Tampereen yliopisto

Valtteri Kostiainen

KUBERNETEKSEN KÄYTTÖÖNOTTO NUTANIX-YMPÄRISTÖSSÄ

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastajat: David Hästbacka
Kari Systä
Marraskuu 2021

TIIVISTELMÄ

Valtteri Kostiainen: Kuberneteksen käyttöönotto Nutanix-ympäristössä
Diplomityö
Tampereen yliopisto
Tietotekniikka, DI
Marraskuu 2021

Karbon on Nutanixin Kubernetes-ratkaisu, joka luo ja konfiguroi Kubernetes-klusterin automaattisesti käyttäjää varten. Karbon yksinään ei kuitenkaan tarjoa samalla tavalla kokonaisratkaisua, kuten esimerkiksi Amazonin Elastic Kubernetes Service. Tässä työssä on toteutettu osa kohdeorganisaation tunnistamista tarpeista, jotta Karbonia voitaisiin käyttää tuotantoympäristönä.

Työssä toteutettiin reititys ulko- ja sisäverkon palveluille, lokien kerääminen keskitettyyn ympäristöön ja varmuuskopiointi Kubernetes-resursseille. Ulkoverkon reitityksessä päädyttiin käyttämään käänteisproxynä Traefikkia, joka integroituu natiivisti Kubernetekseen. Sisäverkon liikenteessä käyttöönotettiin yksinkertainen MetalLB, joka on tarkoitettu yksityispilviin. Varmuuskopioinnissa otettiin käyttöön kolmesta eri teknologiasta koostuva yhdistelmä. MinIO-objektivarasto toimii varmuuskopioiden säilytyspaikkana. Velero varmuuskopioi Kubernetes-oliot ja Restic varmuuskopioi levyillä olevan datan osana Veleron normaalia varmuuskopiointisykliä. Lokien keskittäminen toteutettiin Elastic-pinolla ja Fluent Bitillä.

Avainsanat: Kubernetes, pilvi, reititys, varmuuskopio, loki

ABSTRACT

Valtteri Kostiainen: Deploying Kubernetes into Nutanix environment
Master's thesis
Tampere University
Information technology
November 2021

Karbon is Kubernetes solution from Nutanix. Karbon creates a Kubernetes cluster in a turn-key manner for the client to use. Karbon is not however a similar complete solution like Amazon's Elastic Kubernetes Servie. Some of the key requirements for using Karbon in production were implemented as a part of this thesis.

We implemented networking for services reachable from the public internet and services only available in the organization private network. We used Traefik as the routing technology for the public services, Traefik is a cloud native proxy and integrates to Kuberentes. For the private network we used MetalLB which is ment for bare metal Kubernetes deployments. Backups required us to use three different tools in combination. MinIO is used for storing the backups. Velero is used backing up the Kubernetes resources and Restic is used for backinup volumes. We used the Elastic stack and Fluent Bit for centralized logging.

Keywords: Kubernetes, cloud, networking, backup, log

ALKUSANAT

Haluan kiittää työni kohdeorganisaatiota, Dicodea mahdollisuudesta toteuttaa tämä työ ja erityisesti Janne Sikiötä. Kiitän myös ohjaajaani David Hästbackaa. Ilman hänen apuaan en olisi onnistunut saamaan työtä sen formaatin vaatimiin raameihin. Haluan kiittää myös isääni, joka oli tukenani koko kirjoitusprosessin ajan.

Tämä työ on omistettu äitini Riitta Kostiaisen muistolle.

Tampereella, 7. marraskuuta 2021

Valtteri Kostiainen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

OHJELMA- JA ALGORITMILUETTELO

LYHENTEET JA MERKINNÄT

ACME	Automatic Certificate Management Environment
AES	Advanced Encryption Standard
API	Application Programming Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BaaS	Backend-as-a-Service
BASE	Basically Available, Soft state, Eventual consistency
BSD	Berkeley Software Distribution
CD	Continous Delivery
CI	Continous Integration
CNCF	Clouda Native Computing Foundation
CNI	Container Network Interface
CRUD	Create, Read, Update and Delete
CSI	Container Storage Interface
DNS	Domain Name Service
DSL	Domain Specific Language
EBS	Elastic Block Store
EC	Erasure Coding
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
EKS	Elastic Kubernetes Service
FaaS	Function-as-a-Service
GCP	Google Cloud Platform
HostSNI	Host Server Name Indication
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines Corporation
IP	Internetin protokollaosoite
JSON	JavaScript Object Notation

LPR	Least pending requests
MAC	Media Access Control
OSI	Open Systems Interconnection reference
REST	Representational State Transfer
S3	Simple Storage Service
SFTP	SSH File Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier

1 JOHDANTO

Ohjelmistot ovat tulleet kiinteästi osaksi kaikkien ihmisten arki- ja työelämää viimeistään 2000-luvulla. Tästä johtuen ohjelmistoihin kohdistuu yhä enemmän vaatimuksia käyttäjien puolelta: palvelujen oletetaan olevan saavutettavissa 24/7, uusien ominaisuuksien odotetaan tulevan yhä nopeammin käyttöön, sovelluksen pitäisi olla käytettävissä useammalla alustalla ja toimivan mahdollisimman nopeasti riippumatta siitä, missä päin maailmaa käyttäjä on. [1]

Edellä mainittujen toiveiden täyttäminen asettaa runsaita vaatimuksia infrastruktuurille, jonka päällä sovelluksia ajetaan. Vaatimus siitä, että käyttäjän maantieteellinen sijainti ei vaikuta sovelluksen toimintaan johtaa siihen, että sovelluksen täytyy olla ajossa mahdollisimman lähellä käyttäjää eikä pelkästään esimerkiksi Suomessa tai Yhdysvalloissa. Maantieteellisen hajautuksen takia sovellusten käyttämien resurssien määrä voi vaihdella voimakkaasti riippuen kellonajasta, viikonpäivästä tai kuukaudesta. Sovellus ei myöskään saa olla sidottuna yksittäisiin fyysisiin laitteisiin, jotta laiterikot eivät aiheuta katkoksia toimintaan. [1, 2]

Näitä tarpeita varten on kehitetty ratkaisuja sekä teknologioiden muodossa että uusina ohjelmistokehityksen suuntauksina. Infrastruktuurin puolella pilvipalveluiden suosio on kasvanut voimakkaasti viimeisen viiden vuoden aikana. Hyödyntämällä pilvipalvelutarjoajien alustoja sovelluksia voidaan ajaa missä päin maailmaa tahansa, vikasietoisissa ja skaalautuvissa ympäristöissä ilman, että pilvipalvelujen käyttäjien tarvitsee itse huolehtia virtualisoinnista tai fyysistä laitteista. Infrastruktuurin täysimittainen hyödyntäminen vaatii muutoksia ohjelmistojen rakenteeseen ja siihen miten niitä kehitetään. Esimerkkejä pilvinatiiveista arkkitehtuureista ovat mikropalveluarkkitehtuuri ja palveliton (serverless) arkkitehtuuri. [1, 2, 3]

Microsoftin insinöörin Bill Baker esitti vuonna 2012 lemmikit ja karja-analogian liittyen palvelujen skaalautuvuuteen, joka on myöhemmin laajennettu analogiaksi vanhoista palvelinratkaisuista ja uudemmista pilviteknologioista [4]. Kohdeorganisaation nykyiset virtuaalikoneet ovat analogian lemmikkejä, jotka vaativat jatkuvaa yksilöllistä ylläpitoa ja ongelmatilanteissa korjataan korvaamiseen sijaan. Tästä mallista halutaan päästä kohti karjaa eli vähemmän ylläpitoa vaativiin, nopeasti korvattaviin ympäristöihin.

