytön. Hypervisor eristää virtuaalikoneen fyysisen koneen resursseista, mikä mahdollistaa virtuaalikoneen siirtämisen toisen palvelimen resurssien varaan käynnissä ollessaan. (VMware 2023b.)

Virtualisointi kehitettiin alun perin tarpeeseen ajaa erilaisia käyttöjärjestelmiä samalla fyysisellä koneella. Myöhemmässä vaiheessa virtualisoinnilla saatiin muun muassa käyttöönotettua erillispalvelimien tyhjäkäyntiresurssi. Ennen virtualisointia jokaiselle asiakasohjelmalle piti olla oma palvelimensa. Fyysiset palvelimet mitoitettiin alun perin siten, että ne kykenivät suoriutumaan omasta tehtävästään noin 20–50 prosentin yliresursoinnilla. Palvelinlaitteiden elinkaaripäivityksien myötä saatiin lisää laskentatehoa ja käyttömuistia, vaikka itse asiakasohjelma ei niitä tarvinnutkaan. Muutaman elinkaaripäivityksen jälkeen oltiin tilanteessa, että erillispalvelimilla oli jopa 90 prosenttia tyhjäkäyntiä. (Portnoy 2016, 10.)

Virtualisoinnilla saadaan laskettua yritysten IT-infrastruktuurin kustannuksia. Virtualisoimalla palvelimia säästetään laitesaleille tarvittava tilaa, sähköä, jäähdytyksen tarve vähenee eikä fyysisiä palvelimia tarvitse hankkia niin montaa. (Portnoy 2016, 10.) Kuviossa 1 nähdään visualisointi fyysisten palvelimien vähenemisestä.



Kuvio 1. Fyysiset palvelimet virtuaalisiksi (Portnoy 2016, 11)

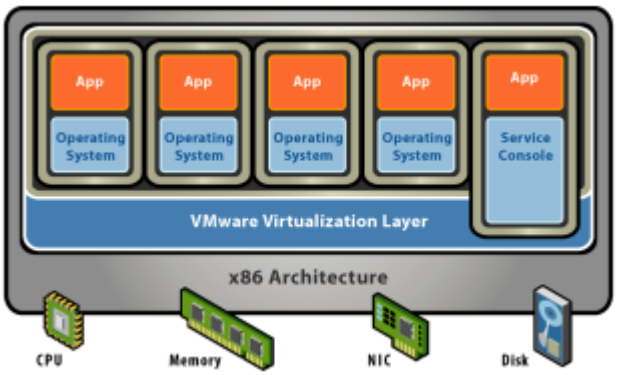
2.2 Erilaiset virtualisointitavat

Virtualisointitapoja on useita. Yleisin virtualisointitapa on tyypin 1 hypervisor, joka toimii suoraan fyysisen raudan kanssa. Tyypin 1 hypervisor keskustelee suoraan fyysisen koneen resurssien kanssa ja hallitsee asennettuja virtuaalikoneita. Tyypin 2 hypervisor on asennettu jonkin käyttöjärjestelmän päälle. Tyypin 2 hypervisor keskustelee tietokoneen resurssien kanssa asennetun käyttöjärjestelmän kautta. (Rathod & Townsend 2014, 11–12.)

Laitevirtualisointi on yleisin virtualisointitapa. Laitevirtualisoinnissa hypervisor tai VMM luovat loogisesti erotetun tietokoneen, joka käyttää fyysisen koneen resursseja. Virtuaalisella koneella on oma käyttöjärjestelmänsä ja omat ohjelmistonsa. Jokaisella erillisellä virtuaalikoneella on oma kernelinsä. (Von Hagen 2008, 4.)

Portnoyn mukaan hypervisor esittää fyysisestä koneesta abstraktion virtuaalikoneelle käytettäväksi. Laitevirtualisoinnissa hypervisor on asennettuna suoraan fyysiseen koneeseen, eikä väliin ole asennettu erillistä käyttöjärjestelmää. (Portnoy 2016, 16.) Laitevirtualisointia kutsutaan yleisesti tyypin 1 hypervisoriksi. Tyypin 1 hypervisor ohjaa suoraan fyysisen koneen resursseja sekä hallitsee virtuaalikoneen käyttöjärjestelmää. Fyysisen koneen resurssit voidaan jakaa useille

eri virtuaalikoneille. (Rathod & Townsend 2014, 11–12.) Kuviossa 2 on esitetty fyysisen isäntäkoneen suhdetta virtuaalikoneen käyttöjärjestelmään. Fyysisen isäntäkoneen ja virtuaalikoneen käyttöjärjestelmän välillä toimii hypervisor tai VMM, joka jakaa fyysisen koneen resursseja virtuaalikoneille.



Kuvio 2. Virtuaalikone suhteessa fyysiseen palvelimeen (VMware 2005, 13)

2.3 Virtualisoinnin hyödyt ja haitat

Virtualisoinnilla saadaan fyysisen tietokoneen resurssit tarkemmin käyttöön. Virtualisoimalla tietojärjestelmäinfrastruktuurin säästyy rahaa sekä perustamis- että ylläpitovaiheessa. (Portnoy 2016, 10.) Hankittavia laitteita on vähemmän, koska useat eri palvelut voidaan perustaa saman fyysisen palvelimen resursseille. Ylläpitotoimet voidaan tehdä keskitetysti.

Virtualisoidulla alustalla saadaan joustavuutta verrattuna fyysisiin palvelimiin. Fyysiset palvelimet ostetaan tietyllä varustuksella ja tietyllä suorituskyvyllä. Virtualisoidussa alustassa suorituskykyä voidaan sovittaa virtualisoidun palvelimen tarpeita vastaavaksi fyysisen palvelimen rajoitukset huomioon ottaen. (Portnoy 2016, 73.)

Virtualisoinnissa virtuaalikoneiden rautavikasietoisuus ei ole niin hyvin turvattu kuin erillispalvelimille asennettujen palvelimien. Tuotantoympäristöissä virtualisointiympäristöt ovat kuitenkin vähintään kahdennettuja laitevikojen väistämiseksi (Portnoy 2016, 246).

Hypervisor on kaikkien asennettujen virtuaalikoneiden yhdistävä tekijä. Hypervisoriin vaikuttamalla voidaan pahimmassa tapauksessa lamauttaa koko virtualisointiympäristö. (IBM 2021.)

2.4 Pilvipalvelu

Yhdysvaltojen valtion virallisen määritelmän mukaisesti pilvipalvelu on pääsy jaettuun määriteltävään laskentaresurssiin, joka sisältää verkko-, palvelin-, tallennus- ja sovelluspalveluja. Palvelut voidaan luoda ja antaa nopeasti käyttöön ilman palveluntarjoajan vaikutusta. (GovInfo 2023.) Kyberturvallisuuskeskuksen määritelmä on muodostettu edellä olevasta National Institute of Standards and Technologyn määritelmästä (Kyberturvallisuuskeskus 2023).

Virtualisointi mahdollisti pilvipalvelujen syntymisen. Isot palveluntarjoajat rakensivat ja rakentavat isoja palvelinkeskuksia, joista yrityksille ja yksityisille ihmisille tarjotaan kapasiteettia maksua vastaan. (Portnoy 2016, 32.)

Opinnäytetyössä käytettiin Amazon Web Services -pilvipalvelutarjoajaa, joka EC2-palvelullaan tarjoaa mahdollisuuden luoda virtuaalikoneita. AWS tarjoaa 12 kuukauden ilmaisen kokeilujakson. Ilmaiseen kokeilujaksolla AWS:n palveluja voi kokeilla rajoituksin. Ilmainen palvelu muuttuu 12 kuukauden jälkeen automaattisesti maksulliseksi, mikäli palvelussa on kokeiluajan jälkeen virtuaalikoneita kuluttamassa AWS:n resursseja. (Amazon Web Services 2023c.)

BPaaS tulee sanoista Business Process as a Service, joka on suomennettuna liiketoimintaprosessi palveluna. BPaaS-mallissa palvelu hankitaan kokonaisuudessaan. BPaaS-mallissa ei hankita välttämättä ollenkaan teknologiaa, vaan se tulee hankittavan prosessin osana. (Valtiovarainministeriö 2020, 20.)

SaaS tulee sanoista Software as a Service, joka on suomennettuna järjestelmä palveluna. Tässä mallissa hankitaan jokin järjestämä palveluntuottajalta ja sitä käytetään internetin yli. (Valtiovarainministeriö 2020, 20.)

PaaS tulee sanoista Platform as a Service, joka on suomennettuna Alusta palveluna (Valtiovarainministeriö 2020, 20). Palvelualusta hankitaan palveluna, eli se

sisältää tietojärjestelmäympäristön ja esimerkiksi ohjelmistokehittämiseen tarkoitettuja sovelluksia (Telia 2018).

IaaS tulee sanoista Infrastructure as a Service, joka on suomennettuna Infrastruktuuri palveluna. Hankitaan tietojärjestelmäalusta palveluna, jolloin ei tarvita omia laitteita tietojärjestelmien alustoiksi vaan käytetään ostettua kapasiteettia. (Valtiovarainministeriö 2020, 20.)

3 TIETOJÄRJESTELMIEN LUOMINEN KOODIN AVULLA

3.1 Määritelmä

Infrastructure as Code eli tietojärjestelmien luominen koodin avulla tarkoittaa Microsoftin määritelmän mukaan tekniikkaa, jolla luodaan, ylläpidetään ja muokataan verkkoja, virtuaalikoneita, kuorman jakajia ja yhteystopologioita. IaC-tekniikan avulla voidaan luoda toistettavia tietojärjestelmäympäristöjä. Ympäristöjen luominen ja ylläpito IaC-tekniikan avulla mahdollistaa ympäristöjen pitämisen yhdenmukaisina. (Microsoft 2022.)

Infrastructure as Code on National Institute of Standards and Technologyn määritelmän mukaan prosessi, jossa luodaan ja hallitaan tietojärjestelmiä koneluettavan koodin avulla. Koodin aktivointi luo, poistaa tai muuttaa tietojärjestelmäympäristöjä ilman fyysisiä toimenpiteitä ja erillisiä asennustyökaluja. (NIST 2023, 19.) IBM:n mukaan IaC automatisoi pilvipalveluun rakennettavien tietojärjestelmäympäristöjen luomisen. IaC-tekniikoiden käyttäminen nopeuttaa, vähentää riskejä ja laskee kustannuksia. (IBM 2023.)

3.2 Edeltävät tekniikat

Ennen IaC-aikaa tietojärjestelmät asennettiin virtuaaliympäristöihin käsin konfiguroimalla, pilvipalvelun tai virtualisointialustan käyttöliittymän kautta. Kehittyneempi tapa oli käyttää ympäristöjen luomisessa esimerkiksi PowerShell-skriptejä. Edellä mainittuja tapoja käyttäen on mahdollista, että ympäristöjä ei kyetä kuitenkaan toistamaan, koska käsin tehden virheen mahdollisuus on todennäköinen. (Modi 2019, 9.)

Käsin tai skriptien avulla tehty konfigurointi ei mahdollista ympäristöjen etukäteisdiagnosointia. Ilman IaC-tekniikoita mahdollisten virheiden etsintä on työläämpää. Edellä mainitut tavat ovat hitaampia ja yrityksille siitä syystä kalliimpia kuin IaC-tekniikat. (Modi 2019, 9.)

IaC-tekniikkojen kehitystä nopeutti esimerkiksi AWS-pilvipalvelujen tuleminen yleisesti saataville. Suurten ympäristöjen hallitseminen skriptien avulla muuttui

monimutkaiseksi. Kehitettiin IaC-tekniikat, joilla voidaan hallita tietojärjestelmäympäristöjä lähdekoodien avulla. IaC-koodeja voitiin käsitellä kuten muitakin ohjelmointikoodeja, käyttäen versionhallintaa. (Hasan, Bhuiyan & Rahman 2020, 8.)

3.3 Ohjelmistokehitystyö

IaC-tekniikkaa käytetään yleisesti ohjelmistokehitystyössä, kun hyödynnetään DevOps-mallia. DevOps lyhenne tulee sanoista Development ja Operations. DevOps kuvaa kehityksen ja tuotannon keskinäistä yhteistyötä. DevOps-mallissa pyritään vastaamaan nopeasti tuotanto- ja kehitystehtäviin. DevOps-malli pyrkii lyhentämään tuotekehitystyön ja käyttöönottamisen välistä aikaa ja silti tuottamaan luotettavasti uusia ominaisuuksia, korjauksia ja päivityksiä. (NIST 2020.)

IaC-tekniikoilla luotavat ympäristöt määritellään koodin avulla ja samaa koodia käytetään tietojärjestelmien luomiseen. Ympäristöjen luontiin käytetyt määrittelykoodit mahdollistavat samanlaisten versionhallintatyökalujen käyttämisen kuin muukin ohjelmointi. Versionhallintatyökalujen avulla saadaan samalla dokumentoitua luodut ympäristöt. Versionhallintatyökalut helpottavat kehitystiimien työskentelyä. (Modi 2019, 9.)

3.4 IaC-työkalut

IaC-työkalut toimivat kahdella eri tavalla. Julistavat työkalut kertovat lopputilanteen, johon halutaan päästä, mutta määrittelytiedostoissa ei kerrota tarkkoja tapoja, siihen miten lopputulos saavutetaan. Kerrotaan resurssit ja ominaisuudet, minkälaiseen lopputulokseen halutaan päästä, ja koodi luo IaC-työkalun avulla halutun tyyppisen ympäristön. (Sokolowski 2022.) Julistava tapa pitää yllä tietoa tietojärjestelmäympäristön senhetkisestä tilasta, jolloin ympäristön poistaminen on hallitumpaa (Red Hat 2022a).

Imperatiivinen lähestymistapa määrittelee yksityiskohtaisesti komennot, joilla haluttuun lopputulokseen päästään (Sokolowski 2022). Imperatiivinen tapa määrittelee myös järjestyksen, jossa komennot tulee suorittaa tavoitteen saavuttamiseksi (Red Hat 2022a).

Tietojärjestelmän muutoksien teko on yksinkertaisempaa julistavalla työkalulla, koska julistava tapa tekee muutokset automaattisesti. Imperatiivisessa tavassa muutoksen tekijän pitää itse tietää, missä järjestyksessä ja miten muutokset tehdään, jotta haluttu lopputulos saavutetaan. (Red Hat 2022a.)

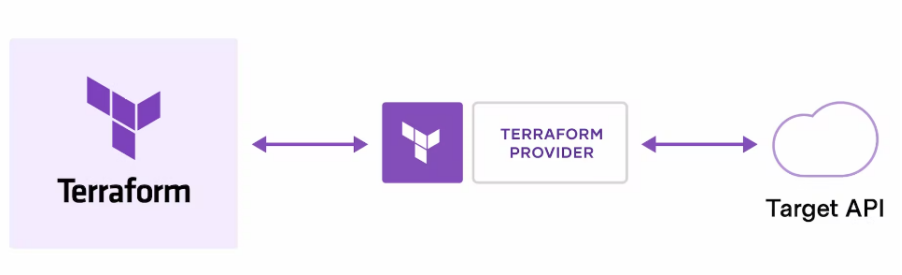
IaC-työkalut keskustelevat virtualisointiohjelmistojen ja pilvipalvelujen kanssa käyttäen sovellusohjelmointirajapintoja. Yleisimpiä käytössä olevia IaC-työkaluja ovat muun muassa Chef, Puppet, Ansible, Saltstack, Terraform, Kubernetes ja Azure Resource Manager.

Opinnäytetyössä käytettiin kahta Hashicorpin tarjoamaa IaC-työkalua nimeltään Terraform ja Vagrant. Terraform-työkalua käytettiin Amazon Web Services -pilvipalvelun kanssa ja Vagrantia käytettiin On Premise -osiossa.

3.5 Terraform

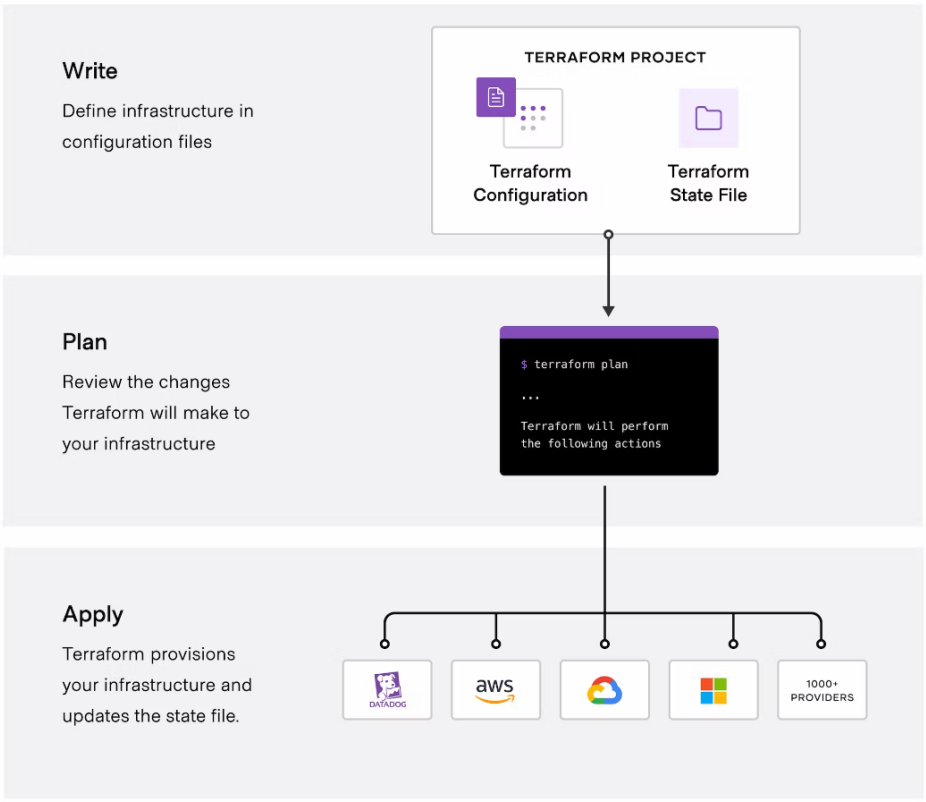
Terraform on yksi tunnetuimmista ja käytetyimmistä IaC-työkaluista, josta löytyy tuki tunnetuimmille pilvipalveluille (Gupta, Chowdary, Bussa & Chowdary 2021, 1). Terraform käyttää Hashicorpin omaa konfigurointikieltä nimeltään HCL. Lyhenne tulee sanoista HashiCorp Configuration Language (Tankov, Valchuk, Golubev & Bryksin 2022, 1168). Terraformin koodi on JSON-muodossa (Gupta ym. 2021, 2). Terraform on avoimen lähdekoodin ohjelmisto. Siitä on saatavilla myös maksullisia versioita kaupalliseen käyttöön (Hashicorp 2023b). Terraform toimii sekä pilvipalveluiden että paikallisten tietojärjestelmäympäristöjen kanssa (Hashicorp 2023d).

Terraform luo ja hallitsee resursseja sovellusohjelmointirajapintojen eli API:en kautta. Kuviossa 3 nähdään Terraformin toimintaperiaate. Terraform tarjoaa tuhansia erilaisia providereita, joiden avulla voidaan hallita suurinta osaa virtualisointi- ja pilvipalvelualustoista (Hashicorp 2023b).



Kuvio 3. Terraformin toimintatapa (Hashicorp 2023d)

Terraformin toiminta perustuu kolmeen toimintovaiheeseen, jotka ovat write, plan ja apply. Write-vaiheessa käyttäjä määrittelee luotavat resurssit. Plan-vaiheessa Terraform luo suunnitelman, siitä miten halutut resurssit saadaan luotua, päivitettyä tai poistettua. Plan-vaiheessa Terraform tarkastaa, onko konfiguraatio toteutuskelpoinen. Apply-vaiheessa Terraform toteuttaa edellä luodun suunnitelman. Kuviossa 4 nähdään toimintovaiheet. (Hashicorp 2023d.)



Kuvio 4. Terraform-toimintovaiheet (Hashicorp 2023d)

Konfigurointitiedosto määritellään seuraavasti. Kaikki konfigurointitieto määritellään terraform {} -lohkoon. Required provider {} -lohkoon määritellään source,

joka määrittelee käytettävän providerin. Kuviossa 5 on määritelty providerin versio, mutta se ei ole pakollinen tieto. Mikäli providerin versiota ei määritellä, käytetään uusinta versiota. Kuvion 5 esimerkissä on käytetty aws provideria, jota käytetään terraform registrystä. (Hashicorp 2023e.)

Source-määrittelyssä tieto hashicorp/ määrittelee, että provideria käytetään terraform registrystä. Terraform registry on kirjasto, jossa on Hashicorpin tarjoamat providerit (Hashicorp 2023f). Provider {} -lohkossa konfiguroidaan itse provideria. Kuvion 5 esimerkissä on määritelty AWS-palvelun alueeksi us-west-2, johon tietojärjestelmäympäristö asennetaan (Amazon Web Services 2023e).

Resource {} -lohkossa määritellään luotavan tietojärjestelmän resurssit. Tässä tapauksessa resurssityypiksi on määritelty aws_instance ja nimeksi on annettu app_server. Tyyppimäärittelyssä annettu aws_ määrittelee käytetyn providerin. Resource {} -lohkoon voidaan määritellä esimerkiksi kiintolevyn koko ja verkkokorttien konfiguraatioita. Kuviossa 5 resource {} -lohkossa on määritelty resurssimäärittelyiksi ami-830c94e3, joka viittaa AWS-palvelun Ubuntu imageen. (Hashicorp 2023f.) AMI-lyhenne tulee sanoista Amazon Machine Image. Instanssi-tyypiksi on määritelty t2.micro, joka viittaa AWS-palvelun ilmaisen kokeilukäyttöjakson virtuaalikonetyyppiin ja sen resursseihin (Amazon Web Services 2023d).

Tags {} -lohkossa määritellään ympäristölle nimi. Kuvion 5 esimerkissä ympäristön nimeksi on määritelty ExampleAppServerInstance (Hashicorp 2023n).

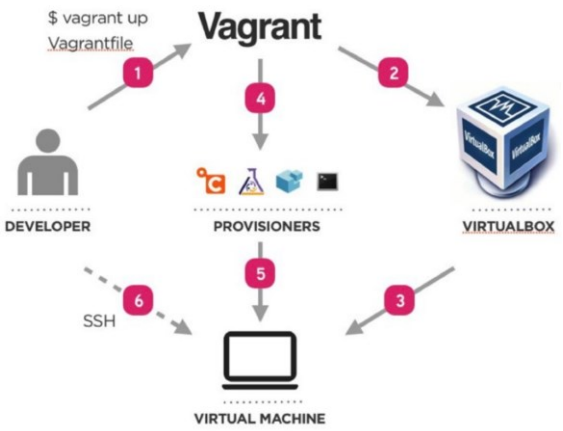
Kuvio 5. Terraform-konfigurointitiedosto

Hashicorpin sivuilta löytyvät selkeät ohjeet tunnetuimpien pilvipalveluiden käyttöön. Sivuilta löytyvät tutoriaalit esimerkiksi Google Cloudille ja Microsoft Azurelle. (Hashicorp 2023c.)

3.6 Vagrant

Vagrant on Terraformin ohella Hashicorpin IaC-työkalu. Vagrant on tarkoitettu kehitysympäristöjen luomiseen, muokkaamiseen ja tuhoamiseen. Terraformista poiketen Vagrant tarjoaa enemmän ominaisuuksia kehitysympäristöjen vaatimiin ominaisuuksiin, kuten kansioiden synkronoimiseen, HTTP-tunnelointiin ja verkkojen automatisointiin. Vagrant on suunniteltu pääasiallisesti paikallisten kehitysympäristöjen luomiseen ja hallinnointiin. (Hashicorp 2023g.) Vagrant tarjoaa tuen tunnetuimmille On Premise -ympäristöille, kuten VirtualBox ja VMware (Hashicorp 2023h). Vagrantilla voidaan hallita pilvipalveluita kuten AWS, mutta pääasiallinen kohde on On Premise -ympäristöt.

Vagrant toimii siten, että ensin luodaan Vagrantfile, jossa määritellään Ruby-ohjelmointikielellä halutun ympäristön resurssit ja mahdolliset ohjelmat. Vagrantfile luodaan komennolla *vagrant init*. *Vagrant init* -komento tulee ajaa siinä hakemistossa, jossa halutaan projektin tietojen olevan. Jokaiseen projektiin tarvitaan oma Vagrantfilensä. Vagrantfile toimii myös projektin dokumentaationa ja se voidaan tallentaa versionhallintaan. Varsinaisen tietojärjestelmäympäristön luominen tapahtuu vagrant up -komennolla (Hashicorp 2023i.) Kuviossa 6 nähdään Vagrantin toimintaperiaate.



Kuvio 6. Vagrant-toimintamalli (Šimec, Držanić & Lozić 2018, 50)

Vagrant käyttää virtuaalikoneiden luomiseen boxeja, jotka sisältävät pohjana toimivan levykuvan, josta virtuaalikone kloonataan. Samaa levykuvaa voidaan käyttää erilaisten ympäristöjen luomiseen. Ympäristöjen luominen ei vaikuta levykuvaan, vaan se säilyy muuttumattomana. Ympäristöjen luomisessa voidaan käyttää verkossa olevia Hashicorpin ylläpitämiä boxeja tai sitten box voidaan itse luoda oman tarpeen mukaisesti. Kuviossa 7 nähdään vagrantfilen perussisältö. Vagranfile sisältä luontivaiheessa seikkaperäiset ohjeet, miten luotavaa ympäristöä voidaan konfiguroida. Kuvion 7 konfiguraatiossa ohjetekstit on poistettu ja jäljellä on vain tarvittavat rivit yhden virtuaalikoneen luomiseksi. Tietue config.vm.box = ”hashicorp/bionic64” määrittelee käytettävän levykuvan. Mikäli boxia ei ole ladattu hallintakoneelle valmiiksi, lataa edellä oleva konfiguraatiorivi määritellyn boxin. Määrittelyrivissä Hashicorp on käyttäjänimi ja bionic64 käytettävä boxi. bionic64 on Linux Ubuntu versio 18.04 LTS 64bit, joka soveltuu Virtualboxille, VMware desktopille ja Hyperveelle. (Hashicorp 2023k.)

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
end
```

Kuvio 7. Vagrantfile box -määrittelyllä (Hashicorp 2023k)

Mikäli käyttäjä haluaa määritellä boxiksi jonkun tietyn version, voidaan se tehdä lisäämällä konfiguraatioon määrittely config.vm.box_version = ”haluttu versionumero” (Hashicorp 2023k). Kuviossa 8 nähdään esimerkki versiomäärittelystä.

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.box_version = "1.0.282"
end
```

Kuvio 8. Box-version määrittely (Hashicorp 2023k)

Boxille voidaan määritellä tarvittaessa URL-osoite, josta se ladataan. Kuviossa 9 nähdään URL-osoite määriteltynä, määritelmällä config.vm.box_url = https://vagrantcloud.com/hashicorp/bionic64. Edellä mainittu määrittely on kuitenkin turha. URL-osoitteen määrittelemättä jättäminen aiheuttaisi sen, että se ladattaisiin juuri edellä mainitusta osoitteesta. (Hashicorp 2023k.)

