Konttiorkestroinnin hyödyntäminen sovellustoimittajan käyttöpalveluissa

Case Triplan Oy

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
kevät, 2021

Karri Lasturanta

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Karri Lasturanta Vuosi 2021

Työn nimi Konttiorkestroinnin hyödyntäminen sovellustoimittajan käyttöpalveluissa
Case Triplan Oy

Ohjaajat Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli selvittää, miten IaC-tyylillä otetaan Kubernetes-klusteri käyttöön vSphere-ympäristössä sekä vertailla muutamaa salaisuuksienhallintajärjestelmää ja levypalvelua. Työtä tehtiin ottaen huomioon yrityksen lähiajan sekä tulevaisuuden tarve. Lopullinen tarve on hyödyntää Kubernetes-klusteria asiakkaille vietävien käyttöpalveluiden tarjoamisessa. Opinnäytetyön toimeksiantaja on yritys Triplan Oy, joka tarjoaa julkishallinnon käyttöön digitaalisia työkaluja tiedon tuottamiseen ja hallintaan.

Opinnäytetyön tietopohja muodostui eri teknologioiden teoriasta. Aluksi selvitettiin konttiteknologiaa ja siihen liittyviä teknologioita, kuten Kubernetestä ja Dockeria. Sitten keskityttiin IaC-teknologiaan, salaisuuksiin, vSphereen ja levypalveluihin. Tarkasteltiin sekä perustietoa jokaisesta aiheesta että syvällisempää tärkeää tietoa muutamista teknologioista ja tuotteista.

Tuloksissa havaittiin Terraformin soveltuvuus Kubernetes-klusterin käyttöönottoon vSphere-ympäristössä sekä saatiin faktoja sekä vertailutietoja salaisuuksienhallintajärjestelmä- ja levypalveluvaihtoehdoista. Todennettiin tiettyjen vaihtoehtojen sopivan yrityksen käyttöön paremmin kuin toisten ja mahdollistettiin päätösten teko seuraavia vaiheita varten Kubernetes-klusterin käyttöönotossa.

Avainsanat Infrastruktuuri koodina, Kubernetes, salaisuuksienhallinta, levypalvelut, Terraform

Sivut 57 sivua ja liitteitä 15 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology **Abstract**

Hämeenlinna University Centre

Author	Karri Lasturanta	Year 2021
Subject	Utilization of container orchestration in application vendor's operating services Case Triplan Oy	
Supervisors	Lasse Seppänen	

ABSTRACT

The purpose of the thesis was to test out how to deploy Kubernetes cluster in vSphere environment using IaC methods and to compare few secret management systems and persistent volume storages. The work was done considering the needs of the company. The final need was to utilize Kubernetes cluster at providing operating services to customers. The client of the thesis was Triplan Oy, which provides public administrations with digital tools for producing and managing information.

The thesis was partly practical and partly theoretical. The knowledge base of the thesis consisted of different technologies and products. The researched technologies consisted of container technologies, secret management technologies, storage technologies and IaC technologies. The researched products consisted of few examples of secret management systems and storage systems that are compatible with the Kubernetes cluster.

The examples created indicate that Terraform suites the requirements of the deployment of the Kubernetes cluster in the vSphere environment. The results also consist of comparison data regarding the secret management systems and storage systems. The certain options were found to be better than others for the needs of the company. The results allow for further decision-making and development of deployment of the Kubernetes cluster.

Keywords Infrastructure as code, Kubernetes, secret management, persistent storage systems, Terraform

Pages 57 pages and appendices 15 pages

Sanasto

REST	HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.
API	Ohjelmointirajapinta, jonka avulla eri ohjelmat voivat keskustella.
JSON	Ihmisen luettava tiedostomuoto tiedonvälitykseen.
IaC	Infrastructure as Code, koodimuotoinen infrastruktuuri.
SaaS	Software as a Service, ohjelma palveluna.
.sh	Komento-skriptin tiedostopääte.
SDN	Software Defined Networking.
DevOps	Toimintamalli sähköisten palvelujen tuotantoon, joka pyrkii automatisoimaan kehitykseen, testaamiseen ja ylläpitoon liittyvät IT-palvelutoiminnot.
CSI	Container Storage Interface
SSH	Secure Shell, salattuun tietoliikenteeseen tarkoitettu protokolla
RPC	Remote Procedure Call
UID	User ID, käyttäjän id-tunniste
GID	Group ID, ryhmän id-tunniste
Backend	Ohjelman taustalla toimiva taso, joka hoitaa esimerkiksi laskennan, tietokannan ja koodin ajon
CLI	Command Line Interface eli komentoliittymä
UI	User Interface eli käyttöliittymä

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Konttiteknologia ja Kubernetes tulevat olemaan yhä tärkeämmässä ja suuremmassa roolissa niin suurissa kuin pienemmissäkin yrityksissä sekä varsinkin teknologia-alalla, sillä ne esimerkiksi tehostavat IT-resurssien ylläpitoa ja hallintaa. Konttiteknologia keskittyy virtuaalisiin kontteihin, jotka sisältävät eristetyn eli kontitetun ohjelman tai ohjelman osan ja sen vaatimat resurssit. Sovelluskonttien suuren määrän hallinta saattaa tulla vaikeaksi, minkä vuoksi on kehitetty orkestrointiohjelmat. Kuberneteksen suosio nousee jatkuvasti sen monipuolisten käyttömahdollisuuksien vuoksi.

Triplan Oy on aikeissa siirtää aiemmin luodun Docker Swarm -ohjelman hallinnoimat kontit Kubernetes alustalle. Opinnäytetyön tarkoitus onkin tutkia Kuberneteksen käyttöönottoa Triplan Oy:n tarpeiden mukaisesti. Aihe on erittäin laaja ja tutkittavia asioita on paljon, joten opinnäytetyössä keskitytään IaC-periaatteisiin käyttöönotossa, levypalveluihin sekä salaisuuksien hallintaan.

Teen opinnäytetyötä yritykselle, jossa myös työskentelen. Aiemmin oppimani Dockeriin sekä konttikuvien tekemiseen liittyvät taidot ja tiedot antavat hyvän pohjan Kuberneteksen opettelulle ja opinnäytetyön tekemiselle. Opinnäytetyön tekeminen edistää osaamistani, jota tulen tarvitsemaan työssäni. Triplanin tulevaisuuden on tarkoitus olla hyvinkin konttipohjainen. Kubernetes-projekti jatkuu vielä opinnäytetyön tekemisen jälkeen. Opinnäytetyötä tehdessä tulee myös tutustuttua moneen muuhun minulle uuteen teknologiaan, kuten Terraformiin ja levypalveluihin.

Tutkimuskysymykset ovat

- Miten voitaisiin käyttää Infrastructure as Code -periaatteita hyödyntäen VMware-ympäristön API-rajapintaa?
- Millä eri keinoin Kubernetes-ympäristö tarjoaa levypalveluita podeille/konteille?
- Mitä vaihtoehtoja salaisuuksien hallintaan Kuberneteksessä on?

2 Konttienhallintajärjestelmän sekä aputeknologioiden tietoperusta

Konttiteknologia nousee jatkuvasti suurempaan rooliin niin yritysten kuin myös yksityisten henkilöidenkin käytössä. Perinteisesti on käytetty suurikokoisia servereitä yksinkertaistenkin palveluiden ja testausten ylläpitoon. Servereiden päivittäminen, ylläpito ja käyttöönotto vaativat paljon aikaa ja työtä. Nykyään siirrytään kohti pienempään osaan jaettuja mikropalveluja. Mikropalveluja voidaan käyttöönottaa, päivittää, kehittää ja skaalata itsenäisesti, mikä auttaa komponenttien nopeassa vaihdossa. Säästetty aika ja säästetyt resurssit ovat yrityksille tärkeitä. (Luksa, 2018)

Tarvittavien komponenttien ja yhä suurempien datakeskusten määrä tuo ongelmia myös mikropalveluille. Vastauksena ongelmaan on konttienhallintatyökalut ja järjestelmät. Kubernetes on yksi konttienhallintajärjestelmä. (Luksa, 2018) Konttienhallintajärjestelmää käytettäessä ja käyttöönotettaessa tulee ottaa huomioon useita asioita, joihin kuuluvat esimerkiksi salaisuuksien hallinta, klusterin käyttöönotto ja pysyvät voluumit sekä levypalvelut.

2.1 Konttiteknologia

Konttiteknologia perustuu kontteihin ja niiden hallinnointiin. Kontti on ajettava ohjelmistoyksikkö, johon ohjelman koodi on paketoitu tarvittavien kirjastojen ja muiden vaatimusten, kuten lisäosien, kanssa. Kontit hyödyntävät käyttöjärjestelmän virtualisoinnin muotoa, jonka avulla useat ohjelmat voivat jakaa käyttöjärjestelmän sekä eristämällä prosessit että hallitsemalla prosessoreiden pääsyä suorittimien, muistin ja levyn määrään. (IBM Cloud Education, 2019)

Kontit mahdollistavat ohjelmille standardoidun ympäristön. Kontit ratkaisevat ongelman, jonka virtuaalikoneiden käyttö tuo esille. Ohjelman ajaminen vaatii paljon muutakin kuin pelkästään ohjelmoijan luoman koodin. Koodi tarvitsee erilaisia kirjastoja, lisäosia sekä asetuksia. (Wallenius, 2019)

Virtuaalipalvelimia voidaan käyttää ohjelman pyörittämiseen, sillä niihin saadaan lisättyä kaikki tarpeellinen. Virtuaalipalvelin sisältää kuitenkin myös paljon muutakin kuin vain

tarpeellisen. Hyvin suunnitellun kontin kuva voi olla jopa 100 kertaa pienempi, sillä se ei sisällä ohjelmia, kirjastoja ja lisäosia, joita ei tarvita. Virtuaalipalvelin sisältää kokonaisen käyttöjärjestelmän, joka vie resursseja, joita voitaisiin käyttää muuhun. Virtuaalipalvelimia siirrettäessä siirretään siis paljon tarpeetonta tietoa. Konttia siirrettäessä siirretään vain tarpeellinen tieto. Konttia käytettäessä ohjelma vaatii siis huomattavasti vähemmän resursseja, ja siirto sekä käynnistys tapahtuu huomattavasti nopeammin. (Wallenius, 2019)

Resurssien säästämisen ja vain oleellisen tiedon siirtämisen mahdollistamisen lisäksi tärkeä konttien etu ovat standardoidut ympäristöt. Standardoidussa ympäristössä kehitys-, testaus ja tuotantoympäristöt voivat olla samanlaisia. Perinteisissä ympäristöissä, kuten virtuaalipalvelimilla, ympäristöjen erilaisuus tuottaa useita ongelmia. Esimerkiksi kehitysympäristö ja tuotantoympäristö useimmiten eroavat toisistaan, ja sen vuoksi useat ongelmat selviävät vasta tuotannossa. Ongelmia voivat tuottaa esimerkiksi Pythonin eri versiot, erilaiset turva-asetukset, eri käyttöjärjestelmäversio tai erilainen verkkotopologia. Standardoidun ympäristön ansiosta kontteja on virtuaalikonetta helpompi liikuttaa konesalista toiseen. (Wallenius, 2019)

Yleensä yksi kontti sisältää ohjelman tietyn ominaisuuden. Kokonainen ohjelma koostuu siis useasta kontista. Ohjelman jakautuminen useaan konttiin mahdollistaa ohjelman päivittämisen paloissa. Pala palalta päivittäessä on aiempaan ympäristöön helppo palata ongelmatilanteen ilmaantuessa. Mikäli vain yksi osa päivitettiin, voidaan aiempaan tilaan palata palauttamalla vain tämä yksi osa. Ohjelman palauttaminen on tämän ominaisuuden ansiota helpompaa, kun ei tarvitse palauttaa kokonaista ohjelmaa. (Wallenius, 2019)

Konttikuva on paketti, joka sisältää kaikki vaatimukset, konfiguroinnit sekä tiedot, joita tarvitaan kontin luonnissa ja suorituksessa. Useimmiten konttikuva koostuu useasta pohjakuvasta, jotka ovat tasoja, joista koostuu kontin tiedostojärjestelmä. Konttikuvaa ei pysty muuttamaan sen luonnin jälkeen. (nishanil et al., 2021)

Dockerfile on tekstitiedosto, joka sisältää rakennusohjeet konttikuvalle. Dockerfile on kuin tiedosto, joka sisältää komentosarjan. Tiedosto alkaa pohjakuvan määrityksestä ja jatkuu esimerkiksi tarvittavien ohjelmien asennusten ja tiedostojen kopioimisten muodossa. (nishanil et al., 2021)

Kontti itsessään edustaa yhden prosessin, ohjelman tai palvelun suoritusta. Kontti sisältää konttikuvan sisällön, suoritettavan ympäristön ja vakio-ohjeet. Kun palvelua skaalataan, luodaan useampi kontin ilmentymä samasta konttikuvasta. (nishanil et al., 2021)

Voluumit tarjoavat kirjoitettavan tiedostojärjestelmän, jota kontti voi käyttää. Koska konttikuvat ovat vain luettavia, mutta useimmat ohjelmat vaativat mahdollisuuden kirjoittaa tiedostojärjestelmään, lisäävät voluumit kirjoitettavan tason konttikuvan päälle. Ohjelmat eivät huomaa, että tiedostojärjestelmä on eri tasolla, vaan toimii normaalisti. Voluumit sijaitsevat isäntäjärjestelmässä ja ovat Dockerin hallinnoimia. Konttikuvissa käytetään merkkejä ja leimoja, jotta eri konttikuvat tai eri versiot samasta konttikuvasta voidaan identifioida. (nishanil et al., 2021)

Arkisto (repository) on kokoelma konttikuvia, jotka on merkitty tageilla. Sama arkisto voi sisältää useita eri versioita samasta konttikuvasta. Konttikuvat ovat tällöin eritelty tageilla. Samassa arkistossa voi olla useita konttikuvia eri alustavaihtoehdoilla, kuten esimerkiksi Ubuntulla ja Windowsilla. (nishanil et al., 2021)

Rekisteri on palvelu, joka mahdollistaa pääsyn arkistoihin. Vakiorekisteri useimmille julkisille konttikuville on Docker Hub. Rekisteri yleensä sisältää arkistoja useista tiimeistä. Yritykset useimmiten käyttävät yksityisiä arkistoja konttikuvien säilömiseen ja hallintaan. (nishanil et al., 2021)

Yksi tunnetuimpia ja käytetyimpiä avoimen lähdekoodin konttihallinnointi-, ja orkestrointijärjestelmiä on Kubernetes. Kubernetes tarjoaa työkalut konttien käyttöönottoon, hallintaan sekä skaalaukseen. Docker taas on yksi tunnetuimpia kaupallisia konttihallinnointijärjestelmiä. Docker tarjoaa integroidun, testatun ja sertifioidun alustan ohjelmien pyörittämiseen konteissa. Monet uskovat konttien olevan virtuaalikoneita vähemmän turvallisia, esimerkiksi mahdollisen kontin isäntäytimen haavoittuvuuden vuoksi. Viimeisinä parina vuotena kuitenkin on käytetty reilusti vaivaa konttien turvallisuuden edistämiseksi. Esimerkiksi Docker sisältää nykyään konttikuvien allekirjoitusinfrastruktuurin, joka antaa pääkäyttäjien allekirjoittaa luotettavat kontit. Haavoittuvuus voidaan kuitenkin havaita vasta allekirjoituksen jälkeen, minkä vuoksi Docker ja vastaavat järjestelmät

tarjoavat konttien turvallisuuden skannaukseen tarkoitettuja ratkaisuja. Konttien turvallisuuteen pystyy halutessa panostamaan paljonkin. (Rubens, 2017)

Vaikka kontit ovat hyödyllisiä eivät ne todennäköisesti lähitulevaisuudessa korvaa täysin kokonaisia virtuaalipalvelimia. Ensinnäkin on vielä yleinen näkemys, että virtuaalikoneet tarjoavat paremman turvallisuuden, sillä ne ovat eristyksissä korkeammalla tasolla. Toiseksi konttien hallintatyökalut eivät ole vielä yhtä kattavia ja selkeitä kuin virtuaalikoneiden vastaavat, kuten esimerkiksi VMwaren vCenter. (Rubens, 2017)

2.2 Docker

Docker oli alun perin Linuxilla toimiva avoimen lähdekoodin projekti, mutta nykyisin sitä kehittää Docker Inc niminen yritys. Dockerista tarjotaan ilmaista Docker Community -versiota sekä maksullista Docker Enterprise -versiota. (Wallenius, 2020)

Docker on avoimen lähdekoodin alusta tarkoitettu ohjelmien kehittämiseen, kuljetukseen ja ajamiseen. Docker mahdollistaa ohjelmien erottamisen muusta infrastruktuurista, mikä nopeuttaa kuljetusta. Käyttämällä Dockerin metodeja testauksella ja käyttöönotolla, on mahdollista lyhentää reilusti koodin kirjoittamisen ja ohjelman ajamisen välistä aikaa. (*Docker Inc*, n.d.)

2.2.1 Dockerin tärkeimmät teknologiat

Docker käyttää löyhästi eristettyjä kontteja ohjelman paketointiin ja ajamiseen. Eristys ja turvallisuus mahdollistavat usean kontin ajamisen samanaikaisesti tietyllä isännällä. Docker tarjoaa työkalut niin ohjelman kehitykseen kuin myös testaukseen ja käyttöönottoon. Kontti toimii kehitysalustana sekä testausalustana. Kun ohjelma on valmis, se voidaan ottaa käyttöön tuotantoon joko konttina tai orkestroituna palveluna Docker Swarmia käyttäen. Docker kontteja voidaan ajaa esimerkiksi kehittäjän kannettavalla tietokoneella, pöytäkoneella, virtuaalikoneella, pilvessä tai eri alustojen yhdistelmässä. Dockerin siirrettävyys ja keveys helpottaa dynaamista kuorman hallintaa, skaalausta ja ohjelmien alasajoa. (*Docker Inc*, n.d.)

Docker käyttää asiakaspalvelinarkkitehtuuria. Dockerin asiakas keskustelee Docker-daemonin kanssa. Daemon hoitaa raskaan konttien rakennuksen, ajon ja distribuution. Nämä voivat olla ajossa joko samalla järjestelmällä tai etäyhteydellä. Docker-asiakas ja daemon kommunikoivat käyttäen REST API -rajapintaa UNIX-liitännän tai verkkoliitännän kautta. (*Docker Inc*, n.d.)

Dockerin daemon on nimeltään dockerd. Dockerd kuuntelee API-pyyntöjä ja hallitsee Dockerin objekteja, kuten konttikuvia, kontteja, verkkoja ja voluumeja. Daemon voi myös kommunikoida muiden daemonien kanssa hallitakseen Docker palveluja. (*Docker inc*, n.d.)

Käyttäjät vuorovaikuttavat Dockerin kanssa pääsääntöisesti Docker asiakkaan avulla, joka toimii komennoilla ja voi kommunikoida usean daemonin kanssa. Komennon, kuten ”docker run”, ajaessa asiakas lähettää komennon dockerdiin, joka suorittaa sen. Docker komennot käyttävät Docker API -rajapintaa. (*Docker Inc*, n.d.)

Docker-rekisteri tallettaa Dockerin konttikuvat. Docker Hub on julkinen rekisteri jota kuka vain voi käyttää. Docker etsii konttikuvat oletuksena Docker Hubista. Docker Hub on julkinen Docker, inc -yrityksen hallinnoima rekisteri. Se keskittää tiedot liittyen organisaatioihin, käyttäjätileihin ja konttikuviin. Docker Hub sisältää web-käyttöliittymän, autentikaation organisaatioille sekä CLI- ja API-yhteyden esimerkiksi komennoilla ”docker login”, ”docker pull” ja ”docker push”. (*Docker Inc*, n.d.)