Useita vanhoja infrastruktuuriratkaisuja täytyy miettiä uudelleen siten, että ne sopivat uusiin ajoympäristöihin. Tyypillinen esimerkki tästä on lokien säilymisen varmistaminen, kun ajoympäristöt ovat ohjelmistokontteja, joilla ei oletuksena ole kontin elinkaaren yli

säilyvää tiedostojärjestelmää. Mikropalveluarkkitehtuuria hyödyntävissä järjestelmissä lokien lähteitä saattaa olla myös niin paljon, että ilman keskitettyä näkymää, josta voi lukea kaikkien järjestelmän palveluiden lokeja, voi kokonaiskuvan muodostaminen olla hankalaa. [5, 6]

1.1 Tutkimuksen tavoitteet ja tutkimuskysymykset

Tämän työn tarkoituksena on ottaa kohdeorganisaatiossa käyttöön Kubernetes Nutanix-ympäristössä. Nutanixin "hoidettu"(managed) Kubernetes-ratkaisu on nimeltään Karbon. Hoidettu Kubernetes-ratkaisu tarkoittaa sitä, että ratkaisua käyttävän tahon ei itse tarvitse ylläpitää klusteritason infrastruktuuria, kuten siihen kuuluvia virtuaalikoneita tai verkkoja, vaan voi keskittyä Kuberneteksen hyödyntämiseen. [7, 8]

Työssä on tarkoituksena toteuttaa osa organisaation tunnistamista tarpeista, joita Karbonin käyttöönotto tuotannossa edellyttää. Osa tunnistetuista tarpeista on rajattu työn ulkopuolelle, jotta työn koko pysyisi kohtuullisena. Kohdeorganisaatio haluaa välttää toimittajaloukkua (vendor lock-in) Nutanixin teknologioihin, joten ratkaisut pyritään toteuttamaan siten, että ne eivät ole riippuvaisia Nutanixin muista palveluista. Nutanixin tarjoamat ratkaisut ovat myös tarkoitettu suurelle skaalalle eivätkä välttämättä sovellu siten kohdeorganisaation itse ylläpitämien sovellusten skaalaan. Näistä tarpeista muodostuu seuraava tutkimuskysymys: "Mitkä ylläpidolliset järjestelmät täytyy rakentaa, jotta Karbonia voidaan käyttää tuotantoympäristönä?"

Tuotantoympäristöllä tarkoitetaan, ajoympäristöä, jossa on sovellusilmentymiä, joita sovelluksien loppukäyttäjät käyttävät. Ylläpidollisilla järjestelmillä tarkoitetaan niitä järjestelmiä, joita tarvitaan varsinaisen Karbonissa olevien sovellusten tueksi. Järjestelmiä tarvitaan, jotta loppukäyttäjät saavat sovelluksiin yhteyden ja että mahdolliset ongelmatilanteet pystytään ratkaisemaan. Järjestelmille asetetut konkreettiset vaatimukset on kuvailtu tarkemmin luvussa 3.

1.2 Tutkimusmenetelmä

Tutkimuskysymys ratkaistaan konstruktiivisella tutkimuksella, jossa on lopputavoitteena tuottaa innovatiivinen ratkaisu johonkin reaalimaailman ongelmaan. Tässä tapauksessa reaalimaailman ongelmia ovat tutkimuskysymyksessä esitellyt työorganisaation tarpeet liittyen Karbonin käyttöönottoon ja hyödyntämiseen tuotantoympäristönä. Ratkaisujen perusta rakennetaan perehtymällä niin teoriaan kuin olemassa oleviin käytännönratkaisuihin. Olennaista on, että saavutetut ratkaisut läpäisevät käytännön testin, jossa ne otetaan käyttöön kohdeorganisaatiossa, mikä samalla osittain validoi käytetyn tutkimusprosessin. [9]

Konstruktiot rakennetaan seuraavalla tavalla:

- Konstruktioiden suunnittelussa käytetään lähteenä alan kirjallisuutta ja tutkimuksia.

- Arviointikriteerinä on kuinka toteutetut ratkaisut vastaavat kohdeorganisaatiossa tunnistettuja tarpeita.

1.3 Työn rakenne

Toisessa luvussa pohjustetaan teknologiat ja teoria, jotka lukijan on hyvä tietää, jotta hän ymmärtää valitut teknologiat. Reitityksen, varmuuskopioinnin ja lokituksen yhteydessä on esitelty perusperiaatteiden lisäksi vastaavat Amazon Web Services (AWS) toteutukset. AWS-toteutukset toimivat esimerkkinä siitä, miten suurin pilvipalvelujen tuottaja ja yleisesti edelläkävijänä pidetty Amazon on ratkaissut kyseisen palvelutarpeen. Kolmannessa luvussa esitellään kohdeorganisaation asettamat vaatimukset suunniteltaville järjestelmille. Neljännessä luvussa esitellään tutkimuskysymykseen valitut teknologiat ja luodut ratkaisut. Viidennessä luvussa analysoidaan luotujen ratkaisujen kelpoisuutta ja jatkokehitysmahdollisuuksia. Kuudes luku on yhteenveto.

Tässä työssä on käytetty samoja suomennoksia Kubernetes-termeille, joita Martikainen on käyttänyt diplomityössään "Tilallisuuden hallinta mikropalvelusovelluksessa". Martikaisen työstä otetut Kubernetes-olioiden suomennokset ja tätä työtä varten laaditut Kubernetes-termien suomennokset esitetään aina *kursivoituina*. Näin toimitaan, jotta lukija erottaa milloin on kyse Kubernetes-oliosta tai resurssista ja milloin yleisestä termistä. Tätä työtä varten tehdyt suomennokset löytyvät liitteestä A. Veleron ja Traefikin omia Kubernetes-olioita ei ole suomennettu. [10]

2 PILVI SUORITUS- JA KEHITYSYMPÄRISTONÄ

Tässä luvussa on esitelty työssä tehtyjen ratkaisujen ja valintojen pohjana olevaa teoriaa. Tässä luvussa esitellään myös työn kannalta tärkein yksittäinen teknologia, Kubernetes. Kubernetes-klusterin sisäistä reititystä käsitellään tarkemmin aliluvussa 4.1.1 ja tiedostojen säilytystä klusterissa käsitellään aliluvussa 4.2.1. Aliluvuissa käytetään soveltuvissa paikoissa AWS palveluja esimerkkeinä käytännön toteutuksista.

2.1 Pilvialustat

Tietotekniikan kontekstissa pilvi määritellään yhdistelmäksi laitteistoresursseja ja sovelluksia, jotka ovat tarvittaessa käyttäjien hyödynnettävissä [11]. Pilvipalvelut ovat yksinkertaisimmillaan virtuaalikoneita ja verkkoja, joita kehittäjät voivat hallita kuten perinteisiä palvelimia [2]. Yhä useammat pilvipalvelut toimivat kuitenkin korkeammilla abstraktiotasoilla. Esimerkiksi Azuren Computer Vision on palvelu, joka tarjoaa rajapinnan, johon käyttäjä voi lähettää kuvamateriaalia analysoitavaksi ilman, että käyttäjän tarvitsee itse tietää konenäön toteutuksesta mitään [12].

Pilvialustojen luonteeseen kuuluu olennaisesti resurssien tarjoaminen nopeasti hetkellisen tarpeen mukaan. Asiakas ostaa resursseja käyttöönsä vain siksi ajaksi, jonka hän todella tarvitsee. Tyypillinen esimerkki on verkkokauppa, joka tarvitsee suuren määrän laskentakapasiteettia vain tiettyinä päivinä vuodesta alennusmyyntien aikaan. Hinnoittelu on tarkimmillaan minuuttikohtaista tai Function-as-a-Service (FaaS) -funktioiden tapauksessa jopa sekuntikohtaista. [2, 11]

Pilvialustat voidaan jakaa karkeasti julkisiin, yksityisiin ja näiden yhdistelmään eli hybridipilveen. Julkisella pilvellä tarkoitetaan useimmiten pilvialustatoimijoita kuten Amazonia tai Microsoftia, jotka tarjoavat laajan kirjon erilaisia palveluja asiakkaidensa käyttöön. Asiakkuus on tarjolla kaikille toimijoille ilman erillistä neuvottelua vaatien vain tilin luomisen palveluun. Motivaationa julkisen pilven hyödyntämiseen on yleensä rahallinen säästö, joka saavutetaan sillä, että pilveä hyödyntävän organisaation ei tarvitse investoida laitteistoresursseihin ja resursseja ylläpitävään henkilöstöön. Yritys voi sen sijaan keskittyä suoraan ydinosaamisensa kehittämiseen, josta se saa parhaan taloudellisen hyödyn ja markkinaedun. [2]