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.box_url = "https://vagrantcloud.com/hashicorp/bionic64"
end
```

Kuvio 9. URL-osoite määriteltynä (Hashicorp 2023k)

Hashicorp ylläpitää kirjastoa, jossa on tarjolla erilaisia boxeja eri virtualisointialustoille. Kirjasto on nimeltään Vagrant Cloud (Hashicorp 2023l). Edellä olevien

määrittelyiden jälkeen voidaan ajaa vagrant up -komento, jonka jälkeen Vagrant luo halutun ympäristön olemassa olevaan paikalliseen virtualisointiympäristöön.

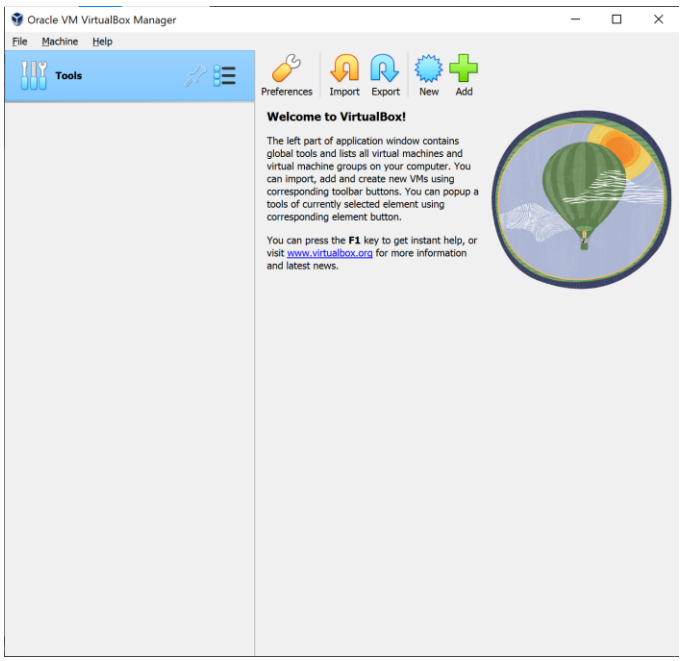
3.7 Chocolatey

Chocolatey on paketinhallintaohjelma Windows-käyttöjärjestelmille. Se mahdollistaa ohjelmien asentamisen ilman asennusvelhoja. Asennukset tapahtuvat komentokehotteen kautta. Chocolatey toimii samaan tapaan kuin Linux Debian jakelussa apt-get -komentoja käyttäen. (Luchian, Filip, Rus, Ivanciu & Dobrota 2016, 3)

Chocolatey käyttää ohjelman asennuksessa kirjastoja, joihin on määritelty ohjelman tarvitsemat palaset. (Luchian ym. 2016, 3). Chocolatey-paketti sisältää kaiken tarvittavan ohjelman asennukseen. Chocolatey-pakettiin on koottu kaikki pakatut tiedot, skriptit ynnä muut asennukseen tarvittavat tiedostotyypit. (Chocolatey 2023b, 3.)

3.8 Virtualbox

VirtualBox on Oraclen omistama avoimen lähdekoodin työpöytäsovellus, jolla voidaan virtualisoida x86 ja AMD64/Intel64 -käyttöjärjestelmiä. VirtualBox toimii useimmilla käyttöjärjestelmillä, kuten Windows, macOS ja Linux. VirtualBox osaa virtualisoida tunnetuimpia käyttöjärjestelmiä. (VirtualBox 2023a.) Kuviossa 10 näkyy VirtualBoxin käyttöliittymä Windows-työpöydällä.



Kuvio 10. VirtualBox-käyttöliittymä

VirtualBox on määritelmän mukaan tyypin 2 hypervisor, eli VirtualBox toimii käyttöjärjestelmän päällä (Virtualbox 2023b, 3). VirtualBox mahdollistaa useiden virtuaalikoneiden käyttämisen yhtä aikaa. Virtuaalikoneiden määrää rajoittaa isäntäkoneen resurssit. (VirtualBox 2023b, 1.) Yksittäiselle koneelle voidaan määritellä halutut resurssit, kuten käyttömuistin määrä, käytettävissä olevan kiintolevyn koko sekä prosessorin ytimien lukumäärä (VirtualBox 2023b, 31–32).

3.9 KVM

KVM-lyhenne tulee sanoista Kernel-based Virtual Machine. KVM on avoimen lähdekoodin virtualisointiohjelmisto, joka toimii Linux-käyttöjärjestelmän päällä. KVM tulee vakiona kaikissa tämänhetkisissä Linux-jakeluissa, jotka käyttävät perustanaan kernel.orgin tuottamaa Linux-kerneliä. (KVM 2023.) KVM tulee Linux-jakelun mukana, mikäli Linux-versio on 2.6.20 tai uudempi (Red Hat 2022b).

KVM muuntaa Linuxin tyypin 1 hypervisoriksi. KVM vaatii toimiakseen x86 tyyppisen prosessorin ja virtualisointituen. KVM muuntaa kaikki virtuaalikoneet Linuxin omiksi prosesseiksi. (Red Hat 2022b.)

3.10 Libvirt

Libvirt on ohjelmointirajapinta virtualisointialustojen hallintaan Linux-käyttöjärjestelmissä. Libvirt tukee muun muassa seuraavia virtualisointialustoja KVM, QEMU ja Xen. Libvirt on avoimen lähdekoodin ohjelmisto. (Libvirt 2023a.) Libvirt mahdollistaa helpon tavan hallita virtuaalikoneita sekä muita virtualisointiin liittyviä ominaisuuksia. Libvirt sisältää muun muassa ohjelmistorajapintakirjaston. Libvirt mahdollistaa eri virtualisointialustojen hallinnan ilman järjestelmäkohtaisia käyttöliittymiä. (Libvirt 2023b.)

Libvirt tarjoaa myös mahdollisuuden tallennustilan hallintaan. Libvirtillä voidaan luoda esimerkiksi levykuvia eli imageja. Libvirt mahdollistaa myös levyjen osioinnin. (Libvirt 2023b.)

4 TIETOTURVA

4.1 Virtualisoinnin tietoturva

Virtualisoinnin etuja tietoturvanäkökulmasta on, että virtuaalikoneet ovat eristettyjä fyysisen koneen käyttöjärjestelmästä ja fyysisistä komponenteista (Von Hagen 2008, 2). Virtualisoitua tietokonetta on kuitenkin käsiteltävä tietoturvanäkökulmasta, kuten fyysistäkin tietokonetta (VMware 2019a). Virtuaalikoneet ovat alttiita muun muassa viruksille ja haittaohjelmille. Jokaisessa virtualisoidussa tietokoneessa suositellaan käytettäväksi virustorjuntaa. Virtuaalikoneiden virusskannaukset kannattaa kuitenkin ajastaa siten, että ympäristön jokainen kone ei ala tekemään tarkistusta yhtä aikaa. Yhtäaikainen virustarkastus voi aiheuttaa suuren kuormituspiikin tietojärjestelmäympäristössä. (VMware 2019b.)

Virtualisoitujen alustojen tietoturvallisuuteen vaikuttavia seikkoja on mainittu seuraavissa kappaleissa.

Resurssien kaappauksella tarkoitetaan sitä, että virtuaalikoneille pitää tehdä suunnitelma resurssien jakamisesta siten, että yksittäinen virtuaalikone ei voi aiheuttaa palvelunestohyökkäystä (Viestintävirasto 2016, 2). Järjestelmän resursseja voidaan jakaa sijoittelemalla virtuaalikoneet käyttötarkoituksen perusteella erilaisiin omiin pooleihinsa (VMware 2019a).

Virheellisesti konfiguroitu virtuaalikone tai virtualisointialustan palvelin aiheuttaa riskin järjestelmään tunkeutumiselle. Esimerkiksi vahingossa päälle jäänyt tarpeeton palvelu tekee järjestelmästä haavoittuvan (VMware 2019a). Tiedostojen jaon salliminen virtuaalikoneiden välillä tekee järjestelmästä haavoittuvan (Liquid Web 2021). VMware suosittelee käyttämään templateja virtuaalikoneiden luonnissa. Template on ennakkoon asennettu, huolellisesti konfiguroitu ja kertaalleen asennettu virtuaalikone, jonka pohjalta voidaan nopeasti luoda kopioimalla uusia koneita. (VMware 2019a.)

Puutteellinen **pääsyn hallinta** käsittää muun muassa käyttäjätunnusten, salasanojen ja mahdollisten fyysisten tunnistautumismenetelmien puutteellisen hallinnan. Puutteellinen pääsyn hallinta mahdollistaa järjestelmään tunkeutumisen ja

sitä kautta vaarantaa koko virtuaalisen tietojärjestelmäympäristön. (CSA 2015, 23.)

Järjestelmän varmistamiseksi otetut **offline varmenteet** eivät sisällä mahdollisesti uusimpia tietoturvapäivityksiä. Mahdollisessa vikatilanteessa käyttöönotettu varmuuskopio voi vaarantaa järjestelmän. (Liquid Web 2021.)

Hypervisor on virtualisoidun tietojärjestelmäympäristön yhdistävä tekijä kaikille ympäristön virtuaalikoneille (Liquid Web 2021). Hypervisor voidaan mahdollisesti lamauttaa aiheuttamalla riittävä määrä kyselyjä virtuaalikoneelta hypervisorille (IBM 2021).

Mikäli virtualisoitualusta sisältää myös julkisia osia, mahdollistavat **sovellusohjelmointirajapinnat** järjestelmään tunkeutumisen. API:en turvallisuus tulee varmistaa tunkeutumisen estämiseksi. API:en käyttäminen tulee ottaa huomioon pääsynhallintaa suunniteltaessa. (CSA 2015, 20–21.)

Virtualisoitua tietojärjestelmäympäristöä hallitsevan tahon tulee luoda prosessit siten, että virtuaalikoneiden elinkaari on jokaisessa kohdassaan hallittua (Kyberturvallisuuskeskus 2020, 17). Järjestelmän varmuuskopiot tulee säilyttää siten, että niillä oleva tieto ei vaarannu. Varmuuskopiot voidaan tarvittaessa teknisesti salata. Virtuaalikoneen varmuuskopio sisältää koko koneen kaikkine tietoineen ja poikkeaa siten fyysisen koneen varmuuskopiosta. (Posey 2023, 4.) Pääsynhallinta tulee olla hallittua siten, että vain järjestelmän ylläpitäjillä ja luvallisilla henkilöillä on pääsy järjestelmään. Järjestelmää tulee valvoa ja järjestelmän tapahtumat tulee lokittaa. Lokit pitää myös auditoida määrävälein, jolloin päästään mahdollisten tietomurtojen tai väärinkäytösten jäljille mahdollisimman pian. Virtualisointialusta tulee pitää päivitysten osalta ajan tasalla. (Kyberturvallisuuskeskus 2020, 35.) Järjestelmän ylläpitämisessä kannattaa käyttää keskitettyä hallintaa. Keskitettyhallinta mahdollistaa alustan tehokkaan resurssien hallinnan ja keskitetyn päivitysten jakamisen. (Liquid Web 2021.)

4.2 IaC-tietoturva

IaC-koodissa olevat virheet ovat myös ympäristöissä, jotka on luotu virheellisellä koodilla. Koska IaC-tekniikalla ympäristöjen luominen on nopeaa, riski virheellisen koodin leviämiseen on suuri. (Tolliver 2022.) Virheellisen koodin leviäminen voi aiheuttaa suuria taloudellisia tappioita, kuten vuonna 2017 Amazonin ylläpitohenkilön antama väärä komento järjestelmien ylläpidossa poisti liian suuren määrän palvelimia ja aiheutti satojen miljoonien dollareiden vahingot ja useiden tuntien mittaisen palvelukatkoksen AWS:n päälle rakennetuissa palveluissa. (DataCenterKnowledge 2017.) Virheellinen koodi AWS-palveluun annettiin IaC-skriptin muodossa. Virheellisen koodin leviämisen estämiseksi tulee IaC-tekniikoiden käyttäjien ottaa käyttöön IaC-koodien etukäteistestaaminen. Koodin testauksen käyttöönotossa haasteena on määritellä se, mitä ja miten halutaan testata. (Hasan ym. 2020, 8.)

Ohjelmistokehitystyössä tietojärjestelmäympäristöjä joudutaan päivittämään usein. Mikäli ympäristöjä joudutaan päivittämään usein, tulee virheellisen koodin käyttöönottaminen todennäköisemmäksi. (Rahman 2018, 476.)

Yksi tapa tunnistaa virheellistä IaC-koodia on etsiä koodissa olevista virheellisyyksistä tunnusmerkkejä ja etsiä niiden perusteella muista koodeista samoja tunnusmerkkejä. Järjestelmän turvallisuuden kannalta tunnusmerkkinä toimii esimerkiksi kovakoodatut käyttäjäoikeudet. (Rahman 2018, 479.)

Toinen tapa virheiden tunnistamiseksi on se, että etsitään toimintatavoista tunnusmerkkejä. Tunnusmerkki voi esimerkiksi olla se, että tehdään suuria muutoksia ympäristöihin kerralla ja sitä kautta virheellistä koodia pääsee ympäristöihin. Tunnusmerkin tunnistamisen perusteella voidaan suositella tehtäväksi pieniä muutoksia kerrallaan. (Rahman 2018, 476.)

Ohjelmoinnissa virheellisestä koodista käytetään termiä **Code Smells**, joka viittaa koodiin, joka voi aiheuttaa ongelmia. Koodihajut eivät välttämättä aiheuta ongelmia käynnissä olevaan ohjelmaan mutta vaativat korjausta. Esimerkiksi identtisten koodilohkojen toistaminen koodissa on laajalti tunnettu koodihaju. Identtiset koodilohkot tulisi muuttaa metodiksi. IaC-koodissa esiintyy koodihajuja samalla

tavalla kuin perinteisessä ohjelmistokoodaamisessa. Tunnistamalla erilaiset koodihajut voidaan IaC-koodin turvallisuutta lisätä. IaC-koodeista on tunnistettu ainakin kolme erilaista koodihajutyyppiä. Tunnusmerkit ovat teknologiauskolliset, teknologiariippuvaiset ja teknologiaspesifiset koodihajut. (Schwarz, Steffens & Lichter 2018, 221–222.)

Progressin mukaan **virheellinen konfiguraatio** voi lähteä leviämään, koska ympäristöjen tekeminen on nopeaa. Virheellinen konfiguraatio voi olla esimerkiksi salaamaton tietokanta. (Tolliver 2022.)

Eri ympäristöt voivat lähteä liukumaan erilleen turvallisuuskäytänteiden osalta. Turvallisuuskäytänteet voivat lähetä liukumaan erilleen, jos ympäristöihin tehdään päivityksiä. Ilmiötä kutsutaan **konfiguraation liukumiseksi.** Konfiguraation liukuminen lähtee leviämään helposti IaC-koodin mukana. (Tolliver 2022.)

Tunnistautuminen on myös tärkeä osa tietoturvallisuutta. Terraform suosittelee käyttämään kaksivaiheista tunnistautumista maksullisessa Terraform Cloud -versiossaan. Käyttäjäoikeuksien määrittelyssä tulee käyttää pienimpiä mahdollisia oikeuksia jokaiselle käyttäjälle. Maksullisessa Cloud-versiossa Terraform mahdollistaa eri käyttäjäryhmille eri API-avaimet. Versionhallintaohjelman pääsyn hallinta pitää myös olla ajan tasalla, koska IaC-koodi on siellä nähtävillä. (Hashicorp 2023m.)

5 PROJEKTIN TOTEUTUS

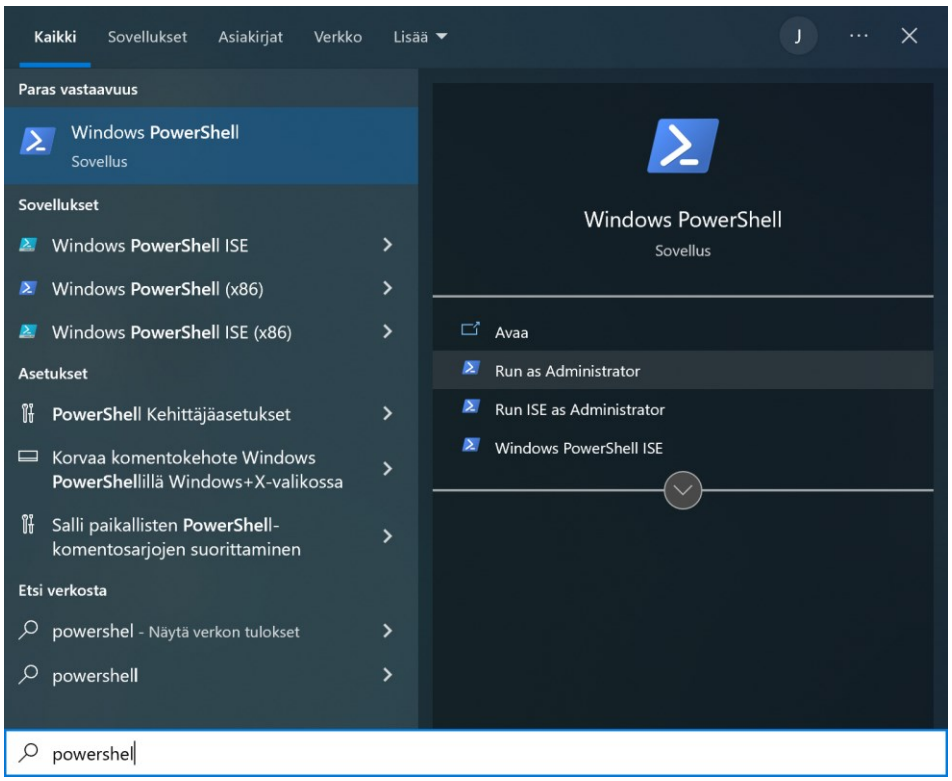
5.1 Tietojärjestelmäympäristön luonti pilvipalveluun

Pilvipalveluosio opinnäytetyön käytännön osuudesta tehtiin käyttäen Terraform-työkalua. Ensiksi kuvataan virtuaalikoneiden luonti Amazon Web Services -palveluun Hashicorpin opasmateriaalia noudattaen. Jatkossa Amazon Web Servicestä käytetään lyhennettä AWS. Toisessa vaiheessa koodia muunnetaan siten, että saadaan luotua kolme Linux-virtuaalikonetta, jotka kykenevät näkemään toisensa verkon yli. Tavoitteena on saada ICMP echo request -viesti yleisimmin ping-viestinä tunnettu sanoma välittymään koneiden välillä.

Terraformin käyttämiseen AWS-palvelussa on olemassa seuraavat vaatimukset: Hallintatyöasemalla pitää olla asennettuna AWS CLI, Terraform CLI -työkalut ja lisäksi pitää olla AWS-tili (Hashicorp 2023a). Tehtävän suorittaminen aloitettiin luomalla tili AWS-palveluun. Opinnäytetyötä varten luotiin ilmainen kokeilukäyttötili, joka on käytössä tietyin rajoituksin vuoden ajan. Opinnäytetyössä ei kuvata AWS-tilin luontia.

5.1.1 Chocolatey-asennus

AWS-resurssien luonti aloitettiin Terraform CLI -työkalun asentamisella Windows 10 pro -käyttöjärjestelmään paikalliselle hallintakoneelle. Asennus aloitetiin asentamalla paketinhallintaohjelmisto Chocolatey. Chocolatey-asennus tehtiin PowerShell-ohjelmalla. PowerShell piti käynnistää järjestelmänvalvojan oikeuksilla (Chocolatey 2023a). Kuviossa 11 nähdään PowerShell-käynnistys.

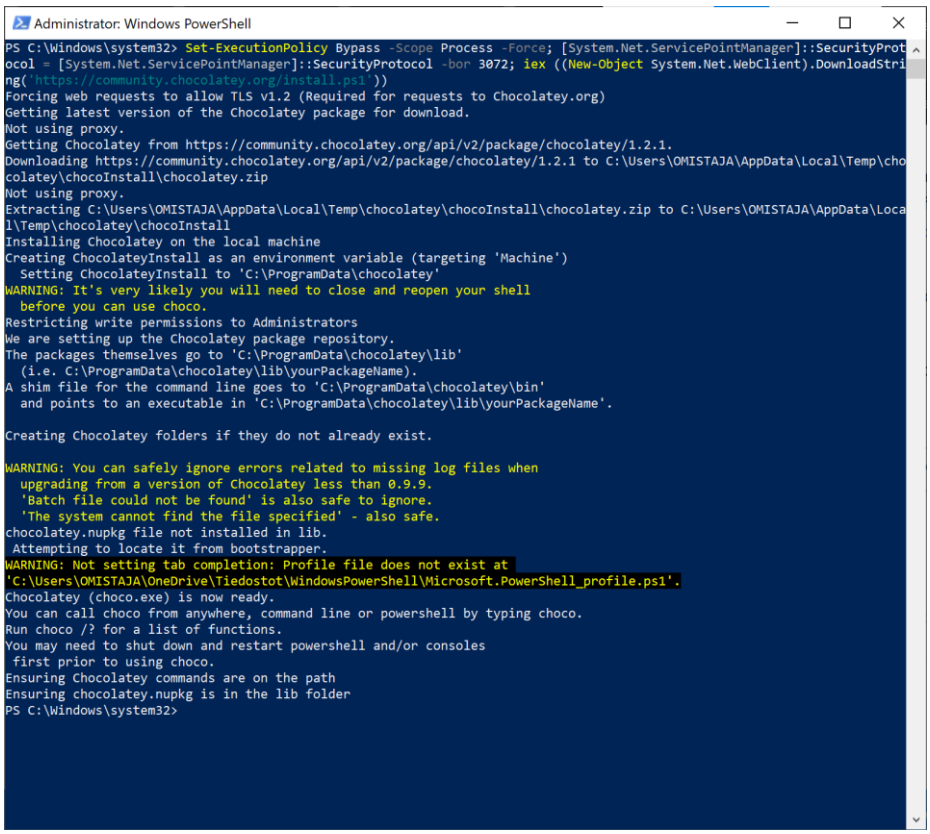


Kuvio 11. PowerShell-käynnistys korotetuilla oikeuksilla

Chocolatey asennettiin komennolla:

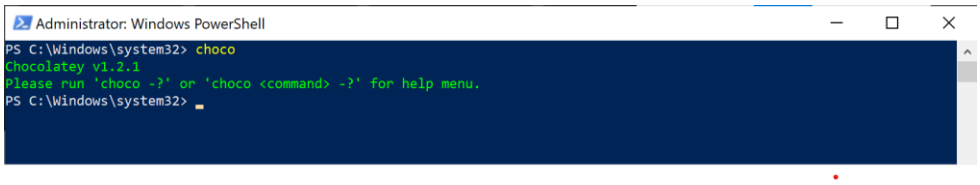
> *Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient)*.DownloadString('https://community.chocolatey.org/install.ps1'))

Komento lataa viimeisimmän Chocolateyn tarvitsemat tiedostot ja luo tarvittavat ympäristömuuttujat (Chocolatey 2023a). Kuviossa 12 näkyy onnistunut asennus. Asennus kehotti käynnistämään PowerShellin uudelleen ennen Chocolatey-paketinhallinnan käyttämistä.



Kuvio 12. Chocolatey-asennus

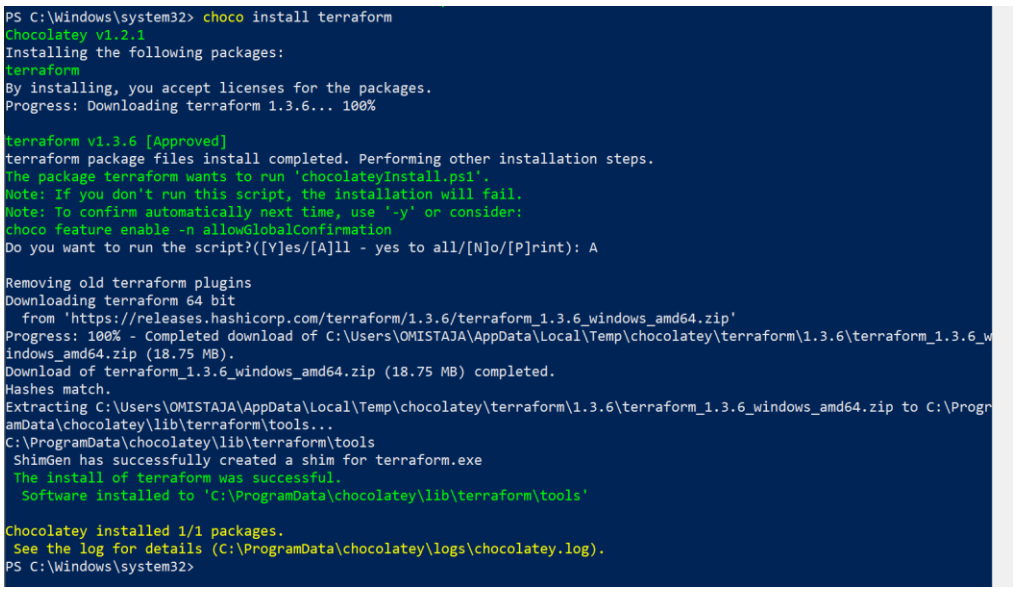
Komennolla *choco* voidaan varmistaa Chocolateyn onnistunut asennus (Chocolatey 2023a). Kuviossa 13 näkyy vastaus komentoon *choco*.



Kuvio 13. Choco-komento

5.1.2 Terraform-asennus

Seuraavaksi jatkettiin varsinaiseen Terraform-asennukseen. Asennus tapahtui käyttämällä edellä asennettua Chocolatey-paketinhallintaa. Asennus aloitettiin antamalla komento *choco install terraform* (Hashicorp 2023a). Asennus pyysi seuraavaksi lupaa käyttää asennusskriptiä. Asennukselle annettiin lupa vastaamalla Y. Kuviossa 14 näkyy onnistunut asennus.



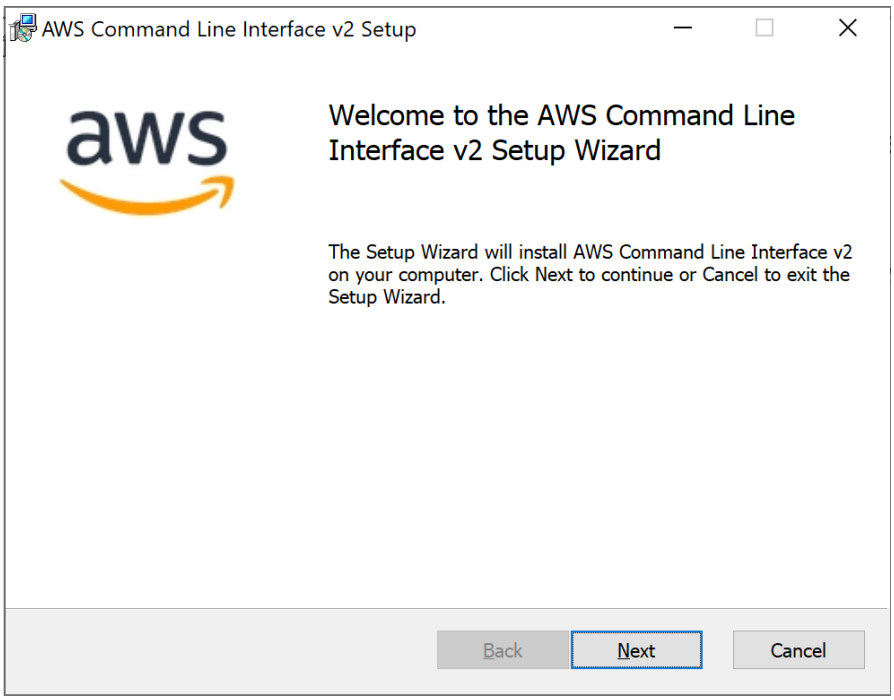
Kuvio 14. Terraform-asennus

Terraform-asennus voidaan varmentaa esimerkiksi komennolla *terraform --version*. Kuviossa 15 näkyy tulostus versiokyselyyn.

Kuvio 15. Terraform-versiokysely

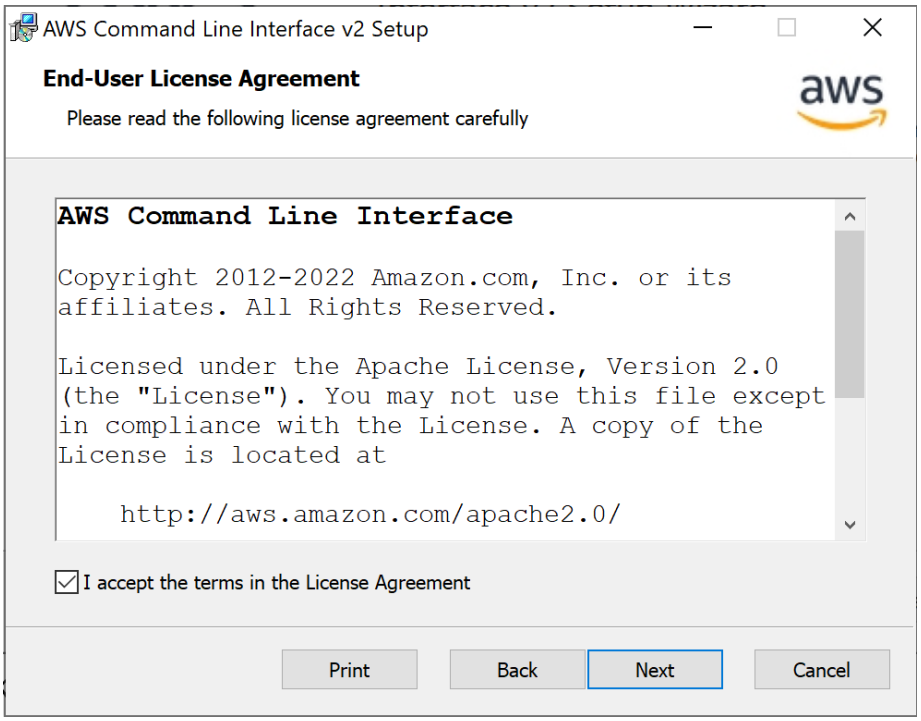
5.1.3 AWS CLI -asennus ja määrittely

Terraform CLI -työkalun asentamisen jälkeen aloitetaan AWS CLI-asennus Windows-koneelle. Asennus aloitettiin lataamalla AWS CLI -asennuspaketti osoitteesta https://awscli.amazonaws.com/AWSCLIV2.msi (Amazon Web Services 2023g). Latauksen jälkeen klikattiin asennuspakettia, jolloin asennusvelho käynnistyi. Kuviossa 16 näkyy asennusvelhon aloitusnäkymä.



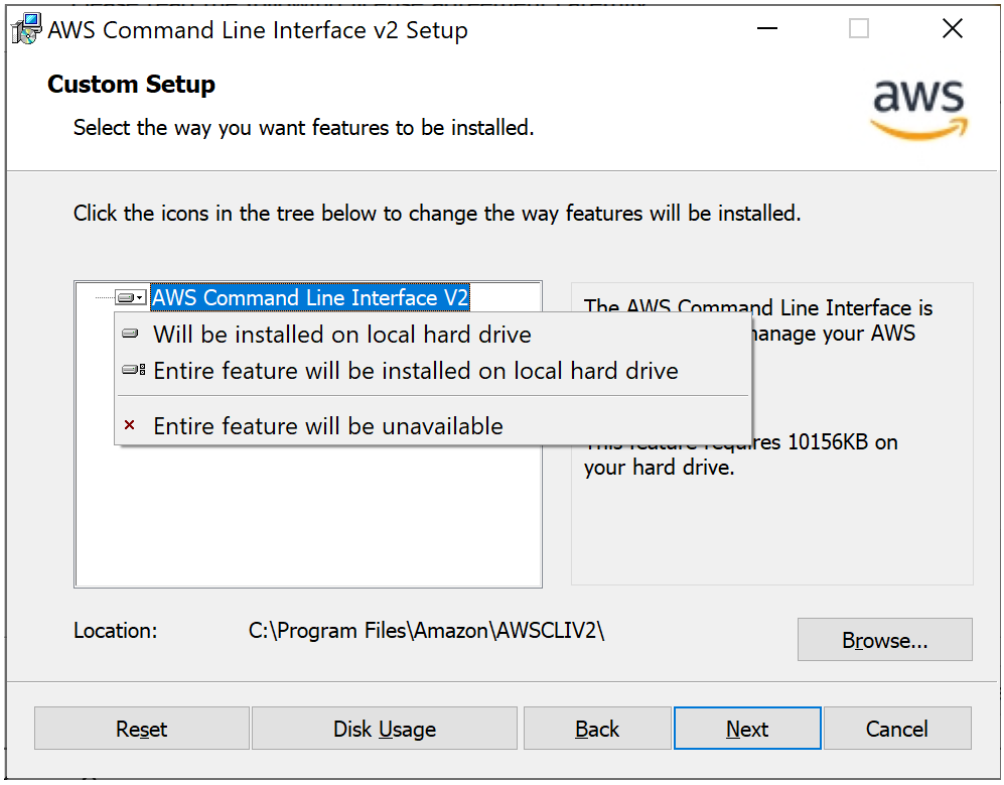
Kuvio 16. AWS CLI -asennusvelho

Asennus eteni valitsemalla Next. Seuraavaksi piti hyväksyä loppukäyttäjän ehdot. Kuviossa 17 näkyvät loppukäyttäjän ehdot. Asennus eteni, kun laitettiin väkänen ehtojen hyväksymisen merkiksi ja valittiin Next.



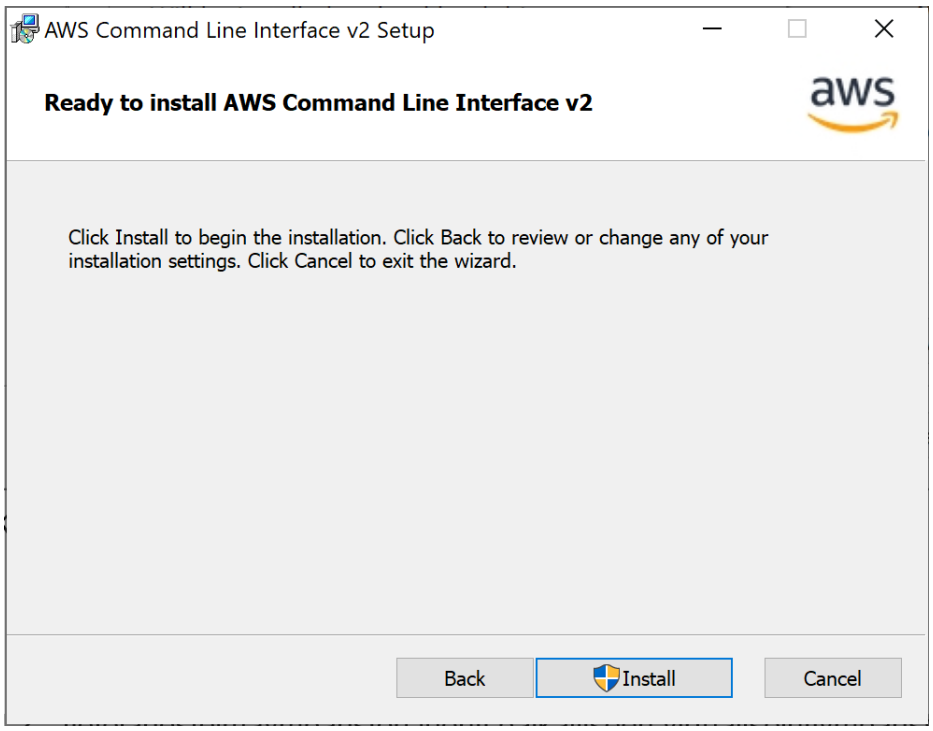
Kuvio 17. Loppukäyttäjän ehdot

Seuraavaksi valittiin asennustapa. Valittiin oletuksena oleva vaihtoehto, jossa asennettiin CLI paikalliselle kovalevylle. Kuviossa 18 nähdään, että tässä vaiheessa voidaan halutessaan valita myös joku muu asennuskansio. Asennusta jatkettiin oletusasetuksilla valitsemalla Next.



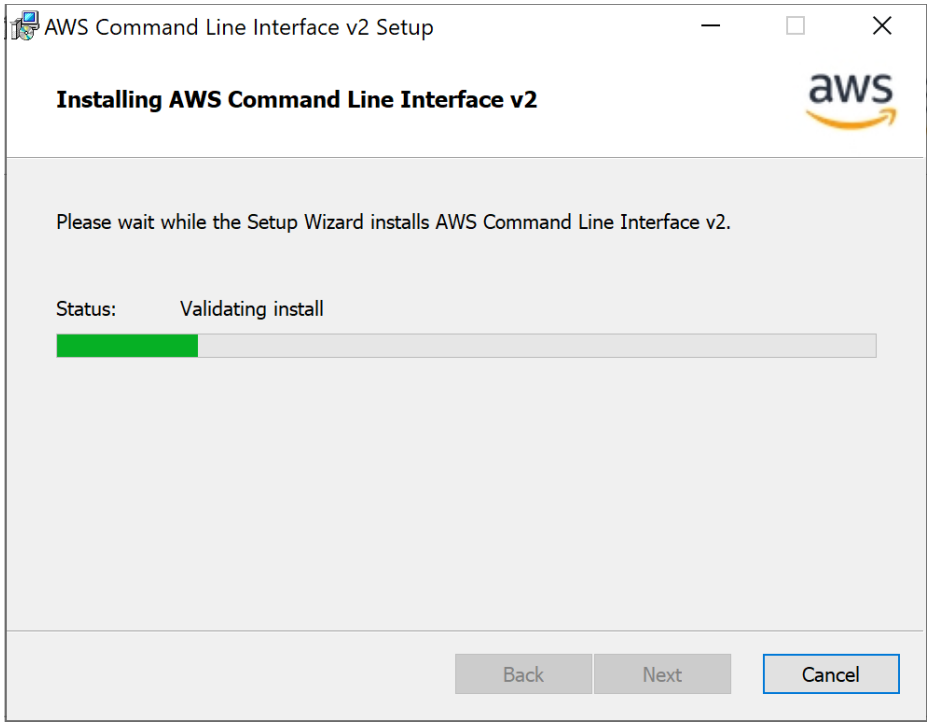
Kuvio 18. Asennustavan valinta

Valitsemalla Install aloitettiin varsinainen asennus. Kuviossa 19 nähdään, että vielä tässä vaiheessa voidaan peruuttaa asennus. Asennusta jatkettiin valitsemalla Install.



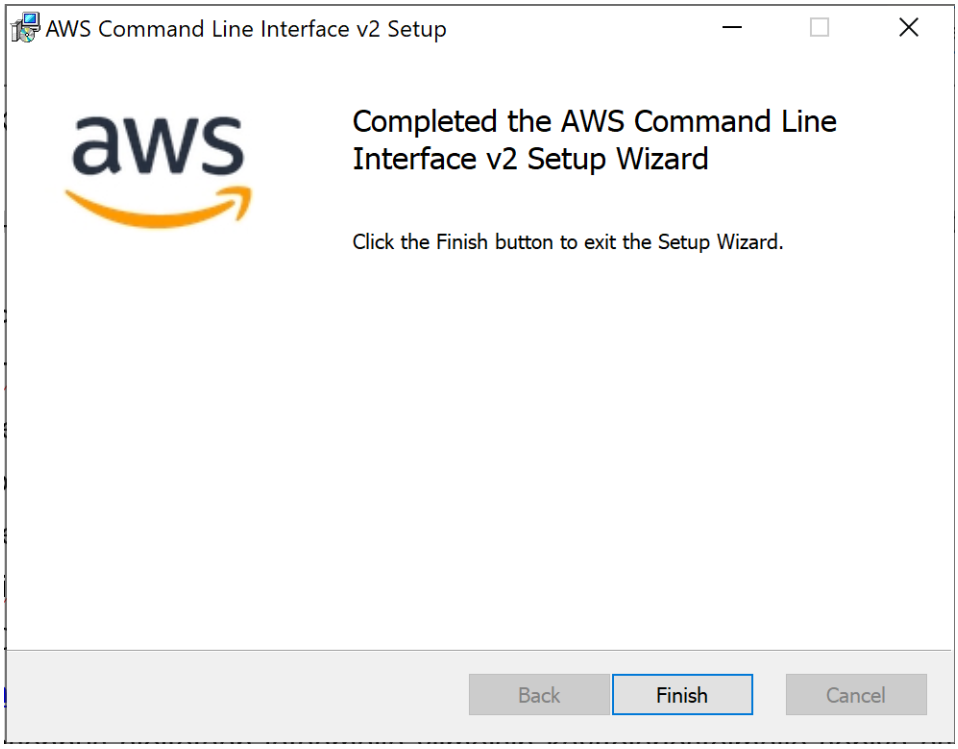
Kuvio 19. Asennuksen aloitus

Kuviossa 20 nähdään asennuksen eteneminen. Asennus kesti muutaman minuutin.



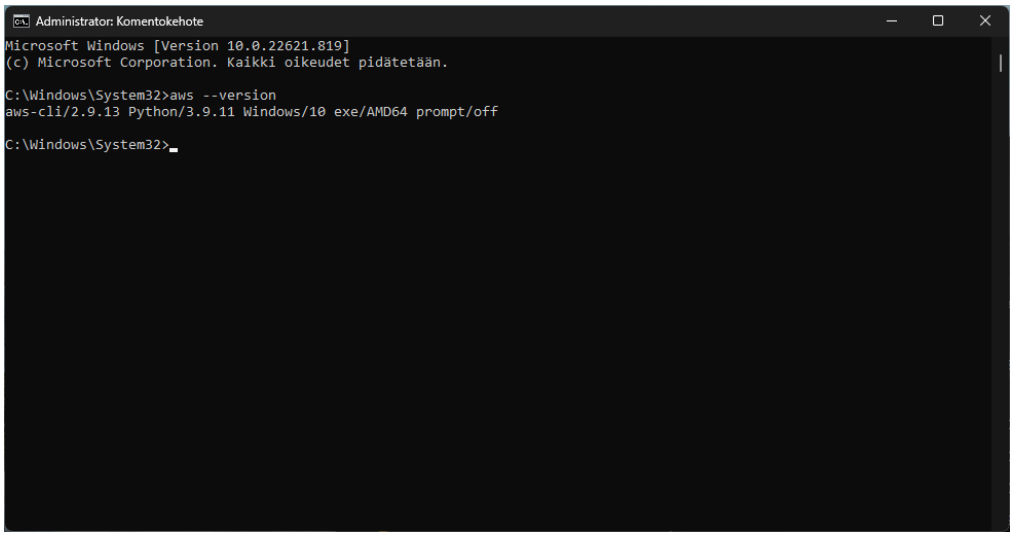
Kuvio 20. Asennuksen eteneminen

Kuviossa 21 nähdään asennuksen päättyminen. Asennus viimeisteltiin valitsemalla Finish.



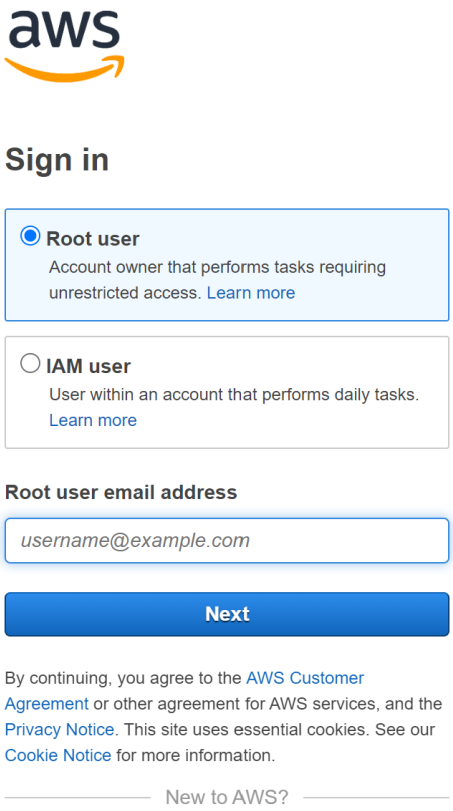
Kuvio 21. Asennuksen päättyminen

AWS CLI -työkalun asennuksen onnistuminen varmistettiin käynnistämällä komentokehote uudelleen ja antamalla komento *aws - -version* (Amazon Web Services 2023g). Kuviossa 22 näkyy vastaus onnistuneeseen asennukseen.



Kuvio 22. AWS CLI -asennuksen varmistaminen

Seuraavaksi käyttöönotossa luotiin AWS-käyttöoikeudet AWS-provideriin. Käyttöoikeuksien luomisessa tarvittiin AWS access key ja secret key. Avainpari luotiin AWS:n selainkäyttöliittymällä, johon pääsee osoitteella *https://signin.aws.amazon.com*. Seuraavaksi kirjauduttiin selaimella AWS-käyttöliittymään käyttäen Root-käyttäjätilin käyttäjätunnusta (Amazon Web Services 2023h). Kuviossa 23 nähdään AWS-kirjautumisikkuna. Valittiin Next käyttäjätunnuksen syötön jälkeen.

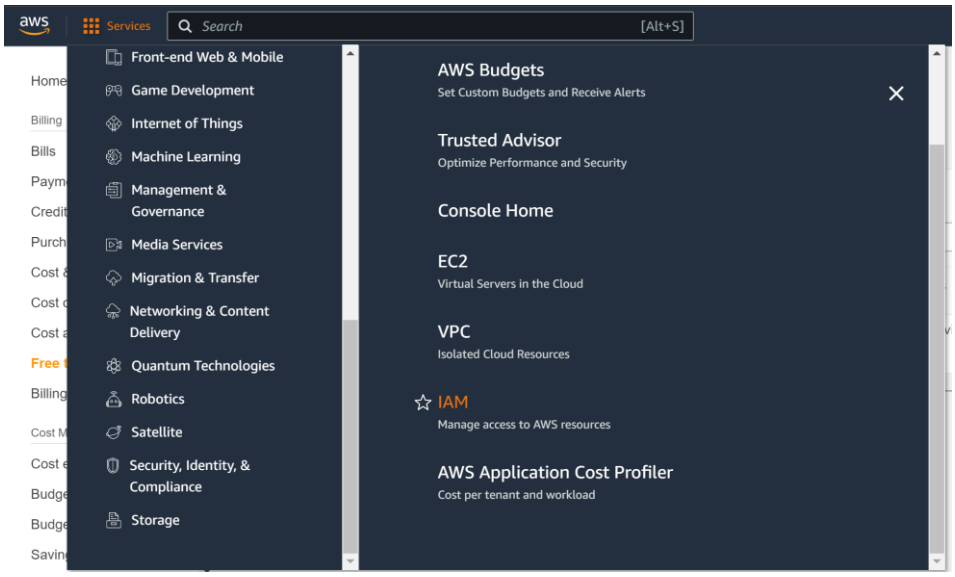


Kuvio 23. Root-käyttäjätunnuksella kirjautuminen

Seuraavaksi kysyttiin Root-käyttäjän salasana. Kuviossa 24 näkyy salasanakysely.

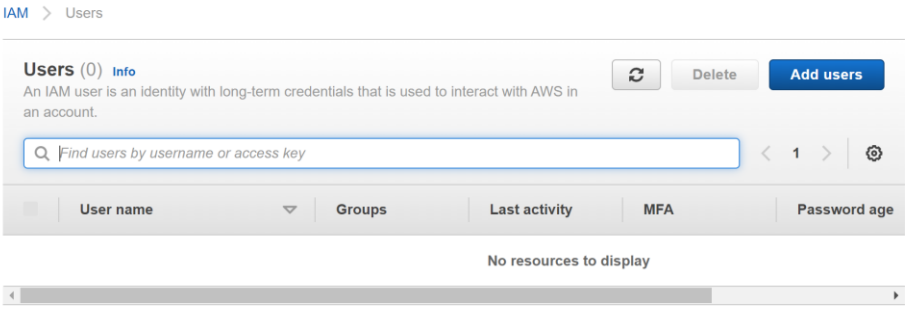
Kuvio 24. Root-käyttäjän salasanakysely

Avainparin luominen aloitettiin seuraavasti: Kuvion 25 näkymässä valittiin yläpalkissa olevasta Services-valikosta IAM manage access to AWS resources.



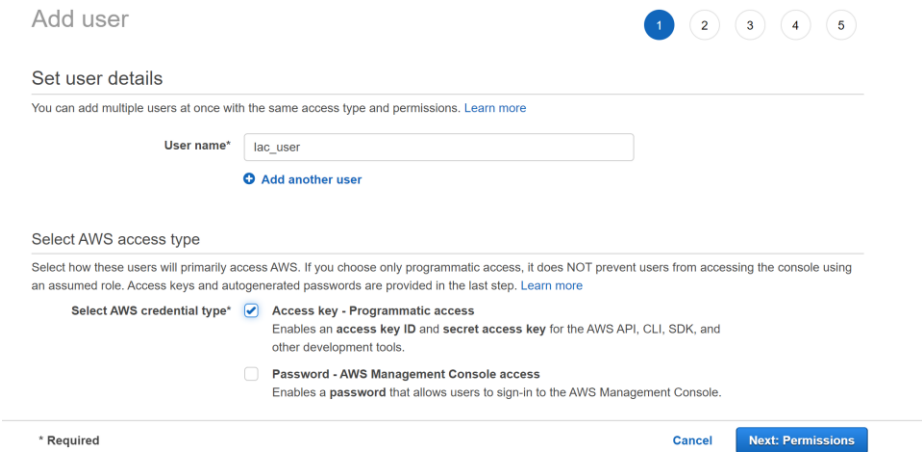
Kuvio 25. Services-valikkossa IAM

Luotiin uusi AWS-käyttäjä, jolla ei ole Root-oikeuksia. Kuvion 26 näkymässä valittiin Add users (Amazon Web Services 2023i).



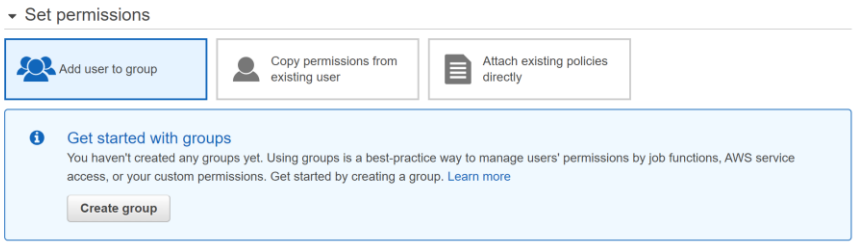
Kuvio 26. AWS-käyttäjän luonti

Käyttäjänluontivelho jatkoi määrittelyä. Kuvion 27 näkymässä annetaan käyttäjälle nimi ja valitaan, että käyttäjä kirjautuu käyttäen access keytä (Amazon Web Services 2023i). Tässä tapauksessa käyttäjätunnukseksi annettiin IaC_user. Määrittelyä jatkettiin valitsemalla Next: Permissions.



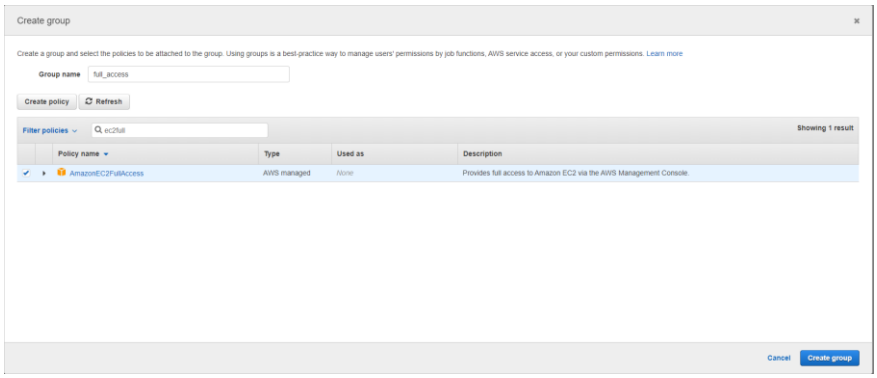
Kuvio 27. Käyttäjätunnuksen ja kirjautumistavan määrittely

Seuraavaksi määriteltiin käyttäjäryhmä, johon edellä luotu käyttäjätunnus liitettiin (Amazon Web Services 2023i). Kuvion 28 näkymässä valitaan Create group.



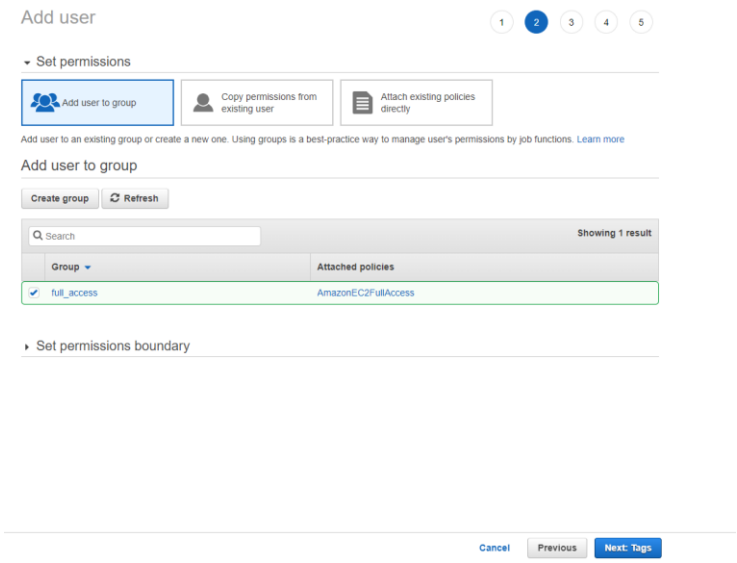
Kuvio 28. Ryhmän luominen

Käyttäjäryhmälle annettiin tässä tapauksessa nimi full_access ja käyttäjäryhmälle lisättiin täydet oikeudet EC2-ympäristöihin antamalla ryhmälle käytännöksi AmazonEC2FullAccess. Opinnäytetyössä ei ole tarkoitus pureutua AWS:n käyttämiseen ja siitä syystä käyttäjälle annetaan kaikki oikeudet EC2-ympäristöihin. Tuotantoympäristöissä käyttäjäoikeuksien antamisessa tulee käyttää tarkkaa harkintaa siitä, mitkä ovat riittävät oikeudet kullekin toimijalle. Käyttäjäryhmän luonti viimeisteltiin valitsemalla Create group kuvion 29 näkymässä. Kuviossa 29 nähdään myös käyttäjäryhmän oikeuksien määrittely.



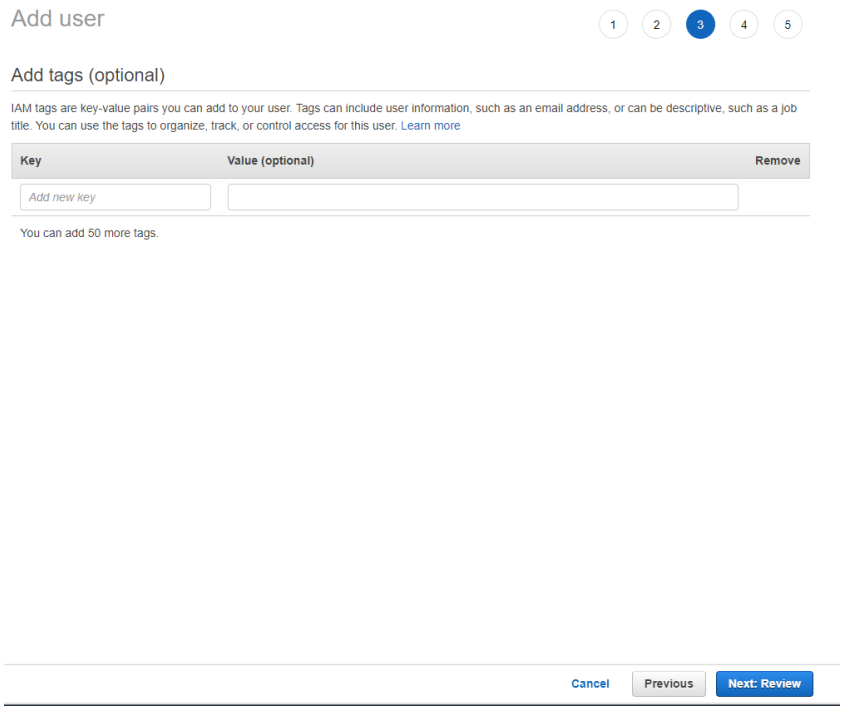
Kuvio 29. Käyttäjäryhmän luonti

Käyttäjä liitettiin edellä luotuun ryhmään valitsemalla full_access kuvion 30 näkymässä. Käyttäjän luominen jatkui valitsemalla näkymästä Next: Tags.



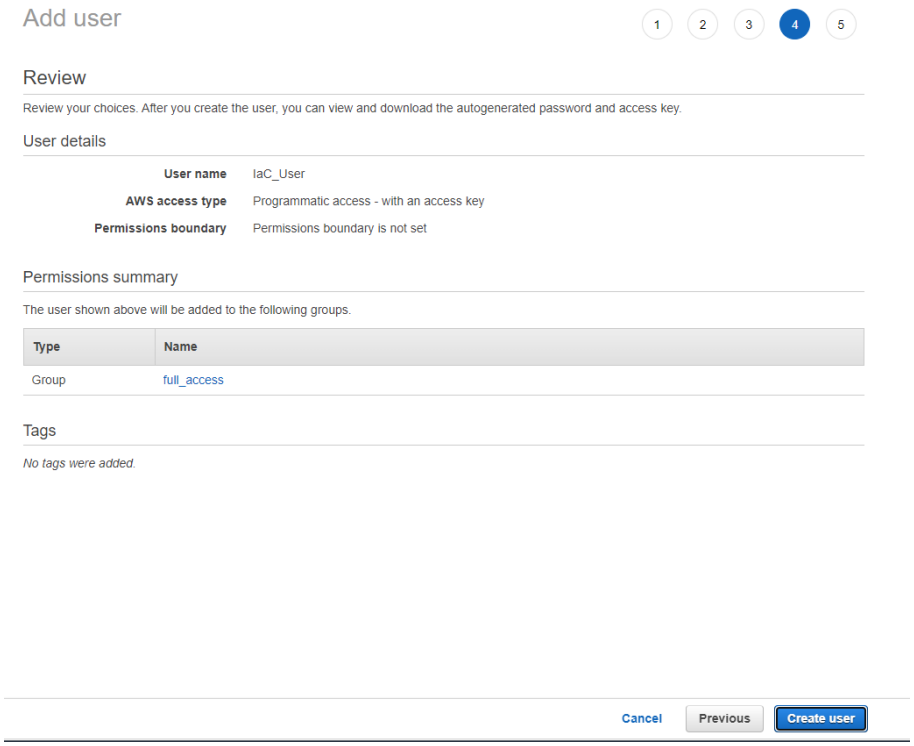
Kuvio 30. Käyttäjäryhmä liitettynä käyttäjään

Kuviossa 31 voidaan lisätä käyttäjälle valinnaisia tietoja, kuten sähköpostiosoite (Amazon Web Services 2023i). Tässä tapauksessa ei annettu käyttäjälle lisätietoja, vaan jatkettiin asennusta valitsemalla Next:Review.



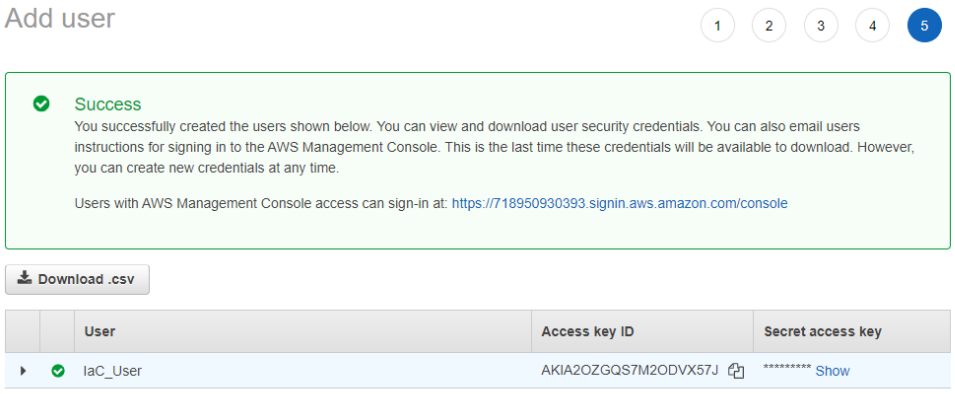
Kuvio 31. Valinnaisten lisätietojen antaminen

Kuviossa 32 nähdään yhteenveto käyttäjän tiedoista. Valittiin kuvion näkymästä Create user.



Kuvio 32. Käyttäjän IaC_User tiedot

Kuviossa 33 nähdään onnistunut käyttäjän luonti. Näkymä esittää myös AWS CLI -työkalun käyttöön tarvittavat Access key ID:n ja Secret access key -avainparin. Avainpari otettiin tässä vaiheessa talteen, koska sitä ei saa myöhemmässä vaiheessa näkyviin (Amazon Web Services 2023i). Luontevinta oli ladata näkymässä esitetty Download .csv -tiedosto talteen, josta avaimet on myöhemmässä vaiheessa helppo kopioida AWS-määrittelytiedostoon.



Kuvio 33. Onnistunut AWS-käyttäjän luonti

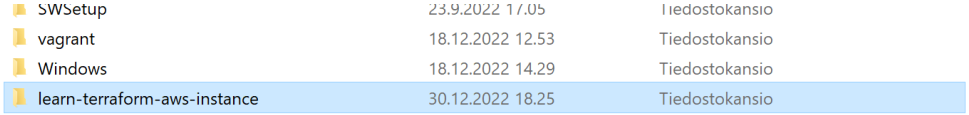
Seuraavaksi määriteltiin edellä luotu avainpari käyttöön kirjoittamalla komentokehotteeseen komento *aws configure*. *Aws configure*-komennon jälkeen piti syöttää kuviossa 34 pyydetyt tiedot. AWS Access key ID ja AWS Secret Access Key löytyvät edellä tallennetusta .csv-tiedostosta. Kuviossa 34 avainpari on tietoturvasyistä piilotettu. Default region määriteltiin tässä tapauksessa käyttämään us-west-2 -aluetta. Default region -alueita on rajatusti käytössä ilmaisen AWS-tilin kautta. Default region -määrityksen täytyy olla sama kuin myöhemmin luotavan Terraform main.tf -tiedoston määrittely. Default output format jätetään tässä tapauksessa määrittelemättä. AWS käyttää oletuksena JSON-muotoa, mikäli ei erikseen määritellä muuta (Amazon Web Services 2023j).

Kuvio 34. Aws configure -komento

Asennuksen päätyttyä voidaan asennuksen onnistuminen varmistaa antamalla *aws - -version* -komento komentokehotteessa (Amazon Web Services 2023g). Kuviossa 35 nähdään onnistuneen asennuksen jälkeinen vastaus.

Kuvio 35. AWS CLI -version kysely

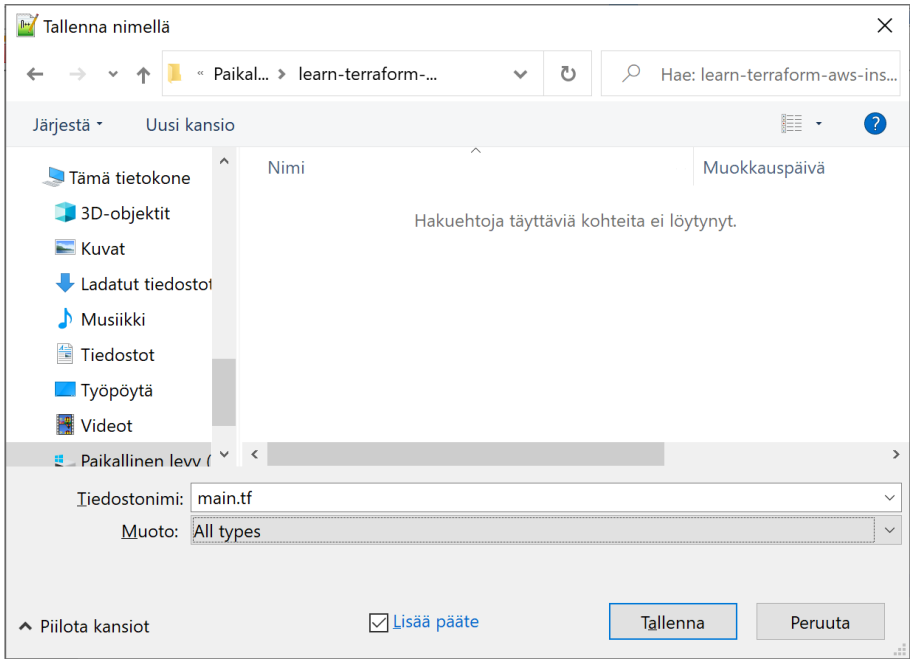
Seuraavaksi jatkettiin Terraformin käyttöönottoa luomalla kansio, jonne tehtiin Terraformin vaatima konfiguraatiotiedosto. Kuviossa 36 näkyy C-aseman juureen luotu kansio nimeltään learn-terraform-aws-instance.



Kuvio 36. Terraform-tiedostojen kansio

Seuraavaksi tehtiin edellä luotuun kansioon Terraform-työkalun määrittelytiedosto nimeltään main.tf. Konfiguraatiotiedoston voi luoda ja muokata käyttäen

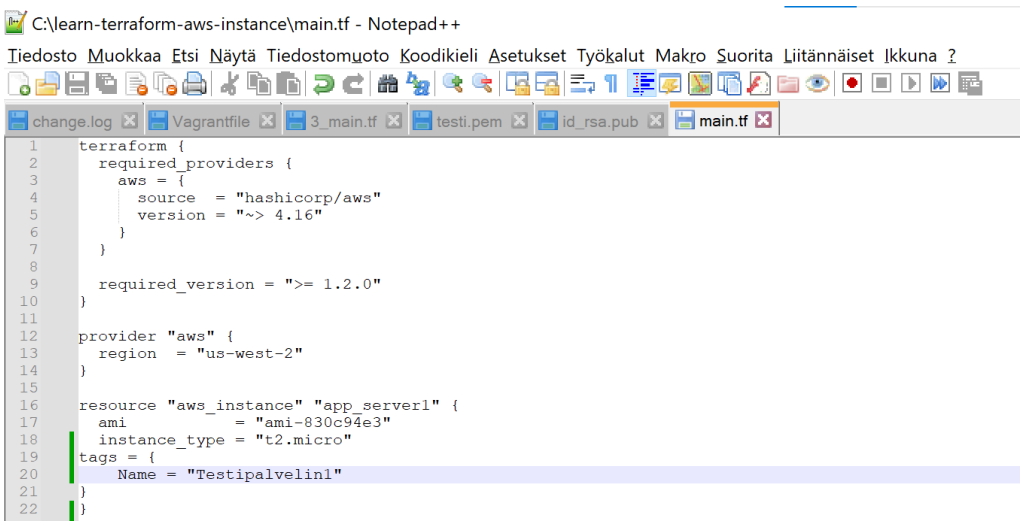
esimerkiksi Visual Studio Code -koodausympäristöä. Kuviossa 37 tiedosto luodaan käyttäen Notepad++ -ohjelmaa. Notepad++ on kevyempi vaihtoehto tiedoston muokkaamiseen. Tiedoston muodoksi valittiin tässä All types, jolloin tiedostoon ei lisätä ylimääräistä tiedostopäätettä. Poistamalla Lisää pääte -valinta tiedoston muodoksi tulee myös All types.



Kuvio 37. Main.tf-tiedoston luominen

5.1.4 Main.tf-määrittely

Seuraavaksi luotiin main.tf-tiedostoon varsinainen sisältö (Hashicorp 2023a). Kuviossa 38 nähdään Terraform AWS -tutoriaalin peruskonfiguraatio, jolla luodaan yksi palvelin AWS-ympäristöön (Hashicorp 2023n).



Kuvio 38. Main.tf-tiedoston sisältö

Main.tf-tiedoston valmistelujen jälkeen voitiin siirtyä ympäristön luomiseen Terraform CLI -työkalulla. Seuraavaksi käynnistettiin komentokehote ja siirryttiin hakemistoon, jossa Main.tf-tiedosto sijaitsee. Tässä tapauksessa tiedosto sijaitsee c:\terraform -hakemistossa. Tiedoston sijainti näkyy kuviossa 39.