Dockee tukee myös yksityisiä rekistereitä, jotka käyttäjä voi luoda itse. Dockerin rekisterit toimivat push- ja pull-komennoilla. Push-komentoa käyttäessä konttikuva työnnetään konfiguroituun rekisteriin. Pull-komentoa käyttäessä konttikuva vedetään konfiguroidusta rekisteristä. (*Docker Inc*, n.d.)

Dockerilla luodaan sekä käytetään konttikuvia, kontteja, verkkoja, voluumeja, lisäosia ja muita objekteja. (*Docker Inc*, n.d.)

2.2.2 Konttikuvat Dockerissa

Konttikuva on vain luettavissa oleva mallipohja, joka sisältää ohjeet Docker kontin luomiseen. Yleensä konttikuva perustuu johonkin aiemmin tehtyyn kuvaan. Esimerkiksi on

mahdollista rakentaa konttikuva, joka perustuu Ubuntu-kuvaan, mutta joka asentaa lisäksi Apache-verkkopalvelimen ja jonkin ohjelman, sekä tarvittavat konfiguraatiot. (*Docker*, n.d.) Konttikuvaa voidaan helposti kopioida ja siirrellä paikasta toiseen. (Wallenius, 2020)

Käyttäjä voi joko luoda omia konttikuvia tai käyttää aiemmin luotuja. Muiden käyttäjien luomia konttikuvia löytyy Docker Hubista paljon useisiin eri tarkoituksiin. Konttikuva rakennetaan luomalla Dockerfile-tiedosto. Dockerfile-tiedosto sisältää yksinkertaisella syntaksilla luodun määritelmän askelista, joilla konttikuva rakennetaan ja ajetaan. Jokainen ohje Dockerfile-tiedostossa luo tason konttikuvaan. Kun Dockerfile-tiedostoa muuttaa ja konttikuvan rakentaa uudestaan, vain muokatut tasot rakennetaan uudestaan. Tämä on yksi syy miksi konttikuvat ovat niin kevyitä, pieniä ja nopeita, kun niitä verrataan muihin virtualisointiteknologioihin. (*Docker Inc*, n.d.)

Kontti on konttikuvan ajettava instanssi. Kontin voi liittää yhteen tai useampaan verkkoon. Oletuksena kontti on suhteellisen hyvin eristetty muista konteista ja isäntäkoneesta. Kuitenkin käyttäjä voi määritellä kuinka eristettyjä kontin verkko, muisti tai muut alisysteemit ovat. Kontti määritellään sen konfiguraatioiden mukaisesti konttikuvan luomisen tai käynnistämisen yhteydessä. Mikäli kontti poistetaan kaikki siihen tehdyt muutokset, joita ei ole talletettu pysyvään muistiin, katoavat. (*Docker*, n.d.) Docker Enginellä ajetaan palvelimen kontteja ja sitä voidaan ohjata komentokehotteella, kuten Powershellillä. (Wallenius, 2020)

2.2.3 Muita Dockerin teknologioita

Docker on kirjoitettu Go-ohjelmointikielellä ja käyttää useita Linux-ytimen ominaisuuksia. Docker käyttää Nimitila-teknologiaa, joka tarjoaa eristetyt kontit Dockerin käyttöön. Kun kontin ajaa, Docker luo joukon nimitiloja kontille. Nämä nimitilat antavat kerroksen eristäytyneisyyttä. Jokainen kontin aspekti ajetaan omalla nimitilalla ja sillä on siis oikeus vain omaan nimitilaan. (*Docker Inc*, n.d.)

Konttien hallintaa ajatellen orkestrointiohjelma on tärkeä. Dockerilla on sisäänrakennettu Docker Swarm -orkestrointiohjelma, mikä on Kubernetestä helpompikäyttöinen, mutta sisältää myös vähemmän toiminnallisuuksia. (Wallenius, 2020)

2.3 Kubernetes

Kubernetes on perustasoltaan järjestelmä, joka on tarkoitettu kontitettujen ohjelmien ajamiseen ja koordinointiin laiteklusterissa. Se on alusta, jonka tarkoitus on hallinnoida kontitettujen ohjelmien ja palveluiden elinkaarta. Kubernetes käyttää metodeja, joiden tarkoitus on tarjota ennustettavuutta, skaalautuvuutta sekä korkeaa käytettävyyttä. (Ellingwood, 2018)

Yksinkertaistettuna Kubernetes on konttiorkestrointialusta, jota voidaan käyttää ohjelmien ajastukseen, käyttöönoton automaatioon, hallintaan ja skaalaukseen. Kuberneteksestä käytetään myös nimityksiä “k8s” ja “kube”. Kubernetestä kehitti aluksi Google, kunnes se tuotiin avoimen lähdekoodin alustaksi vuonna 2014. Kubernetes on kehitetty Googlen sisäisesti käyttämästä Borgista, joka on myös konttiorkestrointiin tarkoitettu alusta. Nykyään Kubernetes sekä muut kontteihin liittyvät teknologiat ovat kypsyneet yleisesti käytetyiksi ja saattavat jopa ohittaa perinteiset virtuaalikoneet modernien pilvi-infrastruktuurien ja pilviohjelmien käytössä. (IBM, 2019)

Yrityksellä saattaa olla käytössä satoja tai jopa tuhansia kontteja, joten on tarvetta niiden ajastukselle, automaatiolle, verkotukselle, skaalaukselle ja saatavuudelle. Tarve konttiorkestrointimarkkinoille on kasvanut siis suureksi. Aikaisessa vaiheessa konttiorkestorintialustat Docker Swarm ja Apache Mesos olivat vahvassa asemassa, mutta nopeasti Kubernetes nousi vahvimpaan asemaan. Kubernetes oli yhdessä vaiheessa nopeiten kasvava projekti avoimen lähdekoodin ohjelmistojen historiassa. Kubernetes valitaan usein sen laajan tuen ja liikuttavuuden vuoksi. Monet eri pilvipalveluiden tarjoajat käyttävät Kubernetestä, joten integraatio ja siirto on sujuvaa. (IBM, 2019)

2.3.1 Kuberneteksen pääkomponentit

Klusterit ja noodit ovat Kuberneteksen pääkomponentteja. Kubernetes koostuu klustereista, jotka koostuvat noodeista. Noodeista jokainen edustaa yhtä isäntää, kuten virtuaalista tai fyysistä konetta. Jokainen klusteri koostuu useasta työntekijänoodista, jotka hoitavat konttisoitujen ohjelmien käyttöönoton, hallinnan ja ajon, sekä yhdestä mestarinoodista, joka kontrolloi ja tarkkailee työntekijänoodeja. (IBM, 2019)

Mestarinoodi eli mestaripalvelin ajaa ajastinpalvelua, joka automatisoi konttien käyttöönoton. Valinta tehdään sen perusteella mitkä ovat asetetut vaatimukset ja missä on kapasiteettia. Jokaisessa työntekijänoodissa on työkalut konttien hallintaan ja käskyjen ajamiseen, kuten Docker ja Kubelet. (IBM, 2019)

Kuberneteksen voidaan ajatella muodostuvan useasta tasosta. Kukin ylempi kerros abstraktioi alemmilla tasoilla havaitun monimutkaisuuden. Kubernetes tuo yhteen yksittäisiä fyysisiä tai virtuaalisia koneita klusterin muodossa. Klusteri käyttää yhteistä verkkoa kommunikointiin. Klusteri on fyysinen alusta, jossa kaikki Kuberneteksen komponentit, ominaisuudet ja työmäärät konfiguroidaan. (Ellingwood, 2018)

Jokaiselle koneelle klusterissa annetaan oma rooli Kuberneteksen ekosysteemissä. Yleisesti yksi palvelin toimii mestaripalvelimena. Mestaripalvelin toimii yhdyskäytävänä ja klusterin älynä. Mestaripalvelin hoitaa ohjelmointirajapinnan julkistamisen käyttäjille ja asiakkaille, muiden servereiden kunnon tarkastamisen, työnmääräyksen sekä orkestrointiin liittyvän kommunikaation. Muut koneet klusterissa toimivat nimettyinä noodeina, eli palvelimina, jotka ovat vastuussa työmääräysten toteutuksesta käytettävissä olevilla resursseilla. Kubernetes pyörittää ohjelmia ja palveluita konteissa, minkä vuoksi jokainen noodi tulee olla varusteltu kontinajo-ohjelmalla, kuten Dockerilla. Noodi luo ja tuhoaa kontteja, sekä pitää huolta verkon toiminnasta, mestaripalvelimen ohjeiden mukaisesti. (Ellingwood, 2018)

Ohjelman tai palvelun käynnistäminen vaatii JSON- tai YAML-muodossa luodut ohjeistukset siitä mitä luodaan ja miten sitä hallinnoidaan. Mestaripalvelin käyttää ohjeistusta ja tutkii kuinka ajaa se infrastruktuurilla järjestelmän vaatimusten ja nykyisen tilan mukaisesti. (Ellingwood, 2018)

Devops on ketterä toimintamalli, joka pyrkii automatisoimaan IT-palvelutoimintoja. Devopsia käytetään metodina, jonka on tarkoitus nopeuttaa ohjelmien luontia, testausta ja julkaisuprosessia. Sen seurauksena on ollut painopisteen muutos infrastruktuurin hallinnasta siihen, kuinka ohjelmia käyttöönotetaan ja skaalataan. Kubernetes tukee tätä mallia, toisin kuin suurin osa muista infrastruktuuripuitteista. Kuberneteksen ohjaimet auttavat infrastruktuurin käyttämistä ohjelman elinkaaren hallinnassa. (Raghuram, 2017)

2.3.2 Mestarinoodin komponentit

Mestarinoodilla on usea komponentti, jotka yhdessä hyväksyvät käyttäjien pyynnöt, päättävät työmäärien ajastuksen, autentikoivat asiakkaat ja noodit, säätävät klusterin laajuista verkkoa ja hallitsevat skaalausta sekä kunnontarkastuksia. Nämä komponentit voidaan asentaa yhdelle koneelle tai jakaa usealle palvelimelle. (Ellingwood, 2018)

Etcd on yksi tärkeimmistä komponenteista. Se toimii maailmanlaajuisesti saatavilla olevana määritysvarastona/muistina. Etcd projektin kehitti CoreOS:n tiimi ja se on kevyt, hajautettu avainarvovarasto, joka voidaan määrittää kattamaan useita noodeja. Kubernetes käyttää etcd:tä varastoidakseen dataa, johon päästään käsiksi jokaisesta klusterin noodista. Tätä voidaan käyttää palvelujen etsimiseen ja se voi auttaa komponentteja konfiguroimaan tai konfiguroimaan itsensä uudelleen ajantasaisten tietojen mukaan. Etcd tarjoaa yksinkertaisen ja suoraviivaisen HTTP/JSON-ohjelmointirajapinnan arvojen asettamiseen ja noutamiseen. Kuten suurin osa muistakin komponenteista, on etcd konfiguroitavissa yhdellä mestaripalvelimella tai tuotantoympäristöissä usealla palvelimella. (Ellingwood, 2018)

Tärkeimpiä mestaripalvelimen palveluita on ohjelmointirajapintapalvelin. Kube-ohjelmointirajapintapalvelin on oleellisin klusterin hallintapiste, sillä sen avulla käyttäjä voi määrittää Kuberneteksen kuormitukset ja organisaatioyksiköt. Se on myös vastuussa etcd-varaston ja käyttöönotetun kontin palvelun yksityiskohdat ovat yhteisymmärryksessä. Ohjelmointirajapinta-palvelu toteuttaa REST-tyylisen käyttöliittymän, joka voi kommunikoida useiden eri työkalujen ja kirjastojen kanssa. Kubectl on oletustyökalu Kubernetes klusterityöskentelyssä paikallisella koneelta. (Ellingwood, 2018)

Kube-controller-manager hallinnoi eri ohjaimia, jotka sääntelevät klusterin tilaa, hallinnoivat työmäärän elinkaarta ja toteuttavat rutiinitehtäviä. Esimerkiksi kopiointiohjain varmistaa, että podille määritelty kopioiden määrä täsmää klusterissa hetkellä käyttöönotettujen määrän kanssa. Operaatioiden yksityiskohdat kirjoitetaan etcd:een, missä ohjaintenhoitaja/controller manager tarkkailee muutoksia ohjelmointirajapinta-palvelimessa. Kun muutos on näkyvissä, controller lukee uuden tiedon ja implementoi menettelyn, joka täyttää toivotun tilan. Tämä voi sisältää ohjelman skaalausta ylös tai alas, päätepisteiden säätöä, jne. (Ellingwood, 2018)

Kube-ajastin on palvelu, joka jakaa työmäärät tietyille noodeille klusterissa. Tämä palvelu lukee työmäärän vaatimukset, analysoi ympäristön infrastruktuurin ja asettaa työn hyväksyttyihin noodeihin. Ajastin on vastuussa isäntien kapasiteettien tarkkailusta, jotta työmäärät eivät ylitä avoimia resursseja. Ajastimen tulee tietää koko kapasiteetti sekä jo annetut työmäärät. (Ellingwood, 2018)

Kubernetes voidaan käyttöönottaa monissa eri ympäristöissä ja se voi vuorovaikuttaa useiden infrastruktuurien tarjoajien ymmärtääkseen ja hallinnoidakseen klusterin resursseja. Vaikka Kubernetes toimii geneeristen resurssien esitysten kuten liitettävän varaston/tallennustilan kanssa, se tarvitsee tavan kartoittaa ne varsinaisten resurssien kanssa, joita tarjoavat ei-homogeeniset pilvipalvelujen tarjoajat. Cloud-controller-manager mahdollistaa Kuberneteksen vuorovaikutuksen erilaisten palveluidentarjoajien kanssa, pystyen samalla pitämään sisäiset rakenteet suhteellisen geneerisinä. Tämä antaa Kuberneteksen päivittää sen tilatiedot pilvipalveluidentarjoajilta saatujen tietojen mukaisesti, säätää pilviresursseja, järjestelmän vaatimusten mukaisesti sekä luoda ja käyttää pilvipalveluita tyydyttääkseen klusterille annetut työvaatimukset. (Ellingwood, 2018)

2.3.3 Noodin palvelinkomponentit

Kuberneteksessä palvelimet, jotka ajavat kontteja, ovat noodeja. Noodipalvelimilla on muutama vaatimus mestarikomponenttien kanssa kommunikointia, konttiverkon konfigurointia ja työmääräyksien ajamista varten. Jokaisen noodin ensimmäiseksi vaatima komponentti on kontinajo-ohjelma. Yleensä tämä vaatimus toteutetaan asentamalla Docker. Muita kontinajo-ohjelmia ovat esimerkiksi rkt ja runc on. Kontinajo-ohjelma käynnistää ja hallitsee kontteja, jotka ovat määritelty klusterille lähetetyissä työmääräyksissä. (Ellingwood, 2018)

Kubelet toimii jokaisen noodin kontaktipisteenä klusteriryhmissä. Kubelet vastaa tietojen välittämisestä ohjaustasopalveluille ja sieltä pois, sekä vuorovaikutuksesta etcd-varaston kanssa määritystietojen lukemiseksi tai uusien arvojen kirjoittamiseksi. Kubelet-palvelu kommunikoi mestarikomponenttien kanssa autentikoidakseen klusteriin ja saadakseen komentoja ja töitä. Työt saadaan manifestin muodossa. Manifesti määrittelee työmäärän ja

toimintaparametrit. Kubelet-prosessi ottaa vastuun noodipalvelimen työn tilan ylläpitämisestä. (Ellingwood, 2018)

Kube-välityspalvelinta ajetaan jokaisella noodilla. Kube-välityspalvelin on pienimuotoinen palvelu, joka hallinnoi yksittäisen isännän aliverkottamisen sekä palvelujen tarjoamisen muiden komponenttien saataville. (Ellingwood, 2018)

Kontit ovat se mekanismi, jota käytetään ohjelmien käyttöönotossa, mutta Kubernetes tarvitsee muitakin tasoja skaalaukseen, joustavuuteen ja elinkaaren hallintaominaisuuksiin. Konttien suoran hallinnan sijaan käyttäjät vuorovaikuttavat esiintymien kanssa. (Ellingwood, 2018) Kontin kaatuessa Kubernetes voi uudelleenkäynnistää tai vaihtaa sen toimivaan automaattisesti. Kubernetes voi myös sulkea kontit, jotka eivät tapaa asetettuja kontin terveysvaatimuksia. (IBM, 2019)

Podit ovat tavallisin Kuberneteksen yksikkö. Kontteja itsessään ei ole määritetty isännille, vaan yksi tai useampi tiukemmin kytketty kontti on kapseloitu podiksi. (Ellingwood, 2018) Podi on lyhyesti tiukka ryhmä kontteja, jotka jakavat samat resurssit ja saman verkon. (IBM Cloud Education, 2019) Podeja voidaan ohjata yhtenä ohjelmana. Yhden podin kontteja käsitellään yksikkönä, ja ne jakavat saman ympäristön, voluumit sekä IP-tilan. Podeja onkin parasta ajatella yhtenä monoliittisena ohjelmana. (Ellingwood, 2018) Podit ovat osa skaalausta. Jos kontti podissa saa enemmän liikennettä kuin se kestää, Kubernetes kopioi podin toisille nodeille klusterissa. (IBM, 2019)

Yleensä podit muodostuvat yhdestä pääkontista, joka vastaa työmäärän yleisestä toiminnasta, ja mahdollisista läheisiä tehtäviä tekevistä apukonteista. Apukonttien ohjelmat ovat tiukasti kiinni pääkontin ohjelmassa, mutta hyötyvät omissa konteissa olosta. Esimerkiksi podissa voi olla yksi pääkontti, joka käyttää ensisijaista sovelluspalvelinta, ja apukontti, joka vetää tiedostot alas jaettuun tiedostojärjestelmään, kun ulkoisessa arkistossa havaitaan muutoksia. (Ellingwood, 2018)

Usein Kuberneteksellä työskennellessä hallinnoidaan, yhden podin sijaan, joukkoa identtisiä, replikoituja podeja. Nämä luodaan podimallista ja niitä voidaan skaalata replikointiohjaimilla ja replikaatiojoukoilla. (Ellingwood, 2018)

Replikointiohjain on objekti, joka määrittelee käytettävän podimallin ja ohjaa skaalaukseen käytettäviä parametrejä. Skaalauksessa voidaan nostaa tai laskea podien identtisten kopioiden määrää. Tämä on yksinkertainen tapa jakaa kuormaa ja lisätä saatavuutta luontaisesti Kuberneteksessä. Replikointiohjain on vastuussa siitä, että käyttöönotettujen podien määrä on sama kuin podien määrä sen konfiguraatiossa. Mikäli podi tai sen isäntä kaatuu, korvaa ohjain sen uudella podilla. Ohjaimen konfiguraatiossa olevien replikoiden määrän muuttuessa, ohjain joko tuhoaa tai käynnistää kontteja päästäkseen samaan lukuun. Replikointiohjaimet voivat myös suorittaa juoksevia päivityksiä siirtääkseen joukon podeja yksitellen uuteen versioon. (Ellingwood, 2018)