Julkista pilveä hyödyntämällä käyttäjä saa myös maantieteellistä hajautusta ja vikasietoisuutta. Amazon ei suoraan julkaise datakeskuksiensa määrää, mutta sillä on datakeskuksia 24 erillisessä maantieteellisessä sijainnissa (region), esimerkiksi Ohiossa, Tuk-

holmassa ja Hong Kongissa. Näissä sijainneissa on yhteensä 77 erillistä Availability Zonea, joissa kussakin on yksi tai useampi datakeskus. Availability Zonet on eristetty toisistaan siten, että niillä on erilliset verkot ja virransaanti. Vastaavan vikasietoisuuden ja maantieteellisen kattavuuden saavuttaminen olisi hyvin kallista yksittäiselle toimijalle siihen verrattuna, että AWS:tä yritys voi ostaa resursseja pelkästään tarvitsemansa määrän haluamistaan sijainneista ja haluamastaan määrästä Availability Zoneja. Tällä tavoin organisoitu toiminta ei maksa juuri enempää kuin se, että kaikki resurssit olisivat samassa fyysisessä sijainnissa AWS:ssä. [13, Section: Availability Zones, Available Regions]

Yksityistä pilveä hyödyntävä organisaatio toteuttaa käytännössä pilvi-infrastruktuurin itse. Tällöin organisaatiolla on täysi kontrolli omasta infrastruktuuristaan, mutta ennen kaikkea pilvessä olevasta datasta. Tarkka tieto datan sijainnista on sitä tärkeämpää mitä arkaluonteisempaa säilytettävä data on. Arkaluonteista dataa on esimerkiksi terveydenhuollon potilastiedot ja yrityksen liikesalaisuudet. Asiakas voi myös ostaa yksityisen pilven ylläpidon kolmannelta osapuolelta siten, että samoilla palvelimilla ei ole muiden asiakkaiden dataa eikä pilven verkossa ole muita toimijoita. [2]

Hybridipilvessä organisaatio hyödyntää pilvi-infrassaan yhdistelmää julkisesta ja yksityisestä pilvestä. Organisaatio jakaa toimintonsa ja datansa niihin, jotka voidaan suorittaa ja säilyttää julkisessa pilvessä ja niihin, jotka pitää suorittaa ja säilyttää yksityisessä pilvessä. Näin pyritään saavuttamaan molempien mallien hyödyt. [2]

Flexeran 2019 suorittaman kyselytutkimuksen perusteella monimuotoinen pilven hyödyntäminen on organisaatioissa hyvin laajalle levinnyt ilmiö. Kyseelyyn vastanneista organisaatioista 94% hyödynsi pilveä, 91% julkista, 72% yksityistä ja 52% hybridipilveä. Organisaatiotioista 84% hyödynsi myös useaa pilveä. Julkista pilveä hyödyntävät organisaatiot käyttivät keskimäärin kahta eri julkista palveluntarjoajaa ja olivat kokeiluvaiheessa 1,8 palveluntarjoajan kanssa. Vastaavat luvut yksityistä pilveä käyttäville organisaatioille ovat 2,7 ja 2. Kyselystä käy myös ilmi se, että pilven hyödyntämiseksi on organisaatioissa perustettu erillisiä tiimejä ja pilveen liittyvien kustannusten optimointi on hyvin korkealla niiden agendalla. Kyselyyn vastasi 786 toimenkuviltaan erilaista teknistä henkilöä ja heidän vastuunsa vaihtelivat käytännön työstä tekniseen johtoon. Kyselyyn vastaajat edustivat laaja-alaisesti eri kokoisia organisaatioita. [14]

Pilviympäristöjen käyttäjät ajautuvat herkästi toimittajaloukkuun, eli tilanteeseen, jossa käyttäjän järjestelmät ovat täysin riippuvaisia tietyn palveluntarjoajan palveluista tai että järjestelmän siirto ja uudelleen konfigurointi toisen pilvitarjoajan ympäristöön on liian työlästä ja aikaavievää [15]. Kritikos et al. esittävät [16] ratkaisuksi multi-cloud management platform (MCMP) -palveluita, jotka mahdollistavat parhaimmillaan yksityisen pilven ja useamman eri julkisen pilven hallinnan yhdestä paikasta. Tutkimuksessa arvioitiin tuotteistettujen palveluiden suoriutumista orkestroinnista eri pilviympäristöissä (cloud orchestration support), sovellusten hallinnassa (cloud application support) ja alustojen älykkäässä hyödyntämisessä (platform intelligence). Tutkitut MCMP:t selviytyivät kaikki hyvin palveluiden orkestroinnissa eri pilvialustojen välillä. Palveluilla oli eniten hankaluuksia alustojen älykkäässä hyödyntämisessä, johtuen siitä että vain yksi MCMP mah-

dollisti laajan metriikoiden hyödyntämisen pilviresurssien optimoinnissa eri pilvien välillä. Muut palvelut kykenivät ainoastaan sovelluskomponenttien skaalaamiseen metriikan avulla. Useilla palveluilla oli myös vaikeuksia pilviresurssien pystyttämisen ulkopuolisen elinkaaren hallinnassa. [16]

Serhane et al. tutkivat [15] miten datan aiheuttamaa tomittajaloukkua voitaisiin välttää julkisia pilvipalveluja hyödynnettäessä. Palveluntarjoajilla on erilaiset API:t säilytyspalveluissaan ja useamman säilytyspalvelun tukeminen tai palvelusta toiseen vaihtaminen edellyttää usein järjestelmän datan varastoinnista vastaavien osien uudelleenkirjoittamista. Ratkaisuksi kirjoittajat ehdottavat järjestelmää, jossa palveluntarjoajia varten kehitetään omat agentit, jotka vuorovaikuttavat palveluiden kanssa. Agenttien yläpuolella on abstraktiotaso, Storage Broker Service (SBS), joka valitsee käyttäjän syötteen perusteella agentit ja koordinoi migraatiot. SBS abstraktoi eri palvelujen API:t oman yhtenäisen API:nsa taakse ja pyrkii minimoimaan siirrosta syntyneet kustannukset ja kuluneen ajan. Siirto-operaation kuluvan ajan optimointi tärkeää, sillä siirrossa käytettävät laskentaresurssit maksavat käytön mukaan. Migraatiot hoidetaan suoraan pilvien välillä, jotta ratkaisu voi hyödyntää pilvipalveluntarjoajien hyviä yhteyksiä ja SBS pyrkii valitsemaan kustannuksien ja suorituskyvyn kannalta optimin datakeskuksen vastaanottavan pilven datakeskuksista. [15]

2.2 Konttiteknologiat

Konttiteknologioiden synty voidaan ajoittaa 2000-luvun alun FreeBSD jail-komentoon ja Linux-VServeriin [17]. Jail-komennolla luodaan eristettyjä, samaa kerneliä hyödyntäviä järjestelmiä ja Linux-VServer on teknologia, joka perustuu kernel-tason eristämiseen [18, 19]. Linux kerneliin lisättiin 2000-luvun loppupuolella control groupit ja käyttäjänimiavaruudet (user namespaces). Conrol groupit mahdollistivat resurssien jakamisen ja käyttäjänimiavaruudet mahdollistivat prosesseille omat käyttäjät sekä prosessiin eristetyt root-oikeudet. IBM lähtöinen Linux Containers -projekti yhdisti vuonna 2008 kernelin tarjoamat mahdollisuudet ja lanseerasi konttitermin käytön [17]. Docker muuttui avoimen lähdekoodin projektiksi vuonna 2013, ja siitä alkoi konttiteknologioiden suosion voimakas kasvu, jonka ajurina toimi Dockerin helppokäyttöisyys [20].