```
C:\Users\OMISTAJA>cd \

C:\>cd learn-terraform-aws-instance

C:\learn-terraform-aws-instance>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\learn-terraform-aws-instance>dir
 Volume in drive C has no label.
 Volume Serial Number is 4E54-45CC

 Directory of C:\learn-terraform-aws-instance

15.01.2023  17.23    <DIR>          .
15.01.2023  17.23    <DIR>          ..
30.12.2022  21.40    <DIR>          .terraform
30.12.2022  21.40             1 407 .terraform.lock.hcl
15.01.2023  16.46             1 229 main.tf
15.01.2023  17.23               181 terraform.tfstate
15.01.2023  17.20            14 198 terraform.tfstate.backup
               4 File(s)         17 015 bytes
               3 Dir(s)  121 446 752 256 bytes free

C:\learn-terraform-aws-instance>
```

Kuvio 39. Main.tf-tiedoston sijainti

Terraform main.tf -tiedoston käyttö valmisteltiin komennolla *terraform init* (Hashicorp 2023n). Kuviossa 40 näkyy onnistunut valmistelu.

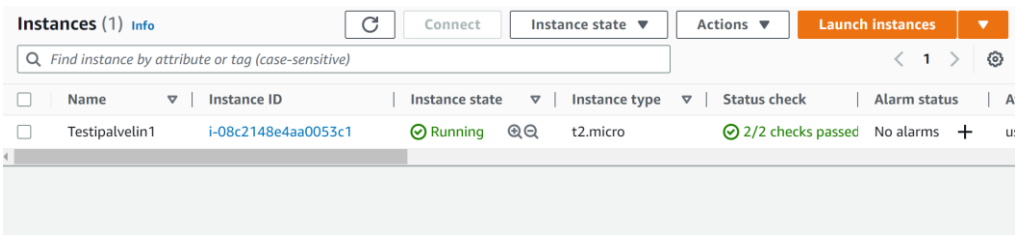
```
C:\learn-terraform-aws-instance>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.48.0

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.