Replikaatiojoukot ovat replikointiohjaimen iterointeja, jotka ovat joustavampia tavassa, jolla ohjaimet tunnistavat mitä podeja niiden tulee hallita. Replikaatiojoukot ovat alkaneet korvata replikointiohjaimet niiden joustavuuden vuoksi, mutta ne eivät tue juoksevia päivityksiä samalla tavalla. Kuten podien tapauksessa replikointiohjaimia ja replikaatiojoukkoja ei yleensä hallinnoida suoraan. (Ellingwood, 2018)

Käyttöönotot ovat yksi yleisimpiä työmääriä, joita suoraan luodaan ja hallitaan. Käyttöönotto käyttää replikointijoukkoja rakennuspalikoina, tuoden joustavan elinkaaren hallintatoiminnon. Käyttöönotot ovat korkean tason objekteja, jotka on suunniteltu helpottamaan replikoitujen podien elämänkaaren hallinnassa. Käyttöönottoja on helppo muokata muuttamalla konfiguraatiota. Kubernetes säätää replikointijoukot, hallinnoi ohjelmaversioissa siirtymistä ja valinnaisesti ylläpitää tapahtumahistoriaa ja kumoaa ominaisuuksia automaattisesti. (Ellingwood, 2018)

Käyttöönotot kontrolloivat kontitetun ohjelman luontia ja tilaa sekä pitää sen käynnissä. Se päättää montako kopiota podista klusterilla tulisi olla ajossa. Jos podi kaatuu, käyttöönotto luo uuden. (IBM Cloud Education, 2019)

Tilalliset sarjat ovat erikoistuneita podiohjaimia, jotka tarjoavat tilaamisen ja ainutlaatuisuuden takuut. Näitä käytetään tarkempaan hallintaan, kun on erityisiä vaatimuksia, jotka liittyvät käyttöönoton tilaamiseen, pysyvään dataan tai vakaan verkon muodostamiseen. Tilalliset sarjat liitetään usein datapohjaisiin sovelluksiin, kuten

tietokantoihin, jotka tarvitsevat pääsyn samoihin volyymeihin, vaikka ne ajoitettaisiin uudelle noodille. (Ellingwood, 2018)

Tilalliset sarjat tarjoavat vakaan verkon tunnisteen luomalla uniikin, numeropohjaisen, nimen jokaiselle podille. Tämä nimi pysyy, vaikka podi jouduttaisiin siirtämään toiselle noodille. Myös pysyvät voluumit voidaan siirtää podin mukana, eivätkä katoa, vaikka podi poistettaisiin. (Ellingwood, 2018)

Käyttöönotossa tai skaalatessa tilalliset joukot suorittavat toiminnot nimessä olevan numeroidun tunnisteen mukaan. Tämä parantaa ennustettavuutta ja suoritusjärjestyksen hallintaa, mikä voi olla hyödyllistä joissakin tapauksissa. (Ellingwood, 2018)

Daemon-joukot ovat toinen erikoistunut podiohjain, joka ajaa podien kopioita jokaisella klusterin noodilla. Tämä on useimmiten hyödyllistä, kun otetaan käyttöön podeja, jotka auttavat ylläpidon suorituksessa ja tarjoavat palveluja noodeille itselleen. (Ellingwood, 2018)

Suosittuja toimintoja daemon-joukoissa ovat esimerkiksi lokien kerääminen ja edelleenlähettäminen, tietojen yhdistäminen ja itse noodin ominaisuuksia lisäävien palveluiden suorittaminen. Koska daemon-joukot tarjoavat usein peruspalveluja ja niitä tarvitaan koko kalustossa, ne voivat ohittaa podi-ajastuksen rajoitukset, jotka estävät muita ohjaimia määrittelemästä podeja tietyille isännille. Esimerkiksi ainutlaatuisten vastuidensa vuoksi pääpalvelin on usein määritetty olemaan poissa käytöstä normaalille podi-ajoitukselle, mutta daemon-joukot voivat ohittaa rajoituksen podi-by-podi-pohjalta varmistaakseen, että välttämättömät palvelut ovat käynnissä. (Ellingwood, 2018)

Jobit ovat tehtäväpohjaisia työmääriä, missä ajettavat kontit sulkeutuvat jonkin ajan päästä tehtävän tehtyään. Jobit ovat hyödyllisiä esimerkiksi kertaluontoisissa tehtävissä sen sijaan että ne olisivat jatkuvassa ajossa olevia palveluita. Kuberneteksen jobit ovat cron jobeja. Kuten perinteinen cron daemon Linuxilla, myös cron jobit Kuberneteksellä tarjoavat käyttöliittymän jobien ajamiseen. Käyttöliittymä sisältää ajastuskomponentin. Jobit voidaan ajastaa ajettavaksi joko tulevaisuudessa tai jatkuvasti tiettynä ajankohtana tai aikavälillä. Kuberneteksen cron jobit ovat muuten hyvin samankaltaisia kuin perinteiset cron jobit, mutta ne käyttävät alustana klusteria eivätkä yksittäistä järjestelmää. (Ellingwood, 2018)

2.3.4 Muut Kubernetes-komponentit

Perinteisesti palvelulla tarkoitetaan, usein verkkoon yhdistettyä, pitkään ajossa olevaa prosessia. Kuberneteksessä palvelu on komponentti, joka toimii perustason sisäisenä kuormituksen tasapainottajana ja lähettiläänä podeille. Palveluryhmät ovat loogisia kokoelmia podeista, jotka suorittavat saman toiminnon näkyäkseen yhtenä kokonaisuutena. Tämän avulla on mahdollista ottaa käyttöön palvelu, joka voi seurata reittiä kaikkiin tietyn tyyppisiin taustakontteihin. Sisäisten käyttäjien tarvitsee tietää ainoastaan palvelun tarjoama vakaa päätepiste. Palvelun abstraktion avulla voi skaalata tai vaihtaa taustatyöyksiköt tarpeen mukaan. Palvelun IP-osoite pysyy vakaana riippumatta siitä, mitä muutoksia se ohjaa podeihin. (Ellingwood, 2018)

Aina kun tarvitaan yhteyttä yhteen tai useampaan podiin, ohjelmaan tai ulkoiseen käyttäjään, tulee konfiguroida palvelu. Palvelu tuo tarvittavan abstraktion esimerkiksi tilanteessa, jossa käyttäjä tahtoo podien ajavan verkkopalvelimia, joilla on pääsy internetiin. Vaikka palvelut oletusarvoisesti ovat käytettävissä vain sisäisesti reititettävän IP-osoitteen avulla, ne voidaan asettaa saataville klusterin ulkopuolelle valitsemalla yksi useista strategioista. NodePort-konfiguraatio toimii avaamalla staattinen portti kunkin noodin ulkoiseen verkkoliitäntään. Liikenne ulkoiseen porttiin ohjataan automaattisesti asianmukaisiin podeihin käyttämällä sisäistä klusterin IP-palvelua. (Ellingwood, 2018)

Vaihtoehtoisesti, LoadBalancer-palvelutyyppi luo ulkoisen kuormituksen tasauksen, reitittääkseen palveluun pilvipalveluntarjoajan Kubernetes-kuormituksen tasaaja -integraation avulla. Cloud controller manager luo sopivan resurssin ja konfiguroi sen sisäisten palvelujen palveluosoitteiden avulla. (Ellingwood, 2018)

2.3.5 Voluumit ja levypalvelut

Container runtimet tarjoavat usein jonkinlaisen mekanismin tallennuksen kiinnittämiseen varastoon, joka säilyy kontin käyttöiän jälkeen, mutta toteutuksista puuttuu tyypillisesti joustavuus. Kubernetes käyttää sen omaa voluumiabstraktiota, joka mahdollistaa tiedon jaon kaikilla konteilla podissa, kunnes podi tuhotaan. Tiukasti kytketyt podit voivat helposti jakaa tiedostoja ilman monimutkaisia ulkoisia mekanismeja. Podissa esiintyvät konttihäiriöt

eivät vaikuta jaettujen tiedostojen käyttöön. Kun podi on tuhottu, myös jaettu asema tuhoutuu, joten se ei ole hyvä ratkaisu oikeasti pysyvään dataan. (Ellingwood, 2018)

Pysyvät voluumit ovat mekanismi vankemman tallennustilan abstraktoimiseksi, joka ei ole sidottu podin elinkaareen. Sen sijaan, ne antavat järjestelmänvalvojille mahdollisuuden määrittää klusterin tallennusresursseja, joita käyttäjät voivat pyytää ja käyttää podeissa. Kun podi on tehty pysyvällä voluumilla, podin palauttamispolitiikka määrittää, pidetäänkö voluumia, kunnes se poistetaan manuaalisesti tai poistetaanko tiedot välittömästi. Pysyviä tietoja voidaan käyttää suojaamaan podipohjaisia vikoja vastaan ja varaamaan suurempia määriä tallennustilaa kuin paikallisesti on käytettävissä. (Ellingwood, 2018)

Pysyvät voluumit määritetään Kubernetekseen PersistentVolume-objekteilla. PersistentVolume-objekti kuvastaa olemassaolevaa verkotettua varastoa. Tähän objektiin voidaan määrittää varasto, joka toimii esimerkiksi NFS- tai iSCI-yhteydellä. Näitä varastoja, joita Kubernetes voi käyttää PersistentVolume-objektin avulla kutsutaan levypalveluiksi. (ahardin-rh et al., 2016)

2.3.6 Leimat ja merkinnät

Kubernetes-leima on semanttinen tunniste, joka voidaan liittää Kubernetes-objekteihin merkitäkseen ne osaksi ryhmää. Ryhmä voidaan sitten valita kohdistettaessa eri instansseihin hallintaa tai reititystä varten. Esimerkiksi jokainen ohjainpohjainen objekti käyttää leimoja tunnistaakseen podit, joita niiden tulisi käyttää. Palvelut käyttävät leimoja ymmärtääkseen mihin taustapodeihin niiden tulisi reitittää pyynnöt. Leimat annetaan yksinkertaisina avainarvopareina. Jokaisella yksiköllä voi olla useampi kuin yksi leima, mutta vain yksi liittyminen kutakin avainta kohtaan. (Ellingwood, 2018)

Merkinnät ovat samankaltainen mekanismi, jonka avulla voit liittää avainarvotietoja objektiin. Leimoja tulisi käyttää semanttisissa tiedoissa, jotka ovat hyödyllisiä sovittamaan podi valintakriteereihin, kun taas merkinnät ovat vapaamuotoisempia ja voivat sisältää vähemmän jäsenneltyjä tietoja. (Ellingwood, 2018)

2.4 Muut teknologiat

Konttiteknologian lisäksi työhön liittyy vahvasti myös muita teknologioita, joilla hallitaan esimerkiksi Kuberneteksen käyttöönottoa, salaisuuksia ja levypalveluita. Terraform on yleinen ja tehokas IaC-työkalu. Salaisuuksien sekä levypalveluiden hallintaan taas on useita vaihtoehtoja.

2.4.1 IaC ja Terraform

Terraform on avoimen lähdekoodin IaC-ohjelma, jonka on kehittänyt HashiCorp. Terraform mahdollistaa lopullisen infrastruktuurin kuvaamisen JSON-tyylisellä HCL-konfiguraatiokielellä. Terraform luo kuvauksen mukaisesti suunnitelman ja toteuttaa sen. (IBM, 2020)

Terraformista on muodostunut yksi suosituimpia infrastruktuurin automatisaatio-ohjelmia, sillä se käyttää yksinkertaista syntaksia ja tekee ympäristöjen uudelleentoteutuksen turvalliseksi sekä tehokkaaksi. (IBM, 2020)

Infrastructure as Code -periaate mahdollistaa ympäristöjen koodiksi ja/tai konfiguraatiokieleksi muuttamisen. IaC-periaate tekee ympäristöjen pystyttämisen automatisoiduksi, nopeaksi ja toistettavaksi, mikä on tärkeää esimerkiksi versionhallinnassa ja jatkuvassa integraatiossa sekä kehityksessä. (IBM Cloud Education, 2020) Terraformia käytetään muun muassa ulkoisten resurssien hallintaan, pilvipalveluiden käyttöönotossa, resurssikokoelmien skaalaukseen, SDN-konfiguraatioon, resurssien ajastukseen ja testiympäristöjen tekoon. (Gillis, 2021)

IaC-periaate nopeuttaa ympäristön pystyttämistä automaatiolla, parantaa luotettavuutta, ehkäisee versio-ongelmia ja tukee kokeilua, testausta sekä optimointia. Terraformin hyviä puolia verrattuna muihin IaC-periaatteen työkaluihin ovat lähdekoodin avoimuus, yhteensopivuus monien pilvipalveluiden kanssa ja ympäristöjen muuttumattomuus. (IBM, 2020)

Moduulilla tulee olla nimeämisrakenne, arkistokuvaus, standardi moduulirakenne, tuettu versionhallintajärjestelmä ja julkaisutunnisteet, jotta se voidaan julkaista Terraformin

rekisterissä. Rekisteri toimii keskitettynä arkistona moduulien jakamiselle ja löytämiselle. Rekisteri muodostuu julkisesta ja yksityisestä rekisteristä. Julkinen on yhteisöä varten ja yksityinen on organisaation sisäinen. (Gillis, 2021)

Terraform muodostuu useasta konseptista ja käyttää tiettyä terminologiaa. Terraform moduulit ovat pienikokoisia ja uudestaan käytettäviä konfiguraatioita useille yhdessä käytettäville ympäristöresursseille. Terraform moduulit mahdollistavat monimutkaisten resurssien automatisaation uudelleen käytettävillä ja konfiguroitavilla rakenteilla. Moduuleilla voi olla lapsimoduuleja ja niitä voidaan kutsua useita kertoja. Terraform providerit ovat laajennuksia, jotka lisäävät resurssityyppejä. Providerit sisältävät kaiken tarpeellisen koodin, jota tarvitaan palveluun yhdistämisessä. Providereita on olemassa lähes kaikille suurimmille pilvipalveluntarjoajille, ja yhteisö sekä palveluntarjoajat luovat jatkuvasti lisää. (IBM, 2020)

Muuttujat ovat arvoja, jotka mahdollistavat kustomoinnin. Tila muodostuu välimuistissa olevasta tiedosta liittyen Terraformin hallinnoimaan ympäristöön ja sen konfiguraatioon. Resurssi viittaa ympäristön objekteihin, kuten virtuaalisiin verkkoihin. Resursseja käytetään infrastruktuurin konfigurointiin ja hallintaan. (Avi, 2020)

Datalähteistä Terraform saa providereiden avulla tietoa ulkoisista objekteista. Ulostuloarvot ovat Terraformille palaavia arvoja, joita moduulit voivat käyttää. Terraformin suunnitelmakomento luo suunnitelman, jossa Terraform tutkii mitä tulee luoda, päivittää tai tuhota, jotta ympäristön sen aikainen tila muuttuu halutun kaltaiseksi. Terraformin käyttökomento aloittaa itse muutosten ajamisen suunnitelman mukaisesti. (Avi, 2020)

2.4.2 Salaisuudet

Salaisuus on lyhyesti digitaalisen todennuksen tunnistetieto. Ne ovat erikseen nimettyjä arkaluonteisia tietosettejä. Salaisuudet käsittävät laajasti erilaisia suojattuja tietoja. Erilaisia salaisuuksia ovat esimerkiksi salasanat, ssh-avaimet, API-avaimet, autentikaatiotunnisteet ja yksityiset sertifikaatit. Parhaiten tunnettu salaisuus on salasana, mutta se ei kuitenkaan ole yleisin tai haastavin säilöä turvallisesti. (Hoffman, 2021)

Digitaalisten salaisuuksien tarve on kasvanut paljon ja tulee kasvamaan uusien ja monimutkaisempien teknologioiden tullessa käyttöön. Esimerkiksi salaisuudet, jotka kulkevat ohjelmasta toiseen lisääntyvät eksponentiaalisesti. Tällaisia salaisuuksia ovat esimerkiksi verkkosivun lähettämät tiedot, API-pyynnöt ja pilvikantoihin pääsy. Salaisuuksia tarvitaan niin paljon ja niin monissa eri konteksteissa, että niiden seuraaminen tulee haastavaksi ilman salaisuuksien hallintaa. (Hoffman, 2021)

Salaisuuksien hallinta pitää salaisuudet turvassa, ehkäisee salaisuuksien leviämistä ja huolehtii siitä, että järjestelmät pystyvät yhdistämään välittömästi toteuttaakseen automaattiset tehtävät. Kun salaisuus luodaan, tulee sen elämänkaarta hallita oikein. Salaisuuksien hallinta on prosessi, jolla hallitaan salaisuuksia turvallisesti ja keskitetysti koko salaisuuden elinajan. Salaisuuden elinkaareen kuuluu luonti, kierto ja peruuttaminen. (Hoffman, 2021)

Salaisuuksia luodaan manuaalisesti, käyttäjien toimesta sekä automaattisesti. Salasanojen tulee yleensä täyttää monimutkaisuusvaatimukset, mutta manuaalisesti luotuna salasana on silti todennäköisesti helpommin murrettavissa. Käytössä olevat salaisuudet tulee vaihtaa tasaisin väliajoin, mikä on monien standardien vaatimuskin. Yleisin salaisuuksien kiertoaika on 90 päivää. Vaihto voidaan tehdä manuaalisesti ja automaattisesti. Salaisuuden vanhetessa käyttö on estetty, kunnes se on päivitetään. Salaisuuden poistaminen käyttäjän tai ohjelman käytöstä on tärkeä ominaisuus. Käytöstä poistamista voidaan käyttää esimerkiksi työntekijän lähtiessä yrityksestä. (Hoffman, 2021)

Salaisuuksien hallinta poistaa tai vähentää ihmisten osallistumista salaisuuksien hallinnassa. Systemaattinen lähestymistapa salaisuuksien hallintaan estää luvattoman pääsyn sensitiiviseen dataan ja ehkäisee tietomurtoja sekä identiteettivarkauksia. (Hoffman, 2021)

Vahvat salaisuuskäytännöt auttavat ongelmissa liittyen salaisuuksien jakamiseen, uudelleenkäyttöön, talletukseen, kiertoon ja peruuttamiseen. Salaisuuksien hallinnassa tulee muistaa turvata kaikki organisaation salaisuudet, luoda yhtenäinen käytäntö, automatisoida kaikki mahdollinen, pakottaa käytäntö ohjelmiin ja käyttäjille sekä erottaa data salaisuuksista. (Hoffman, 2021)

2.4.3 VMware vSphere

vSphere on sateenvarjotermi VMwaren virtualisaatioympäristölle. Termi vSphere pitää sisällään useita tuotteita ja teknologioita, jotka yhdessä tarjoavat kokonaisen virtualisaatioinfrastruktuurin. vSpheren teknologioita/tuotteita ovat esimerkiksi ESXi, vCenter Server, vCenter Client ja VMFS. (Lowe, n.d.)

ESXi on vSpheren ydin, jota ajetaan isäntäkoneella. ESXi hoitaa virtuaalikoneiden suorituksen jakamalla niille resursseja tarpeen mukaisesti. vCenter Server on palvelinohjelma, joka on virtuaalikoneiden luomisen, käynnistämisen ja lopettamisen keskus. vCenterissä voidaan tehdä myös muita vSphere ympäristöön liittyviä hallintatoimenpiteitä. vCenter Client on Windows-ohjelma, jota käytetään vCenterin etäkäyttöön. VMFS (Virtual Machine File System) on tiedostojärjestelmä, jota VSphere käyttää levyn resurssien hallintaan. VMFS:llä voidaan luoda tietovarastoja ja voluumeja, jotta fyysiset levyresurssit saataisiin virtuaalikoneiden käyttöön. (Lowe, n.d.)