Konttien idea on olla eristettyjä, kevyitä ja helposti siirrettäviä ajoympäristöjä sovelluksille [7]. Konttikuviin (container image) pakataan sovellus ja kaikki sen riippuvuudet. Sovelluksien ja riippuvuuksien paketoimisella saavutetaan haluttu siirreltävyys ja eristys. Ainut riippuvuus isäntäjärjestelmään on kernelin kautta. Koska kontit on eristetty isäntäjärjestelmästä ja toisistaan, ei kehittäjän tarvitse enää huolehtia riippuvuuskonflikteista. Kontin sisältä ei pitäisi olla mahdollisuutta päästä käsiksi isännän resursseihin ellei sille ole niitä erikseen annettu. Konteille voidaan antaa esimerkiksi pääsy osaan isännän tiedostojärjestelmästä. Kontit ovat kriittisessä asemassa useista pienistä komponenteista koostuvien järjestelmien toteuttamisessa. Konttien avulla voidaan suorittaa suurempaa määrää toisistaan eristettyjä sovelluksia tai sovelluskomponentteja tietyllä määrällä laitteistore-

sursseja virtuaalikoneisiin verrattuna.[21]

Sovelluskonttien ideana on toimia yksittäisen sovelluksen ajoympäristönä, kun taas järjestelmäkonttien tarkoitus on toimia virtuaalikoneiden kaltaisina ympäristöinä, joissa virtualisoidaan kokonainen käyttöjärjestelmä. Perusperiaatteiltaan nämä kaksi teknologiaa ovat kuitenkin hyvin lähellä toisiaan, joten sen vuoksi on yleistä puhua vain konteista. Tähän todennäköisesti myötävaikuttaa myös Dockerin asema suosituimpana konttiteknologiana, jolloin siitä on tullut lähes synonyymi konteille. Tämän työn kontekstissa konteilla tarkoitetaan Docker-sovelluskontteja (application container). [21]

2.3 Kubernetes

Kubernetes on avoimen lähdekoodin konttienhallintajärjestelmä, joka on suunniteltu siirrettävyys ja jatkokehitys silmällä pitäen. Kubernetes on alun perin lähtöisin Googlen sisäisäisistä Borg ja Omega -hallintajärjestelmistä [7]. Google julkaisi vuonna 2014 Kuberneteksen avoimena lähdekoodina. Version 1.0 julkaisun yhteydessä Google siirsi Kuberneteksen ylläpitovastuun Cloud Native Computing Foundationille [22].

Riippumattomuus laitteistoresursseista ja neutraalius ovat Kuberneteksen keskeisiä ominaisuuksia. Kuberneteksen tavoitteena on jalkautusprosessin samankaltaisuus siitä riippumatta, mihin jalkautus fyysisesti tapahtuu. Kubernetes abstraktoi laitteistoresurssit siten, että ne eivät ole kiinnitetty yksittäisiin fyysisiin tai virtuaalisiin koneisiin, vaan ne ovat käytössä klusteritasolla resurssivarantona. Sovelluskehittäjä voi itse varata esimerkiksi yhden prosessoriytimen ja gigatavun muistia sovellukselleen ja hänen ei tarvitse välittää siitä, miltä laitteelta resurssit tulevat. Kuberneteksen avulla myös järjestelmien siirto eri pilvialustojen välillä helpottuu, kun Kubernetes toimii yhteisenä nimittäjänä jalkautuksissa, ja alustojen käytättäjät voivat käyttää Kuberneteksen API:a molemmilla alustoilla. Kubernetes on neutraali myös käytetyn konttiteknologian (container engine) ja konttien suorittimen (container runtime) suhteen. Se tukee molemmissa kategorioissa useampia teknologioita Dockerin rinnalla, kuten rkt:tä konttiteknologioissa ja containerdiä suorittimena. [7, 20, 23] [24, Section: What is Kubernetes?, Nodes, Container Runtimes]

Kubernetes esittää kaikki klusterissa olevat komponentit ja resurssit Kubernetes-olioina. Olioiden avulla Kubernetes abstraktoi laitteistoresurssit, ajossa olevat kontit ja konfiguraatiot. Kuberneteksen API on deklaratiivinen ja idempotenttinen HTTP-pohjainen Representational State Transfer -API. Kubernetekselle kuvataan haluttu järjestelmän tila API:n kautta ja Kubernetes vie järjestelmän haluttuun tilaan sekä pyrkii ylläpitämään kyseisen tilan. Tilaa muokataan joko suoraan CRUD-operaatioiden kautta tai oliomanifesteilla, joiden avulla voidaan kuvata monimutkaisten olioiden tiloja tiiviissä muodossa. Kubernetekseen on myös mahdollista luoda *omaolioita* (Custom Resources), näille *omaohjaimia* (Custom Controller) ja näin laajentaa Kuberneteksen API:a. [7] [24, Section: Custom Resources]

Poniszewska-Marańda et al. esittävät [25] tuotantokäyttöön soveltuvalle Kubernetes-klusterille yhdeksän piirrettä. Keskitetty lokienhallinta ja monitorointi, korkea saavutettavuus, katko-

ton klusterin päivitys, klusterin tilan palautus ongelmatilanteen jälkeen, tietoturvan hallinta läpi klusterin, klusterin tilan tarkkailu testeillä, infrastruktuurin esittäminen koodina ja automaattinen skaalautuminen kuorman mukaan. Heidän mukaansa tavoitteisiin pääsee parhaiten ja nopeimmin käyttämällä Kubernetes-palveluita (managed services). Itse ylläpidettyä Kubernetesta tulisi hyödyntää, jos tavoitteena on kontrolloida hintaa tarkkaan tai sääntelyn aiheuttamat vaatimukset tai vaatimus täydellisestä kontrollista fyysiseen laitteistoon.

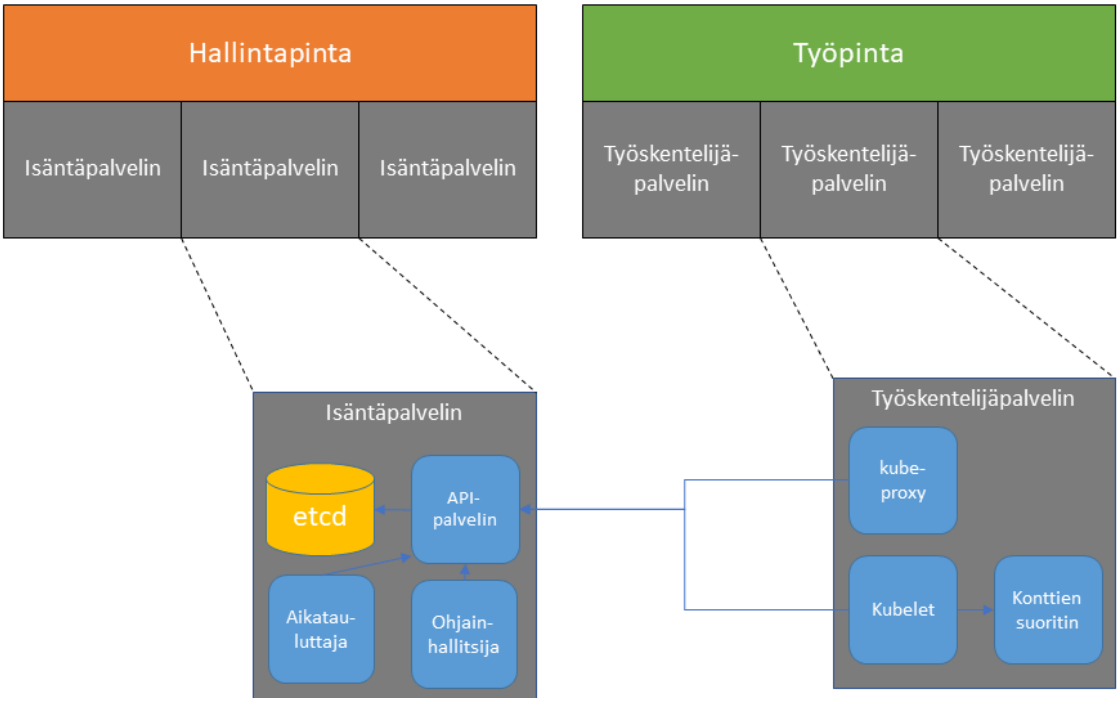
2.3.1 Kubernetes-klusterin rakenne

Kubernetes-klusteri jaetaan karkeasti kahteen eri pintaan: hallintapintaan (Control Plane) ja työpintaan (Workload Plane, joskus myös Data Plane). Hallintapinta kostuu klusterin isäntäpalvelimista (Master Node), jotka tarkkailevat klusterin tilaa ja hallinnoivat sen toimintaa. Jos hallintapinnalla on useita isäntäpalvelimia, ne ovat kopioita toisistaan ja pitävät sisällään samat tiedot. Useamman isäntäpalvelimen klusteri on vikasietoinen eikä sen toiminta häiriinny, jos yksittäinen isäntäpalvelin menee vikatilaan. Kuva 2.1 havainnollistaa klusterin rakennetta. Kuberneteksen API-palvelin (kube-apiserver) on ajossa isäntäpalvelimella. API-palvelin on tilaton, joten tieto siitä mitä olioita on luotu ja missä tilassa ne ovat, tallennetaan etcd-tietokantaan, joka on vikasietoisissa klustereissa hajautettu ja jaettu useammalle palvelimelle. API-palvelin on se konkreettinen komponentti, jonka kanssa käyttäjä on tekemisissä käyttäessään Kuberneteksen REST-API:a. Työntekijäpalvelimet kommunikoivat myös omasta tilastaan ja niillä ajossa olevien olioiden tilasta isäntäpalvelimille API-palvelimen kautta. Isäntäpalvelimella on API-palvelimen lisäksi ajossa *aikatauluttaja* (Scheduler), jonka vastuulla on sijoittaa eri sovellusilmentymät työntekijäpalvelimille ja *ohjainhallitsija* (Controller Manager), jonka vastuulla on konkreettisesti luoda käyttäjän API:n kautta pyytämät oliot. [7] [24, Section: Kubernetes Components]