C:\learn-terraform-aws-instance>
```

Kuvio 40. Main.tf-tiedoston valmistelu

Komennolla *terraform plan* nähtiin main.tf valmistelun tulos. Komento *terraform plan* tulostaa luotavan ympäristön tiedot. (Hashicorp 2023n.) Liitteessä 1 nähdään *terraform plan* -komennon antaminen ja komentoon liittyvä tulostus kokonaan. Tulostus sisältää kaikki määrittelyt, jotka on tehty tai jätetty oletusasetuksiin.

Komennolla *terraform apply* aloitettiin ympäristön luonti AWS-palveluun (Hashicorp 2023n). Työkalu pyysi varmistamaan ympäristön luomisen. Ympäristön luominen varmistettiin syöttämällä sana yes. Liitteessä 2 on nähtävillä koko *terraform apply* -komentoon liittyvä tulostus ja varmistuksen kysely.

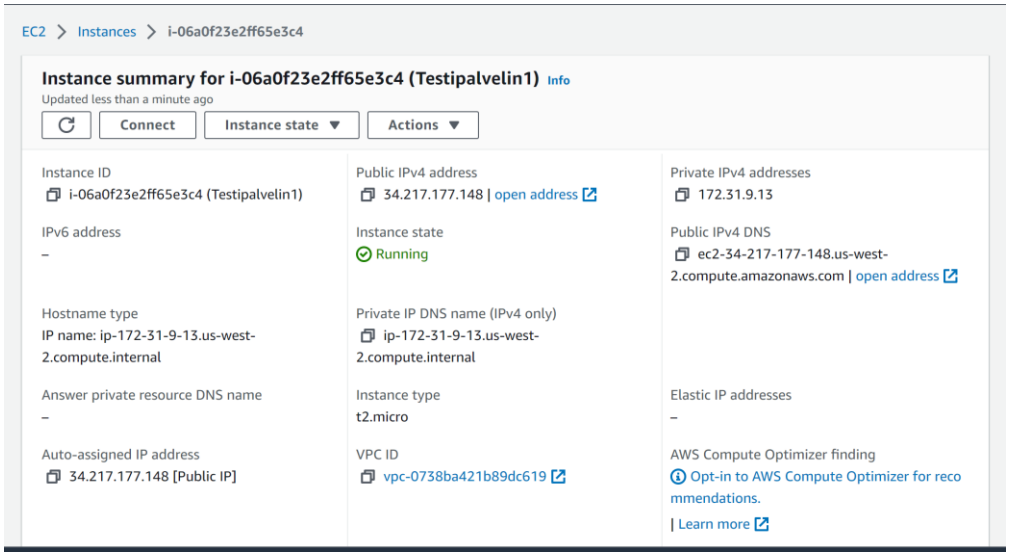
AWS-hallintapaneelista nähtiin, että uusi instanssi oli luotu (Amazon Web Services 2023f). Kuviossa 41 nähdään instances-valinnasta, että Testipalvelin1 luotiin ja se oli käynnissä.



Kuvio 41. Testipalvelin1-instanssi käynnissä

Klikkaamalla Instance ID:tä pääsee tarkastelemaan instanssin tietoja tarkemmin (Amazon Web Services 2023f). Muun muassa julkinen osoite on nähtävillä, johon

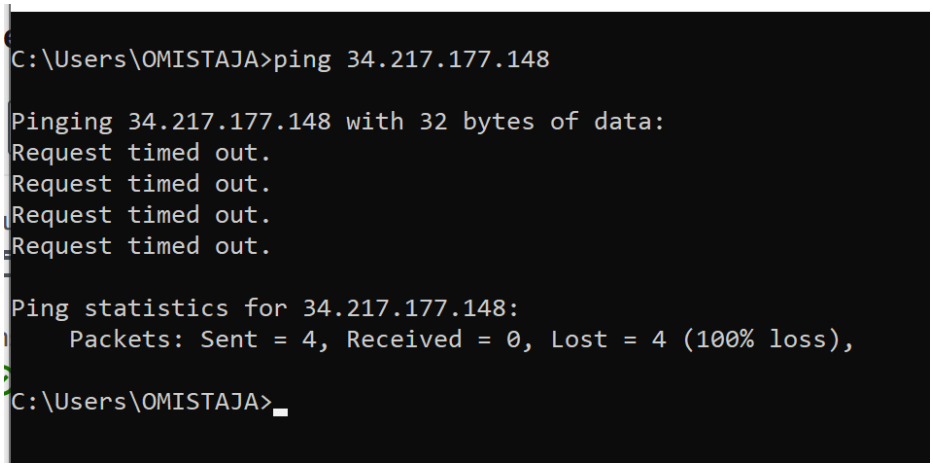
voi kokeilla SSH-yhteyttä. Kuviossa 42 nähdään tiedot aiemmin luodusta instanssista.



Kuvio 42. Instanssin Testipalvelin1 tiedot

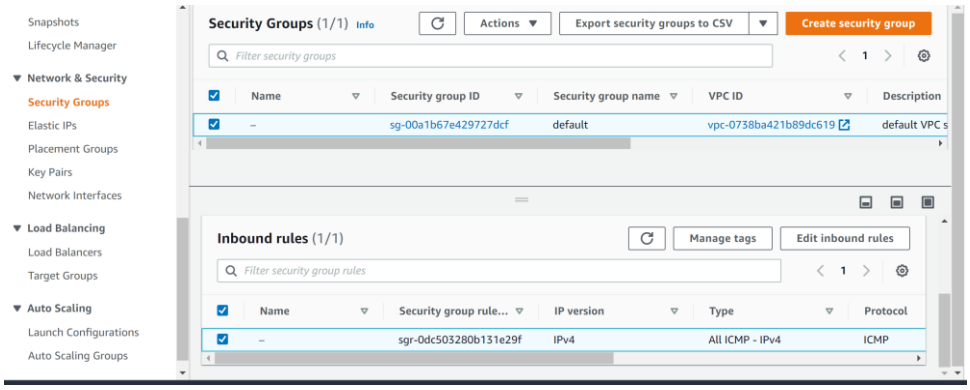
5.1.5 Koneiden välisen yhteyden testaus

Opinnäytetyön tavoitteissa määriteltiin, että luotujen kolmen virtuaalikoneen välillä piti kyetä lähettämään onnistunut ping-kysely. Tässä vaiheessa kokeiltiin, kyetäänkö ping-viesti lähettämään paikalliselta koneelta luodulle virtuaalikoneelle. Ping-viestiin ei vastata, kuten kuviossa 43 nähdään.



Kuvio 43. ping-kysely komentokehotteessa

Virtuaalikoneen security-asetuksiin piti käydä määrittelemässä, että ICMP-viestit on sallittu Inbound-säännöissä (Amazon Web Services 2023l). Kuviossa 44 näkyy security group, joka on määritelty luodulle virtuaalikoneelle ja security group:iin määritelty inbound-sääntö, joka sallii kaiken ICMP-liikenteen kaikista IPv4-osoitteista. (Amazon Web Services 2023a.) Tuotantoympäristöissä kaiken ICMP liikenteen salliva sääntö ei tule kysymykseen, koska se mahdollistaa palvelunestohyökkäyksen konetta kohtaan (Viestintävirasto 2016, 2). Opinnäytetyön tarkoitusta edellä luotu sääntö kuitenkin palvelee, koska tarkoituksena oli testata yhteyksien toimivuus ja poistaa ympäristöt sen jälkeen.



Kuvio 44. Security Group ja Inbound -määrittelyt

Määrittelyn jälkeen ping-viestit menivät läpi ja saivat vastaukset. Kuviossa 45 nähdään vastaukset ping-kyselyihin.

```
C:\Users\OMISTAJA>ping 34.217.177.148

Pinging 34.217.177.148 with 32 bytes of data:
Reply from 34.217.177.148: bytes=32 time=194ms TTL=27
Reply from 34.217.177.148: bytes=32 time=205ms TTL=27
Reply from 34.217.177.148: bytes=32 time=195ms TTL=27
Reply from 34.217.177.148: bytes=32 time=197ms TTL=27

Ping statistics for 34.217.177.148:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 194ms, Maximum = 205ms, Average = 197ms

C:\Users\OMISTAJA>_
```

Kuvio 45. Ping-kyselyiden vastaukset

AWS ei hyväksy ulkoverkon kautta SSH-yhteyksiä, joissa käytetään tunnistautumismenetelmänä käyttäjätunnusta ja salasanaa, vaan on käytettävä julkinen ja yksityinen avain-avainparia (Amazon Web Services 2023m). Yhteyksien testauksen takia luotiin paikallisella koneella RSA-avainpari. Avainpari luotiin komentokehotteessa komennolla *ssh-keygen -t rsa -b 2048* (SSH 2023). Kuviossa 46 nähdään onnistunut avainparin luonti ja tallennuspaikka.

```
C:\learn-terraform-aws-instance>ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\OMISTAJA/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\OMISTAJA/.ssh/id_rsa.
Your public key has been saved in C:\Users\OMISTAJA/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:+TGXM51MBSnmnzFOJcYEU0uHnKIDq9S3P5nCprMFFJI omistaja@DESKTOP-NNMBUCD
The key's randomart image is:
+---[RSA 2048]----+
|      ...   o*==o|
|      E...  +oO+.|
|       ..o + +oo |
|      ..o.+ .=+. |
|     . .S.oo=+++ |
|      .  o.+ o+  |
|         .o. o   |
|        ..+ =    |
|        o= . .   |
+----[SHA256]-----+
```

Kuvio 46. RSA-avainparin luonti

Kun avain on luotu, voidaan julkinen avain liittää main.tf-tiedostoon omaksi resurssikseen (Hashicorp 2023o). Kuviossa 47 näkyy resurssi määriteltynä.

```
resource "aws_key_pair" "deployer"{
key_name ="id_rsa"
public_key="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZ07qefENl7agEI17lnNI2zwONNsM3iU8I9kpZHpXD
}
```

Kuvio 47. Julkisen avaimen liittäminen konfiguraatioon

Edellä luotuun avainpari-resurssiin viitataan luotavan koneen yhteydessä avaimen nimellä (Hashicorp 2023o). Kuviossa 48 nähdään viittaus avaimeen.

```
resource "aws_instance" "app_server1" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
  key_name= "id_rsa"

  tags = {
    Name = "Testipalvelin1"
  }
  }
```

Kuvio 48. Viittaus julkiseen avaimeen

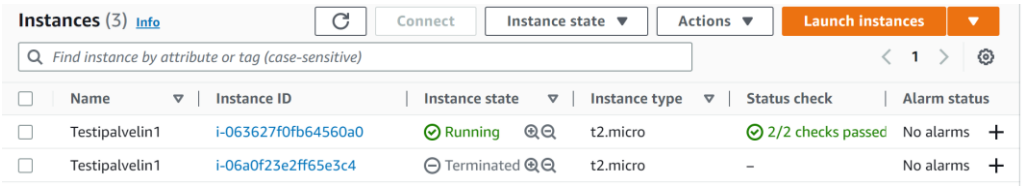
Kuviossa 49 nähdään muokattu main.tf-tiedosto. Kuviossa 49 on myös nähtävissä RSA-avaimen määrittely kokonaisuudessaan.

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

  required_version = ">= 1.2.0"
}

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

resource "aws_instance" "app_server1" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
  key_name= "id_rsa"
tags = {
    Name = "Testipalvelin1"
}
}
resource "aws_key_pair" "deployer"{
key_name ="id_rsa"
public_key="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZ07qefEN17agEI17lnNI2
}
```

Kuvio 49. Muokattu main.tf-tiedosto julkisen avaimen määrittelyllä

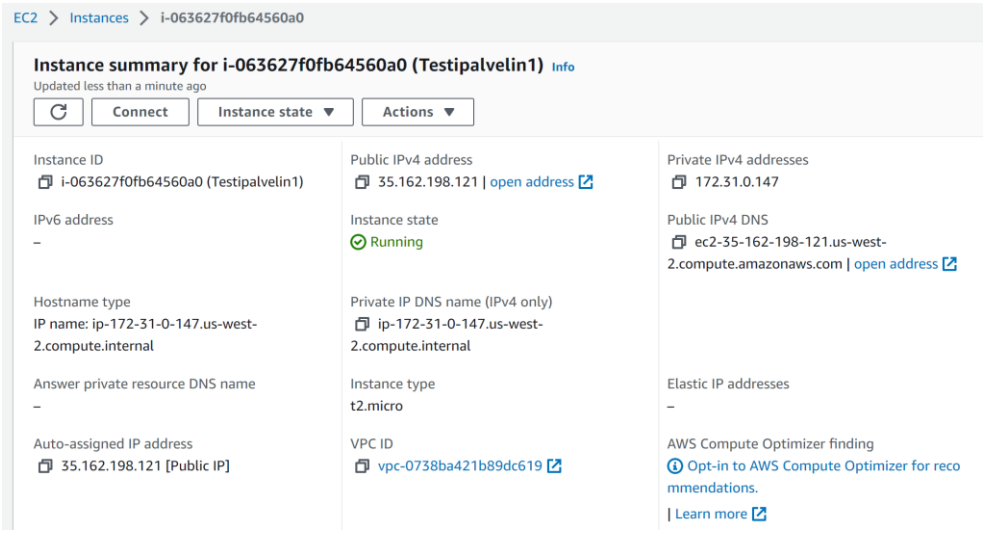
Main.tf-tiedoston RSA-avainpari muutoksen jälkeen piti *terraform apply* -komento ajaa uudelleen. Siten saatiin uusi konfiguraatio käyttöön. Liitteessä 3 nähdään komento *terraform apply* uudelleen ajettuna ja komennon aiheuttama tulostus kokonaisuudessaan.

terraform apply -käskyn uudelleen ajaminen poisti edellä tehdyn virtuaalikoneen ja loi uuden, jossa oli käytössä julkinen avain, kuten kuviosta 50 nähdään. Instances-näkymän Instance state -kohdasta nähdään, että toinen kone on Terminated, mikä viittaa virtuaalikoneen poistamiseen (Amazon Web Services 2023f).



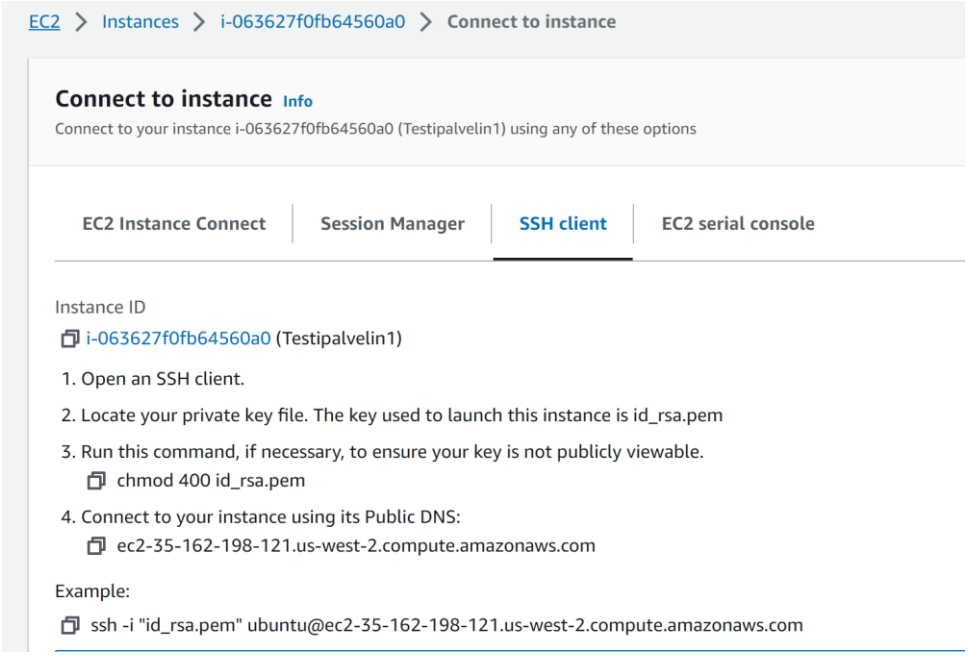
Kuvio 50. Virtuaalikoneen muokkaus

Yhteys saadaan muodostettua paikalliselta koneelta AWS-palvelun koneisiin, jos paikallisella koneella on yksityinen avain tiedossa. Yhteyksien testaus tehtiin tässä tapauksessa Putty-yhteysohjelmistolla. Yhteyden muodostaminen AWS-instanssiin vaatii yksityisen avaimen määrittelemistä yhteysasetuksiin Puttyssä (Amazon Web Services 2023k). Yksityinen avain määriteltiin seuraavasti käyttöön: Yhteysmäärittely aloitettiin kopioimalla AWS-hallintanäkymästä luodun koneen DNS-nimi (Amazon Web Services 2023k). Kuviossa 51 näkymässä valitaan Connect.



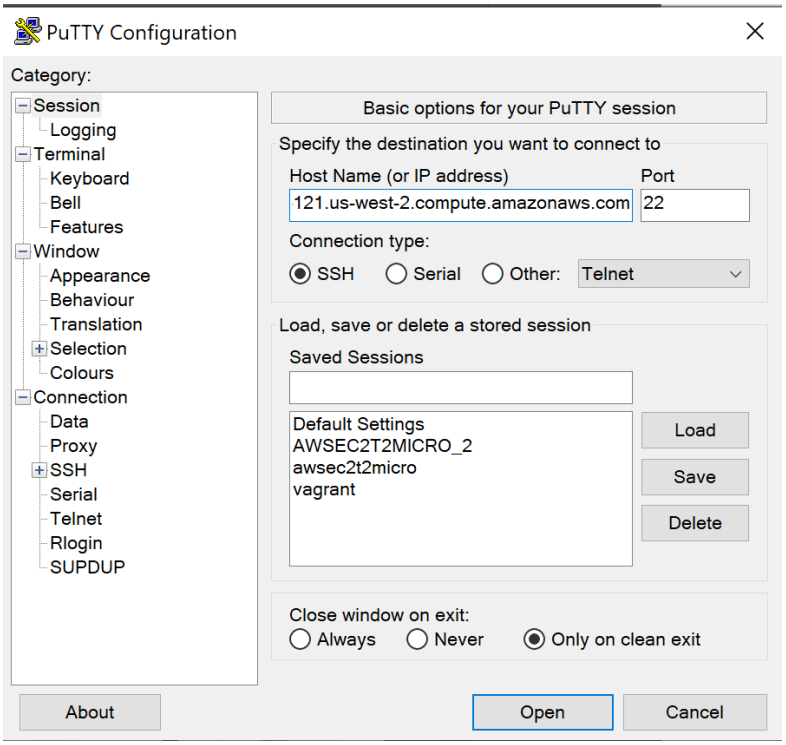
Kuvio 51. Virtuaalikoneen tiedot

Kuvion 52 näkymässä SSH client -välilehdellä nähdään ohjeet yhteyden muodostamiseen SSH-yhteyden yli (Amazon Web Services 2023n). Näkymästä kopioitiin public DNS -osoite.



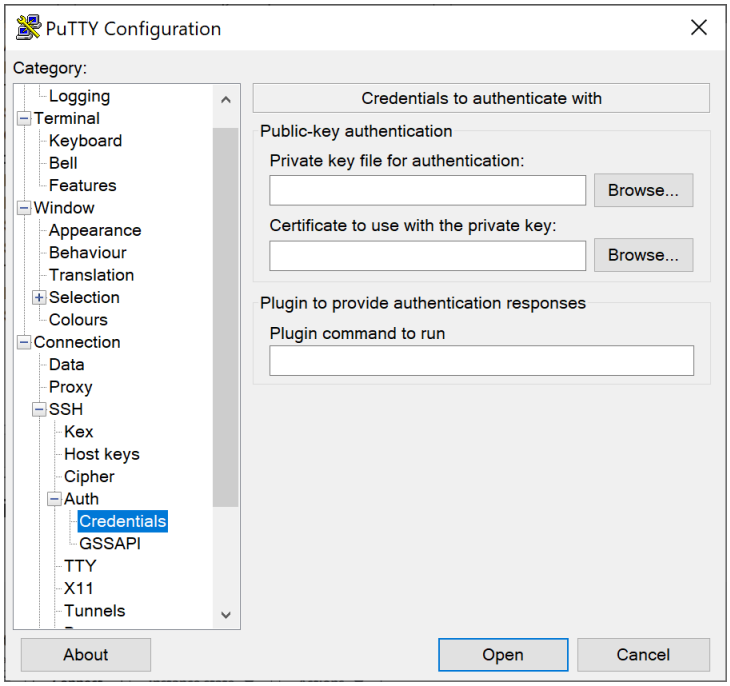
Kuvio 52. Ohje SSH-yhteyden muodostamiseen

Seuraavaksi siirryttiin Putty-ohjelmaan ja aloitettiin yhteyden määrittely. Kuviossa 53 nähdään edellä kopioitu Public DNS -osoite liitettynä Host name -syöttöalueeseen (Amazon Web Services 2023k). Yhteystyypiksi valittiin SSH, kuten kuviossa 53 näkyy. Porttinumero tulee automaattisesti valitun yhteystyypin mukaisesti.



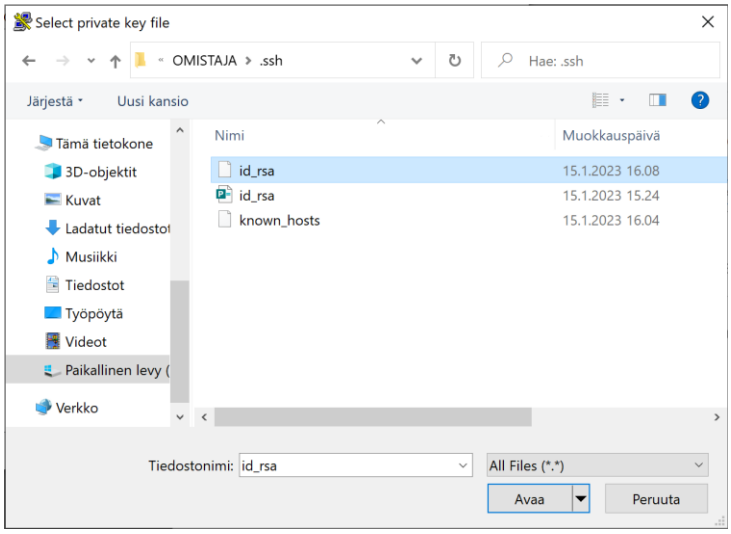
Kuvio 53. Julkinen DNS-osoite

Seuraavaksi siirryttiin Category-valintaikkunassa polkuun SSH->Auth->Credentials (Amazon Web Services 2023k). Kuviossa 54 näkyy valittu polku.



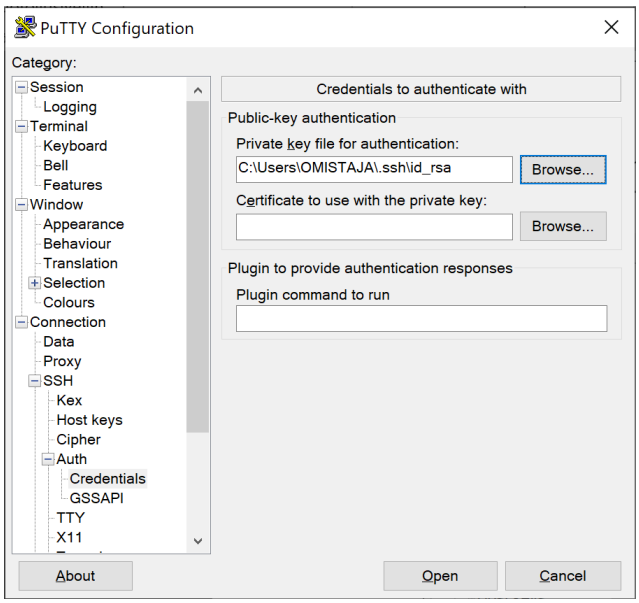
Kuvio 54. Tunnistautumisnäkymä

Näkymässä valittiin private key file for authentication -kohdassa Browse, minkä jälkeen valittiin edellä luotu yksityinen avain (Amazon Web Services 2023k). Kuviossa 55 näkyy valittu avain.



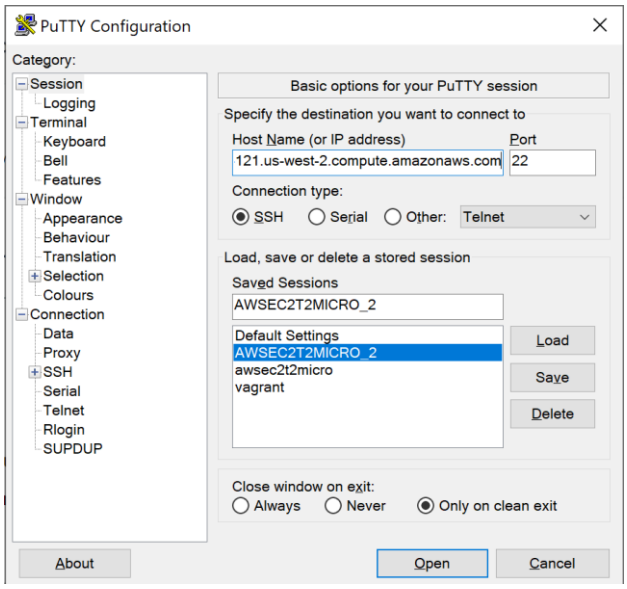
Kuvio 55. Yksityisen avaimen määrittely

Kuviossa 56 nähdään edellä valittu polku. Kuviossa 56 näkyvä polkumäärittely viittaa aiemmin luotuun RSA-avainparin sijaintiin.



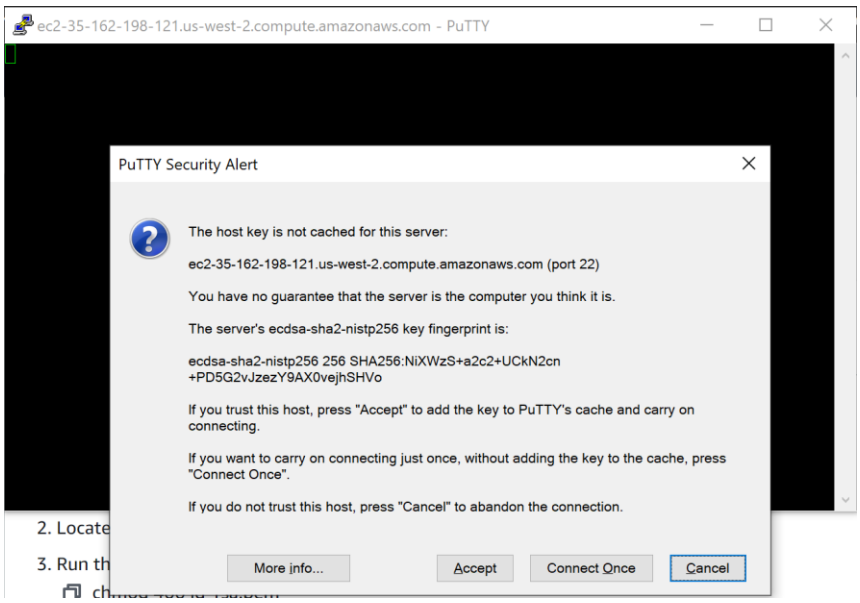
Kuvio 56. Yksityinen avain määriteltynä

Seuraavaksi siirryttiin Category-valintaikkunassa kohtaan Session. Kuvion 57 mukaisesti on suositeltavaa tallentaa edellä luodut yhteysasetukset tulevaa käyttöä varten. Tallentaminen tapahtui antamalla yhteydelle nimi saved sessions -kohtaan ja valitsemalla Save (PUTTY 2023). Yhteyden muodostaminen aloitettiin valitsemalla Open.



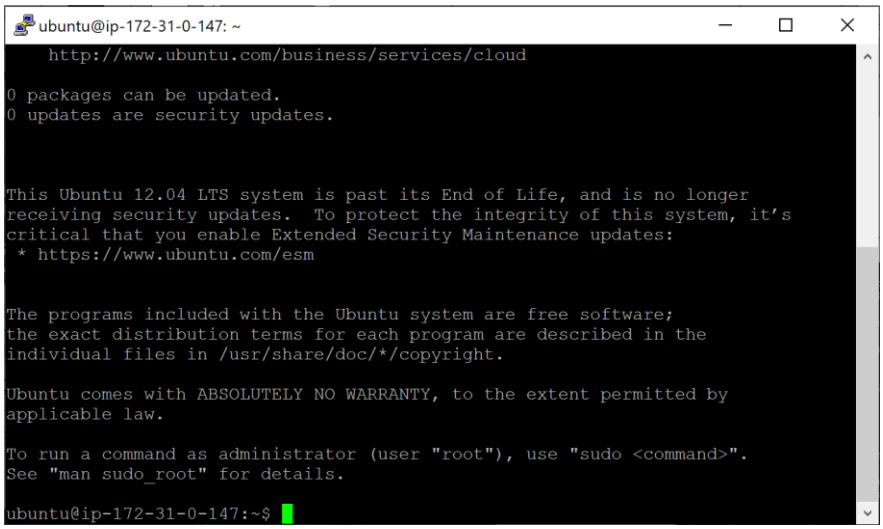
Kuvio 57. Yhteyden tallentaminen ja nimeäminen

Kun SSH-yhteys luotiin aikaisemmin tuntemattomaan koneeseen, Putty varoitti yhteyden muodostamisen riskeistä (Amazon Web Services 2023k). Yhteys koneeseen muodostettiin valitsemalla kuvion 58 näkymästä Accept.



Kuvio 58. Puttyn varoitus yhteyden muodostamisesta

Kun yhteyden muodostamisen riskit oli hyväksytty, Putty yhdistyi edellä luotuun AWS-virtuaalikoneeseen ja kysyi käyttäjätunnusta. Oletuksena ami-830c94e3-koneessa käyttäjätunnuksena on ubuntu (Amazon Web Services 2023n). Kirjautuminen ei vaadi salasanaa, vaan todentaminen tapahtuu yksityisellä avaimella. Kuviossa 59 nähdään onnistunut yhteyden muodostus.



Kuvio 59. Yhteyden muodostaminen yksityisellä avaimella

5.1.6 Kolmen virtuaalikoneen luominen

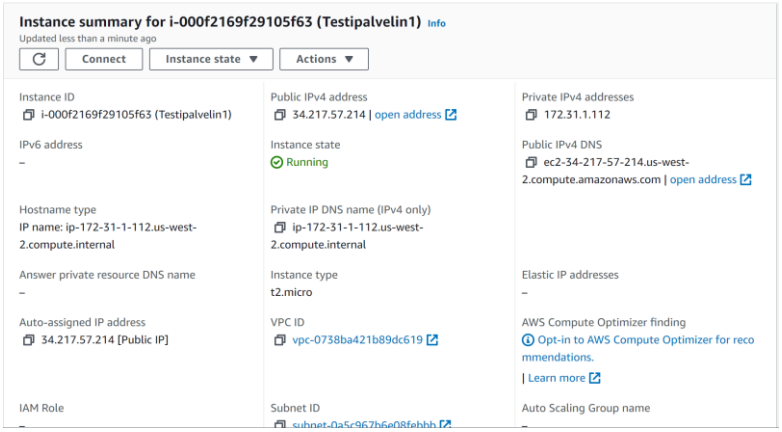
Seuraavaksi luotiin opinnäytetyön tavoitteissa asetetut kolme virtuaalikonetta. Kuviossa 60 nähdään main.tf-tiedosto, joka luo kolme palvelinta nimeltään Testipalvelin1, Testipalvelin2 ja Testipalvelin3. Määrittelytiedosto käyttää kaikkien kolmen palvelimen luomisessa samaa RSA-julkista-avainta, joten yhteydet koneisiin saatiin muodostettua samalla Putty-ohjelman yhteysmäärittelyllä vaihtaen vain konekohtaisen Public DNS -osoitteen. Main.tf-tiedoston muokkauksen jälkeen tiedosto tallennettiin ja ajettiin komentokehotteessa komento *terraform init*, jolloin terraform valmisteli tiedoston käytön (Hashicorp 2023n). *Terraform plan* -komennolla varmistettiin, että tiedosto oli luotu oikein (Hashicorp 2023n). Tarkastuksen jälkeen voitiin ajaa *terraform apply* -komento, jolloin virtuaalikoneet luotiin AWS-palveluun (Hashicorp 2023n).