3 Testaus ja tutkimus

Työn tutkimuskysymyksissä keskitytään IaC-käyttöönottoon, salaisuuksien hallintaan ja levypalveluihin. Käytännön osassa käydään asioita läpi ajatuksella, että lukija on tutustunut Kuberneteksen perustoimintoihin, sillä opinnäytetyön tarkoitus on keskittyä tiettyihin näkökulmiin. Tarkoitus on ottaa käyttöön vSphere-ympäristöön Kubernetes-klusteri IaC-tyylillä, jos se on mahdollista. Terraform on suosittu ja hyväksi todettu monessa yhteydessä, ja soveltuu sen vuoksi IaC-käyttöönoton testaamiseen.

Salaisuudet ovat iso osa Kuberneteksen toimintaa ja niiden turvallisuudesta huolehtiminen on tärkeää. Esimerkiksi asiakasympäristön pystytyksessä pitää asiakkaan pystyä luottamaan ympäristön turvallisuuteen. Salaisuuksien hallinta tekee myös salaisuuksien ylläpidon ja niistä perillä pysymisen helpommaksi ja selkeämmäksi, mikä säästää aikaa ja vaivaa. Levypalveluiden tarkoitus on toimia Kuberneteksen podien yhteisenä voluumien tallennustilana, johon pääsee käsiksi halutuista paikoista ja voidaan taata voluumien pysyvyys.

Salaisuuksien hallintaan ja levypalveluille löytyy monia erilaisia vaihtoehtoja. Työssä on tarkoitus tutustua muutamiin eri vaihtoehtoihin, vertailla niitä ja pohtia mikä olisi paras mihinkin käyttöön. Yritys saa näin tietoa, jota tarvitsee päätöksentekoon, kun Kubernetes-klusteri otetaan käyttöön.

3.1 Virtualbox-testiympäristö

Tehdään aluksi virtuaalikone, jolle luodaan yksinkertainen testiympäristö. Tarkoitus on testata Kuberneteksen asentamista ja käyttöönottoa Linux-pohjaisella koneella. Käytetään CentOS8-kuvaa ja asennetaan virtuaalitietokone käyttäen VirtualBox-ohjelmistoa tai käytetään valmiiksi luotua virtuaalitietokonetta. VirtualBoxiin luodun virtuaalitietokoneen

tiedot kuvattu taulukossa 1. Taulukossa kuvatut määritykset kuvaavat Kubernetes-noodin minimivaatimuksia. Todellisen käytön vaatimukset ovat huomattavasti suuremmat.

Taulukko 1 VirtualBox-ympäristöön luotujen virtuaalitietokoneiden tiedot

Käyttöjärjestelmä:	Centos 8 Stream
Työpöytä:	Kyllä
Muisti:	2790MB
Prosessorit:	2kpl
Videomuisti:	112MB
Tallennustila:	135.59GB
Dynaaminen tila:	Kyllä
Verkko:	Oma NAT
Nimi:	kubemaster

Aluksi tarkistetaan porttien tila komennossa 1 ja avataan portit 6443/tcp ja 10250/tcp Dockerin ja Kuberneteksen käyttöönottoa varten komennoissa 2 ja 3. Peruskäyttöönotto vaatii vain nämä portit, mutta monimutkaisemman ympäristön käyttö vaatii useampia portteja.

Komento 1 palomuurin komento, jolla tarkistetaan aktiiviset portit

```
firewall-cmd –get-active-zones
```

Komento 2 palomuurin komento, jolla avataan pysyvästi portti 6443/tcp

```
firewall-cmd –zone=public –add-port=6443/tcp –permanent
```

Komento 3 palomuurin komento, jolla avataan pysyvästi portti 10250/tcp

```
firewall-cmd –zone=public –add-port=10250/tcp –permanent
```

Haetaan Docker-arkisto komennossa 4, tarkistetaan arkistolistan sisältö komennossa 5 ja asennetaan Containerd ja Docker komennoissa 6 ja 7. Kubernetes vaatii Containerdin ja Docker helpottaa konttien kanssa työskentelyä. Asennuksen jälkeen käynnistetään docker ja asetetaan se käynnistymään automaattisesti palvelimen uudelleenkäynnistyessä komennossa 8. Tarkistetaan komennossa 9 Dockerin tila.

Komento 4 dnf-komento, jolla haetaan docker-ce-arkisto

```
sudo dnf config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo
```

Komento 5 dnf-komento, jolla tarkistetaan arkistolista

```
sudo dnf repolist -v
```

Komento 6 dnf-komento, jolla asennetaan haluttu containerd-paketti

```
sudo dnf install
https://download.docker.com/linux/centos/7/x86_64/stable/Packages/cont
ainerd.io-1.2.10-3.2.el7.x86_64.rpm
```

Komento 7 dnf-komento, jolla asennetaan docker-ce

```
sudo dnf install docker-ce -y –nobest
```

Komento 8 systemctl-komento, jolla käynnistetään Docker ja asetetaan se käynnistymään aina automaattisesti

```
sudo systemctl enable --now docker
```

Komento 9 systemctl komento, jolla tarkistetaan Dockerin tila

```
systemctl status docker
```

Dockerin asentamisen ja käynnistämisen jälkeen asennetaan Kubernetes. Aluksi komennossa 10 luodaan uusi kubernetes.repo-tiedosto, joka toimii arkistona. Tiedostoon lisätään arkiston määritys. Määritys näkyvillä tiedostossa 1.

Komento 10 nano-komento, jolla luodaan kubernetes.repo-tiedosto

```
sudo nano /etc/yum.repos.d/kubernetes.repo
```

Tiedosto 1 kubernetes.repo-tiedosto, Kubernetes arkistoa varten

```
[kubernetes]
```

```
name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64

enabled=1

gpgcheck=1

repo_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
```

Tallennetaan ja suljetaan Kubernetes.repo-tiedosto. Asennetaan tarvittavat Kuberneteksen paketit komennossa 11. Testiympäristössä tarvittavat paketit ovat kubelet, kubeadm ja kubectl.

Komento 11 dnf-komento, jolla asennetaan Kuberneteksen paketit

```
sudo dnf install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes
```

Käynnistetään kubelet-palvelu sekä varmistetaan että se käynnistyy aina automaattisesti koneen uudelleenkäynnistyessä komennossa 12. Luodaan root-käyttäjällä tiedosto k8s.conf, IP-tauluja varten, komennossa 13. Tiedostoon k8s.conf lisätään tarvittavat rivit. Määritysrivit näkyvillä tiedostossa 2.

Komento 12 systemctl-komento, jolla käynnistetään kubelet, sekä asetetaan se käynnistymään aina automaattisesti.

```
sudo systemctl enable --now kubelet
```

Komento 13 nano-komento, jolla luodaan k8s.conf-tiedosto

```
nano /etc/sysctl.d/k8s.conf
```

Tiedosto 2 k8s.conf-tiedosto, Kuberneteksen konfiguraatiota varten

```
net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1
```

Tallennetaan ja suljetaan tiedosto. Päivitetään uusi konfiguraatio komennossa 14. Komennossa 15 otetaan swap pois käytöstä hetkellisesti.

Komento 14 systctl-komento, jolla päivitetään konfiguraatio

```
sysctl --system
```

Komento 15 swapoff-komento, joka poistaa swapin käytöstä

```
sudo swapoff -a
```

Swap saadaan pysyvästi pois muokkaamalla tiedostoa fstab komennolla 16. Swapin tulee olla pysyvästi poissa käytöstä tässä tapauksessa, sillä se estää klusterin toimintaa.

Komento 16 nano-komento, jolla muokataan fstab-tiedostoa

```
sudo nano /etc/fstab
```

Kommentoidaan #-merkillä rivi, joka alkaa: ”/dev/mapper/cl-swap” ja tallennetaan sekä suljetaan tiedosto. Lopuksi luodaan daemon.json-konfiguraatiotiedosto ja lisätään siihen tarvittava konfiguraatio komennossa 17. Konfiguraatio tiedostossa 3.

Komento 17 nano-komento, jolla luodaan daemon.json-tiedosto

```
nano /etc/docker/daemon.json
```

Tiedosto 3 daemon.json-tiedosto, daemon-palvelun konfiguraatiota varten

```
{
 "exec-opts": ["native.cgroupdriver=systemd"],
 "log-driver": "json-file",
 "log-opts": {
  "max-size": "100m"
 },
 "storage-driver": "overlay2",
 "storage-opts": [
  "overlay2.override_kernel_check=true"
 ]
}
```

Tallennetaan ja suljetaan tiedosto. Luodaan uusi systemd-kansio komennossa 18. Käynnistetään daemon- ja Docker-palvelu uudestaan komennoilla 19 ja 20.

Komento 18 mkdir-komento, jolla luodaan kansio docker.service.d

```
mkdir -p /etc/systemd/system/docker.service.d
```

Komento 19 systemctl-komento, jolla uudelleenkäynnistetään deamon-palvelu

```
systemctl daemon-reload
```

Komento 20 systemctl-komento, jolla uudelleenkäynnistetään docker-palvelu

```
systemctl restart docker
```

Poistutaan root-käyttäjältä peruskäyttäjälle, jolla on su-oikeudet. Asennusten jälkeen luodaan uusi kansio .kube komennossa 21 ja kopioidaan Kuberneteksen luoma konfiguraatiotiedosto kyseiseen kansioon komennossa 22. Tämän tekemättä jättäminen aiheuttaa virhetilanteen: ”The connection to server localhost:8080 was refused”, kun ajetaan Kubernetes-klusterin käyttöönottokomento. Komennossa 23 muokataan config-kansion omistajuutta, jotta voidaan olla varmoja siitä, että admin.conf-tiedostoon päästään käsiksi.

Komento 21 mkdir-komento, jolla luodaan .kube-kansio

```
mkdir -p $HOME/.kube
```

Komento 22 cp-komento, jolla kopioidaan admin.conf-tiedosto ~/.kube/config-kansioon

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

Komento 23 chown-komento, jolla muutetaan config-kansion omistajuutta

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Ajetaan klusterin käyttöönottokomento (komento 24) ainoastaan mestarinoodilla. Kubeadm join -komento (komento 25) ajetaan työntekijänoodeilla. Kubeadm join -komento yhdistää työntekijänoodit Kuberneteksen klusteriin. Komento saadaan mestarinoodilta ja sisältää tunnuksen ja sertifikaatin tunnistusta ja yhdistystä varten.

Komento 24 kubeadm-komento, jolla luodaan Kubernetes-klusteri

```
sudo kubeadm init
```

Komento 25 kubeadm-komento, jolla työntekijänoodi liitetään klusteriin

```
kubeadm join <IP>:<port> –token <token> –dicovery-token-ca-cert-hash
<shaHash>
```

3.2 Terraform ja Packer

IaC-tavalla on kätevä pystyttää joko samanlaisia ja/tai halutunlaisia virtuaalikoneita ja ohjelmia vain muutamalla komennolla. Packer on suosittu ja helppokäyttöinen IaC-ohjelma virtuaalikonemallien luomiseen. Terraform taas monikäyttöisempi esimerkiksi virtuaalikoneiden pystyttämisessä ja ohjelmien asentamisessa. Packer ja Terraform käyttävät vSphere-ympäristön API-rajapintaa lähettääkseen vSpherelle vaaditut tiedot, miten ja minne virtuaalitietokoneet luodaan.

Virtualboxissa kuberneteksen testaamisen jälkeen asennetaan yrityksen vSphere-ympäristöön virtuaalitietokone, josta tulee mallipohja Kuberneteksen noodeille. Terraformilla taas luodaan mallipohjan mukaisesti kolmen virtuaalitietokoneen klusteri. Yksi mestarinoodi ja kaksi työntekijänoodia. Pidemmälle vietynä Terraformilla luodaan myös Kubernetes-klusteri mestarinoodille ja asennetaan Kuberneteksen vaatimat ohjelmat työntekijänoodeille.

3.2.1 Mallipohjan luominen vSphere-ympäristöön

Virtuaalitietokoneen mallipohja luodaan vSphereen käyttäen Packer-ohjelmaa. Packer on järjestelmien käyttöönoton automaatioon tarkoitettu työkalu. Packeria varten kirjoitetaan konfiguraatiotiedostoja, jotka kertovat ohjelmalle mihin ympäristöön asennus tehdään sekä millainen virtuaalitietokoneen tulisi olla.

Asennetaan aluksi chocolatey-asennusohjelma. Chocolateylla voidaan asentaa helposti, komentorivin tai Powershellin kautta, erilaisia ohjelmia ja työkaluja Windowsille. Avataan Powershell administrator-tunnuksilla ja aluksi ajetaan komento 26, jolla tarkistetaan suorituskäytännön tila.

Komento 26 Get-ExecutionPolicy-komento, jolla tarkistetaan ajorajoitukset

```
Get-ExecutionPolicy
```

Jos palaute on ”Restricted”, ajetaan komento 27, joka poistaa ajorajoituksen.

Komento 27 Set-Executionpolicy-komento, jolla poistetaan ajorajoitukset

```
Set-ExecutionPolicy AllSigned
```

Ajetaan komento 28, joka asentaa Chocolateyn.

Komento 28 Set-ExecutionPolicy-komento, jolla Chocolatey asennetaan

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex
((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.p
s1'))
```

Jos mitään virheitä ei tule, on Chocolatey asennettu ja valmiina käyttöön. Seuraavaksi asennetaan Packer-ohjelma Chocolatey-asennusohjelmalla Windowsille. Kirjoitetaan komento 29 komentoriville.

Komento 29 choco-komento, jolla Packer asennetaan

```
choco install packer
```

Packeria varten luodaan konfiguraatiotiedostot Ubuntu-18.json (Liite 2), variables.json (tiedosto 3) ja preseed.cfg (Liite 3). Tiedostoista näytetään yksityissyistä vain havainnollistavia osia, joista poistetut osat on merkitty valkoisilla viivoilla.

Ubuntu-18.json (Liite 2) on päätiedosto, joka lähettää tarvittavat tiedot Packerille ja vSpherelle. Tiedosto koostuu builders-osasta ja provisioners-osasta. Builders-osa sisältää määritykset, kuten esimerkiksi virtuaalikoneen nimen, muistin koon, ISO-kuvan sijainnin, vCenterin salasanan ja käyttäjätunnuksen. Builders-osa sisältää myös virtuaalikoneen käynnistyskomennot. Käynnistyskomennot asentavat käyttöjärjestelmän ja asettavat esimerkiksi kielivalinnat automaattisesti.

Käynnistyskomentoihin lisätään yleensä preseed.cfg-tiedosto, johon voidaan määritellä paljon tarkemmin halutut asetukset. preseed.cfg-tiedosto ei kuitenkaan välttämättä toimi enää uudemmilla Ubuntu-versioilla. Niiden sijaan käytetään user-data- ja meta-data-tiedostoja, jotka kirjoitetaan .json-tyylillä.

Variables.json-tiedostoon täytetään muuttujat, joita Ubuntu-18.json-tiedosto käyttää. Muuttujiin voi kuulua esimerkiksi virtuaalikoneen muistin määrä, verkkokortti ja OS-tyyppi. Tässä opinnäytetyössä käytetyssä variables.json-tiedostossa on määritelty vain salattavat tiedot, kuten vCenter palvelimen nimi, vSpheren käyttäjätunnus, ssh-tiedot, salasana ja koneen IP-osoite. Tiedostossa 4 ovat käytetyt muuttujat näkyvillä. Varsinkin jos variables.json-tiedoston haluaa laittaa versionhallintaan, tulee käyttää ympäristömuuttujia tai salaisuuksia, eikä kirjoittaa tietoja suoraan tiedostoon.

Tiedosto 4 variables.json-tiedosto

```
{
    "vcenter_server":"vCenterServerinNimi",
    "username":"vSphereKäyttäjätunnus",
    "password": "vSphereSalasana",
    "datastore":"DatastoreNimi",

    "ssh_password": "sshSalasana"
}
```

3.2.2 Noodien luonti vSphere-ympäristöön

Packerin luomaa valmista mallipohjaa voidaan käyttää Kuberneteksen noodien luomiseen Terraformilla. Terraformilla voidaan luoda sekä virtuaalitietokoneita, että asentaa niihin ohjelmia ja konfiguroida niitä. Seuraavaksi luodaan Terraformilla mallipohjan mukaisesti yksi mestarinoodi ja kaksi työläisnoodia.

Asennetaan Terraform chocolatey-asennusohjelmalla Windowsille komennolla 30. Kirjoitetaan asennuskomento komentoriville.

Komento 30 choco-komento, jolla asennetaan Terraform

```
choco install terraform
```

Konfigurointia varten luodaan tiedostot main.tf (Liite 4), variables.tf (Liite 5) ja terraform.tfvars (Tiedosto 5). Tiedostoista nähtävissä havainnollistavia osia. Kokonaisia tiedostoja ei näytetä yksityissyistä. Tiedostot katkaistaan valkoisten viivojen kohdilta.

Main.tf-tiedosto on nimensä mukaisesti päätiedosto, jossa kaikki määrittely tapahtuu tässä tapauksessa. Main.tf-tiedosto kertoo Terraformille mitä asetuksia ja muuttujia käytetään virtuaalitietokoneiden luonnissa. Tiedosto eritellään muutamiin osioihin. Osioita ovat provider, data, resource ja output. Provider-kohdassa annetaan vSphere serverin nimi ja tunnukset. Data-kohdissa annetaan vSpheressä olevien käytettävien resurssien tiedot, kuten verkko, datastore-klusteri ja mallipohja. Resource-kohdassa annetaan luotavien virtuaalikoneiden ja kansioiden tiedot. Main.tf-tiedoston resource-kohdat on tässä tapauksessa jaettu kolmeen osioon. Yksi osio on kansion luonti vSphereen, jonka sisään virtuaalikoneet luodaan. Toinen osio on mestarinoodin luominen ja kolmas työntekijänoodien luominen. Noodien luomiseen liittyvissä osioissa annetaan esimerkiksi virtuaalikoneiden haluttu määrä, nimi, IP ja muistin määrä.

Variables.tf-tiedosto tarvitaan viemään muuttujat main.tf-tiedostoon. Variables.tf-tiedostossa määritellään muuttujien nimet, oletusarvot, kuvaukset ja tyypit. Itse arvot muuttujille annetaan tiedostossa terraform.tfvars.

Tiedosto 5 terraform.tfvars-tiedosto

```
control_count = "1"
worker_count = "2"

vsphereuser = "vSphereKäyttäjätunnus"
vspherepass = "vSphereSalasana"


vm_cpu = "2"
vm_ram = "4100"


vm_datastore = "DatastoreNimi"
vm_network = "VerkonNimi"
vm_domain = "HaluttuDomain"

vsphere_vcenter = "vCenterServerNimi"
```

Ajetaan komento 31, **terraform init**, joka tutkii konfiguraatiot läpi ja päättää parhaat asetukset sekä provisioinnit.

Komento 31 terraform-komento, jolla terraform asettaa/asentaa parhaat asetukset

```
terraform init
```

Seuraavaksi ajetaan komento 32, **terraform plan**, joka luo suunnitelman, jolla työ toteutetaan. Suunnitelma näytetään käyttäjälle ja siitä voi tarkistaa onko kaikki kunnossa. Komento myös ilmoittaa monista mahdollisista konfiguraation virheistä, jotka tulee korjata.

Komento 32 terraform-komento, jolla Terraform luo suunnitelman