Työpinta koostuu työskentelijäpalvelimista (Worker node), joilla varsinaiset sovellukset ajetaan. Jokaisella työskentelijäpalvelimella on asennettuna kubelet-agentti, joka vastaa palvelimen rekisteröinnistä osaksi klusteria ja käskyttää palvelimen konttien suoritinta (container runtime) luomaan halutut kontit. Kubelet kommunikoi palvelimen olioiden tilan isäntäpalvelimien API-palvelimille. Työntekijäpalvelimilla on myös kube-proxy, joka ei ole nimestään huolimatta varsinainen proxy, vaan nykyään sen tehtävä on hallinnoida työntekijäpalvelimensa iptables-sääntöjä. Kube-proxy saa API-palvelimelta tiedon siitä, mitä *palveluita* (Service) klusteriin on luotu, mitkä IP-osoitteet niiden takana olevilla *kapseleilla* on ja muokkaa palvelimensa iptables-sääntöjä sen mukaan. [7] [24, Section: kube-proxy]

2.3.2 Kuberneteksen tuomat hyödyt

Konttiteknologiat yksinään eivät sovellu laajojen kokonaisuuksien hallintaan, jossa sovellukset koostuvat useammista konteista ja jolloin saman aikaisten konttien määrät voidaan laskea sadoissa konteissa. Kun skaala on iso, tarvitaan avuksi paljon automaatiota, joka pitää huolen siitä, että kontteja on tarpeeseen vaadittava määrä ajossa. Jos ympäristössä



Kuva 2.1. *Isäntä- ja työntekijäpalvelimet sijaitsevat eri pinnoilla. API-palvelimelta on keskitetty kanava komponenttien tarvitsemalle informaatiolle.*

halutaan hyödyntää useampia palvelimia, joille kontteja jaetaan, tarvitaan avuksi konttien hallintajärjestelmiä. Kubernetes mahdollistaa dynamiikan, jota tuotantoympäristöt vaativat. [20]

Otetaan esimerkiksi 15 mikropalvelusta, joka on vielä suhteellisen pieni määrä, koostuva sovellus. 15 palvelun käynnistäminen, alasajo, tilan tarkkailu ja vikatilojen ratkaiseminen manuaalisesti on hyvin työlästä. Kuberneteksen käytön etu on se, että haluttuun tilaan päästään mahdollisimman pienellä manuaalisen työn määrällä sekä se, että haluttu tila myös säilytetään mahdollisimman pienellä työmäärällä. Hallintajärjestelmä voi esimerkiksi tiedustella sovelluksen tilaa ennalta määrätyllä tavalla ja sitten toimia saamansa vastauksen mukaisesti. Kubernetes esimerkiksi mahdollistaa *eloluotaimen* (liveness probe) konfiguroinnin jalkautuksiin. Jos luotain saa tarkkailemaltaan kontilta jonkun muun kuin odotetun tuloksen, kontti on päätynyt vikatilaan ja se käynnistetään uudelleen. [7, 20] [24, Section: Configure Liveness, Readiness and Startup Probes]

Kubernetes mahdollistaa myös resurssien tehokkaan hyödyntämisen, jos kyseessä on sovellus tai ryhmä sovelluksia joiden yksittäisten komponenttien käyttö on epätasaista. Konteille voidaan määritellä minimiresurssit, jotka ne tarvitsevat aina ja maksimin, jonka ne voivat varata käyttöönsä. Näin saadaan kaksi eri lukua, klusterin resurssivarannon minimikäyttö ja teoreettinen maksimi. Resurssien mitoittaminen ei ole suoraviivaista, vaan vaatii usein resurssien kulutuksen tarkkailua ajoympäristössä. Kubernetes myös jakaa kuorman klusterin työntekijäpalvelimille automaattisesti, jolloin klusterin käyttöaste on optimaalinen. [26] [24, Section: Managing Resources for Containers, What is Kubernetes?, Nodes]

Horisontaalinen skaalautuvuus on yksi yleisimmin tavoitelluista eduista, kun siirrytään perinteisistä ajoympäristöistä pilviympäristöihin. Skaalauksen toteuttaminen ilman Kuberneteksen kaltaisia hallintajärjestelmiä olisi kuitenkin hyvin työlästä. Horisontaalisessa skaalauksessa lisätään ohjelmainstanssien määrä kuorman mukaan sen sijaan, että nostettaisiin yksittäisen ilmentymän laskentaresursseja (vertikaalinen skaalaus). Horisontaalinen skaalaus on ketterämpää ja tehokkaampaa kuin resurssien lisäys. Se voidaan hyvin kohdistaa tarkasti suorituksen oikeaan pullonkaulaan, jolloin lopullinen lisättyjen resurssien määrä on todennäköisesti pienempi kuin vertikaalisessa skaalauksessa. Tämä luonnollisesti edellyttää, että järjestelmän eri komponentteja voi skaalata irrallaan toisistaan. Ilmentymisen lisääminen on kuitenkin monimutkaisempaa kuin tehojen lisääminen, sillä ilmentymät pitää luoda ja niiden pitää usein tehdä tiettyjä prosesseja ennen kuin ne ovat valmiita käytettäviksi ja ne voidaan rekisteröidä osaksi järjestelmää. Kubernetes tarjoaa usein näihin kaikkiin työkalut. Varsinkin rekisteröinti osaksi järjestelmää on olennaista, sillä pilvinatiiviin ajatteluun kuuluu, että ilmentymiä voidaan terminoida ja luoda mielivaltaisesti ilman erillistä varoitusta. Skaalaaminen halutaan myös yleensä tehdä automaattisesti ilman, että ylläpidon tarvitsee puuttua asiaan. Hyöty on ilmiselvä, jos ylläpito työskentelee pääasiassa yhdestä maasta ja yhdellä aikavyöhykkeellä, mutta sovelluksen käyttäjäkunta on kansainvälinen ja kuormaa syntyy ympäri vuorokauden. Kubernetes pystyy tarkkailemaan konttien käyttämiä resursseja sekä kuorman saavutettua tietyn osuuden lisäämään instansseja ja tällä tavoin tasaamaan yksittäisten instanssien kuormaa. [7, 26] [24, Section: Deployments]

Nguyen et al. [27] tutkivat miten Kubernetes Resource Metrics (KRM) ja Prometheus Custom Metrics (PCM) -resurssitarkkailijoiden käyttö vaikuttaa horisontaalisen skaalauksen toimintaan. Kuberneteksessa skaalauspäätökset tehdään resurssitarkkailijoiden ilmoittamien tietojen perusteella, joten valittu resurssitarkkailija vaikuttaa oleellisesti skaalauksen toimintaan. KRM on PCM:ää hitaampi ilmoittamaan muutoksista resurssien käytössä johtuen siitä, että se tutkii yksittäisiä datapisteitä. Skaalauspäätökset tehdään luotaussyklien (scraping cycle) jälkeen. PCM pystyy tutkimaan luotauksissa kerättyjen arvojen trendejä ja näin skaalauspäätöksiä voidaan tehdä syklien välissä tarvittaessa. Nguyen et al. [27] esittävät, että KRM:ää käytettäisiin resurssikuormaltaan vakaanpien sovellusten yhteydessä ja PCM:ää niiden sovellusten yhteydessä, joidenka kuorma muuttuu nopeasti. PCM:n aggressiivisempi skaalaus saattaa aiheuttaa resurssihukkaa sovelluksille, joiden kuormat ovat vakaita. KRM pystyy tarkkailemaan vain suorittimen käyttöä kun taas PCM pystyy tarkkailemaan esimerkiksi pyyntöjen määrä ja kestoa. Skaalaukseen valitut metriikat kannattaa valita sovelluskohtaisesti, jotta skaalaus perustuu todelliseen tarpeeseen. Nguyen et al. [27] huomauttavat myös, että useammasta palvelimesta koostuvat klusteri on kuorman jakamisen kannalta vakaampi kuin saman *tehoinen* pienemmästä määrästä palvelimia muodostuva klusteri. Isommassa klusterissa yksittäinen kaatuva palvelin edustaa pienempää osaa klusterista ja sen työkuorman jakaminen jäljellä oleville palvelimille on helpompaa.