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

  required_version = ">= 1.2.0"
}

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

resource "aws_instance" "app_server1" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
  key_name= "id_rsa"

  tags = {
    Name = "Testipalvelin1"
  }
  }
  resource "aws_instance" "app_server2" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
  key_name= "id_rsa"

  tags = {
    Name = "Testipalvelin2"
  }
}
resource "aws_instance" "app_server3" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
  key_name= "id_rsa"

  tags = {
    Name = "Testipalvelin3"
  }
}

resource "aws_key_pair" "deployer"{
key_name ="id_rsa"
public_key="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZ07
}
```

Kuvio 60. Main.tf-tiedosto kolmen virtuaalikoneen määrittelyllä

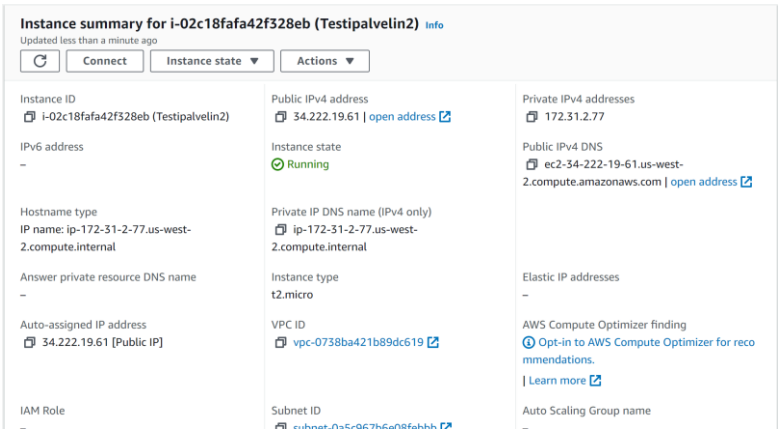
Liitteessä 4 nähdään komennot *terraform init, terraform plan* ja *terraform apply* ja niiden aiheuttamat tulostukset (Hashicorp 2023n). Liitteessä 4 nähdään myös AWS-hallintanäkymästä, että kolme konetta luotiin (Amazon Web Services 2023f).

Edellä luotuihin koneisiin voitiin kokeilla ICMP echo request -viestiä. Palvelimien osoitteet voidaan käydä kopioimassa AWS-hallintapaneelista (Amazon Web Services 2023f) tai käyttämällä AWS CLI -komentoa *aws ec2 describe-instances* (Amazon Web Services 2023o). Edellä mainittu AWS CLI -komento tulostaa kaikki tiedot koneista. Kuvioissa 61 nähdään Testipalvelin1-virtuaalikoneen tiedot.



Kuvio 61. Testipalvelin1-virtuaalikoneen tiedot

Kuviossa 62 nähdään Testipalvelin2-virtuaalikoneen tiedot. Kuviosta 62 nähdään esimerkiksi julkinen DNS-osoite.

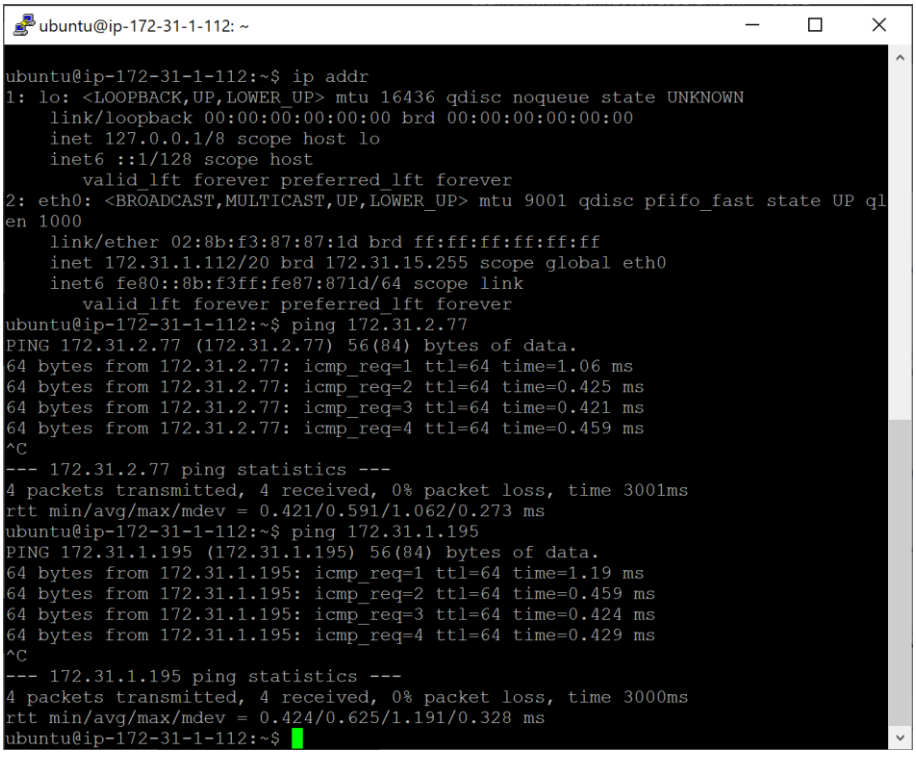


Kuvio 62. Testipalvelin2-virtuaalikoneen tiedot

Kuviossa 63 nähdään Testipalvelin3-virtuaalikoneen tiedot. Kuviossa 63 olevien tietojen perusteella voidaan kokeilla etäyhteyttä luotuun koneeseen.

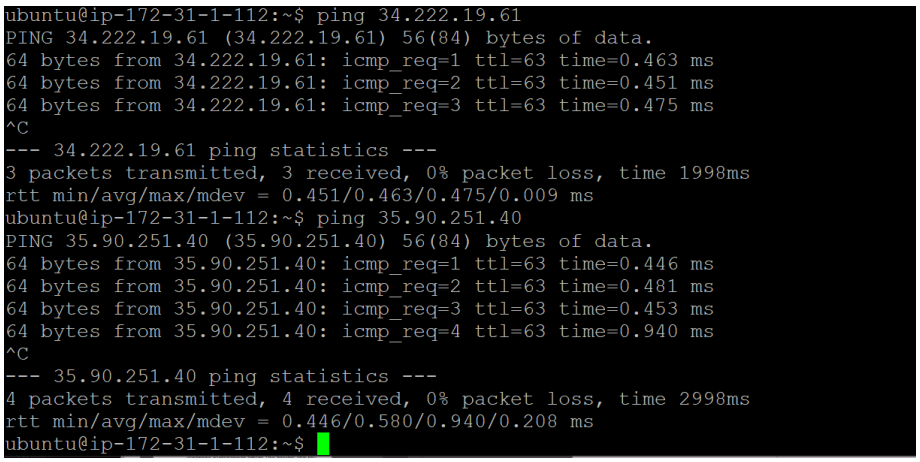
Kuvio 63. Testipalvelin3-virtuaalikoneen tiedot

Kuviossa 64 nähdään onnistuneet vastaukset ping-viesteihin. Tässä tapauksessa yhteyksien todentamisessa käytettiin AWS-palvelun sisäisiä osoitteita.



Kuvio 64. Testit sisäisen verkon osoitteilla

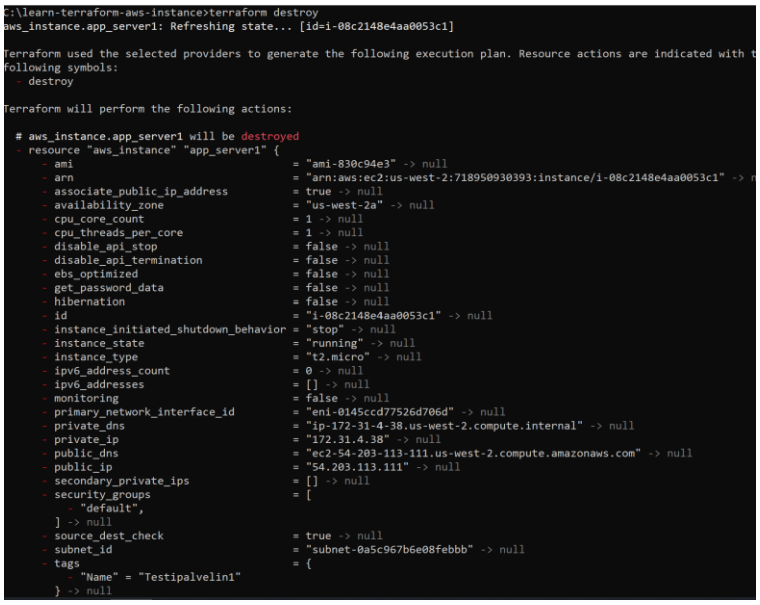
Kuvion 65 näkymässä Testipalvelin1–3-virtuaalikoneiseen testattiin yhteydet käyttäen julkisia osoitteita. Kuviossa 65 nähdään, että ping-viestit onnistuivat koneiden välillä.



Kuvio 65. Testit julkisilla osoitteilla

5.1.7 Ympäristön tuhoaminen

Koneiden luomisen ja yhteyksien testauksen jälkeen edellä luodut koneet poistettiin. Ympäristöjen tuhoaminen tapahtui komennolla *terraform destroy.* (Hashicorp 2023p.) Kuvioissa 66 nähdään komennon antaminen.

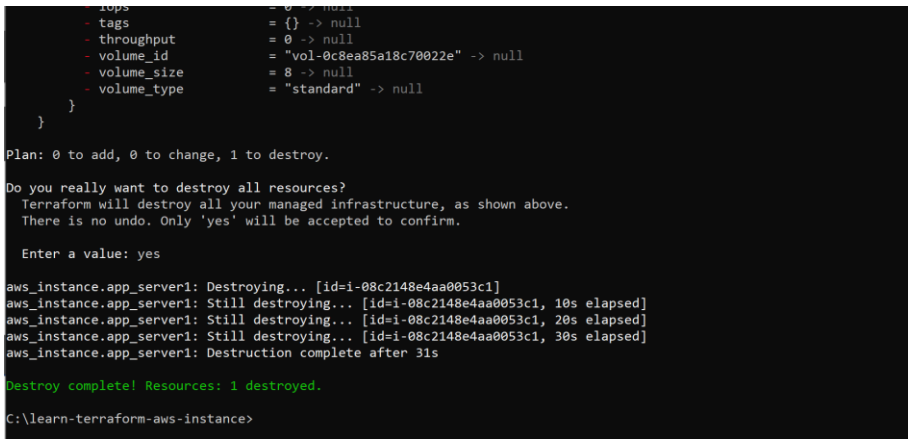


Kuvio 66. Ympäristöjen tuhoaminen

Kuviossa 67 nähdään, että tuhoamiskomento etenee. Komento poistaa edellä luodut ympäristöt.

Kuvio 67. *Terraform destroy* -komento

Kuviossa 68 nähdään, että tuhoamiskomento pitää varmistaa. Varmistin tuhoamiskomennon syöttämällä tekstin *yes*.



Kuvio 68. Ympäristöjen tuhoaminen päättynyt

5.2 Tietojärjestelmän luominen paikalliseen virtualisointiympäristöön

On Premise -ympäristöllä tarkoitetaan paikalliseen tietojärjestelmäympäristöön asennettua virtualisointiympäristöä. Paikallinen ympäristö on yleensä organisaation itsensä hallussa oleva tietojärjestelmäympäristö (itewiki 2023), joka voi olla

hyvinkin laaja. Opinnäytetyössä On Premise -ympäristönä käytettiin yksittäistä kannettavaa tietokonetta, jossa käyttöjärjestelmän päällä ajettiin virtualisointiohjelmistoa.

Paikallisen tietojärjestelmäympäristön luominen suunniteltiin alun perin tehtäväksi myös Terraform-työkalulla. Opinnäytetyössä edetessä selvisi kuitenkin, että Terraform-työkalu ei soveltunut opinnäytetyössä suunniteltuun tarkoitukseen.

Terraformilla yritettiin ensin tehdä On Premise-asennukset Windows-ympäristössä hyödyntäen Oraclen VirtualBox-työpöytävirtualisointia, johon ei ole saatavilla virallisesti ylläpidettyä Terraform provideria. On Premise -asennuksia kokeiltiin tehdä myös Linux-työpöydillä käyttäen KVM-virtualisointia ja Libvirt-ohjelmointirajapintaa virtualisointialustojen hallintaan. Linux-asennukset tehtiin käyttäen Ubuntu 20.04 ja 22.04, Centos 7 ja Fedora 36 -jakeluita. Linux-ympäristöjen On Premise -asennuksiin ei ole myöskään tarjolla virallisesti ylläpidettyä Terraform provideria, vaan on käytettävä kolmannen osapuolen tuottamia providereita. On Premise -tutkimuksessa käytetyt providerit eivät ole virallisesti ylläpidettyjä, eikä Hashicorp takaa niiden toimintaa. Hashicorpilla on toinen jakelu nimeltään Vagrant, joka on tarkoitettu On Premise -ympäristöjen rakentamiseen (Hashicorp 2023q).

Vagrant eroaa Terraformista siten, että Terraform on tarkoitettu isojen infrastruktuurien rakentamiseen pilvipalveluympäristöihin ja Vagrant on tarkoitettu On Premise -ympäristöihin. Molemmat työkalut ovat kuitenkin käytettävissä sekä pilvipalvelu- että On Premise -ympäristöihin. (Hashicorp 2023g.)

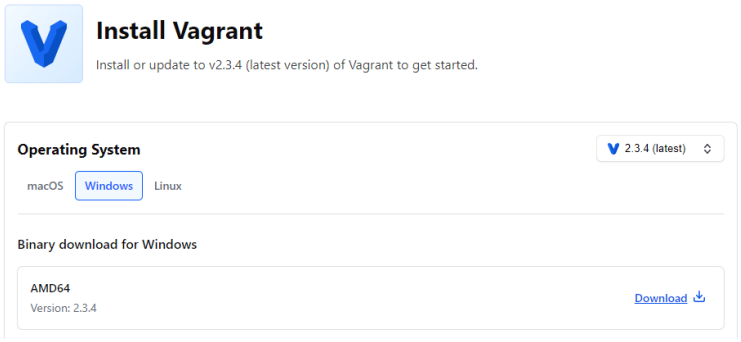
Terraformia käytettäessä Linux KVM -ympäristöt antoivat vikailmoituksen *can't connect to monitor*. Vikailmoituksen perusteella löytyi useita erilaisia korjausehdotuksia, mutta mikään niistä ei suoraan saanut asennuksia onnistumaan. Ainoa toimiva jakelu, jossa On Premise -testit saatiin toimimaan, oli Fedora 36 -jakelu. Myöskään Fedora 36 -jakelun osalta Terraformin käyttöönotto ei toiminut suoraan ohjeiden mukaan, vaan vaati lisäkonfiguraatiota muun muassa kansioiden käyttöoikeuksien osalta.

Vagrantin toiminta-ajatuksena on käyttää joko verkosta löytyviä valmiita levykuvia, joissa on valmiiksi asennettuja käyttöjärjestelmiä tai sitten käyttäjän itse luomia boxeja. Valmiiksi luotuja levykuvia löytyy useimpiin käyttötarkoituksiin.

Terraform päätettiin korvata On Premise -osuudessa Hashicorpin toisella työkalulla nimeltään Vagrant. Vagrant on suunniteltu paikallisten ympäristöjen rakentamiseen. Vagrantista löytyy suoraan tuetut providerit VirtualBoxille, KVM:lle ja Libvirtille (Hashicorp 2023h). On Premise -ympäristön luominen Windows-käyttöjärjestelmään aloitettiin asentamalla Vagrant.

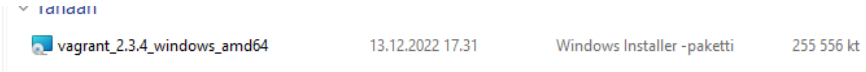
5.2.1 Vagrant-asennus

Vagrant-asennus aloitettiin lataamalla viimeisin käyttöjärjestelmälle sopiva asennuspaketti osoitteesta *https://developer.hashicorp.com/vagrant/downloads* (Hashicorp 2023r). Kuviossa 69 nähdään Vagrant-latauslinkki Windows-käyttöjärjestelmälle.



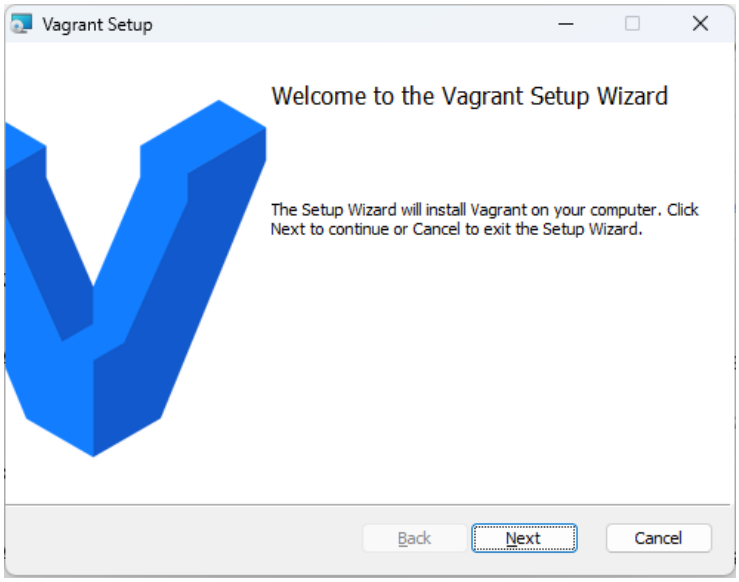
Kuvio 69. Vagrant-asennuspaketti

Asennus aloitettiin klikkaamalla ladattua pakettia. Paketti näkyy kuviossa 70 tietokoneen ladatut tiedostot -hakemistossa.



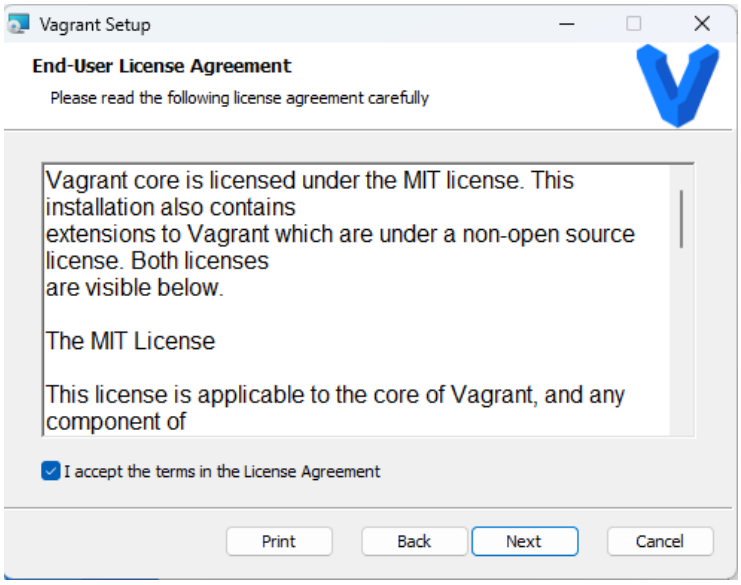
Kuvio 70. Asennuspaketti downloads-kansiossa

Asennuspaketin klikkaus käynnisti asennusvelhon, joka on näkyvissä kuviossa 71. Asennusta jatkettiin valitsemalla Next.



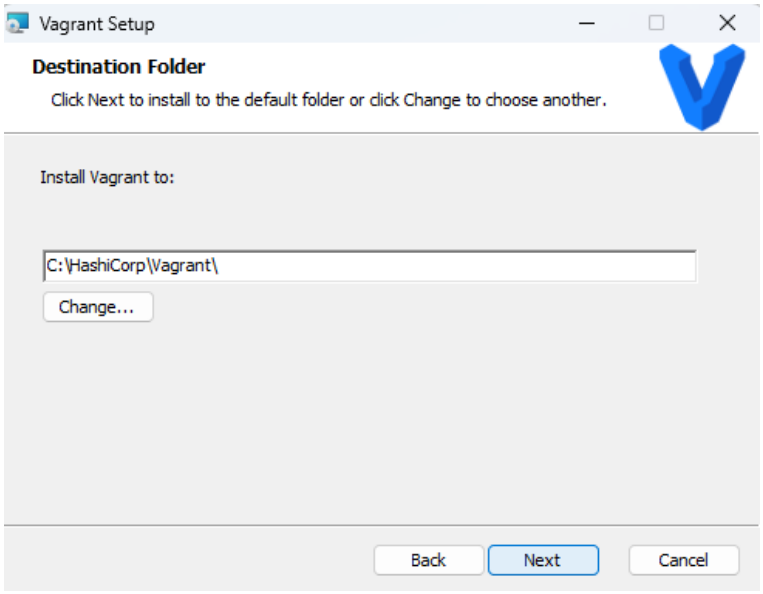
Kuvio 71. Vagrant-asennusvelho

Asennus jatkui, kun lisenssiehdot hyväksyttiin. Hyväksyntäikkuna näkyy kuviossa 72. Lisenssiehdot hyväksyttiin laittamalla valinta alaosan valintaikkunaan I Accept the terms in the Lecence Agreement ja painamalla Next.



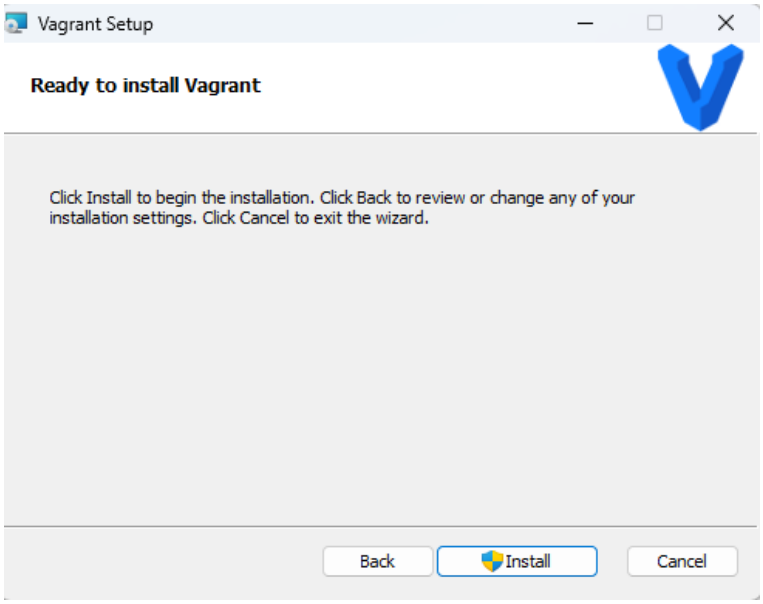
Kuvio 72. Loppukäyttäjän lisenssisopimus

Seuraavaksi valittiin kansio, johon sovellus asennetaan. Käytetty polku näkyy kuviossa 73. Asennus jatkui valitsemalla Next.



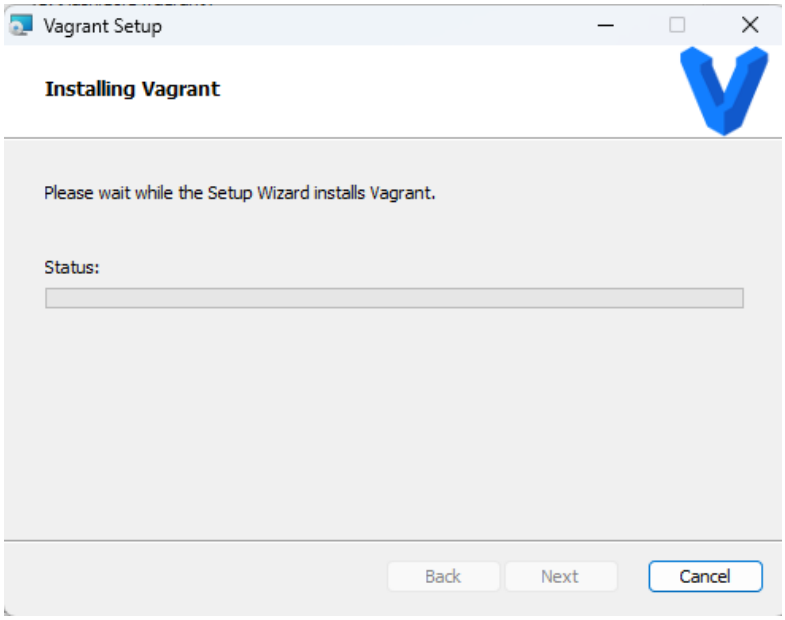
Kuvio 73. Vagrant-asennuskansio

Asennuksen määrittelyiden jälkeen voitiin asennus aloittaa. Asennus aloitettiin kuviossa 74 näkyvästä ikkunasta valitsemalla Install.



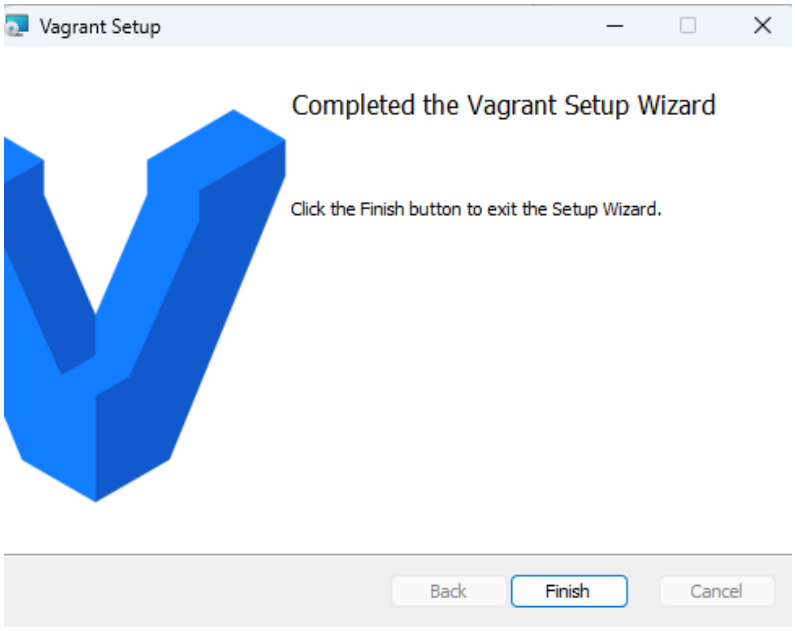
Kuvio 74. Asennuksen aloitusvalinta.

Asennuksen aloituksen jälkeen näkyviin tuli ikkuna, joka näkyy kuviossa 75. Näkyviin tulevasta ikkunasta voitiin seurata asennuksen etenemistä.



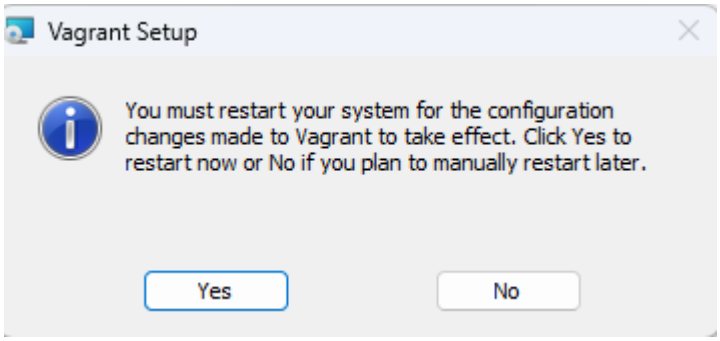
Kuvio 75. Asennuksen eteneminen

Kuviossa 76 asennus on tullut päätökseen. Asennuksen viimeistelemiseksi valittiin Finish.



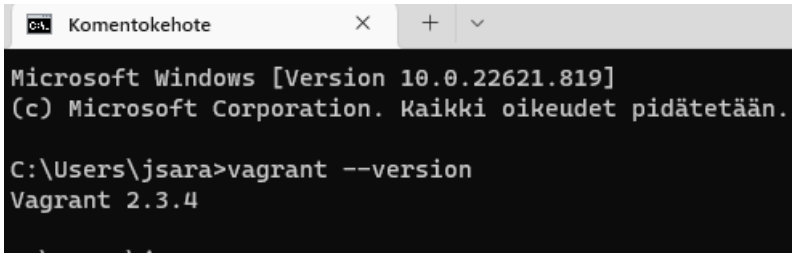
Kuvio 76. Asennuksen viimeistely

Tietokone piti käynnistää uudelleen asennuksen päätteeksi. Kuviossa 77 näkyy uudelleenkäynnistysilmoitus. Kuviosta valittiin Yes.



Kuvio 77. Uudelleenkäynnistyskehotus ja valintamahdollisuus

Tietokoneen uudelleen käynnistyksen jälkeen, asennuksen onnistuminen varmistettin komentokehotteesta. Asennuksen onnistuminen todettiin kirjoittamalla komentokehotteeseen komento *vagrant --version*, jolloin vastaukseksi tuli asennettu versio, kuten kuviossa 78 näkyy. (Hashicorp 2023r.)



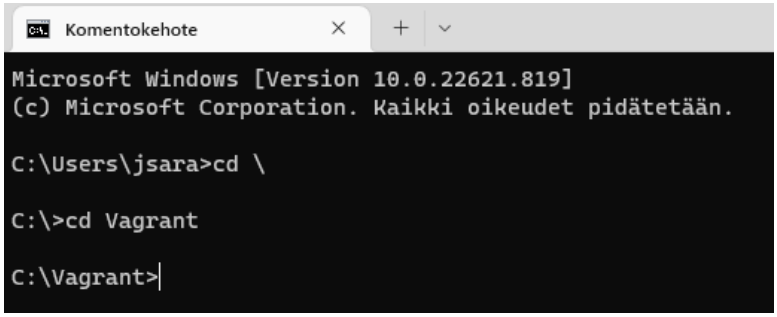
Kuvio 78. Asennetun Vagrantin versionumero

5.2.2 Vagrantfilen luonti

Vagrant-asennuksen jälkeen aloitettiin seuraava vaihe, joka oli Vagrant-tiedoston tekeminen. Vagrant-tiedostossa määriteltiin, minkälaisia virtuaalikoneita halutaan luoda (Hashicorp 2023j). Vagrant-tiedoston luonti aloitettiin luomalla hakemisto, johon Vagrantin käytössä tarvittavat tiedostot tallennetaan. (Hashicorp 2023i.) Hakemisto luotiin suoraan C-aseman juureen. Kuviossa 79 näkyy hakemiston polku.

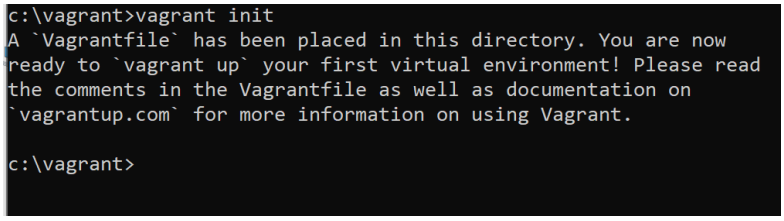
Kuvio 79. Luotu Vagrant-kansio

Seuraavaksi siirryttiin käyttämään komentokehotetta. Kuviossa 80 näkyy siirtyminen äsken luotuun kansioon.



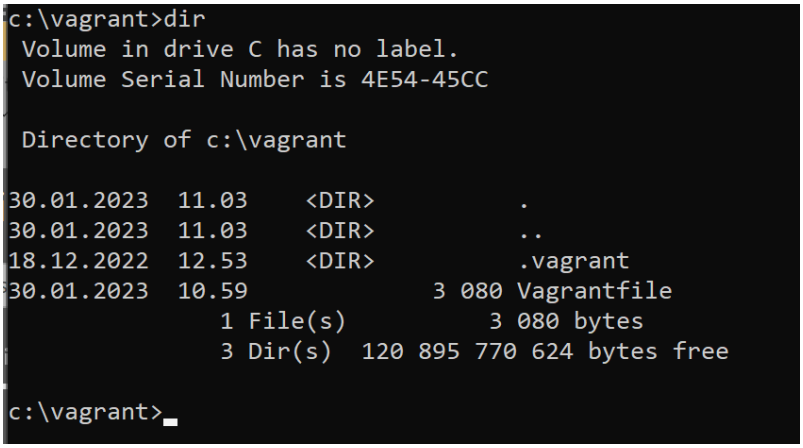
Kuvio 80. Vagrant-hakemisto komentokehotteessa

Seuraavaksi alustettiin projekti kuviossa 81 näkyvällä komennolla. *Vagrant init* -komento luo Vagrantfile-tiedoston (Hashicorp 2023i).



Kuvio 81. Projektin alustus *vagrant init* -komennolla

Kuviossa 82 nähdään alustuskomennon luoma Vagrantfile. Vagrantfile syntyi samaan kansioon kuin missä vagrant init -komento ajettiin.



Kuvio 82. Vagranfile komentokehotteen näkymässä

Edellä luodun Vagrantfilen avulla VirtualBoxiin saatiin luotua uusi virtuaalikone. Kuviossa 83 nähdään konfiguraatiorivit, jossa määritellään virtuaalikoneen luonnissa käytettävä box. Vagrantfile sisältää oletuksena ”base” boxin. (Hashicorp 2023i.)

```
# Every Vagrant development environment requires a box. You can search for
# boxes at https://vagrantcloud.com/search.
config.vm.box = "base"
```

Kuvio 83. Oletus-box määriteltynä

Seuraavaksi muokattiin käytettäväksi boxiksi ubuntu/focal64. Ubuntu/focal64-box sisältää uusimman Linux Ubuntu -version, mitä Vagrant-cloudista on saatavilla (Hashicorp 2023l). Vagrantfilen muokkaus kannattaa Windows-ympäristössä tehdä esimerkiksi Visual Studio Code -koodausympäristöllä. Tässä tapauksessa muokkaukseen käytettiin Notepad++ -ohjelmistoa, joka on kevyempi vaihtoehto. Kuviossa 84 nähdään muokattu box-tietue. Muokkauksen jälkeen vagrantfile tulee tallentaa. Erilaisiin käyttötarkoituksiin on saatavilla valmiita boxeja, joita voi käyttää omiin tarpeisiinsa (Hashicorp 2023l).

```
# https://docs.vagrantup.com.

# Every Vagrant development environment requires a box. You can search for
# boxes at https://vagrantcloud.com/search.
config.vm.box = "ubuntu/focal64"
```

Kuvio 84. Valittu box määriteltynä käyttöön

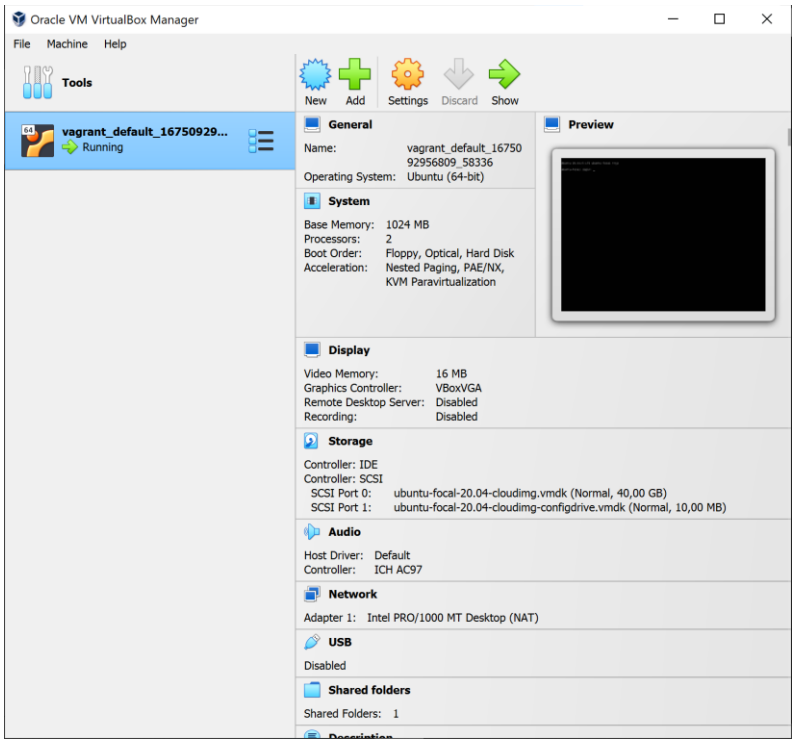
5.2.3 Ensimmäisen ympäristön luominen

Seuraavaksi ajettiin *vagrant up* -komento. Kuviossa 85 nähdään vagrant up -komennon eteneminen. Vagrant up -komento luo Vagrantfilen määrittelyjen mukaisen virtuaalikoneen (Hashicorp 2023i).

```
c:\vagrant>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/focal64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/focal64' version '20230128.0.0' is up to date...
==> default: Setting the name of the VM: vagrant_default_1675092956809_58336
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection reset. Retrying...
    default: Warning: Connection aborted. Retrying...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 6.1.38
    default: VirtualBox Version: 7.0
==> default: Mounting shared folders...
    default: /vagrant => C:/vagrant

c:\vagrant>
```

Kuvio 85. Vagrant up -komento ajettuna

VirtualBoxiin ilmestyi uusi kone, joka on luotu Vagrantilla. Kuviossa 86 nähdään VirtualBoxin näkymä edellä luodusta koneesta.



Kuvio 86. Vagrantin luoma uusi virtuaalikone

5.2.4 Yhteyden muodostaminen vagrant-komennolla

Virtuaalikoneeseen voitiin liittyä komentokehotteesta käyttämällä vagrant-komentoja (Hashicorp 2023s). Ensiksi selvitettiin luodun virtuaalikoneen nimi komennolla *vagrant status* (Hashicorp 2023t). Kuviossa 87 nähdään vastaus komentoon.

```
c:\vagrant>vagrant status
Current machine states:

default                   running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.

c:\vagrant>
```

Kuvio 87. Vastaus vagrant status -komentoon

Seuraavaksi voitiin liittyä luotuun koneeseen käyttämällä komentoa *vagrant ssh default* (Hashicorp 2023s). Kuviossa 88 nähdään onnistunut yhdistäminen.