```
terraform plan
```

Lopuksi ajetaan komento 33, **terraform apply**, joka toteuttaa työn. Komentoriviltä voi seurata virtuaalikoneiden luomista sekä skriptien ajoa. Työ voi keskeytyä, mikäli terraform havaitsee ongelmia. Ongelmat johtuvat yleensä virheistä konfiguraatiotiedostoissa ja ne tulee korjata tai Terraform ei saa työtä loppuun.

Komento 33 terraform-komento, jolla konfiguraatio ajetaan ja noodit luodaan

```
terraform apply
```

3.2.3 Kubernetes-klusterin luonti vSphere-ympäristöön

Terraformia voidaan käyttää myös itse Kubernetes-klusterin luomiseen lisäämällä määrityksiä ja .sh-skriptejä. Luodaan uudet Terraform-määritykset, joilla luodaan vSphereen virtuaalikoneiden lisäksi myös Kubernetes-klusteri.

Konfigurointia varten luodaan konfiguraatiotiedostot main.tf (Liite 6), output.tf, terraform.tfvars (Liite 7), variables.tf (Liite 13), kubernetes-master.tf (Liite 8) ja kubernetes-node.tf (Liite 9). Kubernetes-klusteria varten luodaan .sh-tiedostot configure_part1.sh (Tiedosto 7), configure_part2.sh (Liite 10), configure_part3.sh (Liite 11), configurek8node_part1 (Tiedosto 8) ja configurek8node_part2 (Liite 12).

Tässä tapauksessa main.tf-tiedosto keskittyy provider- ja data-osioihin. Provider-osassa haetaan ja viedään Terraformille vSphereen kirjautumistiedot. Data-osioissa määritetään vSpheressä käytetyt resurssit, kuten verkko ja mallipohja. Output.tf-tiedosto on lyhyt ja sen on vain tarkoitus tuoda takaisin päin ympäristöstä halutut tiedot tai arvot. Tässä tapauksessa tuodaan noodien ip:t.

Terraform.tfvars-tiedosto on samantyylinen kuin aiemmassakin esimerkissä, mutta sisältää enemmän arvoja. Kaikki arvot on annettu tässä tiedostossa, eikä suoraan muualla. Tiedosto on jaettu vCenter-osaan, Virtuaalikoneiden-osioon ja Kubernetes-osioon. Kubernetes osiossa määritetään esimerkiksi noodien määrä, Kuberneteksen versio ja työntekijänoodien ip:t. Variables.tf-tiedosto toimii aivan kuten aiemmassakin esimerkissä.

Kubernetes-master.tf-tiedosto on samankaltainen kuin aiemman esimerkin main.tf-tiedoston osio, joka keskittyi mestarinoodiin. Tiedostossa on määritykset mestarinoodin resursseille ja esimerkiksi ip:lle. Tiedostossa on myös useampi provisioner-osio sekä connection-osio. Provisioner-osiot lähettävät tiedostoja ja komentoja virtuaalikoneelle. Tässä tapauksessa ne lähettävät .sh-tiedostoja ja ajavat niiden ajokomentoja sekä Dockerin ja Kuberneteksen asennuskomentoja. Connection-osiot ovat tarpeellisia provisioner-osioita varten, sillä niillä yhdistetään virtuaalikoneeseen ssh-yhteydellä, jotta komennot voidaan ajaa ja tiedostot lähettää.

Kubernetes-node.tf-tiedosto on rakenteeltaan samanlainen kuin kubernetes-master.tf-tiedosto, mutta se on työntekijänoodeja varten. Muuttujien arvojen määrityksessä on tosin joutunut ottamaan huomioon sen, että työntekijänoodeja on useampi. Nimet ja ip:t on annettu aloittamalla yhdellä arvolla ja lisäämällä haluttu määrä lukuun aina seuraavassa ilmentymässä. Tässä esimerkissä luodaan kaksi työntekijänoodia.

Configure_part1.sh-, configure_part2.sh- ja configure_part3.sh-tiedostot kopioidaan ssh-yhteydellä mestarinoodille. Mestarinoodin .sh-tiedostot sisältävät skriptiä, jolla konfiguroidaan Dockeria ja Kubernetestä. Configurek8node_part1- ja configurek8node_part2.sh-tiedostot konfiguroivat Kuberneteksen ja Dockerin työntekijänoodeille sekä asentavat Dockerin.

Tiedosto 6 output.tf-tiedosto

```
output "Master_IP_Address" {
  value = "${var.vsphere_ipv4_address}"
}

output "Node_IP_Addresses" {
  value = "${vsphere_virtual_machine.K8-NODE.*.default_ip_address}"
}
```

Tiedosto 7 configure_part1.sh-tiedosto

```
systemctl stop firewalld
```

```
systemctl disable firewalld
```

```
cat << EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-
key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
EOF
```

Tiedosto 8 configurek8node_part1.sh-tiedosto