Kubernetes mahdollistaa myös ohjelmistojen jalkautusten kuvaamisen hyvin tiiviissä muodossa *jalkautuksina* (Deployment). *Jalkautukset*, kuten kaikki Kubernetes-oliot, voidaan

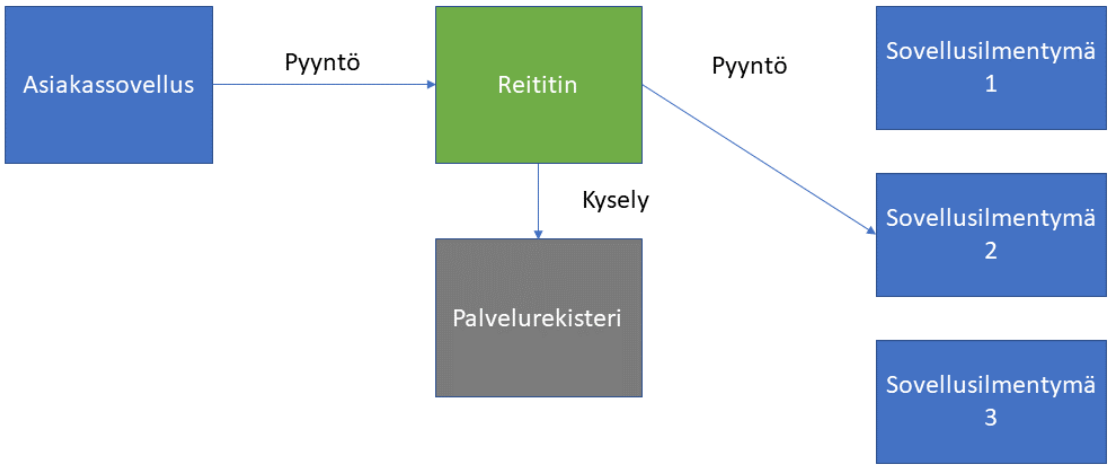
esitettää oliomanifesteina, jotka ovat helposti versioitavissa, siirrettävissä ja ymmärrettävissä. *Jalkautuksen* oliomanifestiin määritellään esimerkiksi mitä kontteja käynnistetään, kuinka monta niitä halutaan ja minkälaiset resurssit niille annetaan. Manifestin perusteella luodaan tarvittavat *ilmentymäjoukot* (Replica Set) ja *kapselit* (pod). *Jalkautus* mahdollistaa sen hallitseman kokonaisuuden päivittämisen deklaratiivisesti. Kun kokonaisuuden johonkin konttiin tulee päivitys riittää se, että *jalkautuksen* tietoihin vaihdetaan konttikuvan uusi versiotieto ja Kubernetes hoitaa päivityksen itsenäisesti valitun päivitysstrategian perusteella. Päivitys voi tapahtua, joko rullaavana päivityksenä (rolling update) tai uudelleen luontina (recreate). Rullaavassa päivityksessä Kubernetes luo päivitettyjä kontteja ja poistaa samanaikaisesti vanhoja, niin että palvelun käyttöön ei synny käyttökatkoja. Uudelleen luonnissa kaikki vanhat *kapselit* poistetaan ennen uusien luontia. Rullaava päivitys on oletus päivitysstrategia Kuberneteksessa, mutta sitä kannattaa käyttää vain, jos päivitettävä sovellus tukee useamman eri version ajoa samanaikaisesti. [7, 26] [24, Section: Deployments]

Kontit tarvitsevat usein ympäristökohtaista konfiguraatiota, jotka usein ratkaistaan ympäristömuuttujien kautta. Kuberneteksessa ympäristömuuttujat voidaan määritellä suoraan konttikohtaisesti *kapseleihin* tai ne voidaan hakea *asetuskorista* (ConfigMap). Ympäristömuuttujat, jotka sisältävät tietoturvakriittistä tietoa voidaan lisätä kapseleihin *salaisuuksilla* (Secret), joita säilytetään salattuna Kuberneteksessa. [7, 26] [24, Section: Define Environment Variables for a Container]

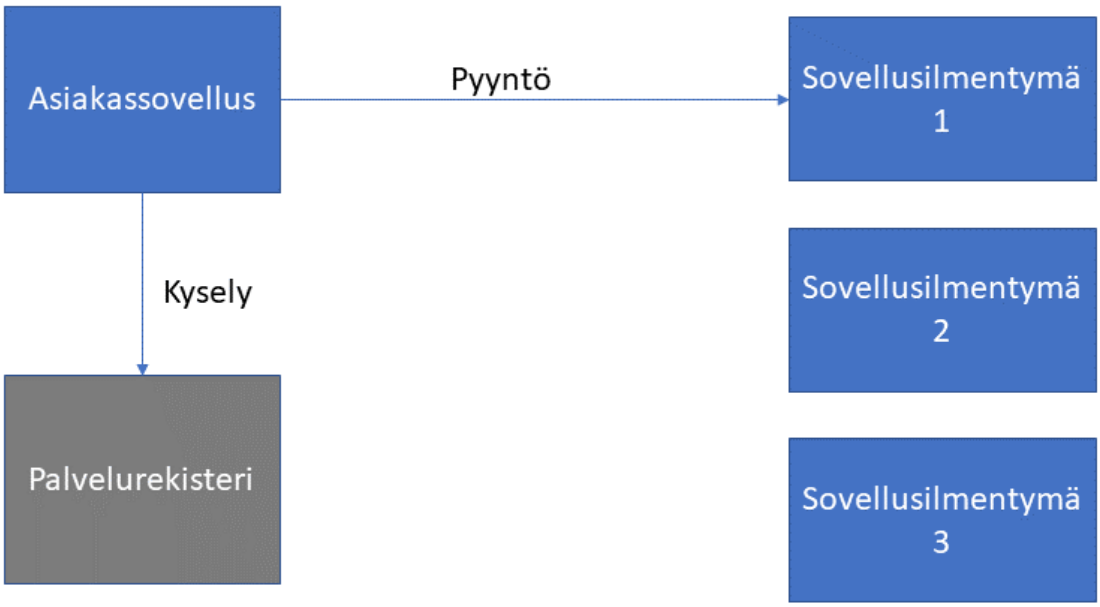
2.4 Pilvinatiivi reititys

Pilvinatiivissa reitityksessä on huomioitava sovellusten skaalautuminen horisontaalisesti ja yksittäisen sovelluksen ilmentymän hetkellisyys. Sovelluksen ilmentymien hetkellisyyden takia pyyntöjä käsittelevän komponentin pitää olla sekä tietoinen uusien ilmentymien synnystä että vanhojen poistumisesta. Kuten aliluvussa 2.3.2 kuvataan, pilvinatiivissa ympäristössä hallintajärjestelmä voi terminoida ja luoda uudelleen sovelluksien ilmentymiä milloin tahansa. Tyypillinen esimerkki on kuorman jako työpalvelimien välillä tai vikatilassa olevan sovelluksen kontin tuhoaminen ja luominen uudelleen. Pilvinatiivin järjestelmän reitityksen tila siis elää jatkuvasti. Reititys, jossa uusien ilmentymien tiedot päivitettäisiin ja tuhottujen ilmentymien tiedot poistettaisiin käsin reitityksestä vastaaviin komponentteihin, ei ole käytännöllinen. Järjestelmää, jossa yhteystietoja ylläpidetään käsin, sanotaan hauraaksi (brittle). [1, 28]

Tyypillinen pilvinatiivi ratkaisu on lisätä asiakkaan (client) ja sovelluksen konkreettisen ilmentymien väliin komponentti, joka abstraktoi ilmentymät taakseen ja toimii näin reittinä ilmentymiin. Tätä abstraktiotasoa kutsutaan palvelurekisteriksi (service registry) tai yksinkertaisemmin sovelluksen palveluksi (service). Se miten asiakassovellukset vuorovaikuttavat palvelurekisterin kanssa voidaan jakaa karkeasti kahteen tapaan: palvelinpuolen palveluhaku (server-side service discovery) ja asiakaspuolen palveluhaku (client-side service discovery). Kuvassa 2.2 on havainnollistettu kuinka palvelinpuolen palvelu-