```
c:\vagrant>vagrant ssh default
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-137-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Jan 30 15:45:18 UTC 2023

  System load:  0.08              Processes:              116
  Usage of /:   3.6% of 38.70GB   Users logged in:        0
  Memory usage: 20%               IPv4 address for enp0s3: 10.0.2.15
  Swap usage:   0%


0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


vagrant@ubuntu-focal:~$
```

Kuvio 88. SSH-yhteys default-koneeseen

5.2.5 Ympäristön poistaminen

Seuraavaksi poistettiin edellä luotu virtuaalikone komennolla *vagrant destroy* (Hashicorp 2023s). Kuviossa 89 nähdään edellä luodun ympäristön poistaminen.

```
c:\vagrant>vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...

c:\vagrant>
```

Kuvio 89. Virtuaalikoneen poistaminen

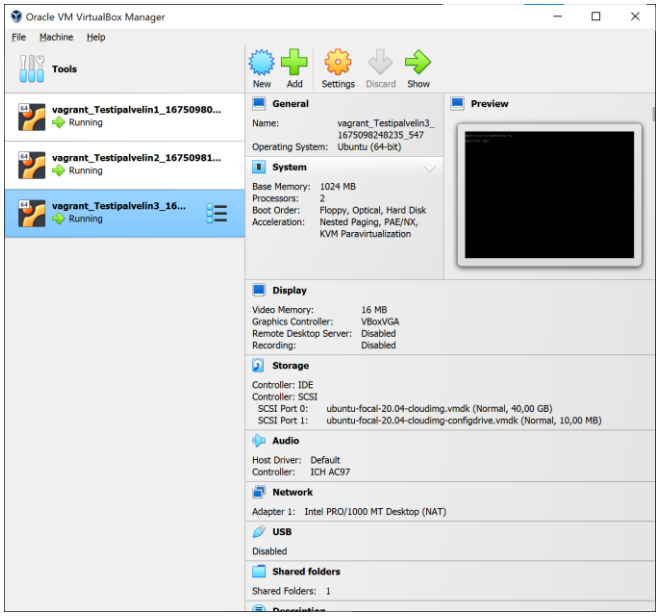
5.2.6 Kolmen palvelimen luominen

Seuraavaksi luotiin VirtualBoxiin kolme palvelinta, joiden välillä toimii verkkoyhteydet (Hashicorp 2023u). Kuviossa 90 nähdään vagrantfilen sisältö. Koodissa annettiin luotaville koneille tunnistettavat nimet Testipalvelin1, Testipalvelin2 ja Testipalvelin3.

```
Vagrant.configure("2") do |config|

config.vm.define "Testipalvelin1" do |tp1|
    tp1.vm.box = "ubuntu/focal64"

end
    config.vm.define "Testipalvelin2" do |tp2|
    tp2.vm.box = "ubuntu/focal64"

  end
  config.vm.define "Testipalvelin3" do |tp3|
    tp3.vm.box = "ubuntu/focal64"

end
end
```

Kuvio 90. Kolmen virtuaalikoneen vagrantfile määriteltynä

Kolme palvelinta saatiin luotua kuten edellä käyttäen *vagrant up* -komentoa. Liitteessä 5 nähdään, miten virtuaalikoneiden luominen etenee komentokehotteessa. Kuviossa 91 nähdään VirtualBoxin hallintanäkymästä edellä luodut palvelimet luotuna ja käynnissä.



Kuvio 91. Kolme käynnissä olevaa palvelinta

Edellä luotuihin koneisiin saatiin yhteys SSH:lla ainoastaan käyttämällä *vagrant ssh* -komentoa (Hashicorp 2023s). Koneisiin voitiin liittyä käyttämällä koneiden nimeä, joka saatiin selville vagrantfile -tiedostosta. Toinen vaihtoehto nimien selvittämiseksi on antaa komento *vagrant status* (Hashicorp 2023t). Kuviossa 92 nähdään vastaus annettuun komentoon.

```
c:\vagrant>vagrant status
Current machine states:

Testipalvelin1            running (virtualbox)
Testipalvelin2            running (virtualbox)
Testipalvelin3            running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.

c:\vagrant>
```

Kuvio 92. Vagrant status -komento

Komennolla *vagrant ssh Testipalvelin1* liityttiin Testipalvelin1-virtuaalikoneeseen (Hashicorp 2023t). Kuviossa 93 nähdään, miten liittyminen näkyy komentokehotteessa.

```
c:\vagrant>vagrant ssh Testipalvelin1
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-137-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Jan 30 17:06:44 UTC 2023

  System load:  0.02              Processes:               119
  Usage of /:   3.6% of 38.70GB   Users logged in:         0
  Memory usage: 21%               IPv4 address for enp0s3: 10.0.2.15
  Swap usage:   0%


0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


vagrant@ubuntu-focal:~$
```

Kuvio 93. Vagrant ssh -komento

5.2.7 Verkkoyhteydet

Opinnäytetyössä asetettiin vaatimukseksi, että luotavien virtuaalikoneiden välillä tulee kyetä lähettämään ICMP echo request -viesti ja saada vastaus kyselyyn. Peruskonfiguraatiolla vagrantfile ei tee VirtualBoxiin verkkoyhteyttä, jolla luotujen

koneiden välillä voitaisiin lähettää ping-viestejä. Opinnäytetyön tavoitteisiin päästiin luomalla kaikkiin koneisiin uusi internal network -tyypin verkkokortti ja antamalla niille kiinteät osoitteet (Hashicorp 2023v). Kuviossa 94 nähdään luotu konfiguraatio verkkoyhteyden lisäämiseksi.

```
Vagrant.configure("2") do |config|

config.vm.define "Testipalvelin1" do |tp1|
    tp1.vm.box = "ubuntu/focal64"
    tp1.vm.network "private_network", ip: "192.168.50.4",
    virtualbox__intnet: true

end
    config.vm.define "Testipalvelin2" do |tp2|
    tp2.vm.box = "ubuntu/focal64"
    tp2.vm.network "private_network", ip: "192.168.50.5",
    virtualbox__intnet: true

  end
  config.vm.define "Testipalvelin3" do |tp3|
    tp3.vm.box = "ubuntu/focal64"
    tp3.vm.network "private_network", ip: "192.168.50.6",
    virtualbox__intnet: true

end
end
```

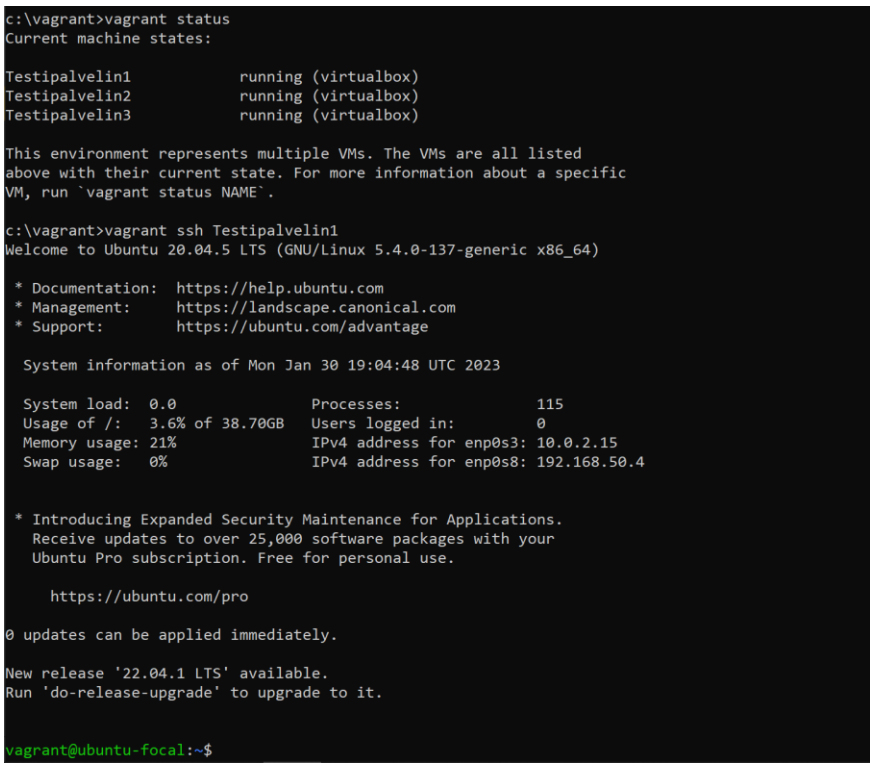
Kuvio 94. Uusien verkkoyhteyksien luominen

Seuraavaksi poistettiin edellä luodut koneet komennolla *vagrant destroy* (Hashicorp 2023s). Kuviossa 95 nähdään komento ajettuna.

```
c:\vagrant>vagrant destroy
    Testipalvelin3: Are you sure you want to destroy the 'Testipalvelin3' VM? [y/N] y
==> Testipalvelin3: Forcing shutdown of VM...
==> Testipalvelin3: Destroying VM and associated drives...
    Testipalvelin2: Are you sure you want to destroy the 'Testipalvelin2' VM? [y/N] y
==> Testipalvelin2: Forcing shutdown of VM...
==> Testipalvelin2: Destroying VM and associated drives...
    Testipalvelin1: Are you sure you want to destroy the 'Testipalvelin1' VM? [y/N] y
==> Testipalvelin1: Forcing shutdown of VM...
==> Testipalvelin1: Destroying VM and associated drives...

c:\vagrant>
```

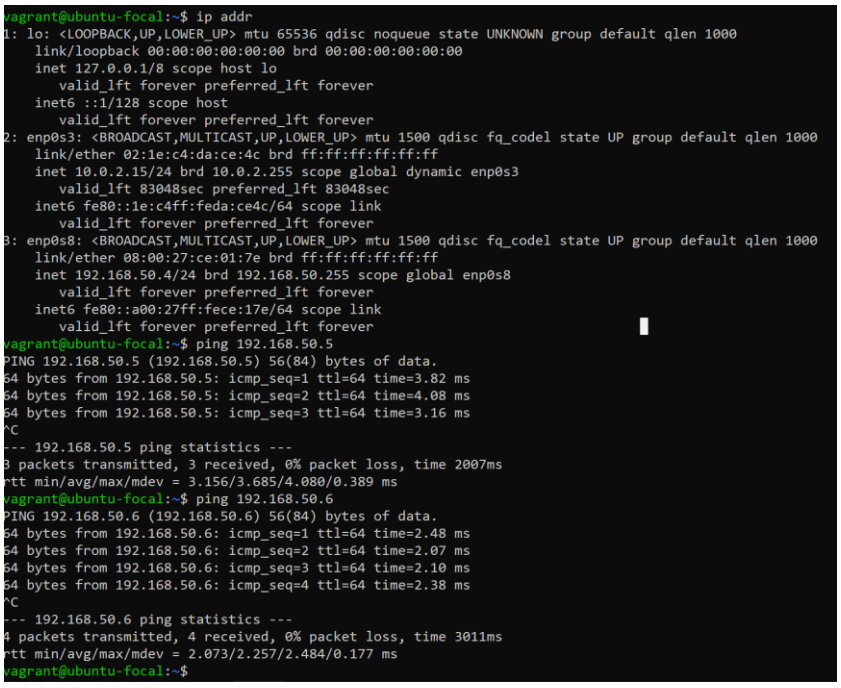
Kuvio 95. Vagrant destroy -komento

Uudet koneet saatiin luotua komennolla *vagrant up* (Hashicorp 2023i) ja päästiin kokeilemaan yhteyksiä. Liityttiin ensin Testipalvelin1-virtuaalikoneeseen komennolla *vagrant ssh Testipalvelin1* (Hashicorp 2023s). Testipalvelin1-koneelta voitiin kokeilla ping-viestiä Testipalvelin2- ja Testipalvelin3-koneille. Kuvioissa 96 nähdään palvelinten tila haettuna *vagrant status* -komennolla.



Kuvio 96. *vagrant ssh* -komento

Kuviossa 97 nähdään Testipalvelimen1:n osoite ja vastaukset ping-viestiin Testipalvelin2- ja Testipalvelin3-koneilta. Yhteydet toimivat uusien verkkokorttien määrittelyn jälkeen.



Kuvio 97. Ping-viestien vastaukset

6 POHDINTA

Opinnäytetyön toiminnallisen osuuden perusteella voi sanoa, että IaC-työkalut ovat erittäin tuottavia toimiessaan. Työkaluissa kannattaa tukeutua pelkästään virallisesti tuettuihin providereihin. Epävirallisia providereita on saatavilla suoraan Hashicorpin sivuilta ja myös muista lähteistä. Epävirallisia providereita ei ylläpidetä samoin kuin virallisia, ja siitä johtuen ne toimivat vain tiettyjen käyttöjärjestelmä- ja ohjelmistoversioiden sekä laitekonfiguraatioiden kanssa. Epävirallisten providereiden kanssa työskennellessä joudutaan helposti tilanteeseen, jossa ympäristöjen luonti ei onnistu, vaan tarvitaan mahdollisesti suuri määrä vian selvitystä ja erinäköisiä korjaustoimenpiteitä riippuen käyttöjärjestelmäversioista, ohjelmistoversioista ja käytetyn koneen ominaisuuksista. Ohjelmistoyhtiöt tekevät myös räätälöityjä providereita asiakkaiden tarpeiden mukaisesti.

Asennusten toteutus aloitettiin luomalla tili AWS-palveluun. AWS-palveluun voi luoda ilmaisen tilin, jolla kyettiin tekemään opinnäytetyön tavoitteissa määritellyt tehtävät. Virtuaalikoneiden tuottaminen AWS-palveluun Terraform-työkalulla onnistui ohjeita seuraamalla. Ohjeet ovat selkeät ja koneet saadaan luotua. Virallisia ohjeita Terraformin käyttämiseen löytyy sekä AWS-palvelusta että Hashicorpin sivuilta. Lisäksi useista foorumeista on saatavilla apua työkalun käyttämiseen.

Opinnäytetyön tavoitteisiin pääseminen yhteyksien testauksen osalta vaati kuitenkin selvittelyä sekä AWS-palvelun että Putty-ohjelmiston osalta. Yhteyksien testaaminen tavoitteiden mukaisesti vaati RSA-avainparin luomisen ja käyttöönottamisen. Lisäksi AWS-palveluun piti luoda security group, joka salli ICMP- ja SSH-liikenteen.

Asennusten toteutuksen toisessa vaiheessa oli tarkoitus luoda paikallisella koneella olevaan VirtualBoxiin virtuaalikoneita käyttäen Terraform-työkalua. Terraform-työkaluun ei ollut tarjolla Hashicorpin tuottamaa provideria, mutta epävirallisia providereita oli tarjolla. Terraform-työkalun avulla ei saatu luotua toimivia virtuaalikoneita Windows-ympäristöön lukuisista yrityksistä ja selvitystyöstä huolimatta. Seuraavaksi siirryttiin yrittämään Linux KVM -ympäristöön virtuaalikoneiden luomista. Toimivien koneiden luominen KVM-virtualisointiympäristöön ei myöskään onnistunut. Linux-ympäristön ongelmien korjaamiseen löytyi erittäin

paljon epävirallisia ohjeita, mutta millään löydetyistä ohjeista ei saatu Terraformia toimimaan moitteettomasti.

Tässä vaiheessa keskustelimme opinnäytetyön ohjaajan kanssa, miten käytännön osuutta jatkettaisiin. Olin tässä vaiheessa jo testannut Hashicorpin toista IaC-työkalua nimeltään Vagrant, joka on tarkoitettu käytettäväksi pieniin paikallisiin ympäristöihin. Tulimme ohjaajan kanssa siihen tulokseen, että käytännön osuutta oli järkevää jatkaa käyttäen työkaluna Vagrantia On Premise -asennusten osalta.

Paikalliset asennukset saatiin tehtyä kohtalaisella työllä käyttäen Vagrant-työkalua. Vagrant tukee muun muassa VirtualBoxia oletuksena. Vagrantin käyttöön löytyy paljon ohjeita Hashicorpin sivuilta ja ohjeita noudattamalla asennukset saatiin onnistumaan tavoitteiden mukaisesti.

Opinnäytetyön tekeminen kehitti osaamistani IaC-työkalujen käytössä. En ollut aikaisemmin käyttänyt IaC-työkaluja virtuaalikoneiden luomisessa, vaikka olen tehnyt ylläpitotöitä virtuaalikoneiden kanssa useiden vuosien ajan. Uskon, että työmarkkinoilla olisi valtti, jos työntekijällä on valmiiksi kokemusta IaC-työkalujen käytöstä eri alustoilla. Tällä hetkellä lehdissä on uutisia, että osa yhtiöistä on siirtämässä palvelujaan takaisin On Premise -ympäristöihin muun muassa pilvipalveluiden korkeiden hintojen takia. IaC-työkalujen osaaminen On Premise -ympäristöissä on tarpeellinen taito jatkossakin. IaC-työkaluille ja niiden käyttämisen osaamiselle olisi hyvä olla opintojakso saatavilla, jolloin tekniikasta kiinnostuneet voisivat laajentaa osaamistaan.

Opinnäytetyöhön liittyen jatkotutkimusideoiksi näkisin IaC-koodin tutkimisen, opetusympäristöjen luonnin pilvipalveluun tai Private Cloudiin IaC-työkalujen avulla, IaC-koodin tietoturvan tutkimisen, oman providerin luonnin valittuun virtualisointiympäristöön Terraform-työkalulle ja omien Vagrant-boxien luominen johonkin tiettyyn käyttötarkoitukseen.

LÄHTEET

Amazon Web Services 2023a. Enable SSH Connections for your Linux WorkSpaces. Viitattu 2.2.2023 https://docs.aws.amazon.com/workspaces/latest/adminguide/connect-to-linux-workspaces-with-ssh.html.

Amazon Web Services 2023b. What is an API? Viitattu 5.2.2023 https://aws.amazon.com/what-is/api/.

Amazon Web Services 2023c. AWS Free Tier. Viitattu 18.2.2023 https://aws.amazon.com/free/?all-free-tier.sort by=item.additionalFields.SortRank&all-free-tier.sort order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all.

Amazon Web Services 2023d. General Purpose. Viitattu 26.2.2023 https://aws.amazon.com/ec2/instance-types/.

Amazon Web Services 2023e. Regions and Zones. Viitattu 26.2.2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

Amazon Web Services 2023f. Finding an instance ID or IP address, AMS. Viitattu 10.3.2023 https://docs.aws.amazon.com/managedservices/latest/userguide/find-instance-id.html.

Amazon Web Services 2023g. Installing or updating the latest version of the AWS CLI. Viitattu 11.3.2023 https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html.

Amazon Web Services 2023h. Sign in to the AWS Management Console. Viitattu 11.3.2023 https://docs.aws.amazon.com/signin/latest/userguide/console-sign-in-tutorials.html#introduction-to-root-user-sign-in-tutorial.

Amazon Web Services 2023i. Creating an IAM user in your AWS account. Viitattu 11.3.2023 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html.

Amazon Web Services 2023j. Setting the AWS CLI output format. Viitattu 11.3.2023 https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-output-format.html.

Amazon Web Services 2023k. Connect to your Linux instance from Windows using PuTTY. Viitattu 11.3.2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html.

Amazon Web Services 2023l. Amazon EC2 security groups for Linux instances. Viitattu 11.3.2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html.

Amazon Web Services 2023m. Amazon EC2 key pairs and Linux instances. Viitattu 11.3.2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html.

Amazon Web Services 2023n. Set up to connect to your instance. Viitattu 11.3.2023 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-get-info-about-instance.

Amazon Web Services 2023o. describe-instances. Viitattu 11.3.2023 https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html.

Chocolatey 2023a. Installing Chocolatey. Viitattu 5.2.2023 https://chocolatey.org/install#individual.

Chocolatey 2023b. The Package Manager for Windows Modern Software Automation. Viitattu 3.3.2023 https://chocolatey.org/.

CSA 2015. Best Practices for Mitigating Risks in Virtualized Environments. Viitattu 10.3.2023 https://downloads.cloudsecurityalliance.org/whitepapers/Best_Practices_for%20_Mitigating_Risks_Virtual_Environments_April2015_4-1-15_GLM5.pdf.

DataCenterKnowledge 2017. AWS Outage that Broke the Internet Caused by Mistyped. Viitattu 9.3.2023 Commandhttps://www.datacenterknowledge.com/archives/2017/03/02/aws-outage-that-broke-the-internet-caused-by-mistyped-command.

Fortinet 2023. What Is a Thin Client? Viitattu 5.2.2023 https://www.fortinet.com/resources/cyberglossary/thin-client.

GeeksforGeeks 2022. Kernel in Operating System. Viitattu 5.2.2023 https://www.geeksforgeeks.org/kernel-in-operating-system/.

GovInfo 2023. The NIST Definition of Cloud Computing. Viitattu 15.2.2023 https://www.govinfo.gov/app/details/GOVPUB-C13-74cdc274b1109a7e1ead7185dfec2ada.

Gupta, M., Chowdary, M., Bussa, S. & Chowdary, C. 2021. Deploying Hadoop Architecture Using Ansible and Terraform. Viitattu 25.2.2023 https://ieeexplore-ieee-org.ez.lapinamk.fi/document/9702299.

Hasan, M., Bhuiyan, F. & Rahman, A. 2020. Testing practices for infrastructure as code. Viitattu 9.3.2023 https://dl-acm-org.ez.lapinamk.fi/doi/pdf/10.1145/3416504.3424334.

Hashicorp 2023a. Install Terraform. Viitattu 5.2.2023 https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli.

Hashicorp 2023b. Automate infrastructure on any cloud. Viitattu 25.2.2023 https://www.hashicorp.com/products/terraform.

Hashicorp 2023c. Get Started. Viitattu 25.2.2023 https://developer.hashicorp.com/terraform/tutorials.

Hashicorp 2023d. What is Terraform. Viitattu 25.2.2023 https://developer.hashicorp.com/terraform/intro.

Hashicorp 2023e. Build Infrastructure. Viitattu 26.2.2023 https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build.

Hashicorp 2023f. Providers. Viitattu 26.2.2023 https://registry.terraform.io/browse/providers.

Hashicorp 2023g. Vagrant vs. Terraform. Viitattu 26.2.2023 https://developer.hashicorp.com/vagrant/intro/vs/terraform.

Hashicorp 2023h. Introduction to Vagrant. Viitattu 26.2.2023 https://developer.hashicorp.com/vagrant/intro.

Hashicorp 2023i. Initialize a Project Directory. Viitattu 28.2.2023 https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-project-setup.

Hashicorp 2023j. Vagrantfile. Viitattu 28.2.2023 https://developer.hashicorp.com/vagrant/docs/vagrantfile.

Hashicorp 2023k. Install and Specify a Box. Viitattu 28.2.2023 https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-boxes.

Hashicorp 2023l. Discover Vagrant Boxes. Viitattu 28.2.2023 https://app.vagrantup.com/boxes/search.

Hashicorp 2023m. Terraform Cloud Security Model. Viitattu 9.3.2023 https://developer.hashicorp.com/terraform/cloud-docs/architectural-details/security-model#recommendations-for-securely-using-terraform-cloud.

Hashicorp 2023n. Build Infrastructure. Viitattu 26.2.2023 https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build.

Hashicorp 2023o. Resource: aws_key_pair. Viitattu 11.3.2023 https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/key_pair.

Hashicorp 2023p. Destroy Infrastructure. Viitattu 11.3.2023 https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-destroy.

Hashicorp 2023q. Get Started. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/tutorials/getting-started.

Hashicorp 2023r. Install Vagrant. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-install.

Hashicorp 2023s. Boot an Environment. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-up.

Hashicorp 2023t. Status. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/docs/cli/status.

Hashicorp 2023u. Multi-Machine. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/docs/multi-machine.

Hashicorp 2023v. Public Networks. Viitattu 11.3.2023 https://developer.hashicorp.com/vagrant/docs/networking/public_network.

Herbert, A. 2015. Symposium on Operating Systems Principles, Virtualization definition video. Viitattu 1.2.2023 https://dl-acm-org.ez.lapinamk.fi/doi/10.1145/2830903.2830909.

IBM 2021. IBM Support. Viitattu 10.3.2023 https://www.ibm.com/support/pages/security-bulletin-powervm-hypervisor-vulnerable-specially-crafted-sequence-hypervisor-calls-partition-can-lead-system-crash.

IBM 2023. What is Infrastructure as Code (IaC)? Viitattu 21.2.2023 https://www.ibm.com/topics/infrastructure-as-code.

itewiki 2023. Pilvipalvelut. Viitattu 11.3.2023 https://www.itewiki.fi/opas/pilvipalvelut/.

KVM 2023. Kernel Virtual Machine. Viitattu 1.3.2023 https://www.linux-kvm.org/page/Main_Page.

Kyberturvallisuuskeskus 2020. Pilvipalveluiden turvallisuuden arviointikriteeristö (PiTuKri). Viitattu 3.3.2023 https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/file/Pilvipalveluiden_turvallisuuden_arviointikriteeristo_PiTuKri_v1_1.pdf.

Kyberturvallisuuskeskus 2023. Pilvipalveluiden turvallisuus. Viitattu 15.2.2023 https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/file/Pilvipalveluiden_tietoturva_organisaatioille.pdf.

Libvirt 2023a. Libvirt virtualization API. Viitattu 1.3.2023 https://libvirt.org/.

Libvirt 2023b. Libvirt FAQ. Viitattu 1.3.2023 https://wiki.libvirt.org/FAQ.html#what-is-libvirt.

Liquid Web 2021. Top Eight Virtualization Security Issues and Risks. Viitattu 10.3.2023 https://www.liquidweb.com/kb/virtualization-security-issues-and-risks/.

Luchian, E., Filip, C., Rus, A., Ivanciu, I & Dobrota, V. 2016. Automation of the infrastructure and services for an openstack deployment using chef tool. Viitattu 3.3.2023 https://ieeexplore-ieee-org.ez.lapinamk.fi/stamp/stamp.jsp?tp=&arnumber=7753200.

Microsoft 2022. What is infrastructure as code (IaC)? Viitattu 21.2.2023 https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.

Modi, R. 2019. Azure resource manager templates quick start guide: Create, deploy, and manage azure resources with ARM templates using best practices (1st edition.). Packt Publishing Ltd. Viitattu 22.3.2023 https://ebookcentral-proquest-com.ez.lapinamk.fi/lib/ulapland-ebooks/reader.action?docID=5721781&query=%22Azure+resource+manager+templates+quick+start+guide%3A+Create%2C+deploy%2C+and+manage+azure+resources+with+ARM+templates+using+best+practices%22.

NIST 2020. Development Operations (DevOps). Viitattu 22.2.2023 https://csrc.nist.gov/glossary/term/development_operations.

NIST 2023. Implementation of DevSecOps for a Microservices-based application with Service Mesh. Viitattu 21.2.2023 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf.

Portnoy, M. 2016. Virtualization Essentials. New Jersey: John Wiley & Sons, Incorporated.

Posey, B. 2023. 10 Essential Best Practices for Virtual Server Backups. Viitattu 10.3.2023 https://www.altaro.com/assets/10-Essential-Best-Practices-for-Virtual-Server-Backups.pdf.

PUTTY 2023. Loading and storing saved sessions PuTTY. Viitattu 11.3.2023 https://documentation.help/PuTTY/config-saving.html.

Rahman, A. 2018. Characteristics of Defective Infrastructure as Code Scripts in DevOps. Viitattu 9.3.2023 https://dl-acm-org.ez.lapinamk.fi/doi/pdf/10.1145/3183440.3183452.

Rathod, H. & Townsend, J. 2014. Virtualization 2.0 for Dummies. Viitattu 4.2.2023 https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/solutions/vmware-virtualization-for-dummies.pdf.

Red Hat 2017. Enable agility with infrastructure-as-code. Viitattu 2.2.2023 https://www.redhat.com/en/blog/enable-agility-infrastructure-code.

Red Hat 2022a. What is Infrastructure as Code (IaC)? Viitattu 25.2.2023 https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac.

Red Hat 2022b. What is KVM? Viitattu 1.3.2023 https://www.redhat.com/en/topics/virtualization/what-is-KVM.

Schwarz, J., Steffens, A. & Lichter, H. 2018. Code Smells in Infrastructure as Code. Viitattu 9.3.2023 https://ieeexplore-ieee-org.ez.lapinamk.fi/stamp/stamp.jsp?tp=&arnumber=8590193.

Šimec, A., Držanić, B. & Lozić, D. 2018. Isolated Environment Tools for Software Development. Viitattu 3.3.2023 https://ieeexplore-ieee-org.ez.lapinamk.fi/stamp/stamp.jsp?tp=&arnumber=8955312.

Sokolowski, D. 2022. Advancing Computing as a Science & Profession. Infrastructure as Code for Dynamic Deployments. Viitattu 23.2.2023 https://dl-acm-org.ez.lapinamk.fi/doi/pdf/10.1145/3540250.3558912.

SSH 2023. How to Use ssh-keygen to Generate a New SSH Key? Viitattu 12.3.2023 https://www.ssh.com/academy/ssh/keygen.

Tankov, V., Valchuk, D., Golubev, Y. & Bryksin, T. 2022. Infrastructure in Code: Towards Developer-Friendly Cloud Applications. Viitattu 25.2.203 https://dl-acm-org.ez.lapinamk.fi/doi/pdf/10.1109/ASE51524.2021.9678943.

Telia 2018. Pilven monet kasvot – IaaS, PaaS ja SaaS. Viitattu 14.3.2023 https://www.inmicsnebula.fi/fi/blogi/pilven-monet-kasvot-iaas-paas-ja-saas?language_content_entity=fi.

Tolliver, K. 2022. Infrastructure as Code (IaC) Security Best Practices. Progress 13.6.2022. Viitattu 8.6.2023 https://www.chef.io/blog/5-best-practices-for-iac-security.

Valtiovarainministeriö 2020. Tuottavuutta pilvipalveluilla, ohje julkisen hallinnon pilvipalvelujen hyödyntämiseen. Viitattu 22.3.2023 https://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/162451/VM_2020_66.pdf?sequence=4&isAllowed=y.

Viestintävirasto 2016. Kyberturvallisuuskeskus ohje. Viitattu 2.2.2023 https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/file/Ohje_3_2016_liite_1_Palvelunestohyokkaysten_tekniikkaa_puolustajille_0.pdf.

VirtualBox 2023a. Welcome to VirtualBox.org! Viitattu 28.2.2023 https://www.virtualbox.org/.

VirtualBox 2023b. Oracle VM VirtualBox User Manual. Viitattu 28.2.2023 https://download.virtualbox.org/virtualbox/7.0.6/UserManual.pdf.

VMware 2005. WMware ESX Server Virtual Infrastructure Node Evaluator’s Guide. Viitattu 15.2.2023 https://www.vmware.com/pdf/esx_vin_eval.pdf.

VMware 2019a. Virtual Machine Security Best Practices. Viitattu 10.3.2023 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-14CCC8CD-D90D-4227-B2C3-0A93D3C023BA.html.

VMware 2019b. General Virtual Machine Protection. Viitattu 10.3.2023 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-8B93E01D-AB37-41F2-A225-892E40BAFB35.html.

VMware 2023a. What is Infrastructure as Code (IaC)? Viitattu 2.2.2023 https://www.vmware.com/topics/glossary/content/infrastructure-as-code.html.

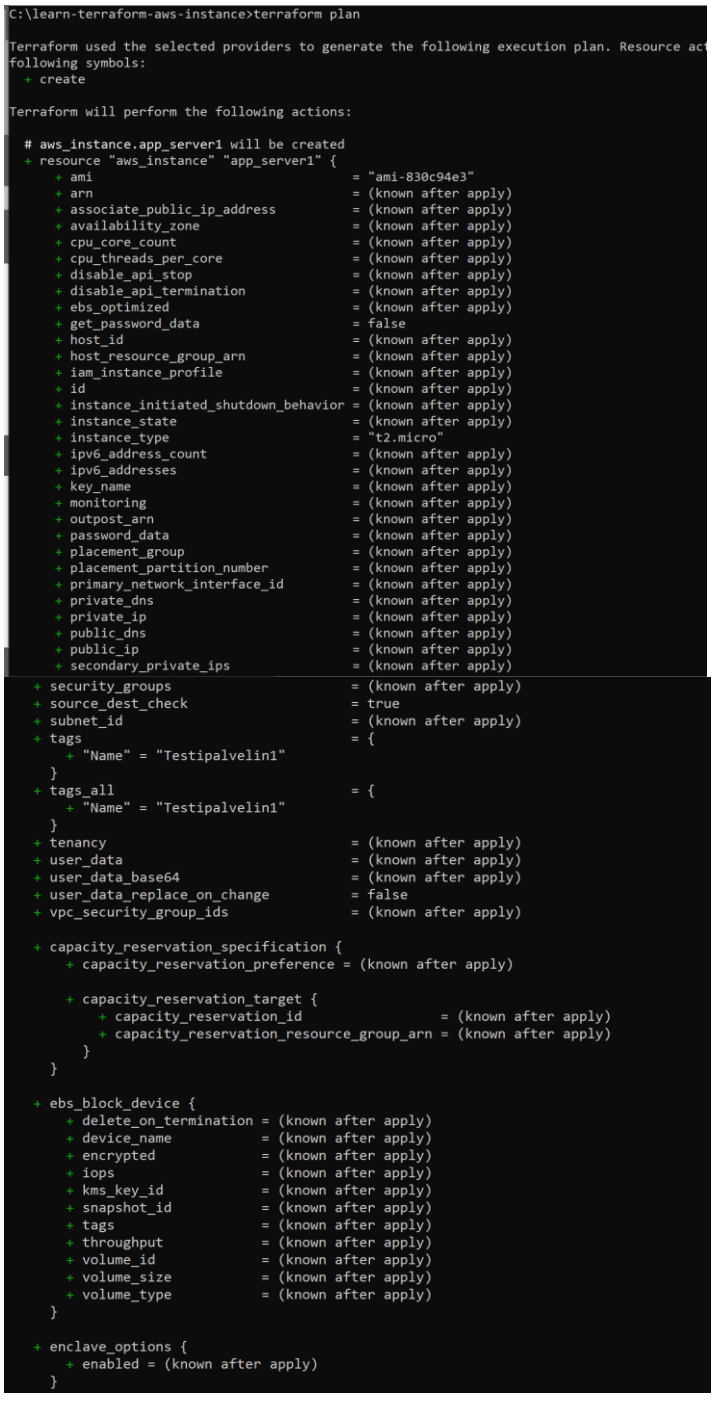
VMware 2023b. What is a hypervisor? Viitattu 5.2.2023 https://www.vmware.com/topics/glossary/content/hypervisor.html.

Von Hagen, W. 2008. Professional Xen virtualization (1st edition.). Indianapolis, Wiley Pub. Viitattu 22.3.2023 https://ebookcentral-proquest-com.ez.lapinamk.fi/lib/ulapland-ebooks/reader.action?do-cID=331669&query=Professional+Xen+virtualization.

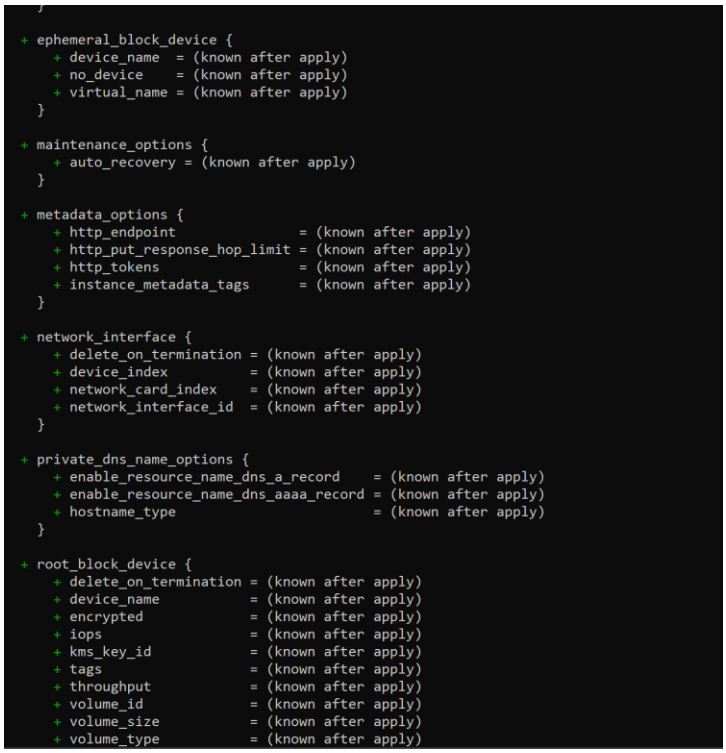
LIITTEET

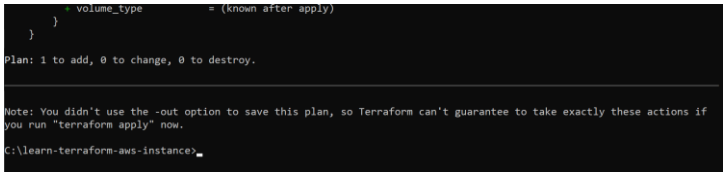
Liite 1 1(2). Terraform plan- komennon tulostus

Tässä liitteessä on terraform plan -komennon jälkeen tulleet tulostukset kokonaisuudessaan.



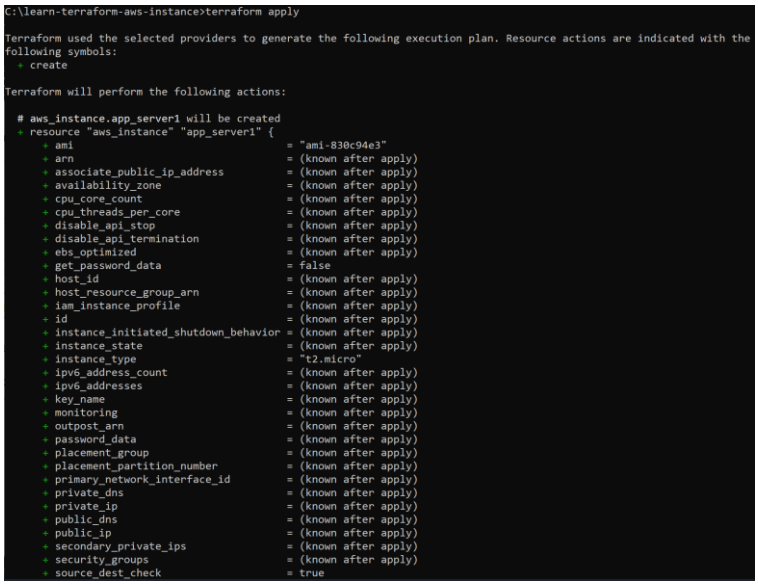
Liite 1 2(2). Terraform plan -komennon tulostus



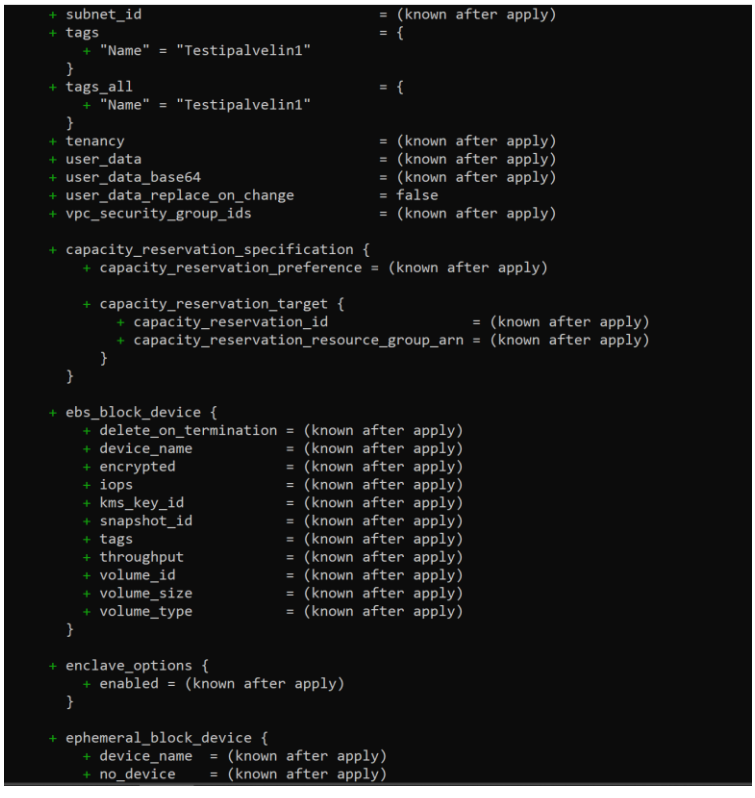


Liite 2 1(2). *Terraform apply* -komennon tulostus

Tässä liitteessä nähdään *terraform apply* -komennon tulostus kokonaisuudessaan.

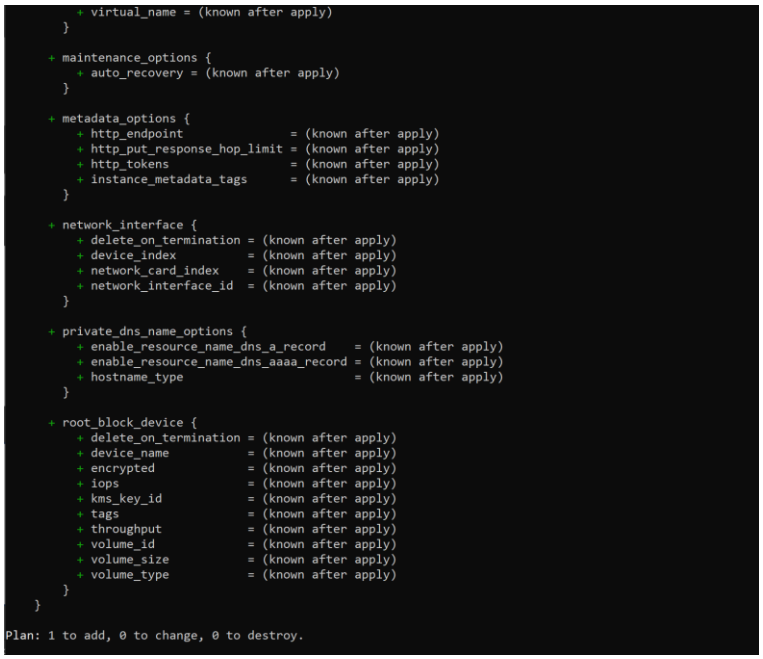