```
systemctl stop firewalld
systemctl disable firewalld
```

```
cat << EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-
key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
EOF
```

Ajetaan terraform komennot: **terraform init**, **terraform plan** ja **terraform apply**, sekä korjataan mahdolliset virheet konfiguraatiossa.

Yrityksen vSphere-ympäristöön on pystytetty Terraformilla Kubernetes-klusteri aiempien kohtien mukaisesti. Terraform-tiedostojen ollessa valmiita, onnistuu klusterin pystytys hetkessä. Kolmen Terraform-komennon ajaminen voidaan nopeuttaa entisestään käyttämällä komentotiedostoa, mutta jos toimivuus ei ole täysin varmaa se ei kannata, sillä ensimmäisten kahden komennon aikana tulee ilmi ongelmat konfiguraatiossa. Ongelmien tullessa ilmi ne on selkeämpi korjata.

3.3 Salaisuuksien hallinta

Salaisuuksien hallinta on yritykselle tärkeää kaikkia teknologioita käytettäessä. Salaisuuksien turvallisuudesta tulee pitää erityistä huolta, kun ollaan luomassa ympäristöä asiakkaalle. Asiakas luottaa, että heille luotu ympäristö on turvallinen, varsinkin kun ympäristöön lisätään arkaluonteista materiaalia. Kuberneteksen käytössä välitetään konteille esimerkiksi salasanoja ja SSH-avaimia. Konteille voidaan viedä salaisuudet suoraan konttikuvan kautta tai komentokehotteesta konttia luodessa. Konttikuvassa, varsinkin selkotekstinä, ne ovat erittäin suuressa riskissä. Komentokehotteen kautta annettuna ne ovat hiukan paremmassa turvassa, mutta se on myös työläämpää. Salaisuuksienhallintatyökalut tuovat turvallisuuden lisäksi helpotusta salaisuuksista perillä pysymisessä. Salaisuudet pysyvät turvassa, tallessa sekä lajiteltuna ja niiden oikeuksia esimerkiksi käyttäjien, konttien, podien ja palvelimien välillä voidaan hallita. Työssä on tarkoitus valita muutama salaisuuksien hallintaan soveltuva työkalu, vertailla niitä ja pohtia mikä olisi hyvä mihinkin tarkoitukseen. Salaisuuksienhallintatyökalut voivat olla ilmaisia tai maksullisia. Työhön valittiin Hashicorpin Vault, Akeyless ja Kubernetes Secrets. Hashicorpin Vaultista on ilmainen ja maksullinen versio.

3.3.1 Hashicorp Vault

Vault on Hashicorpin luoma työkalu, joka on tarkoitettu salaisuuksien turvalliseen käyttöön. Vault tarjoaa yhtenäisen käyttöliittymän kaikenlaisille salaisuuksille. Vaultia voidaan käyttää komentokehotteella sekä visuaalisella URL-käyttöliittymällä. Vaultin tärkeimpiä ominaisuuksia ovat turvattu salaisuusvarasto, dynaamiset salaisuudet, tietojen salaus, salaisuuksien vuokraaminen, vuokrauksien uusiminen ja salaisuuksien peruuttaminen. Vault kirjoittaa salaisuudet talteen varastoon, kuten levylle, tietokantaan tai Consuliin. (*HashiCorp*, n.d.)

Vaultin avoimen lähdekoodin -versio saa jatkuvasti käyttäjiltä uutta sisältöä esimerkiksi integraatioiden muodossa. Lisäksi ohjeita käytöstä löytyy paljon ja Hashicorpin omat ohjeet ovat kattavat ja niitä on reilusti, mutta ne eivät ole aina täysin ajankohtaiset. Avoimen lähdekoodin -version voi ottaa käyttöön ilmaiseksi, mutta on olemassa myös maksullinen vaihtoehto Vault Cloud. Vault Cloud on HashiCorpin ylläpitämä IaaS-tyylinen ratkaisu ja

sisältää hiukan enemmän ominaisuuksia, kuten parempi tuki, varmuuskopiot ja nimitilat. Vault Cloudin hinnaksi on ilmoitettu vähintään 0.03 dollaria tunnissa. (*HashiCorp*, n.d.) Hinta oletettavasti nousee salaisuuksien lukumäärän ja käyttömäärien mukaisesti.

3.3.1.1 Consul

Vault tarvitsee backend-varaston salaisuuksien tallettamista varten. Erilaisia backend-varastovaihtoehtoja on useita ja jokaisella on hyviä ja huonoja puolia. Vaihtoehtoina ovat esimerkiksi tiedostojärjestelmä, etcd, erilaiset tietokannat ja Consul. (*HashiCorp*, 2021) Tiedostojärjestelmä ja Consul ovat virallisesti tuettuja backend-varastoja. Backend-varastoa valittaessa on tärkeä ottaa huomioon tukeeko se korkeaa saatavuutta. Consul ja etcd tukevat, mutta jotkin tietokannat ja tiedostojärjestelmä eivät (*HashiCorp*, 2021; *HashiCorp*, 2021).

Consul on palveluverkkoratkaisu, joka tarjoaa täysin varustellun ohjaustason palvelujen löytämis-, määritys- ja segmentointitoiminnoilla. Jokaista toimintoa voidaan käyttää erikseen ja yhdessä. Consul vaatii tietotason ja tukee sekä välityspalvelinta että natiivia integraatiomallia. KV Store (key/value store) on yksi Consulin tärkeimmistä ominaisuuksista. KV Storella on yksinkertainen HTTP API, joka tekee siitä helpon käyttää. (*HashiCorp*, 2021) Vault voi käyttää KV Storea backend-varastona. Vaultin ja Consulin yhteiskäytöstä löytyy hyvin ohjeita ja artikkeleja. Consul on tuettu, helppokäyttöinen ja lisäominaisuuksia tuova backend-varasto Vaultille, minkä vuoksi se on varteenotettava vaihtoehto.

3.3.1.2 Kubernetes-ajuri

Vault CSI Provider mahdollistaa Vaultin salaisuuksien käyttämisen Kuberneteksen podeissa käytettäessä CSI Secrets Store -voluumeja. Käytännössä CSI Secrets Store -ajuri antaa käyttäjän luoda SecretProviderClass-objektin Kubernetekseen. Objektissa määritetään mitä salaisuuksienhallinnan tarjoajaa käytetään ja mitä salaisuuksia haetaan. Kun podin CSI-voluumit luodaan, CSI Secrets Store -ajuri lähettää pyynnön Vault CSI Providerille. Vault CSI Provider käyttää objektin määrityksiä, hakee halutut salaisuudet Vaultista ja kiinnittää ne podin CSI-voluumiin. Salaisuus saadaan Vaultista ja liitetään CSI Secrets Store -voluumiin kontinluonti-vaiheen aikana, minkä vuoksi podi ei käynnisty ennen kuin salaisuudet on luettu Vaultista ja kirjoitettu voluumiin. (*HashiCorp*, 2021)

3.3.2 Akeyless

Akeyless Vault -alusta on SaaS-tyylinen salaisuuksienhallintajärjestelmä. Akeylessillä pystyy tallettamaan, suojaamaan, kääntämään ja dynaamisesti luomaan salaisuuksia. Työmääräimet ja käyttäjät ovat vuorokaudessa Akeylessin kanssa useiden kanavien kautta. Kanavia ovat esimerkiksi verkkokonsoli, Akeyless Remote Access Portal, CLI ja useat lisäosat. Akeyless käyttää suurta määrää erilaisia autentikaatiometodeja, joiden ansiota voidaan hallita kenellä tai millä on oikeus mihinkin salaisuuteen. (*Akeyless*, 2021)

Akeyless käyttää Akeyless Distributed Fragments (Akeyless DFC) -teknologiaa, joka mahdollistaa SaaS-ratkaisun, jossa Akeyless ei pysty saavuttamana tai purkamaan käyttäjän salaisuuksia. (*Akeyless*, 2021) Akeyless DFC -teknologian ansiota yksi avaimen osista voi olla käyttäjän ympäristössä, jolloin Akeylessillä ei ole pääsyä siihen. (*Akeyless*, 2021)

Akeyless tukee monia integraatioita ja uusia voi ehdottaa verkkosivuilla. Huomionarvoisia integraatioita ovat esimerkiksi Kubernetes, Terraform, Microsoft AD, GitHub, Chrome, Azure DevOps ja Ansible. (*Akeyless*, n.d.) Akeylessin Kubernetes-lisäosa mahdollistaa staattisten ja dynaamisten salaisuuksien käytön konttiympäristössä. (*Akeyless*, 2021) Käyttäessä Kubernetes-lisäosaa, ohjelmien tarvitsee vain löytää salaisuuden tiedostopolku. Ohjelman ei siis tarvitse hallinnoida tunnuksia tai yhdistää ulkoiseen API:iin. Akeylessin Kubernetes-lisäosan käyttämät Sidecar-kontit hakevat salaisuudet ennen kuin ohjelma käynnistyy. (*Akeyless*, 2021)

Akeyless on maksullinen palvelu, minkä vuoksi käyttäjän ei tarvitse itse ylläpitää palvelinta tai varastoa. Valittavissa on neljä sopimusta. Pienin vaihtoehto on ilmainen ja suurin kallein. Pienin vaihtoehto sisältää käytön kolmelle asiakkaalle, joita ovat esimerkiksi ohjelmat, palvelut ja käyttäjät. Salaisuuksia pienimmällä voi tallettaa 50 kappaletta. Loput vaihtoehdot lisäävät asiakasmääriä ja salaisuuksien määriä suuresti, mutta tuovat myös reilusti lisähintaa. (*Akeyless*, n.d.) Sopimus tuleekin valita sen mukaan mikä riittää omaan Kubernetes-ympäristöön. Tulee ottaa myös huomioon eri sopimuksiin sisältyvät lisäominaisuudet. Esimerkiksi ilmaisversio ei sisällä korkeaa käytettävyyttä tai hätäpalautusta. Sopimusversioiden suurimmat erot ja hinnat näkyvillä taulukossa 2.

Taulukko 2 Akeyless-sopimusten hinnat ja erot

Sopimus	Community	Business	Enterprise	Enterprise+
Hinta	Ilmainen	2000$/kk	4500$/kk	Sopimuksen mukaan
Asiakasmäärä	3	100	250	Loputtomasti
Salaisuuksien määrä	50	5000	20000	Loputtomasti
Lisäominaisuudet	Ei	Sisäänrakennettu monivuokraus ja korkea käytettävyys ja hätäpalautus	Sisäänrakennettu monivuokraus ja korkea käytettävyys ja hätäpalautus	Sisäänrakennettu monivuokraus ja korkea käytettävyys ja hätäpalautus. Vaihtoehtoisesti vain yksi vuokraus.
Ylimääräiset asiakkaat ja salaisuudet	40$/kk, 0.3$/kk	19$/kk, 0.2$/kk	16$/kk, 0.2$/kk	Ei lisähintaa

Akeylessille löytyy hyvin virallisia asennus -ja käyttöohjeita. Kaikki ohjeet eivät tosin ole aina täysin ajankohtaisia, mutta käyttäjäkunta on suuri ja yhteisö on aktiivinen, mikä parantaa epävirallista tukea.

3.3.3 Kubernetes Secrets

Kubernetes Secrets on Kuberneteksen oma yksinkertainen salaisuuksienhallinta toiminto. Kuberneteksen salaisuudet voidaan liittää podiin joko ympäristömuuttujana tai liitetiedostona. (*Kubernetes*, 2021) Kuberneteksen salaisuudet ovat reilusti turvallisempia ja joustavampia kuin salaisuuksien suora liittämien podiin tai Dockerin konttikuvaan, mutta Kubernetes Secrets -toiminnolla on kuitenkin heikkouksia.

Kubernetes Secrets tallettaa käyttäjätunnukset ja salasanat base64-muotoon koodattuina merkkijonoina. Salaisuudet pysyvät piilossa arkikäytössä, mutta base64-koodaus ei ole turvallinen. Talletetut salaisuudet pysyvät näkyvissä vain klusterin sisällä. Tämä voi aiheuttaa ongelmia joissain pilviympäristöissä, sillä kaikki toiminnot eivät välttämättä ole Kuberneteksen konteissa. (*Kubernetes*, 2021)

Kubernetes Secretsin potentiaaliset riskit on hyvin selvillä ja niitä voidaan lieventää kolmannen osapuolen työkaluilla sekä laajennuksilla. Salaisuudet talletetaan etcd:een, joten tulee huolehtia etcd:n turvallisuudesta. Kuberneteksen salaisuuksia voidaan luoda käyttäen JSON -ja YAML-tiedostoja, joten riskinä on, että salaisuudet päätyvät esimerkiksi arkistoon tai GitHubiin. Salaisuustiedostojen käsittelyssä tuleekin olla tarkka. Käyttäjät, joilla on

oikeudet podiin, pääsevät myös käsiksi podin käyttämiin salaisuuksiin, vaikka ne olisikin suojattu erikseen. Kontteja ajetaan noodeilla, joten salaisuuksiin on mahdollista päästä käsiksi Kubernetes API:n avulla, jos käyttäjällä on juurioikeudet. Tämä voidaan tehdä matkimalla kubelet-palvelua. Noodien turvallisuus on siis yhtä tärkeä asia kuin podienkin. (Kubernetes Advocate, 2021)

3.3.4 Salaisuuksien hallintajärjestelmien/palveluiden vertailu

Kuberentes Secrets ei ole vaihtoehdoista kovinkaan turvallinen, eikä se tarjoa lähellekään saman tasoista salaisuuksienhallintaa. Ominaisuuksia on paljon vähemmän ja se on tarkoitettu juuri Kubernetestä varten, joten muissa mahdollisissa tapauksissa siitä ei ole hyötyä. Hashicorpin Vault ja Akeyless tarjoavat paljon turvallisemman, monipuolisemman ja hyödyllisemmän ympäristön. Vaultin mukana saattaa olla järkevin vaihtoehto ottaa myös käyttöön Consul, joka tuo lisää useita ominaisuuksia. Consulin kanssa toimiva Vault ja Akeyless tarjoavat korkeaa saatavuutta.

Hashicorp Vaultin käyttöönotto on aluksi varsin hankalaa ja monimutkaista, vaikka ohjeilla pääsee hyvään alkuun. Vaultin käyttöönotto vie siis aikaa ja vaatii paneutumista. Consul vaatii myös paneutumista ja ajankäyttöä, varsinkin jos haluaa hyötyä sen muistakin ominaisuuksista kuin vain backend-varastosta.

Akeyless on SaaS-tyyppinen järjestelmä, joka ei vaadi lainkaan asennuksia. Käytön voi aloittaa helposti ja nopeasti kirjautumalla verkkokäyttöliittymään. Asennusten puute tarkoittaa myös sitä, ettei päivityksiä tai huoltoja tarvitse tehdä. Akeylessin tukitiimi korjaa mahdolliset esiin tulevat ohjelman virheet. SaaS-tyyli helpottaa lisäksi skaalausta, kun järjestelmä otetaan käyttöön useammalle käyttäjälle. Vault ja Akeyless ovat kumpikin skaalattavissa minkä kokoiselle organisaatiolle vain. Akeylessin SaaS-tyyli tarkoittaa myös sitä, että sitä käyttävien palvelinten tulee olla yhteydessä ulkoverkkoon päästäkseen käsiksi salaisuuksiin. Vault taas voidaan pystyttää sisäiseen verkkoon, johon ei ole ulkoverkkoyhteyttä.

Akeyless on lähes täysin maksullinen palvelu, jonka hinta määräytyy halutun sopimuksen sekä käytön mukaisesti. Hashicorp Vault taas on ilmainen avoimen lähdekoodin ympäristö,

josta löytyy myös maksullinen versio. Pienen tai pienehkön yrityksen tarpeisiin voivat Akeylessin ilmainen tai halvin sopimus sopia. Kalliimmat sopimukset ovat suurille yrityksille, joissa käyttäjiä ja tarpeita on satoja, jopa tuhansia. Vaultin tapauksessa päätös siitä ottaako yritys ilmaisen vai maksullisen version riippuu esimerkiksi siitä, että millainen tuki tahdotaan. Avoimen lähdekoodin versiolle tuki on pääsääntöisesti yhteisön tarjoamaa, kun taas maksullisessa Hashicorp tarjoaa tuen. Yhteisö on kuitenkin suuri ja aktiivinen, joten tukea löytyy reilusti. Akeyless tarjoaa tuen kaikissa sopimuksissa, myös ilmaisessa. Akeylessin panostus tuen tarjoamiseen tosin nousee, mitä kalliimpi sopimus on. Näiden kahden vaihtoehdon välillä ilmenee siis suuri ero kustannuksissa. Hashicorp Vaultin voi saada suurella skaalalla käyttöön ilman suoria kustannuksia, kun taas Akeyless pääsääntöisesti kustantaa joka kuukausi. Vaultin ja Consulin käyttö tosin saattaa myös lisätä kustannuksia, esimerkiksi palvelimien ylläpidon vuoksi. Vaultin ja Consulin käyttö ja varsinkin käyttöönotto myös vievät aikaa, jota voitaisiin käyttää muuhunkin.

Lyhyesti, Kubernetes Secrets on vähiten turvallinen, skaalautuva ja joustava. Akeyless ja Vault ovat periaatteen tasolla lähes yhtä tehokkaita ja turvallisia, mutta Akeyless voittaa helppokäyttöisyydessä ja käyttöönotossa. Akeyless tarjoaa myös DFC-teknologian, joka lisää turvallisuutta. Vaihtoehtojen hinnoissa taas on suuria eroja. Kubernetes Secrets on ilmainen Kuberneteksen mukana tuleva palvelu, Vaultilla on ilmainen sekä maksullinen vaihtoehto ja Akeyless on pääsääntöisesti maksullinen. Akeylessin maksullisuus on tosin skaalattavissa hyvin tarpeen mukaisesti.

3.4 Levypalvelut

Yksinkertainen Kuberneteksen käyttö ei vaadi levypalveluita, mutta ne tuovat paljon joustavuutta ja hallittavuutta. Levypalvelut tuovat mahdollisuuden pitää klusterin tallennustilaa yllä muualla kuin Kubernetes-palvelimella, joten mahdollinen klusterin noodin kaatuminen ei tuhoa noodin käyttämää talletettua dataa, kuten avaimia tai skriptitiedostoja. Noodien käyttämän datan ollessa yhteisellä tallennustilalla, voivat ne myös käyttää tätä dataa yhdessä.

Levypalvelut tässä yhteydessä koostuvat pääsääntöisesti varastointijärjestelmästä, Kubernetes-klusterista ja teknologiasta, joka yhdistää nämä. Kubernetes-klusteri käyttää

käytännössä levytilaa pysyvien voluumien muodossa. Pysyvät voluumit eivät tuhoudu, vaikka podi/kontti, joka niitä käyttää, tuhoutuisi. Pysyvillä voluumeilla voidaan määritellä esimerkiksi kuinka ison siivun podi/kontti saa levytilasta.

Opinnäytetyöhön kuuluu muutamaan levypalveluun tutustuminen ja niiden vertailu sekä pohdinta, voidaanko jotain niistä käyttää yrityksen tarkoitusperissä. IBM tarjoaa ainakin kaksi eri mahdollisuutta ja NFS on vanha teknologia, jota voidaan mahdollisesti myös käyttää Kubernetes-klusterin pysyvissä voluumeissa.

3.4.1 IBM Spectrum connect

IBM Spectrum Connect tarjoaa yhdellä palvelimella toimivan backend-sijainnin ja keskittää IBM varastojärjestelmien resurssit eri tarkoituksiin (*IBM*, 2021). Opinnäytetyön puolesta kiinnostava tarkoitus on konttiympäristön pysyvien voluumien käyttö. IBM Spectrum Connect tarjoaa myös verkkoselaimessa toimivan käyttöliittymän, jonka avulla konfiguraatiosta tulee helpompaa ja nopeampaa (*IBM*, 2021).
Kuberneteksen pysyvät voluumit IBM Spectrum Connectin avulla toimivat kolmen ympäristön avulla/välillä. Itse varastointijärjestelminä toimivat IBM:n FlashSystem-järjestelmät. Kontrollointiin käytetään palvelinta, jolla toimii IBM Spectrum Connect -hallintapalvelu. Kubernetes-ympäristössä taas on asennettu ja käyttöönotettu noodeille IBM Storage Enabler for Containers -ohjelma. (*IBM*, 2021)

Uusin IBM Spectrum Connectin versio, joka tukee varmasti IBM Storage Enabler for Containersia on 3.5.0 vuodelta 2018, sillä sen dokumentaatiota ei enää pidetä uudemmissa versioissa mukana. Uusin IBM Storage Enablerin versio on 2.1.0 vuodelta 2019. Näiden kahden yhteyksistä kertovat tiedot on kerrottu sen verran sekavasti, että en löydä varmaa tietoa tukeeko uudemmat IBM Spectrum Connectin versiot IBM Storage Enabler for Containersia. Tosin myös uudet versiot mainitsevat kontit, joten voisi olettaa, että tukevat sitä tai jotain muuta tapaa, josta tietoa en löytänyt. IBM Storage Enabler for Containeristakaan ei löydy 2019 jälkeen uutta tietoa, eikä sitä ole päivitetty sen jälkeen, joten vaikuttaa siltä, että sen tuki on loppunut. Hyviä ohjekirjojakin löytyy, mutta nekin kirjoitettu vuonna 2018 tai aiemmin. Koska IBM Storage Enabler for Containersista ei löydy tuoretta tietoa, ei ole myöskään varmaa toimiiko se uudempien Kubernetes-versioiden

kanssa. Uusin Kuberneteksen versio, jonka kanssa Enabler for Containers varmasti toimii, on 1.12 (*IBM*, 2019).

IBM Storage Enabler for Containers tukee tiettyjä varastointijärjestelmiä ja käyttöjärjestelmiä. Tuetut varastointijärjestelmät ovat pääsääntöisesti IBM:n FlashSystem-tuotteita. Kaikki tuetut järjestelmät listassa 1. (*IBM*, 2019) Tuettuja käyttöjärjestelmiä ovat RHEL 7, Ubuntu 16.04 ja sen uudemmat versiot sekä SUSE Linux Enterprise Server 12 (*IBM*, 2019). IBM Storage Enabler for Containers -palvelulla on kaksi tunnettua rajoitusta. Palvelu ei tue samanaikaista IBM blockin ja IBM Spectrum Scalen käyttöä. Palvelu ei myöskään tue voluumimigraatiota, jos käytetään IBM Hyper-Scale Mobilitya. (*IBM*, 2019)

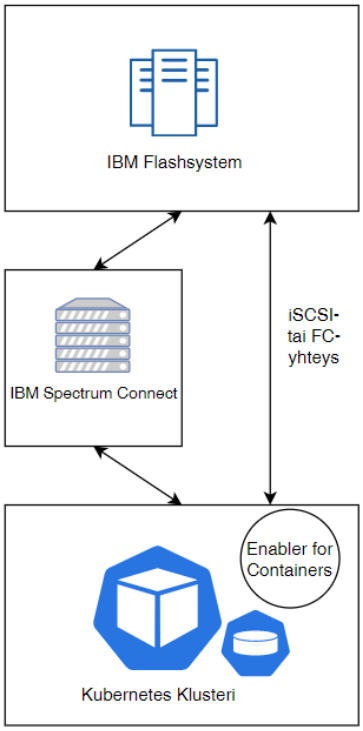
Lista 1 IBM Storage Enabler for Containers -palvelun tukemat varastointijärjestelmät

- IBM DS8870 (7.5.1)
- IBM DS8880 (8.0.1, 8.1.x, 8.2.x, 8.3.x, 8.4.x, 8.5.x)
- IBM FlashSystem 9100 (8.2.x)
- IBM FlashSystem A9000 (12.0.x, 12.1.x, 12.2.x, 12.3.x)
- IBM FlashSystem A9000R (12.0.x, 12.1.x, 12.2.x, 12.3.x)
- IBM FlashSystem V9000 (7.6.0, 7.7.0, 7.8.x, 8.1.x, 8.2.x)
- IBM SAN Volume Controller (7.4.0, 7.5.0, 7.6.0, 7.7.0, 7.8.x, 8.1.x, 8.2.x)
- IBM Spectrum Accelerate (11.5.x)
- IBM Spectrum Scale (5.x.x)
- IBM Spectrum Virtualize as software only (7.7.1, 7.8.x, 8.1.x, 8.2.x)
- IBM Storwize® V3500 (7.4.0, 7.5.0, 7.6.0, 7.7.0, 7.8.x)
- IBM Storwize V3700 (7.4.0, 7.5.0, 7.6.0, 7.7.0, 7.8.x)
- IBM Storwize V5000 (7.4.0, 7.5.0, 7.6.0, 7.7.0, 7.8.x, 8.1.x, 8.2.x)
- IBM Storwize V7000 (7.4.0, 7.5.0, 7.6.0, 7.7.0, 7.8.x, 8.1.x, 8.2.x)
- IBM XIV® Storage System (11.6.x)

Mitä tulee ottaa huomioon, kun otetaan käyttöön IBM Spectrum Connect levypalveluita ajatellen Kubernetes-klusteria varten? Kubernetes-klusteriin tulee asentaa ja konfiguroida IBM Storage Enabler for Containers -ajuri, tarvitaan jokin tuetuista varastointijärjestelmistä, asennetaan ja konfiguroidaan hallintaohjelma varastointijärjestelmälle, jos ei niin ole vielä tehty. Esimerkiksi FlashSystem A9000 -varastointijärjestelmä käyttää Hyper-Scale Manageria, kun taas FlashSystem 5000 käyttää IBM Spectrum Virtualizea. Enabler for Containers asennetaan jokaiselle työntekijänoodille. Yhdelle klusterista erillään olevalle palvelimelle asennetaan IBM Spectrum Connect, josta tulee hallintapalvelin. Yhteys

varastointijärjestelmän, hallintapalvelimen ja Kubernetes-klusterin välillä on havainnollistettu kuvassa 1.

Kuva 1 Kuva, joka havainnollistaa yksinkertaisesti IBM Spectrum Connectin ja IBM Storage Enabler for Containersin toiminnan



3.4.2 IBM Block CSI driver

IBM Block CSI driver on ajuri, jonka voi asentaa Kubernetes-klusterin noodeille. Sen avulla voidaan yhdistää tuettu IBM:n varastointijärjestelmä klusteriin pysyvien voluumien käyttöä varten. Kuberneteksen pysyvät voluumit voivat hyödyntää IBM block storage CSI driver -ohjelmaa tilaa sisältävien konttien kanssa käytetyn lohkotallennuksen tarjoamiseksi dynaamisesti. (*IBM*, 2021)

IBM block storage CSI driver -ohjelma perustuu IBM:n avoimen lähdekoodin projektiin CSI driver. IBM: n säilöille tarkoitettu säilytysjärjestely antaa yrityksille mahdollisuuden toteuttaa nykyaikaisen konttiohjatun hybridi monipilvi-ympäristön, joka voi vähentää IT-kustannuksia ja parantaa liiketoiminnan ketteryyttä samalla, säilyttäen myös yhä nykyisten järjestelmien arvon. (*IBM*, 2021)

Hyödyntämällä IBM-tallennusjärjestelmien CSI-ajureita, Kuberneteksen pysyvät volyymit voidaan varata dynaamisesti lohko- tai tiedostotallennukselle, joita käytetään tilallisten säilöjen, kuten tietokantaohjelmien, kanssa Kubernetes-klusterissa. Tallennustilan valmistelu voidaan automatisoida klusteri-orkestrointijärjestelmien lisätuella, jotta konttisovelluksia voidaan käyttöönottaa, skaalata ja hallita automaattisesti. (*IBM*, 2021)

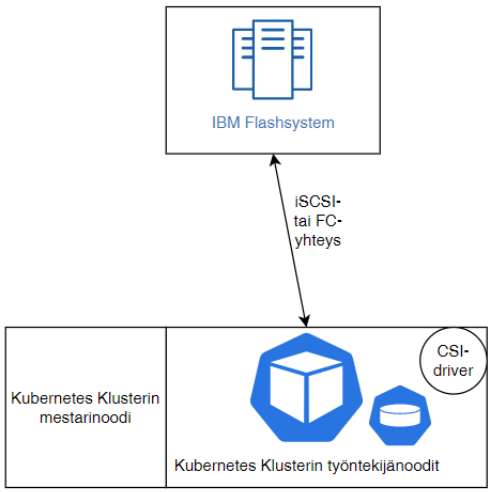
IBM Block CSI driver tukee tiettyjä varastointijärjestelmiä ja käyttöjärjestelmiä. Tuetut varastointijärjestelmät ovat pääsääntöisesti IBM:n FlashSystem-tuotteita. Kaikki tuetut järjestelmät listassa 2. Tuettuja käyttöjärjestelmiä ovat RHEL 7 ja Red Hat Enterprise Linux CoreOS. (*IBM*, 2021)

Lista 2 IBM Block CSI Driverin tukemat varastointijärjestelmät

- IBM Spectrum Virtualize Family including IBM SAN Volume Controller (SVC) and IBM FlashSystem® family members built with IBM Spectrum® Virtualize (FlashSystem 5010, 5030, 5100, 5200, 7200, 9100, 9200, 9200R)
- IBM FlashSystem A9000 and A9000R
- IBM DS8000 Family

Mitä tulee ottaa huomioon, kun otetaan käyttöön IBM Block CSI driver levypalveluita ajatellen Kubernetes-klusteriin? Kubernetes-klusteriin tulee asentaa ja konfiguroida CSI driver, tarvitaan jokin tuetuista varastointijärjestelmistä, asennetaan ja konfiguroidaan hallintaohjelma varastointijärjestelmälle, jos ei niin ole vielä tehty. Esimerkiksi FlashSystem A9000 -varastointijärjestelmä käyttää Hyper-Scale Manageria, kun taas FlashSystem 5000 käyttää IBM Spectrum Virtualizea. CSI Driver asennetaan jokaiselle työntekijänoodille, joista yhdestä tulee CSI Driverin ohjain. Yhteys varastointijärjestelmän ja Kubernetes-klusterin välillä on havainnollistettu kuvassa 2.

Kuva 2 Kuva, joka havainnollistaa yksinkertaisesti IBM Block CSI driverin toiminnan



Käydään lyhyesti läpi yksinkertaisen esimerkin kohdat CSI driverin käytöstä pysyvien voluumien luonnissa. RHEL-käyttöjärjestelmää käytettäessä tulee ensimmäiseksi varmistaa iSCSI-yhteys. Konfiguroidaan noodit FlashSystemin ohjeiden mukaisesti yhteyttä varten. Konfiguroidaan FlashSystemiin Kuberneteksen noodit ja yhteydet. Asennetaan CSI driver Kuberneteksen noodeille GitHubista. Luodaan ensimmäiseksi noodille Secret tai salaisuus, jossa määritetään tallennustilan kredentiaalit. Seuraavaksi luodaan StorageClass, jossa määritetään tallennustilan käyttöä. Sitten luodaan PersistentVolumeClaim, joka mahdollistaa pysyvien voluumien käytön ja antaa valita esimerkiksi käytettävän tilan suuruuden. Lopuksi luodaan StatefulSet tilallisia ohjelmia varten ja määritellään pysyvä voluumi käyttöön.

3.4.3 NFS

Konteille on kolme erilaista tallennustilan ylläpito tapaa. Konteilla on oma tiedostojärjestelmä, johon voidaan tallentaa tiedostoja. Konttien tiedostojärjestelmä on pystyssä vain niin kauan kuin konttikin. Voluumit, joita podi ja sen kontti käyttää on hiukan kestävämpi tallennustapa. Voluumit ovat pystyssä niin kauan kuin podikin. Pysyvät voluumit ovat reilusti kestävämpiä. Pysyvät voluumit ovat pystyssä niin kauan kuin klusterikin. Kolmesta tallennustavasta yksi on siis huomattavasti muita kestävämpi. Työssä keskitytään pysyviin voluumeihin, sillä ne tukevat levypalveluita. (Palmer, n.d.)

3.4.3.1 Perustietoa

NFS (Network File System) on Sun Microsystemsin kehittämä järjestelmä, joka mahdollistaa datan lukemisen, säilömisen, päivittämisen ja tiedostojen jakamisen etäkoneella, kuin se olisi paikallinen kone. NFS:n ensimmäinen versio oli tarkoitettu vain Sun Microsystemsin sisäiseen käyttöön, mutta toinen versio tuotiin myös julkiseen käyttöön. Toista versiota käytettiin tiedostojen jakamiseen UNIX-käyttöjärjestelmillä. Jokainen uusi versio on tuonut uusia ominaisuuksia ja parantanut vanhoja, kuten kasvattanut enimmäistiedostokokoa ja parantanut turvallisuutta. (Mesevage, 2019)

Käyttöönotettaessa Kuberneteksen pysyvissä voluumeissa NFS-palvelin tulee olla jo valmiiksi olemassa, sillä Kuberneteksen podit vain yhdistävät siihen käyttääkseen sen jakamaa kansiota pysyvänä voluumina. NFS tuo kaksi tärkeää ominaisuutta. NFS ei tuhoudu, vaikka sitä käyttävä podi tuhoutuisi. Lisäksi NFS:ää voi käyttää usea podi samaan aikaan, mikä mahdollistaa podien välisen datan siirron. Tämä mahdollistaa ohjelman, joka vaatii useamman palvelimen, toiminnan Kuberneteksessä. (Palmer, n.d.)

NFS:n yksi positiivinen ominaisuus on se, että autentikaatio voidaan suorittaa yksinkertaisesti IP:llä. Lisätään NFS:n tiedostoon palvelimen IP, jonka halutaan pystyvän yhdistämään NFS:ään. Muita positiivisia ominaisuuksia ovat esimerkiksi se, ettei manuaalista virkistystä tarvita uusien tiedostojen yhteydessä ja se että NFS voidaan turvata palomuurilla sekä Kerberoksella. (Mesevage, 2019) NFS:ään voidaan määritellä minkä UID:n käyttäjillä on oikeudet jaettuun kansioon. Mikäli oikeudet annetaan esimerkiksi UID:lle 300, tulee kaikilla jakoa käyttävillä koneilla olla käyttäjä, jolla UID on 300. (Nyffeneger, n.d.)

NFS:stä on olemassa useita eri versioita. Versiota NFS v3 käytetään eniten. NFS-asiakas-palvelimen protokolla alkaa mount-komennolla, joka määrittää asiakkaan ja NFS-palvelimen asetukset ja ominaisuudet. NFS v4 on tilallinen protokolla, jolla on yli 30 asetusta liittyen esimerkiksi kirjoituslohkon kokoon ja datan turvallisuuteen. Tärkeitä NFS-protokollan ominaisuuksia ovat myös välimuisti ja tiedostojen lukitus, jotka ovat yhteydessä toisiinsa. Molempien tulee olla spesifioitu oikein, jotta jaettuihin tiedostoihin pääsy onnistuu. Jos tiedoston data on ainoastaan palvelimen välimuistissa ja toinen palvelin yrittää lukea samaa

tiedostoa, luettu data voi olla väärin. Tämän takia jokaisella NFS:ään yhdistetyllä palvelimella tulee olla samat lukitus ja välimuisti asetukset. (Orlando, 2021)

NFS:llä on myös ongelmia, jotka vaikuttavat sen käytettävyyteen eri käyttötarkoituksissa. NFS perustuu RPC:iin (Remote Procedure Call), mikä ei ole kovinkaan turvallinen, minkä vuoksi NFS:ää tulisi käyttää ainoastaan palomuurilla suojatussa ja luotetussa verkossa. NFS-protokolla on erittäin puhelias. NFS-vaatii useita pieniä toimenpiteitä datan lukemiseen ja kirjoittamiseen, eikä siis toimi pienillä pyyntömäärillä. Vaikka NFS on perustasoltaan yksinkertainen, on tiedostojen jako usean palvelimen välillä monimutkaista johtuen vaaditusta lukituksen ja välimuistin konfiguraatiosta. Nykyisin tiedostojen rinnakkainen käyttö on usein pakollista. Ominaisuus on tullut vasta versioon NFS v4 eikä kovinkaan moni järjestelmä vielä tue sitä. Nykyinen NFS-protokolla kykenee siirtämään vain 1MB verran dataa yhden luku- tai -kirjoituspyynnön aikana. Tämä johtaa siihen, että pyyntöjä vaaditaan suuri määrä. (Orlando, 2021)

NFS:llä on lisäksi muutamia rajoituksia. Näitä ovat esimerkiksi serverin kyvyn rajoitukset, korkean saatavuuden puuttuminen ja suorituskyky ongelmat, kun usea asiakas/kone/podi yrittää samaan aikaan lukea palvelimelta ja/tai kirjoittaa palvelimelle. Tiedostojen yhteiskoko on kiinni palvelimen tallennuskapasiteetista. Korkean saatavuuden puuttuminen voi aiheuttaa helposti ongelmia, jos palvelin kaatuu. (Nyffeneger, n.d.)

3.4.3.2 NFS-perustesti

Seuraavaksi toteutetaan yksinkertainen esimerkki NFS-palvelimen käytöstä Kubernetes podin voluumina. Tämä tapa ei ole paras mahdollinen, mutta tarkoitus on käydä läpi NFS:n perustoiminta. Asennetaan aluksi NFS server Ubuntu 18.04 palvelimelle, joka on luotu vSphere ympäristöön ja joka ei kuulu Kubernetes-klusteriin. Päivitetään palvelimen pakettitiedot komennolla 34 ja asennetaan nfs-kernel-server-paketti komennolla 35.

Komento 34 apt-komento, jolla päivitetään pakettikirjasto