Kuva 2.2. *Yksinkertaistettu esimerkki palvelinpuolen palvelujen löytämisestä*



Kuva 2.3. *Yksinkertaistettu esimerkki asiakaspuolen palvelujen löytämisestä*

haussa käyttäjä on aina yhteydessä komponenttiin, joka hakee palvelurekisteristä tiedon palvelun sijainnista. Kuvassa 2.3 on havainnollistettu kuinka asiakaspuolen palveluhaussa asiakassovellus itse kommunikoi palvelurekisterin kanssa. Rekisterin olisi myös hyvä tarkkailla ilmentymien tilaa, ettei se ohjaa liikennettä vikatilassa oleville ilmentymille. Rekisterin rinnalle tarvitaan vielä nimipalvelu Domain Name Service (DNS), joka yhdistää rekisterissä olevat IP-osoitteet kehittäjien määrittämiin URL:eihin. Nimipalvelimen avulla konfiguraatioissa voidaan aina viitata kehittäjien käsissä olevaan URL:iin, ei alustan dynaamisesti allokoimaan Internetin protokollaosoite (IP)-osoitteeseen. [1, 28, 29, 30]

Sovellukset täytyy rekisteröidä osaksi palvelurekisteriä tai niiden täytyy itse rekisteröityä osaksi sitä, jotta tiedot uusista ilmentymistä ja vanhojen poistumisesta päivittyvät rekisteriin ja rekisteri pystyy ohjaamaan asiakkaiden pyynnöt oikeisiin paikkoihin. Itserekisteröitymisen ongelma on se, että vikatilaan joutunut sovellus ei todennäköisesti kykene

itse poistamaan itseään palvelurekisteristä ja jokainen sovellus joutuu toteuttamaan rekisteröitymistoiminnallisuuden. Jos kolmas osapuoli hoitaa sovellusten rekisteröinnin, ei sovellusten itsenä tarvitse toteuttaa rekisteröitymiseen tarvittavaa toiminnallisuutta, mutta jos rekisteröijäkomponentti ei ole osa pilvialustaa niin se täytyy asentaa, konfiguroida ja ylläpitää erikseen. [28]

Taakanjako (load balancing) mahdollistaa horisontaalisen skaalauksen täysimittaisen hyödyntämisen. Taakanjaosta vastaava komponentti päättää, mitä ilmentymää milloinkin kutsutaan. Taakkaa voidaan jakaa joko verkon tasolla (network layer) tai sovellustasolla (application layer). Verkkotasolla toimittaessa taakanjakajalla pystyy analysoimaan liikennettä pelkästään pakettien otsikkotietojen (header) kautta, mikä tarkoittaa käytännössä paketin lähdettä ja kohdetta. Tämän vuoksi verkkotason reititys toimii osittain satunnaisuuden ja arvailun varassa. Esimerkkejä verkkotason taakanjakoalgrotimeista ovat kiertovuorottelu (round robin), jossa pyyntöjä ohjataan eri ilmentymille vuorotellen ja "pienin yhteyksien määrä"(least connections), jossa pyynnöt ohjataan niille ilmentymille, joilla on vähiten aktiivisia yhteyksiä. Sovellustasolla toimittaessa taakanjakaja näkee pyynnön sisällön, otsikkotiedot ja evästeet, mikä mahdollistaa suurempaan tietomäärän perustuvien päätösten tekemisen ja liikenteen tarkkailun molempiin suuntiin. Esimerkki sovellustason reititysalgoritmista on "pienin avointen pyyntöjen määrä"(Least Pending Requests, LPR) [31]. LPR:ssä taakanjakaja tarkkailee avoimien HTTP-pyyntöjen määrää ja ohjaa ne ilmentymille, joilla on vähiten kuormaa. LPR pystyy tekemään päätökset muun muassa sen perusteella, kuinka paljon aikaa erityyppiset pyynnöt useimmiten ottavat ja näin jakamaan liikennettä [31]. [1, 32]

AWS:ssä on verkkoihin ja reititykseen 16 eri palvelua, jotka on jaoteltu 5 eri käyttötapaukseen. Tämän työn kannalta oleellinen käyttötapaus on sovellusten verkot ja reititys (Application networking). Elastic Load Balancing on AWS:n tyypillisin taakanjakopalvelu. Palvelulla pystyy jakamaan liikennettä sovellus- ja verkkotasolla sekä myös toimimaan taakanjakoa hoitavana yhdyskäytävänä 3. osapuolen palvelujen välillä (Gateway Load Balancer). Julkiverkossa näkyvät taakanjakajat saavat julkisen IP:n automaattisesti ja niihin voi liittää TLS-varmenteet AWS Certicate Manager -palvelulla. Varmenteet pitää luoda käsin, mutta AWS Certicate Manager uusii ne automaattisesti tarvittaessa. Palvelimet tai kontit, joita halutaan skaalata horisontaalisesti, liitetään Auto Scaling -ryhmään, jonka uudet ilmentymät siihen liitetty taakanjakaja tunnistaa automaattisesti ja alkaa jakaa niille liikennettä. Jos uusia ilmentymiä ei luoda automaattisesti, niin taakanjaon kohteena olevat ilmentymät lisätään käsin taakanjakajalle. [33, Section: Networking, Elastic Load Balancing, AWS Certificate Manager, Certificate Manager FAQs]

AWS:n Cloud Map on AWS:n palvelurekisteriratkaisu. Cloud Map:issa asiakasovellus on yhteydessä siihen ja saa siltä tiedot tarvitsemiensa palveluiden osoitteista. Clou Mappiin luodaan nimiavaruuksia, joihin rekisteröidään palvelukategorioita. Kategorian alle rekisteröitäville palveluille annetaan avain-arvo-pareja, jotka yhdistämällä nimiavaruuteen ja kategoriaan asiakassovellus löytää tarvitsemansa palvelun. Palveluiden löytäminen perustuu joko API-kutsuilla tai DNS-kyselyillä. Kun palvelut rekisteröidään osaksi Cloud Map

-nimiavaruutta, niille voidaan määrittää myös terveystarkastuksia (health check), jotka pitävät huolen siitä, että Cloud Map ei tarjoa vikatilassa olevien ilmentymien tietoja asiakassovelluksille. Cloud Map voi rekisteröidä AWS-klusterissa olevia palveluja kuten EC2 virtuaalikoneita, Amazon DynamoDB tietokantoja, palvelittomia Lambda-funktoita ja taakanjakajia. Cloud Map ei jaa julkisia IP-osoitteita, vaan ne konfiguroidaan palveluille erikseen. [33, Section: AWS Cloud Map, AWS Cloud Map features] [34]

Amazonin oma Kubernetes-palvelu Elastic Kubernetes Service käyttää Kuberneteksen omia mekanismeja klusterin sisäiseen reititykseen. Kuberneteksen sisäistä reititystä on kuvattu tarkemmin aliluvusssa 4.1.1. EKS integroituu Cloud Mapin kanssa ja Elastic Load Balancing -palvelun kanssa. ExternalDNS komponentin avulla EKS:ssä ajossa olevat kontit rekisteröityvät osaksi Cloud Mappia ja ovat löydettävissä EKS-klusterin ulkopuolella. Kun klusteriin luodaan *sisääntulo* olio, AWS provisioi klusterille sovellustason taakanajakajan (OSI-mallin 7. taso) ja *palvelu* oliolle luodaan verkkotasolla (OSI-mallin 4. taso). Molemmat voivat saada tarpeen mukaan joko julkiverkon IP-osoitteen tai AWS:n sisäverkon osoitteen. [13, Section:Network load balancing on Amazon EKS, Application load balancing on Amazon EKS, Cluster VPC considerations]

2.5 DevOps osana pilven hyödyntämistä

Ohjelmistojen odotetaan olevan yhä vikasietoisempia, paremmin saavutettavissa ja skaalautuvampia. Samalla aikajänne, jolla uusia ominaisuuksia kehitetään sekä viedään tuotantoon, halutaan mahdollisimman pieneksi. Ohjelmistot muuttuvat näiden tavoitteiden johdosta monimutkaisemmiksi kokonaisuuksi, jotka koostuvat varsinaisen itsekirjoitetun ohjelmistokomponentin lisäksi erilaisista resursseista, kuten verkoista ja tietokannoista sekä 3. osapuolien palveluista. [1, 26]