```
C:\learn-terraform-aws-instance>terraform apply

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:

  # aws_instance.app_server1 will be created
  + resource "aws_instance" "app_server1" {
      + ami                                  = "ami-830c94e3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
```

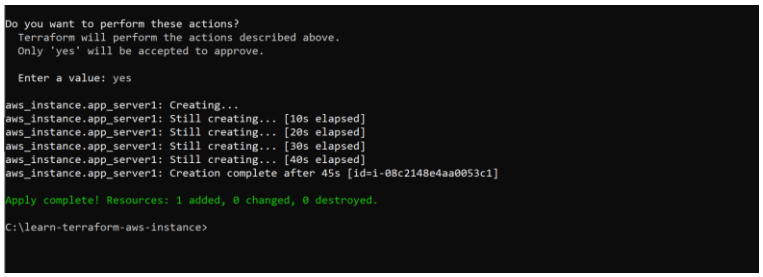


```
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "Testipalvelin1"
        }
      + tags_all                             = {
          + "Name" = "Testipalvelin1"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification {
          + capacity_reservation_preference = (known after apply)

          + capacity_reservation_target {
              + capacity_reservation_id                = (known after apply)
              + capacity_reservation_resource_group_arn = (known after apply)
            }
        }

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }

      + enclave_options {
          + enabled = (known after apply)
        }

      + ephemeral_block_device {
          + device_name  = (known after apply)
          + no_device    = (known after apply)
```

Liite 2 2(2). *Terraform apply* -komennon tulostus



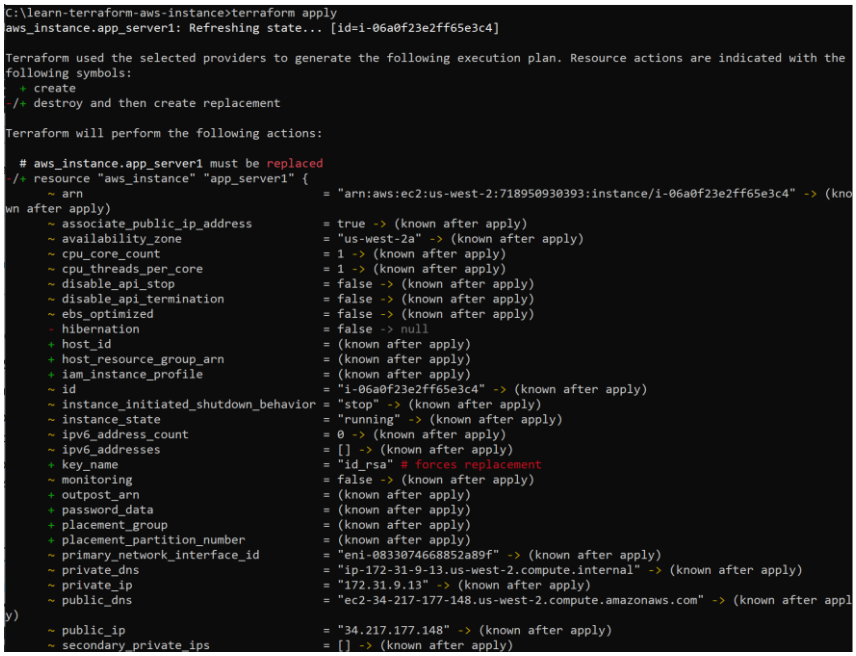
```
          + virtual_name = (known after apply)
        }

      + maintenance_options {
          + auto_recovery = (known after apply)
        }

      + metadata_options {
          + http_endpoint               = (known after apply)
          + http_put_response_hop_limit = (known after apply)
          + http_tokens                 = (known after apply)
          + instance_metadata_tags      = (known after apply)
        }

      + network_interface {
          + delete_on_termination = (known after apply)
          + device_index          = (known after apply)
          + network_card_index    = (known after apply)
          + network_interface_id  = (known after apply)
        }

      + private_dns_name_options {
          + enable_resource_name_dns_a_record    = (known after apply)
          + enable_resource_name_dns_aaaa_record = (known after apply)
          + hostname_type                        = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

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



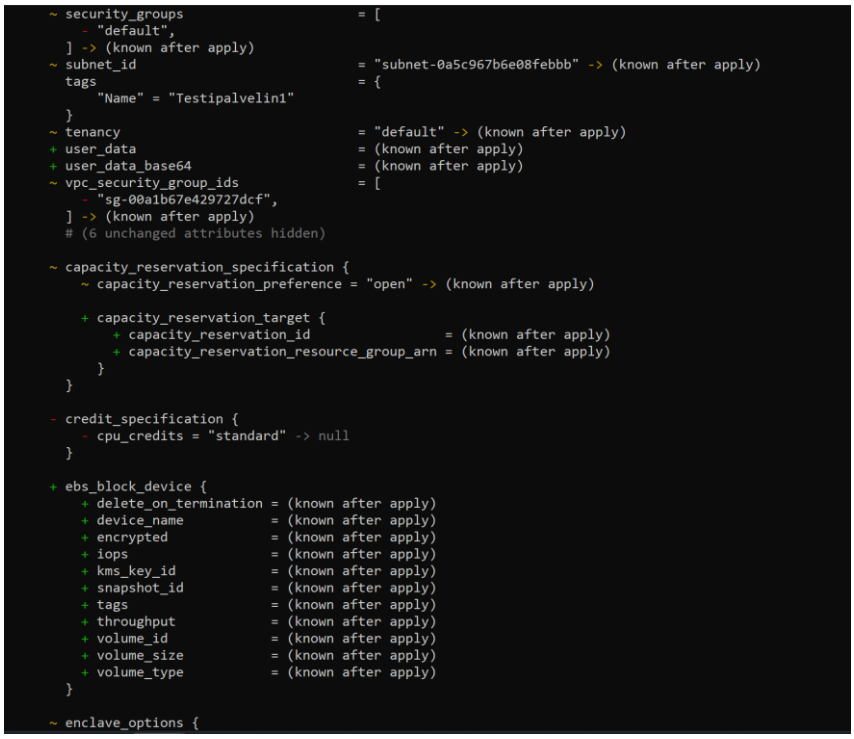
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.app_server1: Creating...
aws_instance.app_server1: Still creating... [10s elapsed]
aws_instance.app_server1: Still creating... [20s elapsed]
aws_instance.app_server1: Still creating... [30s elapsed]
aws_instance.app_server1: Still creating... [40s elapsed]
aws_instance.app_server1: Creation complete after 45s [id=i-08c2148e4aa0053c1]

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

C:\learn-terraform-aws-instance>
```

Liite 3. 1(2) *Terraform apply* -komento ajettuna uudelleen

Tässä liitteessä nähdään *terraform apply* -komento ajettuna uudelleen julkisen avaimen määrittelemiseksi.

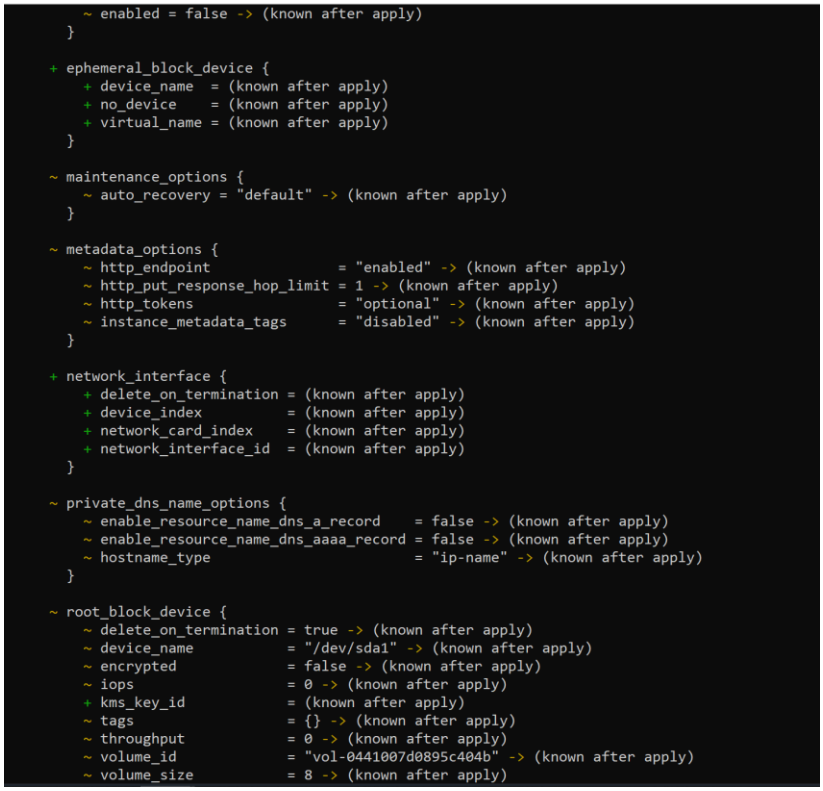


```
C:\learn-terraform-aws-instance>terraform apply
aws_instance.app_server1: Refreshing state... [id=i-06a0f23e2ff65e3c4]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_instance.app_server1 must be replaced
-/+ resource "aws_instance" "app_server1" {
      ~ arn                                  = "arn:aws:ec2:us-west-2:718950930393:instance/i-06a0f23e2ff65e3c4" -> (kno
wn after apply)
      ~ associate_public_ip_address          = true -> (known after apply)
      ~ availability_zone                    = "us-west-2a" -> (known after apply)
      ~ cpu_core_count                       = 1 -> (known after apply)
      ~ cpu_threads_per_core                 = 1 -> (known after apply)
      ~ disable_api_stop                     = false -> (known after apply)
      ~ disable_api_termination              = false -> (known after apply)
      ~ ebs_optimized                        = false -> (known after apply)
      - hibernation                          = false -> null
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      ~ id                                   = "i-06a0f23e2ff65e3c4" -> (known after apply)
      ~ instance_initiated_shutdown_behavior = "stop" -> (known after apply)
      ~ instance_state                       = "running" -> (known after apply)
      ~ ipv6_address_count                   = 0 -> (known after apply)
      ~ ipv6_addresses                       = [] -> (known after apply)
      + key_name                             = "id_rsa" # forces replacement
      ~ monitoring                           = false -> (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      ~ primary_network_interface_id         = "eni-0833074668852a89f" -> (known after apply)
      ~ private_dns                          = "ip-172-31-9-13.us-west-2.compute.internal" -> (known after apply)
      ~ private_ip                           = "172.31.9.13" -> (known after apply)
      ~ public_dns                           = "ec2-34-217-177-148.us-west-2.compute.amazonaws.com" -> (known after appl
y)
      ~ public_ip                            = "34.217.177.148" -> (known after apply)
      ~ secondary_private_ips                = [] -> (known after apply)
```

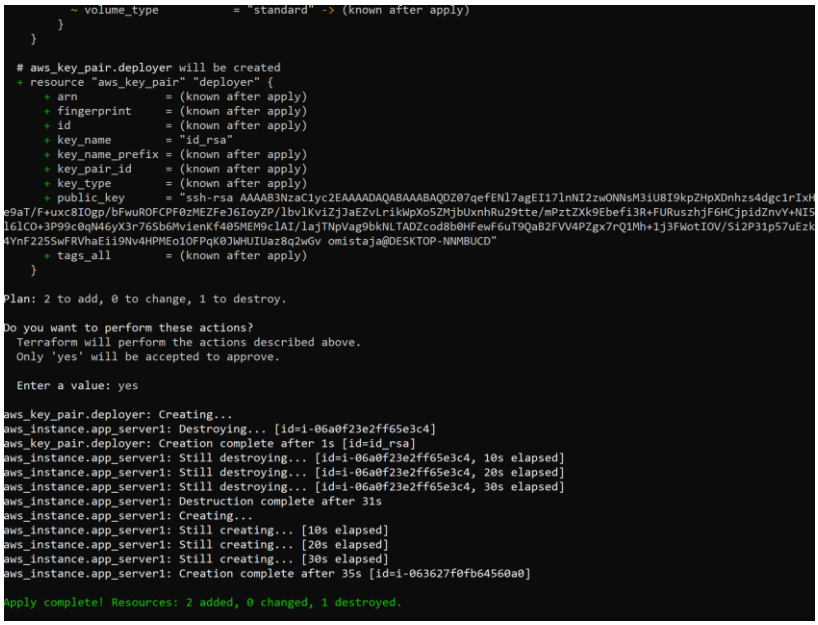


```
      ~ security_groups                      = [
          - "default",
        ] -> (known after apply)
      ~ subnet_id                            = "subnet-0a5c967b6e08febbb" -> (known after apply)
        tags                                 = {
            "Name" = "Testipalvelin1"
        }
      ~ tenancy                              = "default" -> (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      ~ vpc_security_group_ids               = [
          - "sg-00a1b67e429727dcf",
        ] -> (known after apply)
        # (6 unchanged attributes hidden)

      ~ capacity_reservation_specification {
          ~ capacity_reservation_preference = "open" -> (known after apply)

          + capacity_reservation_target {
              + capacity_reservation_id                = (known after apply)
              + capacity_reservation_resource_group_arn = (known after apply)
            }
        }

      - credit_specification {
          - cpu_credits = "standard" -> null
        }

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }

      ~ enclave_options {
```

Liite 3. 2(2) *Terraform apply* -komento ajettuna uudelleen



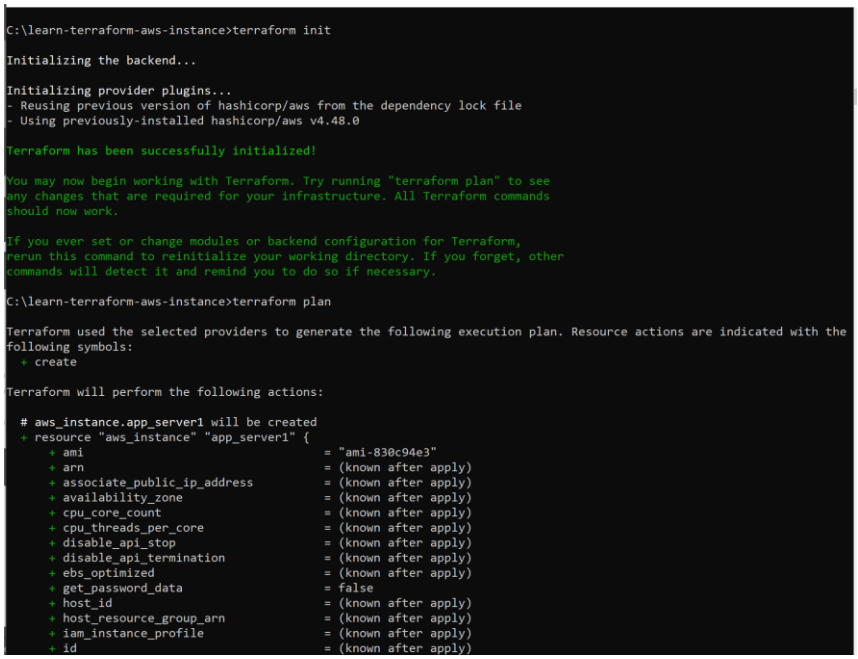
```
        ~ enabled = false -> (known after apply)
      }

    + ephemeral_block_device {
        + device_name  = (known after apply)
        + no_device    = (known after apply)
        + virtual_name = (known after apply)
      }

    ~ maintenance_options {
        ~ auto_recovery = "default" -> (known after apply)
      }

    ~ metadata_options {
        ~ http_endpoint               = "enabled" -> (known after apply)
        ~ http_put_response_hop_limit = 1 -> (known after apply)
        ~ http_tokens                 = "optional" -> (known after apply)
        ~ instance_metadata_tags      = "disabled" -> (known after apply)
      }

    + network_interface {
        + delete_on_termination = (known after apply)
        + device_index          = (known after apply)
        + network_card_index    = (known after apply)
        + network_interface_id  = (known after apply)
      }

    ~ private_dns_name_options {
        ~ enable_resource_name_dns_a_record    = false -> (known after apply)
        ~ enable_resource_name_dns_aaaa_record = false -> (known after apply)
        ~ hostname_type                        = "ip-name" -> (known after apply)
      }

    ~ root_block_device {
        ~ delete_on_termination = true -> (known after apply)
        ~ device_name           = "/dev/sda1" -> (known after apply)
        ~ encrypted             = false -> (known after apply)
        ~ iops                  = 0 -> (known after apply)
        + kms_key_id            = (known after apply)
        ~ tags                  = {} -> (known after apply)
        ~ throughput            = 0 -> (known after apply)
        ~ volume_id             = "vol-0441007d0895c404b" -> (known after apply)
        ~ volume_size           = 8 -> (known after apply)
```



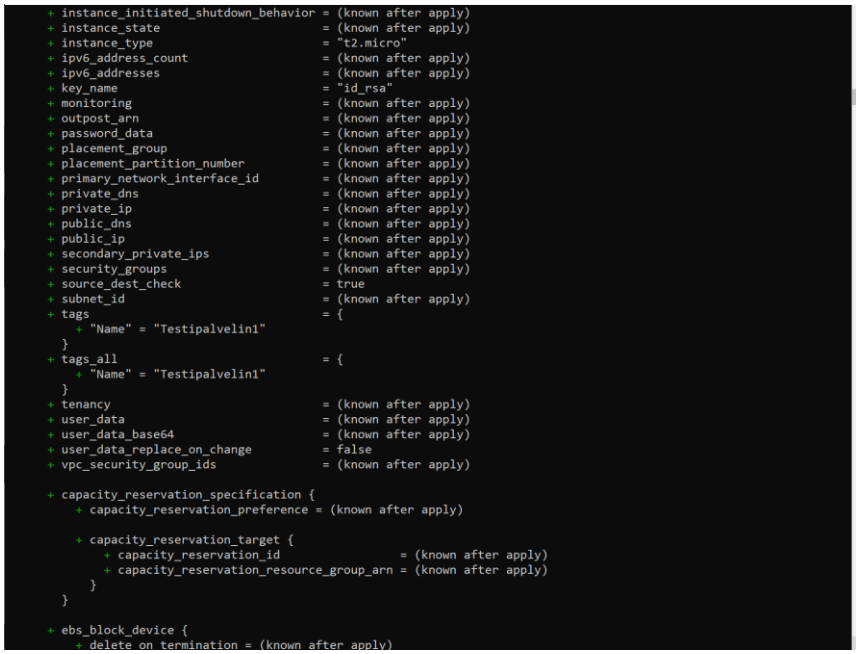
```
        ~ volume_type           = "standard" -> (known after apply)
      }
  }

  # aws_key_pair.deployer will be created
  + resource "aws_key_pair" "deployer" {
      + arn             = (known after apply)
      + fingerprint     = (known after apply)
      + id              = (known after apply)
      + key_name        = "id_rsa"
      + key_name_prefix = (known after apply)
      + key_pair_id     = (known after apply)
      + key_type        = (known after apply)
      + public_key      = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZ07qefENl7agEI17lnNI2zwONNsM3iU8I9kpZHpXDnhzs4dgc1rIxH
e9aT/F+uxc8IOgp/bFwuROFCPF0zMEZFeJ6IoyZP/lbvlKviZjJaEZvLrikWpXo5ZMjbUxnhRu29tte/mPztZXk9Ebefi3R+FURuszhjF6HCjpidZnvY+NI5
l6lCO+3P99c0qN46yX3r76Sb6MvienKf405MEM9clAI/lajTNpVag9bkNLTADZcod8b0HFewF6uT9QaB2FVV4PZgx7rQ1Mh+1j3FWotIOV/Si2P31p57uEzk
4YnF225SwFRVhaEii9Nv4HPMEo1OFPqK0JWHUIUaz8q2wGv omistaja@DESKTOP-NNMBUCD"
      + tags_all        = (known after apply)
  }

Plan: 2 to add, 0 to change, 1 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: yes

aws_key_pair.deployer: Creating...
aws_instance.app_server1: Destroying... [id=i-06a0f23e2ff65e3c4]
aws_key_pair.deployer: Creation complete after 1s [id=id_rsa]
aws_instance.app_server1: Still destroying... [id=i-06a0f23e2ff65e3c4, 10s elapsed]
aws_instance.app_server1: Still destroying... [id=i-06a0f23e2ff65e3c4, 20s elapsed]
aws_instance.app_server1: Still destroying... [id=i-06a0f23e2ff65e3c4, 30s elapsed]
aws_instance.app_server1: Destruction complete after 31s
aws_instance.app_server1: Creating...
aws_instance.app_server1: Still creating... [10s elapsed]
aws_instance.app_server1: Still creating... [20s elapsed]
aws_instance.app_server1: Still creating... [30s elapsed]
aws_instance.app_server1: Creation complete after 35s [id=i-063627f0fb64560a0]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.
```

Liite 4 1(10). Kolmen virtuaalikoneen luominen

Tässä liitteessä nähdään kolmen virtuaalikoneen luominen käyttämällä *terraform init, terraform plan ja terraform apply* -komentoja.



```
C:\learn-terraform-aws-instance>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.48.0

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.