```
sudo apt update
```

Komento 35 apt-komento, jolla asennetaan nfs-kernel-server

```
sudo apt install -y nfs-kernel-server
```

Seuraavaksi luodaan kansio, joka jaetaan podien käyttöön. Luodaan komennolla 36 kansio NFS-palvelimelle. Poistetaan komennolla 37 kansion käyttörajoitukset ja annetaan komennolla 38 luku, kirjoitus ja ajo -oikeudet kansioon.

Komento 36 mkdir-komento, jolla luodaan uusi kansio /home/mount/nfs

```
sudo mkdir -p /home/mount/nfs
```

Komento 37 chown-komento, jolla määritetään, että kukaan ei omista kansiota /nfs/

```
sudo chown –R nobody:nogroup /home/mount/nfs/
```

Komento 38 chmod-komento, jolla määritetään kansiolle /nfs/ luku-, kirjoitus- ja ajo-oikeudet

```
sudo chmod 777 /home/mount/nfs/
```

Tiedosto /etc/exports sisältään NFS:n luvalliset yhteydet ja kongifuraatiot. NFS-palvelimen konfigurointi voidaan tehdä kahdella eri tavalla. Ensimmäinen tapa on editoida manuaalisesti konfiguraatiotiedostoa /etc/exports. Toinen tapa on lisätä halutut konfiguraatiot exportfs-komentoon. Editoidessa konfiguraatiotiedostoa, kirjoitetaan rivejä, joissa on jaettava kansio, isännän tai verkon IP ja halutut asetukset, tiedoston X mukaisesti. Tiedoston rivin perusmuoto on: ”export host(options)”. (Suchánek Marek et al., n.d.)

Avataan nano-tekstieditorilla tiedosto /etc/exports komennolla 39 ja lisätään tarvittava rivi (Tiedosto 9). Tiedostoon voidaan määritellä yksittäisiä palvelimia, joilla on oikeudet jaettuun kansioon, mutta tässä tapauksessa annetaan pääsy koko aliverkolle. Asetetaan luku- ja kirjoitusoikeudet, vaaditaan muutosten kirjoitus levylle ennen muutosten käyttöönottoa ja otetaan alipuun tarkistus pois käytöstä.

Komento 39 nano-komento, jolla avataan exports-tiedosto muokattavaksi tekstieditoriin

```
sudo nano /etc/exports
```

Tiedosto 9 /etc/exports-tiedosto, jossa määritellään NFS-jaon autentikaatiot

```
/mount/nfs <Aliverkon IP>/24(rw,sync,no_subtree_check)
```

Julkistetaan /mount/nfs-kansio komennolla 40 ja päivitetään konfiguraatio käynnistämällä nfs-kernel-server uudestaan (Komento 41).

Komento 40 exportfs-komento, joka vie kaikki exports-tiedoston konfiguraatiot

```
sudo exportfs -a
```

Komento 41 systemctl-komento, jolla uudelleenkäynnistetään nfs-kernel-server

```
sudo systemctl restart nfs-kernel-server
```

Lopuksi annetaan NFS:lle pääsy palomuurin läpi komennolla 42.

Komento 42 ufw-komento, jolla avataan NFS:n vaatimat portit

```
sudo ufw allow from <Aliverkon IP>/24 to any port nfs
```

Siirrytään Kubernetes-klusterin noodille. Luodaan .yaml-tiedosto, joka luo podin Kubernetekseen (Tiedosto 10) (Palmer, n.d.). Tiedosto luodaan normaalisti, mutta siihen lisätään volumes-osio ja volumeMounts-osio, joissa määritetään NFS-palvelimen jaetun kansion käyttö voluumina. Käytetään esimerkkinä hyvin yksinkertaista podia, joka käyttää alpine-kuvaa. Podiin määritetään komento, joka kirjoittaa tiedostoon date.txt viiden sekunnin välein päivämäärän ja kellonajan. Komennon avulla voidaan tarkistaa, että voluumi toimii.

Tiedosto 10 Podin, joka käyttää NFS-voluumia, määrittelytiedosto

```
kind: Pod
apiVersion: v1
```

```yaml
metadata:
  name: nfs-podi

spec:
  volumes:
    - name: nfs-voluumi
      nfs:
        server: <NFS-palvelimen IP>
        path: /home/mount/nfs

  containers:
    - name: app
      image: alpine

      volumeMounts:
        - name: nfs-voluumi
          mountPath: /var/nfs

      command: ["/bin/sh"]
      args: ["-c", "while true; do date >> /var/nfs/dates.txt; sleep 5; done"]
```

Podi luodaan normaalisti komennolla 43.

Komento 43 kubectl-komento, jolla otetaan .yaml-tiedoston määritys käyttöön, eli luodaan podi

```
kubectl apply -f podi.yaml
```

Voluumin toimivuus voidaan testata joko podin tai nfs-serverin kautta. Nfs-serverillä voidaan tarkistaa käymällä jaetussa kansiossa ja tarkastamalla date.txt-tiedosto. Podin kautta asia tarkastetaan esimerkiksi komennolla 44, jolla päästään podin ”sisälle”. Sitten ajetaan komento 45, joka näyttää date.txt-tiedoston sisällön komentorivillä.

Komento 44 kubectl-komento, jolla päästään käsiksi podiin

```
kubectl exec -it nfs-podi sh
```

Komento 45 cat-komento, jolla luetaan komentoriville dates.txt-tiedoston sisältö

```
cat /var/nfs/dates.txt
```

Komentoriville piirtyvät ajankohdat näyttävät voluumin, yhteyden ja NFS:n toimivan kuten pitääkin. Lisäämällä voluumin ja mountin podin .yaml-tiedostoon, tiedosto X:n tavoin, pystyy yhdistämään lisää podeja NFS serveriin ja toisiinsa. Esimerkin lisäksi on muitakin parempia, vaikkakin asteen monimutkaisempia tapoja käyttää NFS-palvelinta pysyvinä voluumeina.

Selkeä ja kätevä tapa on luoda Kubernetekseen pysyvä voluumi ja yhdistää se esimerkiksi podin tai käyttöönoton luontiin. Tehdään siis useampi askel, kuin aiemmassa esimerkissä. NFS-palvelin luodaan normaalisti. Käydään toinen esimerkki lyhyesti läpi.

Kubernetes-klusterissa luodaan pysyvä voluumi, joka konfiguroidaan yhdistämään NFS-palvelimen jaettuun kansioon. Pysyvä voluumi tehdään luomalla .yaml-tiedosto, jossa määritellään PersistentVolume (Tiedosto 11). Tiedostossa määritetään paljonko tallennustilaa voluumille annetaan, voluumin nimi ja NFS-palvelimen tiedot. Tiedosto ajetaan normaalisti kubectl:n create-komennolla. Seuraavaksi luodaan .yaml-tiedosto, jossa määritetään PersistentVolumeClaim (Tiedosto 12). Tiedostossa määritellään yhden voluumin käyttöönoton saava tallennustila, PersistentVolumeClaimin nimi ja luodun pysyvän voluumin nimi. PersistenVolumeClaim luodaan normaalisti kubectl:n create komennolla.

Lopuksi luodaan normaalisti Deployment ja lisätään siihen kohdat volumes ja volumeMounts, kuten aiemmassa esimerkissä. Tiedostossa 11 nähtävillä miten kohdat eroavat aiemman esimerkin vastaavista. Pääsääntöinen ero on, että määritetään PersistentVolumeClaim eikä NSF-palvelin. Käyttöönotto otetaan normaalisti käyttöön kubectl:n create-komennolla ja testaus voidaan tehdä esimerkiksi samalla tavalla kuin aiemmassa esimerkissä. Tätä tapaa voidaan soveltaa myös esimerkiksi käyttämällä StorageClass-objektia PersistentVolumen sijaan.

Tiedosto 11 PersistentVolume-tiedosto

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-voluumi
  labels:
    volume: nfs-PV-voluumi
spec:
```

```
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    Retain
  nfs:
    server: <NFS-palvelin IP>
    path: /home/mount/nfs
    readOnly: false
```

Tiedosto 12 PersistentVolumeClaim-tiedosto

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-PV-claim
spec:
  selector:
    matchLabels:
      volume: nfs-PV-voluumi
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

Tiedosto 13 Voluumiin liittyvä osa käyttöönotto-tiedostosta.

```
spec:
  volumes:
    - name: NFS-testi
      persistentVolumeClaim:
        claimName: nfs-PV-claim
  containers:
  - name: NFS-testi
    image: alpine
    volumeMounts:
      - name: NFS-testi
        mountPath: /var/nfs
```

3.4.4 Levypalveluiden vertailu

Vuoden 2019 jälkeen löytyy vain vähän tietoa liittyen IBM Storage Enabler for Containersiin, joten saattaa olla, että sitä ei tueta enää. Vuoden 2019 versio voi olla varsin hyväkin ja sopia

käyttötarkoitukseen. Mutta tiedon puutteen ja tulevan tuen mahdollisen puutteen takia kyseinen vaihtoehto voi olla huono. Vaikuttaisikin siltä, että IBM Block CSI driver on tullut korvaamaan Storage Enabler for Containersin. CSI driverista löytyy tietoa ja ohjeita, jotka on kirjoitettu lähiaikoina.

NFS on perustasoltaan yksinkertainen ymmärtää ja helppo käyttää. Kuitenkin NFS on verrattuna IBM:n ratkaisuihin hyvin rajoittunut. Suurimpana ongelmana on varmasti se, että NFS ei tue korkeaa saatavuutta. NFS-palvelimen kaatuessa tulee nopeasti paljonkin ongelmia Kubernetes-klusterille, kun pysyvät voluumit eivät toimikaan enää. NFS on myös vanha teknologia, johon tulee virallisia päivityksiä suhteellisen hitaasti, eivätkä kaikki järjestelmät tue uusinta versiota. Toisaalta NFS:ää on testattu paljon, joten sen toiminta on tunnettua ja siten suhteellisen luotettavaa. IBM:n FlashSystem-järjestelmät ovat rakennettu korkeaa saatavuutta ajatellen, joten oikein konfiguroituna samaa ongelmaa ei ole kuin NFS:ää käytettäessä.

FlashSystem-järjestelmät ovat valmiiksi jo pitkälle konfiguroituja ja laitteina ovat nimenomaan tarkoitettu käytettäväksi tallennustilana esimerkiksi voluumeja ajatellen. Ohjeistusta löytyy myös paljon ja IBM tarjoaa tukea asennukseen, käyttöönottoon ja ongelmatilanteisiin. Toisaalta se on myös huono puoli, että IBM:n tuotteet vaativat FlashSystem-järjestelmien tai muiden IBM:n järjestelmien käyttöä. Järjestelmät saattavat tulla hintaviksi ja ne myös sitouttavat. Yrityksen tuleekin pohtia haluaako se käyttää näitä IBM:n järjestelmiä.

NFS:n käytöstä, konfiguraatiosta ja asennuksista löytyy hyvin käyttäjien luomaa tietoa ja esimerkkejä. IBM:n ratkaisuista taas löytyy paljon virallista tietoa ja ohjekirjoja, mutta ei oikeastaan lainkaan käyttäjien kirjoituksia tai esimerkkejä. IBM:n sivustot, joista tieto löytyy ovat myös hyvin sekavat ja tietoa on ripoteltu ympäriinsä. Olisi myös paljon kätevämpää tutkia ratkaisuja, jos niiden kaikki tiedot, kuten tukitiedot olisivat ajankohtaiset ja selvästi kirjoitetut. IBM:n ratkaisuja pohtiessa onkin todennäköisesti tärkeää ottaa yhteyttä palvelutukeen ja kysyä kaikkea mikä kirjoituksissa jää epäselväksi.

4 Johtopäätökset ja pohdinta

Kuberneteksen käyttöönotto yrityksessä vaatii paljon tutkimista sekä suunnittelua. Opinnäytetyö raapaisee vain osaa kokonaisuudesta, joka on valtava. Kuitenkin opinnäytetyö tuo hyvin tietoa liittyen IaC-tyyliseen käyttöönottoon, salaisuuksiin sekä levypalveluihin, mikä on ollut tarkoituskin.

Terraform vaikuttaa erittäin pätevältä tavalta hoitaa klusterin käyttöönotto vSpheressä. Opinnäytetyössä tarkasteltiin, miten Kubernetes-klusteri asennetaan palvelimelle ja seuraavaksi tutustuttiin miten Packerin ja Terraformin avulla saadaan Kubernetes-klusteri vSphere-ympäristöön. Packerilla ja Terraformilla testattu käyttöönotto kuvattiin opinnäytetyöhön ohjeen kaltaisessa muodossa. Tavallisen klusterin luominen on suhteellisen yksinkertainen projekti, mutta lisävaatimusten ja lisätiedon tullessa tarvitaan esimerkiksi reilusti enemmän .sh-skriptejä. On myös epävarmaa, pystyykö kaiken tarpeellisen tekemään skripteillä vai joutuuko joitain asioita pakosti tekemään tai lisäämään manuaalisesti. Yksi tällainen tapaus on jo työntekijänoodien liittäminen klusteriin. Lisäselvityksellä ratkaisu voidaan tietenkin löytää ongelmaan. Mitä enemmän Terraformilla pystytään tekemään, sitä enemmän siitä on hyötyä käyttöönoton nopeuttamisessa, selkeyttämisessä ja yhtenäistämisessä. Terraformia voidaan myös myöhemmin käyttää esimerkiksi ohjelmien käyttöönoton automaatiossa Kubernetes-klusteriin.

Salaisuudet ovat tärkeässä asemassa, kun pohditaan Kubernetes-klusterin turvallista käyttöönottoa sekä tulevaisuuden käyttöä. Opinnäytetyössä vertailtiin kolmea erilaista vaihtoehtoa. Opinnäytetyössä vertailtiin vaihtoehtoja ensin tutkimalla faktoja ja sen jälkeen vertailemalla ilmi tulleita asioita. Yksi vaihtoehdoista ei saavuttanut tavoitteita läheskään yhtä hyvin kuin loput tutkitut vaihtoehdot. Kubernetes Secrets jää selvästi heikompaan asemaan kuin kaksi muuta vaihtoehtoa. Hashicorpin Vault ja Akeyless tarjoavat tärkeitä ominaisuuksia esimerkiksi skaalauksen ja turvallisuuden puolesta. Opinnäytetyössä ilmi tulleet tiedot toivottavasti auttavat päättämään minkä salaisuuksienhallintajärjestelmän yritys ottaa lisäselvitykseen ja lopulta käyttöön. Mahdollisesti tärkein kysymys päätöksessä on se, onko yritys valmis maksamaan palvelusta vai onko ilmaisuus tai avoin lähdekoodi tärkeä asia.

Salaisuuksienhallintajärjestelmien lisäksi opinnäytetyössä tuli tutustua levypalveluihin ja tutkia vaihtoehtoja. Opinnäytetyössä tutkittiin kolmea vaihtoehtoa, joita lopuksi vertailtiin. Vaihtoehdoista kaksi ovat hyvin lähellä toisiaan, sillä kumpikin on IBM:n samankaltaista teknologiaa. Kolmas vaihtoehto NFS on hyvin erilainen kuin kaksi muuta vaihtoehtoa ja vaikuttaa köyhältä verrattuna kahteen muuhun vaihtoehtoon, vaikkakin myös helppokäyttöisemmältä. IBM:n tuotteiden erot ovat pääsääntöisesti siinä montako palvelinta vaaditaan ja siinä onko tuki jatkuvaa. Toinen tuotteista vaikuttaa jääneen pois tuesta, kun taas toisesta löytyy hyvin tietoa ihan lähiviikoiltakin. Vaikuttaa siis siltä, että mikäli yritys on valmis käyttämään FlashSystem-järjestelmää ja IBM:n ohjelmia, IBM block CSI driver on järkevin vaihtoehto. Tätä asiaa tulee yrityksen sisäisesti vielä pohtia ja tehdä lisäselvitystä, mutta toivon mukaan opinnäytetyöhön kerätty tieto auttaa päätöksen teossa. Jos mitään vaihtoehdoista ei oteta käyttöön, tulee etsiä uusia mahdollisia vaihtoehtoja.

Opinnäytetyön tilaajan palaute oli positiivista. Opinnäytetyöstä on hyötyä yritykselle sekä Kubernetes-projektissa että työn aiheisiin liittyvissä koulutuksissa. Tilaaja sai uutta ja hyödyllistä tietoa erityisesti Terraformiin liittyvässä osuudessa, sillä sitä ei vielä ollut edistetty kovinkaan paljoa. Muistakin aiheista opinnäytetyö tuo esille hyödyllistä tietoa sekä konkreettisia esimerkkejä. Tilaajan mukaan opinnäytetyö on suoraan hyödynnettävissä yrityksen sisäisessä kehitysprojektissa sekä toimii toimivana pohjana konseptin jatkokehittämiselle.

5 Yhteenveto

Tutkimuskysymyksiä oli kolme, ja ne keskittyivät varsin laajoihin aiheisiin. IaC-aiheeseen tutustuttiin konkreettisella esimerkillä siitä, miten Kubernetes-ympäristön pystytys VMwareen onnistuu Terraformilla. Esimerkki on todettu toimivaksi ja hyödylliseksi tulevaisuutta ajatellen. Yritys sai pystyyn yksinkertaisen Kubernetes-ympäristön, jota voidaan lähteä laajentamaan tulevaisuudessa ominaisuuksien ja turvallisuuden osalta.

Toinen sekä kolmas tutkimuskysymys keskittyivät erilaisten ratkaisujen etsimiseen ja vertailuun. Opinnäytetyössä vertailtiin erilaisia levypalveluita ja salaisuuksienhallintaohjelmia. Vaihtoehdot ovat yleisesti suosittuja, joten tiedon saaminen niistä oli tärkeää. Vertailu auttaa valitsemaan käyttääkö yritys yhtä vaihtoehdoista vai siirrytäänkö tutkimaan muita vaihtoehtoja. Tutkimuskysymyksiin vastaaminen onnistui hyvin, sillä opinnäytetyö sisältää konkreettisia esimerkkejä, tietoa ja vertailua, joka auttaa yritystä eteenpäin jatkamisessa.

Opin opinnäytetyötä tehdessä useista eri teknologioista ja tuotteista. Salaisuuksienhallinta sekä levypalvelut esimerkiksi eivät olleet lainkaan tuttuja minulle. Työtä tehdessä ymmärrys kasvoi ja erilaisten vaihtoehtojen tutkiminen sekä vertailu lisäsi ymmärrystä entisestään. Työn päätarkoitus oli auttaa Kubernetes-ympäristön pystyttämisessä, mutta työssä pääsääntöisesti tutustuttiin useisiin muihin teknologioihin. Aihe oli erittäin laaja ja erilaisia teknologioita sekä tuotteita oli monta. Suurikokoisen projektin työstäminen ja kirjoittaminen opetti myös ajanhallinnasta, aiheenvalinnasta sekä tiettyjen asioiden priorisoinnista. Valintoja oli pakko tehdä jatkuvasti projektin aikana, jotta projekti muodostui mahdollisimman järkevästi ja aika ei loppunut kesken. Konttiteknologian potentiaali kasvoi mielessäni työtä tehdessä.

Opinnäytetyö toimii aloituksena isommalle projektille, jonka tarkoituksena on pystyttää Kubernetes-ympäristö yritykselle. Ympäristön tulee toteuttaa yrityksen asettamat vaatimukset. Työ tuo tarpeellista tietoa yrityksen tarpeisiin. Seuraavaksi voidaan tehdä valinnat ja lähteä ympäristön pystyttämiseen konkreettisemmassa mielessä.

Lähteet:

ahardin-rh, adellape, vikram-redhat, Poitras, T., markturansky, bfallonf, & tnguyen-rh. (2016). *Persistent Storage - Additional Concepts | Architecture | OpenShift Enterprise 3.1*. https://docs.openshift.com/enterprise/3.1/architecture/additional_concepts/storage.html

Avi. (2020, November 28). *An Introduction to Terraform for Beginners*. https://geekflare.com/terraform-for-beginners/

HashiCorp. (2021). https://www.vaultproject.io/docs/configuration/storage/consul

Docker Inc. (n.d.). Retrieved May 17, 2021, from https://docs.docker.com/get-started/overview/

Ellingwood, J. (2018, May 2). *An Introduction to Kubernetes | DigitalOcean*. https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

HashiCorp. (2021). https://www.vaultproject.io/docs/configuration/storage/filesystem

Gillis, A. S. (2021, February). *What is Terraform?* https://searchitoperations.techtarget.com/definition/Terraform

HashiCorp. (n.d.). Retrieved June 8, 2021, from https://www.hashicorp.com/products/vault/pricing

Hoffman, B. (2021, March 4). *What is Secrets Management and why is it so important?* https://thycotic.com/company/blog/2021/03/04/secrets-management/

IBM Cloud Education. (2019, November 4). *What is Kubernetes? | IBM*. https://www.ibm.com/cloud/learn/kubernetes

IBM Cloud Education. (2020, February 13). *What is Terraform? | IBM*. https://www.ibm.com/cloud/learn/terraform#toc-terraform--Z5Eg4toU

Akeyless. (n.d.). Retrieved June 7, 2021, from https://www.akeyless.io/integrations/

HashiCorp. (2021). https://www.consul.io/docs/intro

HashiCorp. (n.d.). Retrieved June 7, 2021, from https://www.vaultproject.io/docs/what-is-vault

IBM. (2021). https://www.ibm.com/docs/en/stg-block-csi-driver/1.5.0?topic=introduction

IBM. (2021). https://www.ibm.com/docs/en/spectrum-connect/3.8.0?topic=SS6JWS_3.8.0/UG/sc_ug_sc_intro.html

Kubernetes Advocate. (2021, February 16). *Secrets Management in Kubernetes. Kubernetes Secrets are secure objects… | by Kubernetes Advocate | AVM Consulting Blog | Medium*. https://medium.com/avmconsulting-blog/secrets-management-in-kubernetes-378cbf8171d0

IBM. (2019). https://www.ibm.com/docs/en/stgenablercontainers/2.1.0?topic=notes-limitations

Lowe, D. (n.d.). *What Is vSphere? - dummies*. Retrieved June 2, 2021, from https://www.dummies.com/programming/networking/what-is-vsphere/

Luksa, M. (2018). *Kubernetes in Action*.

Docker. (n.d.). Retrieved May 17, 2021, from https://docs.docker.com/develop/develop-images/image_management/

Mesevage, T. G. (2019, May 23). *What is Network File System (NFS) File Share?* https://www.datto.com/blog/what-is-nfs-file-share

nishanil, sughosneo, mthalman, mvelosop, sguitardude, mairaw, nschonni, & ThePiranha. (2021, January 13). *Docker terminology | Microsoft Docs*. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-terminology

Nyffeneger, R. (n.d.). *NFS - Network File System*. Retrieved May 31, 2021, from https://renenyffenegger.ch/notes/Linux/filesystem/NFS

Orlando, L. (2021, April 15). *What is Network File System? | Benefits, Accelerating NFS Performance*. https://www.weka.io/learn/what-is-network-file-system/

Palmer, M. (n.d.). *Kubernetes Volumes Guide – Examples for NFS and Persistent Volume - Kubernetes Book*. Retrieved May 26, 2021, from https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-volumes-example-nfs-persistent-volume.html

Akeyless. (2021, March). https://docs.akeyless.io/docs/akeyless-overview

Akeyless. (n.d.). Retrieved June 7, 2021, from https://www.akeyless.io/pricing/

Akeyless. (2021, June 1). https://docs.akeyless.io/docs/how-to-provision-secret-to-your-k8s

Raghuram, S. (2017, February 23). *4 reasons you should use Kubernetes | InfoWorld*. https://www.infoworld.com/article/3173266/4-reasons-you-should-use-kubernetes.html

Rubens, P. (2017, June 27). *What are containers and why do you need them? | CIO*. https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html

Kubernetes authors. (2021). https://kubernetes.io/docs/concepts/configuration/secret/

HashiCorp. (2021). https://www.vaultproject.io/docs/configuration/storage

Suchánek Marek, Bhide Apurva, Navrátil Milan, East Jacquelynn, Domingo Don, Bacik Josef, Dudka Kamil, Goede Hans de, Hoyer Harald, Keefe Dennis, Ledford Doug, Novotny Daniel, Straz Nathan, Walsh Andy, Wysochanski David, Christie Michael, Prabhu Sachin, Evers Rob, Howells David, … Snitzer Mike. (n.d.). *8.6. Configuring the NFS Server Red Hat Enterprise Linux 7 | Red Hat Customer Portal*. Retrieved May 31, 2021, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/nfs-serverconfig

IBM. (2021). https://www.ibm.com/docs/en/stg-block-csi-driver/1.5.0?topic=requirements-supported-operating-systems

IBM. (2019). https://www.ibm.com/docs/en/stgenablercontainers/2.1.0?topic=requirements-supported-operating-systems

IBM. (2019). https://www.ibm.com/docs/en/stgenablercontainers/2.1.0?topic=requirements-supported-orchestration-platforms

IBM. (2021). https://www.ibm.com/docs/en/stg-block-csi-driver/1.5.0?topic=requirements-supported-storage-systems

IBM. (2019). https://www.ibm.com/docs/en/stgenablercontainers/2.1.0?topic=requirements-supported-storage-systems

HashiCorp. (2021). https://www.vaultproject.io/docs/platform/k8s/csi

Wallenius, N. (2019, November 6). *Konttiteknologia - mitä kontit tarkoittavat ja mitä hyötyä niistä on*. https://niklaswallenius.fi/konttiteknologia-mita-hyotya/

Wallenius, N. (2020, November 16). *Mikä on Docker ja mitä hyötyä siitä on? – Liiketoimintalähtöistä IT-konsultointia*. https://niklaswallenius.fi/mika-on-docker/

Akeyless. (2021, February). https://docs.akeyless.io/docs/zero-knowledge

Liite 1: Aineistonhallintasuunnitelma

Projektin aikana kerätään ja kirjoitetaan aineistoa, johon kerätään teknistä ja teoreettista tietoa projektista. Tämä tieto analysoidaan opinnäytetyötä varten. Aineistoa säilytetään tekijän työtietokoneen C-asemalla, joka sijaitsee yrityksen tiloissa, ja siitä tehdään säännöllisesti varmuuskopioita muistitikulle, joka pidetään yrityksen tiloissa. Aineistoa säilytetään C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta.

Aineisto sisältää yrityksen autentikaatioon liittyvää tietoa sekä muuta yrityksen sisäistä ja yksityistä tietoa. Kyseisen kaltainen tieto pidetään yrityksen verkossa/laitteissa. Opinnäytetyöhön laitetusta materiaalista siivotaan kaikki yrityksen sisäinen ja yksityinen tieto. Pääosa dokumentoiduista tuloksista kirjoitetaan myös yrityksen sisäiseen intraan ja versionhallintaan.

Liite 2: Ubuntu-18.json -tiedosto

```
{
    "builders": [
      {
        "type": "vsphere-iso",

        "vcenter_server": "{{user `vcenter_server`}}",
        "username": "{{user `username`}}",
        "password": "{{user `password`}}",
        "insecure_connection": "true",
        "host": "{{user `hosti`}}",
        "vm_name": "vmNimi",

        "CPUs":             2,
        "RAM":              2048,
        "RAM_reserve_all": false,
        "disk_controller_type":  "pvscsi",
        "storage": [
            {
                "disk_size": 32768,
                "disk_thin_provisioned": true
            }
        ],

        "floppy_files": [
          "./preseed.cfg"
        ],

        "boot_command": [
          "<enter><wait><f6><wait><esc><wait>",
          "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
          "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
          "<bs><bs><bs>",
          " priority=critical",
          " locale=fi_FI",
          " file=/media/preseed.cfg",

          "<enter>"
        ]
      }
    ],

    "provisioners": [
      {
        "type": "shell",
```

```
        "inline": ["sudo apt dist-upgrade -y"]
      }
    ]
  }
```

Liite 3: preseed.cfg -tiedosto

```
d-i auto-install/enable boolean true
d-i debconf/priority select critical

d-i debian-installer/locale string fi_FI.UTF-8
d-i localechooser/supported-locales select fi_FI.UTF-8
d-i console-setup/ask_detect boolean false
```

```
d-i clock-setup/utc boolean true
d-i time/zone string Etc/UTC
d-i clock-setup/ntp boolean true
```

```
d-i tasksel/first multiselect none
d-i pkgsel/include string open-vm-tools openssh-
server git python python3 vim tmux software-properties-common python3-apt
d-i pkgsel/upgrade select full-upgrade
```

```
d-i finish-install/reboot_in_progress note
```

Liite 4: Ensimmäinen main.tf-tiedosto

```
provider "vsphere" {
    user = var.vsphereuser
    password = var.vspherepass
    vsphere_server = var.vsphere-vcenter
}

data "vsphere_datastore_cluster" "datastore_cluster" {
    name = var.vm-datastore
    datacenter_id = data.vsphere_datacenter.dc.id
}

resource "vsphere_virtual_machine" "control" {
    count = var.control-count
    name = "${var.vm-prefix}_k8sControl"

    num_cpus = var.vm-cpu
    memory = var.vm-ram
    guest_id = var.vm-guest-id

    network_interface {
        network_id = data.vsphere_network.network.id
    }

resource "vsphere_virtual_machine" "worker" {
    count = var.worker-count
    name = "${var.vm-prefix}-k8sWorker${count.index + 1}"

    num_cpus = var.vm-cpu
    memory = var.vm-ram

    network_interface {
        network_id = data.vsphere_network.network.id
    }

output "control_ip_addresses" {
 value = vsphere_virtual_machine.control.*.default_ip_address
}
```

Liite 5: Ensimmäinen variables.tf-tiedosto

```
variable "vsphereuser" {
    type = string
}
variable "vspherepass" {
    type = string
}
```

```
variable "vsphere-datacenter" {
    type = string
}
variable "vsphere-cluster" {
    type = string
}
variable "control-count" {
    type = string
    description = "Amount of the control VM's"
    default     =  1
}
```

Liite 6: Toinen main.tf-tiedosto

```
provider "vsphere" {
  vsphere_server = "${var.vsphere_vcenter}"
  user           = "${var.vsphere_user}"
  password       = "${var.vsphere_password}"


}

data "vsphere_datacenter" "dc" {
  name = "${var.vsphere_datacenter}"
}


data "vsphere_virtual_machine" "template" {
  name          = "${var.vsphere_vm_template}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_resource_pool" "resource_pool" {
  name          = "${var.vsphere_vm_resource_pool}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

Liite 7: terraform.tfvars-tiedosto

```
# vCenter yhteys
vsphere_vcenter = "vCenter"
vsphere_user = "vSphereTunnus"
vsphere_password = "vSphereSalasana"


# VM-tiedot
vsphere_datacenter = "DatacenterNimi"
vsphere_vm_folder = "KansioNimi"
vsphere_vm_name = "ControlKoneenNimi"
vsphere_vm_template = "VMtemplateNimi"
vsphere_vm_resource_pool ="ResourcePoolNimi"


vsphere_vcpu_number = "2"
vsphere_memory_size = "3500"
vsphere_datastore_cluster = "DatastoreKlusteriNimi"
vsphere_port_group = "NetworkNimi"
vsphere_ipv4_address = "ControlIP"
vsphere_ipv4_netmask = "netmask(24)"
vsphere_ipv4_gateway = "Gateway"


vsphere_time_zone = "UTC"


# K8s ja työntekijänoodit
vsphere_k8_nodes = "2(Nodejen määrä)"
vsphere_k8_version = "1.15.3(KubernetesVersio)"
```

Liite 8: Kubernetes-master.tf-tiedosto

```
resource "vsphere_virtual_machine" "K8-MASTER" {
  name             = "${var.vsphere_vm_name}"
  resource_pool_id = "${data.vsphere_resource_pool.resource_pool.id}"
  datastore_cluster_id     = "${data.vsphere_datastore_cluster.datastore_cluster.id
}"
```

```
  # Resurssit
  num_cpus = "${var.vsphere_vcpu_number}"
  memory   = "${var.vsphere_memory_size}"

  # OS
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"

  # Tallennustila
  disk {
    label            = "${var.vsphere_vm_name}.vmdk"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
```

```
  # VM
  clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"

    customize {
      linux_options {
        host_name = "${var.vsphere_vm_name}"
        domain    = "${var.vsphere_domain}"
      }
```

```
    }
  }

    # Kubernetes
    provisioner "file" {
        source      = "configure_part1.sh"
        destination = "/tmp/configure_part1.sh"

        connection {
            type     = "ssh"
            user     = "root"
            password = "${var.vsphere_vm_password}"
            host = "${var.vsphere_vm_host2}"
        }
    }
}
```

Liite 9: Kubernetes-node.tf-tiedosto

```
resource "vsphere_virtual_machine" "K8-NODE" {

  # Noodien määrä

  count = "${var.vsphere_k8_nodes}"

  name             = "${var.vsphere_vm_name_k8node}${count.index + 1}"
  resource_pool_id = "${data.vsphere_resource_pool.resource_pool.id}"
  datastore_cluster_id     = "${data.vsphere_datastore_cluster.datastore_cluster.id
}"
  folder           = "${var.vsphere_vm_folder}"
  tags             = ["${data.vsphere_tag.tag.id}"]


  # Resurssit
  num_cpus = "${var.vsphere_vcpu_number}"
  memory   = "${var.vsphere_memory_size}"

  # OS
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"
```

```
  # VM
  clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"

    customize {
      linux_options {
        host_name = "${var.vsphere_vm_name_k8node}${count.index + 1}"
        domain    = "${var.vsphere_domain}"
      }

      network_interface {
        ipv4_address = "${var.vsphere_ipv4_address_k8node_network}${"${var.vsphere_
ipv4_address_k8node_host}" + count.index}"
        ipv4_netmask = "${var.vsphere_ipv4_netmask}"
      }

      ipv4_gateway    = "${var.vsphere_ipv4_gateway}"
      dns_server_list = ["${var.vsphere_dns_servers}"]
      dns_suffix_list = ["${var.vsphere_domain}"]
    }
  }

provisioner "file" {
        source      = "configurek8node_part1.sh"
```

```
        destination = "/tmp/configurek8node_part1.sh"

        connection {
            type     = "ssh"
            user     = "root"
            password = "${var.vsphere_vm_password}"
            host = "${var.vsphere_vm_host2}"
        }
    }
```

Liite 10: configure_part2.sh-tiedosto

```
systemctl enable docker && systemctl start docker
systemctl enable kubelet && systemctl start kubelet

cat << EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

sysctl --system
```

Liite 11: configure_part3.sh-tiedosto

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

export kubever=$(kubectl version | base64 | tr -d '\n')
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"
kubectl get pods --all-namespaces
kubectl get nodes
```

Liite 12: configurek8node_part2.sh-tiedosto

```
systemctl enable --now kubelet

yum install -y docker

systemctl enable docker && systemctl start docker

sysctl --system

echo '1' > /proc/sys/net/ipv4/ip_forward
echo "1" > /proc/sys/net/bridge/bridge-nf-call-iptables
```

Liite 13: Toinen variables.tf-tiedosto

```
variable "vsphere_user" {
  description = "vSphere username"
}

variable "vsphere_password" {
  description = "vSphere password"
}

variable "vsphere_vcenter" {
  description = "vCenter server name or IP"
}
```

```

variable "vsphere_time_zone" {
  description = "Timezone of the VM"
  default     = "UTC"
}

variable "vsphere_vm_host" {
  description ="hosti"
}
```