C:\learn-terraform-aws-instance>terraform plan

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:

  # aws_instance.app_server1 will be created
  + resource "aws_instance" "app_server1" {
      + ami                                  = "ami-830c94e3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
```



```
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "id_rsa"
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "Testipalvelin1"
        }
      + tags_all                             = {
          + "Name" = "Testipalvelin1"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification {
          + capacity_reservation_preference = (known after apply)

          + capacity_reservation_target {
              + capacity_reservation_id                = (known after apply)
              + capacity_reservation_resource_group_arn = (known after apply)
            }
        }

      + ebs_block_device {
          + delete_on_termination = (known after apply)
```

Liite 4 2(10). Kolmen virtuaalikoneen luominen

```
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }

    + enclave_options {
        + enabled = (known after apply)
      }

    + ephemeral_block_device {
        + device_name  = (known after apply)
        + no_device    = (known after apply)
        + virtual_name = (known after apply)
      }

    + maintenance_options {
        + auto_recovery = (known after apply)
      }

    + metadata_options {
        + http_endpoint               = (known after apply)
        + http_put_response_hop_limit = (known after apply)
        + http_tokens                 = (known after apply)
        + instance_metadata_tags      = (known after apply)
      }

    + network_interface {
        + delete_on_termination = (known after apply)
        + device_index          = (known after apply)
        + network_card_index    = (known after apply)
        + network_interface_id  = (known after apply)
      }

    + private_dns_name_options {
        + enable_resource_name_dns_a_record    = (known after apply)
        + enable_resource_name_dns_aaaa_record = (known after apply)
```

```
        + hostname_type                        = (known after apply)
      }

    + root_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }
  }

# aws_instance.app_server2 will be created
+ resource "aws_instance" "app_server2" {
    + ami                                  = "ami-830c94e3"
    + arn                                  = (known after apply)
    + associate_public_ip_address          = (known after apply)
    + availability_zone                    = (known after apply)
    + cpu_core_count                       = (known after apply)
    + cpu_threads_per_core                 = (known after apply)
    + disable_api_stop                     = (known after apply)
    + disable_api_termination              = (known after apply)
    + ebs_optimized                        = (known after apply)
    + get_password_data                    = false
    + host_id                              = (known after apply)
    + host_resource_group_arn              = (known after apply)
    + iam_instance_profile                 = (known after apply)
    + id                                   = (known after apply)
    + instance_initiated_shutdown_behavior = (known after apply)
    + instance_state                       = (known after apply)
    + instance_type                        = "t2.micro"
    + ipv6_address_count                   = (known after apply)
    + ipv6_addresses                       = (known after apply)
    + key_name                             = "id_rsa"
    + monitoring                           = (known after apply)
    + outpost_arn                          = (known after apply)
    + password_data                        = (known after apply)
    + placement_group                      = (known after apply)
```

Liite 4 3(10). Kolmen virtuaalikoneen luominen

```
    + placement_partition_number           = (known after apply)
    + primary_network_interface_id         = (known after apply)
    + private_dns                          = (known after apply)
    + private_ip                           = (known after apply)
    + public_dns                           = (known after apply)
    + public_ip                            = (known after apply)
    + secondary_private_ips                = (known after apply)
    + security_groups                      = (known after apply)
    + source_dest_check                    = true
    + subnet_id                            = (known after apply)
    + tags                                 = {
        + "Name" = "Testipalvelin2"
      }
    + tags_all                             = {
        + "Name" = "Testipalvelin2"
      }
    + tenancy                              = (known after apply)
    + user_data                            = (known after apply)
    + user_data_base64                     = (known after apply)
    + user_data_replace_on_change          = false
    + vpc_security_group_ids               = (known after apply)

    + capacity_reservation_specification {
        + capacity_reservation_preference = (known after apply)

        + capacity_reservation_target {
            + capacity_reservation_id                = (known after apply)
            + capacity_reservation_resource_group_arn = (known after apply)
          }
      }

    + ebs_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
```

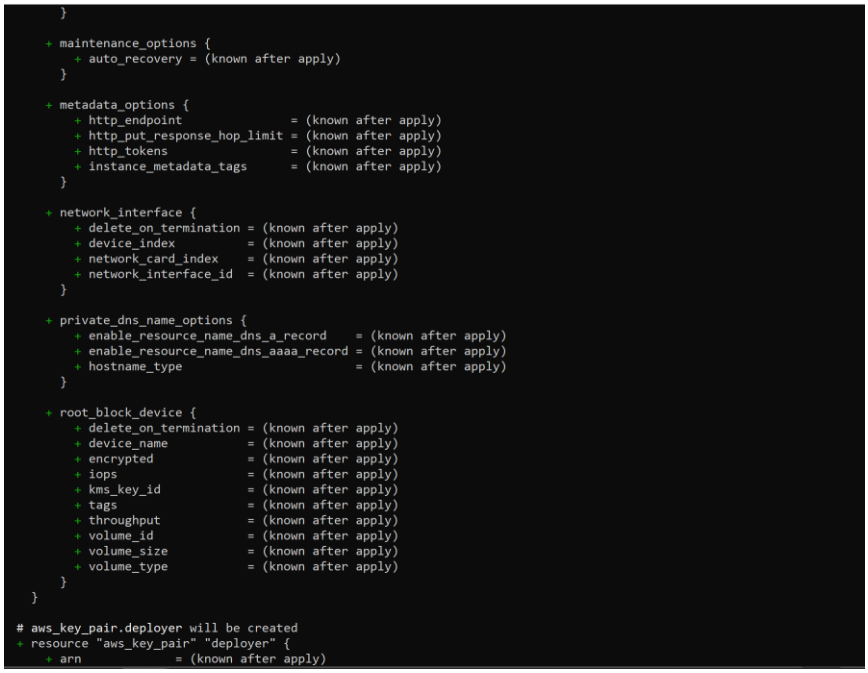
```
      }

    + enclave_options {
        + enabled = (known after apply)
      }

    + ephemeral_block_device {
        + device_name  = (known after apply)
        + no_device    = (known after apply)
        + virtual_name = (known after apply)
      }

    + maintenance_options {
        + auto_recovery = (known after apply)
      }

    + metadata_options {
        + http_endpoint               = (known after apply)
        + http_put_response_hop_limit = (known after apply)
        + http_tokens                 = (known after apply)
        + instance_metadata_tags      = (known after apply)
      }

    + network_interface {
        + delete_on_termination = (known after apply)
        + device_index          = (known after apply)
        + network_card_index    = (known after apply)
        + network_interface_id  = (known after apply)
      }

    + private_dns_name_options {
        + enable_resource_name_dns_a_record    = (known after apply)
        + enable_resource_name_dns_aaaa_record = (known after apply)
        + hostname_type                        = (known after apply)
      }

    + root_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags                  = (known after apply)
```

Liite 4 4(10). Kolmen virtuaalikoneen luominen

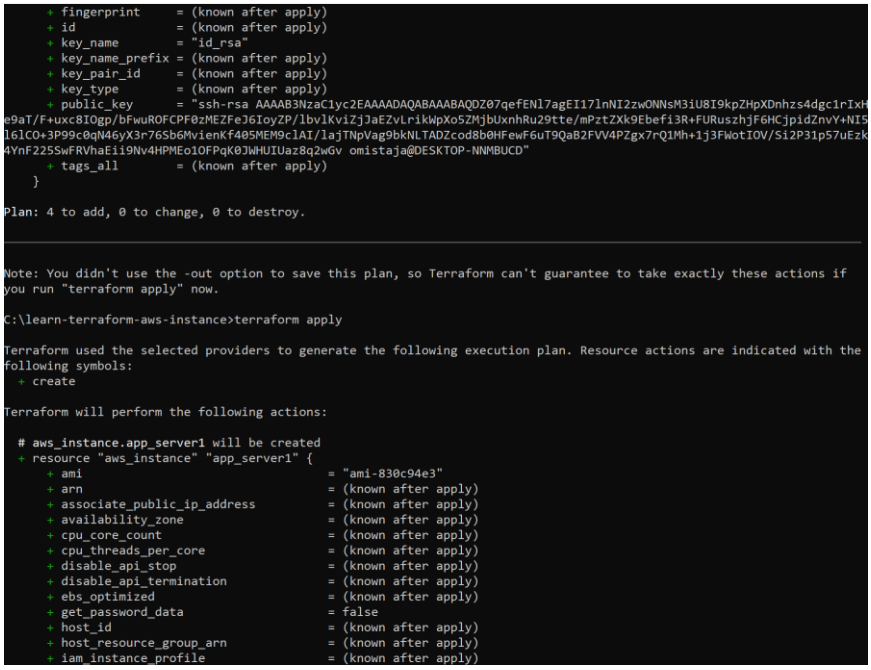
```
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }
  }

# aws_instance.app_server3 will be created
+ resource "aws_instance" "app_server3" {
    + ami                                  = "ami-830c94e3"
    + arn                                  = (known after apply)
    + associate_public_ip_address          = (known after apply)
    + availability_zone                    = (known after apply)
    + cpu_core_count                       = (known after apply)
    + cpu_threads_per_core                 = (known after apply)
    + disable_api_stop                     = (known after apply)
    + disable_api_termination              = (known after apply)
    + ebs_optimized                        = (known after apply)
    + get_password_data                    = false
    + host_id                              = (known after apply)
    + host_resource_group_arn              = (known after apply)
    + iam_instance_profile                 = (known after apply)
    + id                                   = (known after apply)
    + instance_initiated_shutdown_behavior = (known after apply)
    + instance_state                       = (known after apply)
    + instance_type                        = "t2.micro"
    + ipv6_address_count                   = (known after apply)
    + ipv6_addresses                       = (known after apply)
    + key_name                             = "id_rsa"
    + monitoring                           = (known after apply)
    + outpost_arn                          = (known after apply)
    + password_data                        = (known after apply)
    + placement_group                      = (known after apply)
    + placement_partition_number           = (known after apply)
    + primary_network_interface_id         = (known after apply)
    + private_dns                          = (known after apply)
    + private_ip                           = (known after apply)
    + public_dns                           = (known after apply)
    + public_ip                            = (known after apply)
    + secondary_private_ips                = (known after apply)
    + security_groups                      = (known after apply)
    + source_dest_check                    = true
    + subnet_id                            = (known after apply)
```

```
    + tags                                 = {
        + "Name" = "Testipalvelin3"
      }
    + tags_all                             = {
        + "Name" = "Testipalvelin3"
      }
    + tenancy                              = (known after apply)
    + user_data                            = (known after apply)
    + user_data_base64                     = (known after apply)
    + user_data_replace_on_change          = false
    + vpc_security_group_ids               = (known after apply)

    + capacity_reservation_specification {
        + capacity_reservation_preference = (known after apply)

        + capacity_reservation_target {
            + capacity_reservation_id                = (known after apply)
            + capacity_reservation_resource_group_arn = (known after apply)
          }
      }

    + ebs_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }

    + enclave_options {
        + enabled = (known after apply)
      }

    + ephemeral_block_device {
        + device_name  = (known after apply)
        + no_device    = (known after apply)
        + virtual_name = (known after apply)
```

Liite 4 5(10). Kolmen virtuaalikoneen luominen

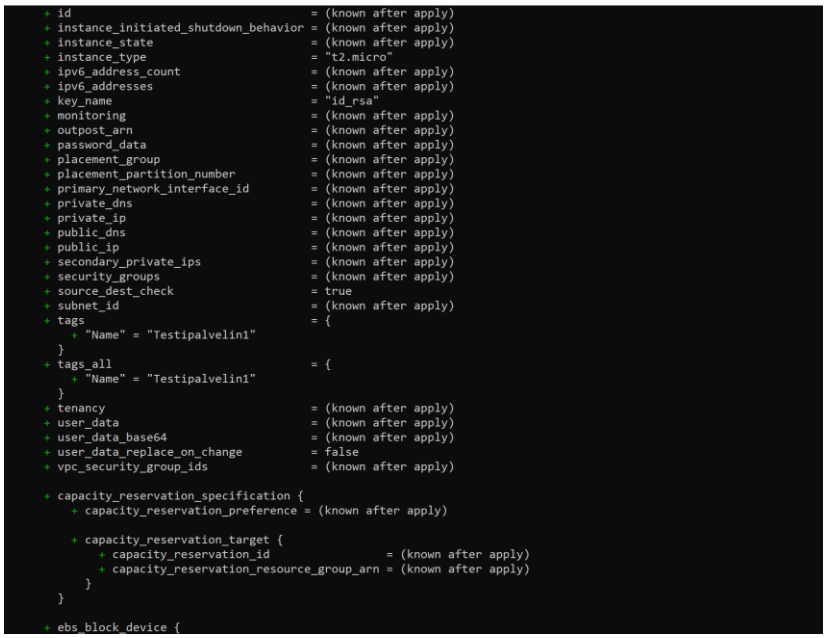


```
      }

    + maintenance_options {
        + auto_recovery = (known after apply)
      }

    + metadata_options {
        + http_endpoint               = (known after apply)
        + http_put_response_hop_limit = (known after apply)
        + http_tokens                 = (known after apply)
        + instance_metadata_tags      = (known after apply)
      }

    + network_interface {
        + delete_on_termination = (known after apply)
        + device_index          = (known after apply)
        + network_card_index    = (known after apply)
        + network_interface_id  = (known after apply)
      }

    + private_dns_name_options {
        + enable_resource_name_dns_a_record    = (known after apply)
        + enable_resource_name_dns_aaaa_record = (known after apply)
        + hostname_type                        = (known after apply)
      }

    + root_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }
  }

  # aws_key_pair.deployer will be created
  + resource "aws_key_pair" "deployer" {
      + arn             = (known after apply)
```

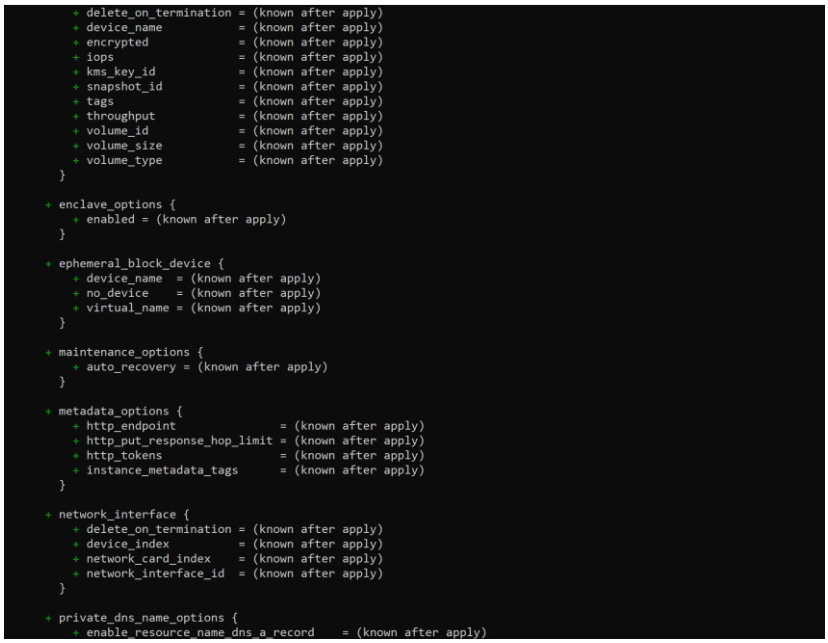


```
      + fingerprint     = (known after apply)
      + id              = (known after apply)
      + key_name        = "id_rsa"
      + key_name_prefix = (known after apply)
      + key_pair_id     = (known after apply)
      + key_type        = (known after apply)
      + public_key      = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDZ07qefENl7agEI17lnNI2zwONNsM3iU8I9kpZHpXDnhzs4dgc1rIxH
e9aT/F+uxc8IOgp/bFwuROFCPF0zMEZFeJ6IoyZP/lbvlkviZjJaEZvLrikWpXo5ZMjbUxnhRu29tte/mPztZXk9Ebefi3R+FURuszhjF6HCjpidZnvY+NI5
l6lCO+3P99c0qN46yX3r76Sb6MvienKf405MEM9clAI/lajTNpVag9bkNLTADZcod8b0HFewF6uT9QaB2FVV4PZgx7rQ1Mh+1j3FWotIOV/Si2P31p57uEzk
4YnF225SwFRVhaEii9Nv4HPMEo1OFPqK0JWHUIUaz8q2wGv omistaja@DESKTOP-NNMBUCD"
      + tags_all        = (known after apply)
    }

Plan: 4 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if
you run "terraform apply" now.

C:\learn-terraform-aws-instance>terraform apply

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:

  # aws_instance.app_server1 will be created
  + resource "aws_instance" "app_server1" {
      + ami                                  = "ami-830c94e3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
```

Liite 4 6(10). Kolmen virtuaalikoneen luominen



```
+ id                                   = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_state                       = (known after apply)
+ instance_type                        = "t2.micro"
+ ipv6_address_count                   = (known after apply)
+ ipv6_addresses                       = (known after apply)
+ key_name                             = "id_rsa"
+ monitoring                           = (known after apply)
+ outpost_arn                          = (known after apply)
+ password_data                        = (known after apply)
+ placement_group                      = (known after apply)
+ placement_partition_number           = (known after apply)
+ primary_network_interface_id         = (known after apply)
+ private_dns                          = (known after apply)
+ private_ip                           = (known after apply)
+ public_dns                           = (known after apply)
+ public_ip                            = (known after apply)
+ secondary_private_ips                = (known after apply)
+ security_groups                      = (known after apply)
+ source_dest_check                    = true
+ subnet_id                            = (known after apply)
+ tags                                 = {
    + "Name" = "Testipalvelin1"
  }
+ tags_all                             = {
    + "Name" = "Testipalvelin1"
  }
+ tenancy                              = (known after apply)
+ user_data                            = (known after apply)
+ user_data_base64                     = (known after apply)
+ user_data_replace_on_change          = false
+ vpc_security_group_ids               = (known after apply)

+ capacity_reservation_specification {
    + capacity_reservation_preference = (known after apply)

    + capacity_reservation_target {
        + capacity_reservation_id                = (known after apply)
        + capacity_reservation_resource_group_arn = (known after apply)
      }
  }

+ ebs_block_device {
```

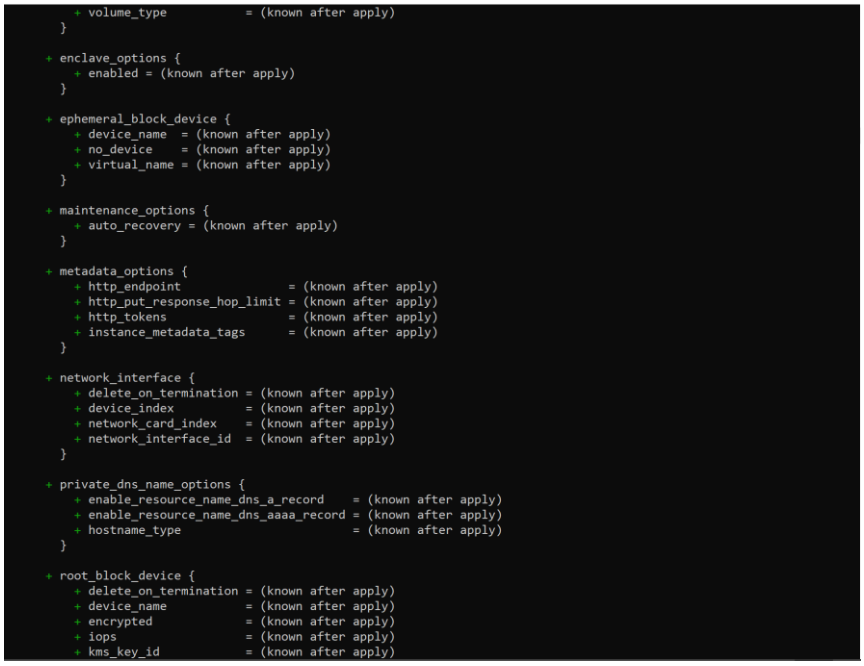


```
    + delete_on_termination = (known after apply)
    + device_name           = (known after apply)
    + encrypted             = (known after apply)
    + iops                  = (known after apply)
    + kms_key_id            = (known after apply)
    + snapshot_id           = (known after apply)
    + tags                  = (known after apply)
    + throughput            = (known after apply)
    + volume_id             = (known after apply)
    + volume_size           = (known after apply)
    + volume_type           = (known after apply)
  }

+ enclave_options {
    + enabled = (known after apply)
  }

+ ephemeral_block_device {
    + device_name  = (known after apply)
    + no_device    = (known after apply)
    + virtual_name = (known after apply)
  }

+ maintenance_options {
    + auto_recovery = (known after apply)
  }

+ metadata_options {
    + http_endpoint               = (known after apply)
    + http_put_response_hop_limit = (known after apply)
    + http_tokens                 = (known after apply)
    + instance_metadata_tags      = (known after apply)
  }

+ network_interface {
    + delete_on_termination = (known after apply)
    + device_index          = (known after apply)
    + network_card_index    = (known after apply)
    + network_interface_id  = (known after apply)
  }

+ private_dns_name_options {
    + enable_resource_name_dns_a_record    = (known after apply)
```

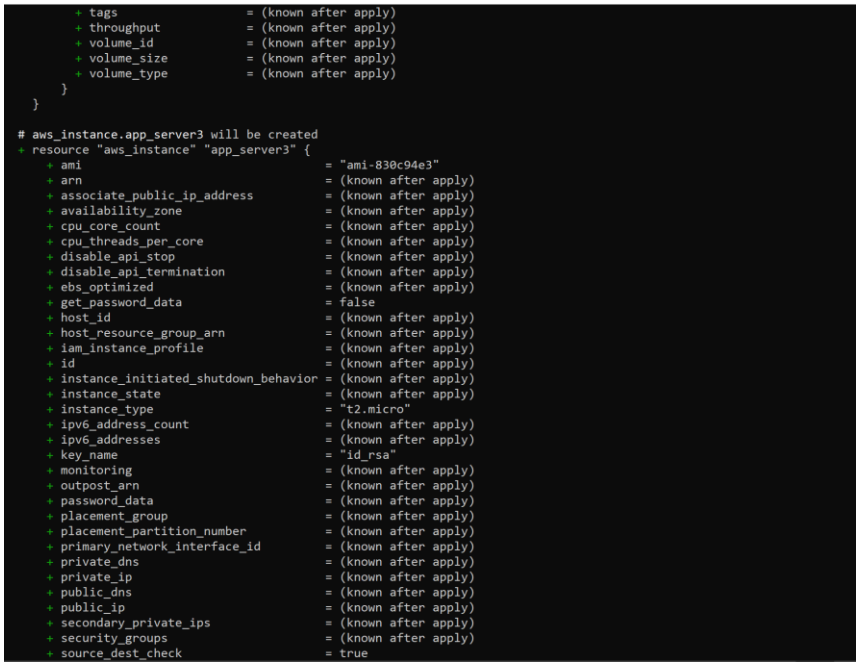
Liite 4 7(10). Kolmen virtuaalikoneen luominen

```
        + enable_resource_name_dns_aaaa_record = (known after apply)
        + hostname_type                        = (known after apply)
      }

    + root_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }
  }

# aws_instance.app_server2 will be created
+ resource "aws_instance" "app_server2" {
    + ami                                  = "ami-830c94e3"
    + arn                                  = (known after apply)
    + associate_public_ip_address          = (known after apply)
    + availability_zone                    = (known after apply)
    + cpu_core_count                       = (known after apply)
    + cpu_threads_per_core                 = (known after apply)
    + disable_api_stop                     = (known after apply)
    + disable_api_termination              = (known after apply)
    + ebs_optimized                        = (known after apply)
    + get_password_data                    = false
    + host_id                              = (known after apply)
    + host_resource_group_arn              = (known after apply)
    + iam_instance_profile                 = (known after apply)
    + id                                   = (known after apply)
    + instance_initiated_shutdown_behavior = (known after apply)
    + instance_state                       = (known after apply)
    + instance_type                        = "t2.micro"
    + ipv6_address_count                   = (known after apply)
    + ipv6_addresses                       = (known after apply)
    + key_name                             = "id_rsa"
    + monitoring                           = (known after apply)
    + outpost_arn                          = (known after apply)
    + password_data                        = (known after apply)
    + placement_group                      = (known after apply)
    + placement_partition_number           = (known after apply)
    + primary_network_interface_id         = (known after apply)
    + private_dns                          = (known after apply)
    + private_ip                           = (known after apply)
    + public_dns                           = (known after apply)
    + public_ip                            = (known after apply)
    + secondary_private_ips                = (known after apply)
    + security_groups                      = (known after apply)
    + source_dest_check                    = true
    + subnet_id                            = (known after apply)
    + tags                                 = {
        + "Name" = "Testipalvelin2"
      }
    + tags_all                             = {
        + "Name" = "Testipalvelin2"
      }
    + tenancy                              = (known after apply)
    + user_data                            = (known after apply)
    + user_data_base64                     = (known after apply)
    + user_data_replace_on_change          = false
    + vpc_security_group_ids               = (known after apply)

    + capacity_reservation_specification {
        + capacity_reservation_preference = (known after apply)

        + capacity_reservation_target {
            + capacity_reservation_id                = (known after apply)
            + capacity_reservation_resource_group_arn = (known after apply)
          }
      }

    + ebs_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
```

Liite 4 8(10). Kolmen virtuaalikoneen luominen



```
      + volume_type           = (known after apply)
    }

  + enclave_options {
      + enabled = (known after apply)
    }

  + ephemeral_block_device {
      + device_name  = (known after apply)
      + no_device    = (known after apply)
      + virtual_name = (known after apply)
    }

  + maintenance_options {
      + auto_recovery = (known after apply)
    }

  + metadata_options {
      + http_endpoint               = (known after apply)
      + http_put_response_hop_limit = (known after apply)
      + http_tokens                 = (known after apply)
      + instance_metadata_tags      = (known after apply)
    }

  + network_interface {
      + delete_on_termination = (known after apply)
      + device_index          = (known after apply)
      + network_card_index    = (known after apply)
      + network_interface_id  = (known after apply)
    }

  + private_dns_name_options {
      + enable_resource_name_dns_a_record    = (known after apply)
      + enable_resource_name_dns_aaaa_record = (known after apply)
      + hostname_type                        = (known after apply)
    }

  + root_block_device {
      + delete_on_termination = (known after apply)
      + device_name           = (known after apply)
      + encrypted             = (known after apply)
      + iops                  = (known after apply)
      + kms_key_id            = (known after apply)
```

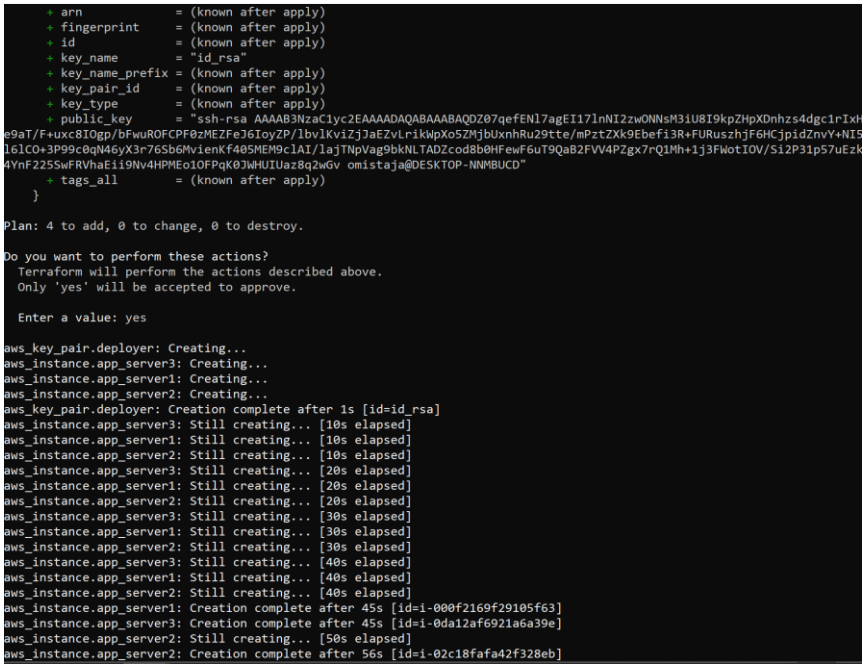


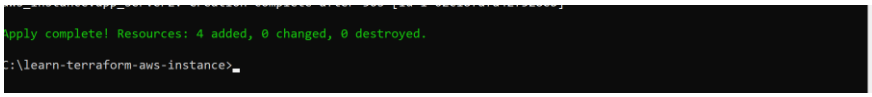
```
      + tags                  = (known after apply)
      + throughput            = (known after apply)
      + volume_id             = (known after apply)
      + volume_size           = (known after apply)
      + volume_type           = (known after apply)
    }
}

# aws_instance.app_server3 will be created
+ resource "aws_instance" "app_server3" {
    + ami                                  = "ami-830c94e3"
    + arn                                  = (known after apply)
    + associate_public_ip_address          = (known after apply)
    + availability_zone                    = (known after apply)
    + cpu_core_count                       = (known after apply)
    + cpu_threads_per_core                 = (known after apply)
    + disable_api_stop                     = (known after apply)
    + disable_api_termination              = (known after apply)
    + ebs_optimized                        = (known after apply)
    + get_password_data                    = false
    + host_id                              = (known after apply)
    + host_resource_group_arn              = (known after apply)
    + iam_instance_profile                 = (known after apply)
    + id                                   = (known after apply)
    + instance_initiated_shutdown_behavior = (known after apply)
    + instance_state                       = (known after apply)
    + instance_type                        = "t2.micro"
    + ipv6_address_count                   = (known after apply)
    + ipv6_addresses                       = (known after apply)
    + key_name                             = "id_rsa"
    + monitoring                           = (known after apply)
    + outpost_arn                          = (known after apply)
    + password_data                        = (known after apply)
    + placement_group                      = (known after apply)
    + placement_partition_number           = (known after apply)
    + primary_network_interface_id         = (known after apply)
    + private_dns                          = (known after apply)
    + private_ip                           = (known after apply)
    + public_dns                           = (known after apply)
    + public_ip                            = (known after apply)
    + secondary_private_ips                = (known after apply)
    + security_groups                      = (known after apply)
    + source_dest_check                    = true
```

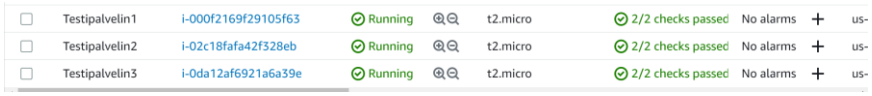
Liite 4 9(10). Kolmen virtuaalikoneen luominen

```
    + subnet_id                            = (known after apply)
    + tags                                 = {
        + "Name" = "Testipalvelin3"
      }
    + tags_all                             = {
        + "Name" = "Testipalvelin3"
      }
    + tenancy                              = (known after apply)
    + user_data                            = (known after apply)
    + user_data_base64                     = (known after apply)
    + user_data_replace_on_change          = false
    + vpc_security_group_ids               = (known after apply)

    + capacity_reservation_specification {
        + capacity_reservation_preference = (known after apply)

        + capacity_reservation_target {
            + capacity_reservation_id                = (known after apply)
            + capacity_reservation_resource_group_arn = (known after apply)
          }
      }

    + ebs_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }

    + enclave_options {
        + enabled = (known after apply)
      }

    + ephemeral_block_device {
        + device_name  = (known after apply)
        + no_device    = (known after apply)
        + virtual_name = (known after apply)
      }

    + maintenance_options {
        + auto_recovery = (known after apply)
      }

    + metadata_options {
        + http_endpoint               = (known after apply)
        + http_put_response_hop_limit = (known after apply)
        + http_tokens                 = (known after apply)
        + instance_metadata_tags      = (known after apply)
      }

    + network_interface {
        + delete_on_termination = (known after apply)
        + device_index          = (known after apply)
        + network_card_index    = (known after apply)
        + network_interface_id  = (known after apply)
      }

    + private_dns_name_options {
        + enable_resource_name_dns_a_record    = (known after apply)
        + enable_resource_name_dns_aaaa_record = (known after apply)
        + hostname_type                        = (known after apply)
      }

    + root_block_device {
        + delete_on_termination = (known after apply)
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags                  = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = (known after apply)
        + volume_type           = (known after apply)
      }
  }

# aws_key_pair.deployer will be created
+ resource "aws_key_pair" "deployer" {
```

Liite 4 10(10). Kolmen virtuaalikoneen luominen







Liite 5 1(2). *Vagrant up* -komento

Tässä liitteessä nähdään *vagrant up* -komennon tulostus kokonaisuudessaan.

```
c:\vagrant>vagrant up
Bringing machine 'Testipalvelin1' up with 'virtualbox' provider...
Bringing machine 'Testipalvelin2' up with 'virtualbox' provider...
Bringing machine 'Testipalvelin3' up with 'virtualbox' provider...
==> Testipalvelin1: Importing base box 'ubuntu/focal64'...
==> Testipalvelin1: Matching MAC address for NAT networking...
==> Testipalvelin1: Checking if box 'ubuntu/focal64' version '20230128.0.0' is up to date...
==> Testipalvelin1: Setting the name of the VM: vagrant_Testipalvelin1_1675098953393_94301
==> Testipalvelin1: Clearing any previously set network interfaces...
==> Testipalvelin1: Preparing network interfaces based on configuration...
    Testipalvelin1: Adapter 1: nat
==> Testipalvelin1: Forwarding ports...
    Testipalvelin1: 22 (guest) => 2222 (host) (adapter 1)
==> Testipalvelin1: Running 'pre-boot' VM customizations...
==> Testipalvelin1: Booting VM...
==> Testipalvelin1: Waiting for machine to boot. This may take a few minutes...
    Testipalvelin1: SSH address: 127.0.0.1:2222
    Testipalvelin1: SSH username: vagrant
    Testipalvelin1: SSH auth method: private key
    Testipalvelin1: Warning: Connection reset. Retrying...
    Testipalvelin1: Warning: Connection aborted. Retrying...
    Testipalvelin1:
    Testipalvelin1: Vagrant insecure key detected. Vagrant will automatically replace
    Testipalvelin1: this with a newly generated keypair for better security.
    Testipalvelin1:
    Testipalvelin1: Inserting generated public key within guest...
    Testipalvelin1: Removing insecure key from the guest if it's present...
    Testipalvelin1: Key inserted! Disconnecting and reconnecting using new SSH key...
==> Testipalvelin1: Machine booted and ready!
==> Testipalvelin1: Checking for guest additions in VM...
    Testipalvelin1: The guest additions on this VM do not match the installed version of
    Testipalvelin1: VirtualBox! In most cases this is fine, but in rare cases it can
    Testipalvelin1: prevent things such as shared folders from working properly. If you see
    Testipalvelin1: shared folder errors, please make sure the guest additions within the
    Testipalvelin1: virtual machine match the version of VirtualBox you have installed on
    Testipalvelin1: your host and reload your VM.
    Testipalvelin1:
    Testipalvelin1: Guest Additions Version: 6.1.38
    Testipalvelin1: VirtualBox Version: 7.0
==> Testipalvelin1: Mounting shared folders...
    Testipalvelin1: /vagrant => C:/vagrant
```

```
==> Testipalvelin2: Matching MAC address for NAT networking...
==> Testipalvelin2: Checking if box 'ubuntu/focal64' version '20230128.0.0' is up to date...
==> Testipalvelin2: Setting the name of the VM: vagrant_Testipalvelin2_1675099025883_73250
==> Testipalvelin2: Fixed port collision for 22 => 2222. Now on port 2200.
==> Testipalvelin2: Clearing any previously set network interfaces...
==> Testipalvelin2: Preparing network interfaces based on configuration...
    Testipalvelin2: Adapter 1: nat
==> Testipalvelin2: Forwarding ports...
    Testipalvelin2: 22 (guest) => 2200 (host) (adapter 1)
==> Testipalvelin2: Running 'pre-boot' VM customizations...
==> Testipalvelin2: Booting VM...
==> Testipalvelin2: Waiting for machine to boot. This may take a few minutes...
    Testipalvelin2: SSH address: 127.0.0.1:2200
    Testipalvelin2: SSH username: vagrant
    Testipalvelin2: SSH auth method: private key
    Testipalvelin2: Warning: Connection reset. Retrying...
    Testipalvelin2: Warning: Connection aborted. Retrying...
    Testipalvelin2:
    Testipalvelin2: Vagrant insecure key detected. Vagrant will automatically replace
    Testipalvelin2: this with a newly generated keypair for better security.
    Testipalvelin2:
    Testipalvelin2: Inserting generated public key within guest...
    Testipalvelin2: Removing insecure key from the guest if it's present...
    Testipalvelin2: Key inserted! Disconnecting and reconnecting using new SSH key...
==> Testipalvelin2: Machine booted and ready!
==> Testipalvelin2: Checking for guest additions in VM...
    Testipalvelin2: The guest additions on this VM do not match the installed version of
    Testipalvelin2: VirtualBox! In most cases this is fine, but in rare cases it can
    Testipalvelin2: prevent things such as shared folders from working properly. If you see
    Testipalvelin2: shared folder errors, please make sure the guest additions within the
    Testipalvelin2: virtual machine match the version of VirtualBox you have installed on
    Testipalvelin2: your host and reload your VM.
    Testipalvelin2:
    Testipalvelin2: Guest Additions Version: 6.1.38
    Testipalvelin2: VirtualBox Version: 7.0
==> Testipalvelin2: Mounting shared folders...
    Testipalvelin2: /vagrant => C:/vagrant
```

Liite 5 2(2). *Vagrant up* -komento