Useista eri komponenteista, resursseista ja palveluista koostuvan ohjelmiston kehittäminen edellyttää sitä, että kehittäjät ovat tietoisia niistä tekijöistä, jotka ovat perinteisesti olleet ylläpidon murheita, kuten palomuurit, verkot ja nimipalvelimet. Ylläpidon on taas hyvä ymmärtää itse ohjelmiston toimintaperiaatteita, sillä se vaikuttaa olennaisesti ylläpidollisiin tehtäviin ja ajoympäristön suunnitteluun. DevOpsin ytimessä onkin irrotautuminen mallista, jossa ylläpito ja ohjelmistokehitys on jaettu kokonaan eri henkilöiden tai kokonaan eri tiimien välille. Pilviympäristön tehokas hyödyntäminen edellyttääkin, että jokaisessa tiimissä on henkilö, joka vastaa sovelluksen suorituskyvystä ja suoritumisesta pilvessä. [26]

Pilviympäristön täysimittainen hyödyntäminen vaatii myös DevOpsin toista suurta painotusta eli automaatiota. Automaation avulla ohjelmistojen julkaisuprosessi, joka kattaa esimerkiksi kääntämisen, Docker-kuvan rakentamisen ja testaamisen, saadaan standardoitua. Oikein toteutettuna se nostaa sekä ohjelmistojen itsensä että julkaisuprosessin laatua, kun tietokone luotettavasti tekee samat asiat samassa järjestyksessä. Toinen merkittävä etu on nopeus, kun käsisyötteisiä komentoja ei enää tarvita. Kun infrastruktuuri pystytään esittämään koodimaisesti, sen pystytyksen ja päivityksen automatisointi on

huomattavasti helpompaa. Myös infrastruktuurin versiointi ja sitä kautta muutosten seuraaminen helpottuu merkittävästi. [26, 35]

Jos keskitetylle ylläpitotiimille on tarvetta, sen tehtäviksi jäävät usein DevOps-infrastruktuurin ylläpito. Näitä ovat esimerkiksi jatkuvan CI/CD-työkalun, versionhallinnan ja tiketöintijärjestelmien ylläpito. Ylläpitotiimi voi myös huolehtia pilviklusterien terveydestä ja hallita klusterien resursseja, jotka pilvialusta abstraktoi kehittäjien käyttöön. [26]

Russo et al. ovat kartoittaneet [36] kyselytutkimuksella DevOps-periaatteiden käyttöä turvallisuuskriittisillä teollisuudenaloilla. Kyselyyn vastanneiden yritysten tietojärjestelmät ovat kyberfyysisiä eli ne ohjaavat realimailmassa toimivaa turvallisuuskriittistä konetta, minkä vuoksi itse järjestelmät ja niiden kehitys ovat tiukasti säännösteltyjä. Yritysten suurin ongelma DevOpsin käyttöönotossa on ollut tiukkoihin turvallisuusvaatimuksiin vastaaminen, mikä on hidastanut DevOpsin-menetelmien käyttöönottoa merkittävästi näillä aloilla. Kirjoittajat ehdottavat ratkaisuksi Secure DevOpsia, jossa alakohtaiset ja säännöstelyn asettamat tarpeet on huomioitu. Tavoitteena on lisätä nopeasti iteroitaviin päivityksiin turvallisuusulottuvuus, jossa aikeisemmin manuaaliset turvallisuustehtävät on automatisoitu. Myös turvallisuuteen liittyvät tarkkailu- ja raportointitehtävät tulee automatisoida. Turvallisuusaspektin tulee ulottaa myös yrityskulttuuriin, mikä näkyy kehitystiimien kokoonpanoissa ja vastuissa.

Laukkarinen et al. [37] mukaan DevOpsin automatisoitujen prosessien ja säännösteltyjen järjestelmien edellyttämän oikeellisuuden välillä ei ole ristiriitaa. DevOps-työkalujen kehityksessä ei ole vain huomioitu säännösteltyjen järjestelmien vaatimuksia. Esimerkkinä käytetään lääketieteellisten laitteiden elinkaaristandardia IEC 63204:ää, joka edellyttää, että jokaisen järjestelmälle asetetun vaatimuksen ja siihen liittyvän ohjelmistokomponentin välillä on dokumentoitu yhteys. Työkalujen tulisi automatisoida mahdollistaa vaatimuksien linkittäminen ohjelmistokomponentteihin, niihin liittyvien muutosten seuraaminen versionhallinnassa ja testien tuloksien tarkkailu ja testitulosten raportointi. Vastaavasti koodimuutosten pitäisi automaattisesi asettaa tietyt testit hylätty tilaan ja näyttää muutokset niihin liittyvän vaatimuksen yhteydessä. Kehitystiimin olisi myös hyvä nähdä tarvittavan dokumentaation suoraan työkaluista. [37]

DevOps-tekniikoita voidaan hyödyntää myös koneoppimismallien kehityksessä. Karamitsos et al. [38] mukaan hyödyntämällä CI/CD-putkia ja DevOps-työskentelyn periaatteita koneoppimismallien koulutusta ja testausta voidaan nopeuttaa ja kustannuksia pienentää. CI/CD-putkien avulla datatieteilijät pystyvät nopeasti iteroimaan mallien arkkitehtuureja ja etsimään datasta hyödyllisiä piirteitä (features). Koneoppimismallit kerryttävät teknistä velkaa nopeasti, koska velkaa kertyy sekä mallin koodissa, että datassa. Datan käsittelyvaiheiden automatisoinnilla pyritään tuottamaan mahdollisimman korkeatasoista dataa mallien kolutukseen ja testaukseen. DevOpsille tyypillisesti dataa myös päivitetään säännöllisesti ja iteratiivisesti ja jokainen päivitys ajetaan CI/CD-putken läpi tuottaen lopputuloksen, joka on vertailukelpoinen aiempien iteraatioiden kanssa. Tuotantoon asti päässeen mallin suoriutumista myös tarkkaillaan jatkuvasti (performance testing). Jos malli suoriutuu huonosti, voidaan CI/CD-putki käynnistää automaattisesti uudelleen ja

yrittää luoda päivitetty malli, joka suoriutuu muuttuneessa ympäristössä paremmin. [38]

2.6 Pilvinatiivit sovellusarkkitehtuurit

Pilvinatiivit sovellusarkkitehtuurit pyrkivät hyödyntämään pilven pilvialustojen tarjoamia mahdollisuuksia ja ottamaan huomioon niiden asettamia haasteita. Tämän vuoksi pilvinatiiveille sovellusarkkitehtuureille ovat tyypillisiä seuraavat piirteet: pyrkimys tilattomuuteen aina kun mahdollista, skaalautumisen mahdollistaminen, sovelluskomponenttien pitäminen yksinkertaisina, tietoturvan huomioiminen läpi koko järjestelmän (defence in depth) ja päivitysten mahdollistaminen ilman käyttökatkoja. Nämä piirteet ovat hyvin voimakkaasti kytkeytyneitä toisiinsa ja usein ratkaisuja jonkin toisen piirteen aiheuttamiin haasteisiin. [1, 21]

Luvussa 2.1 käytiin läpi, miten pilvialustat tukevat horisontaalista skaalaamista tarjoamalla laskentaresursseja dynaamisesti tarpeen mukaan. Luvussa 2.3 käsiteltiin sitä, miten Kubernetes pystyy abstraktoimaan laskentaresurssit resurssivarannoksi ja jalkauttamaan halutut komponentit näille resursseille sekä skaalaamaan niiden ilmentymiä horisontaalisesti siten, että järjestelmä on halutussa tilassa. Jotta horisontaalinen skaalaus olisi mahdollista, täytyy sovellusarkkitehtuurin tukea sitä. Usein ensimmäinen ratkaistava ongelma on tilallisen tiedon hallinta. Tilallisia komponentteja on huomattavasti vaikeampi skaalata kuin tilattomia. Tilallisuus edellyttää esimerkiksi usein sitä, että sama komponentti vastaa aina samalta käyttäjältä tuleviin pyyntöihin. Usein kaikki sovelluksien osat, eivät voi olla tilattomia. Tästä johtuen sovellusarkkitehtuuria suunniteltaessa on tehtävä selkeä jako tilallisiin ja tilattomiin komponentteihin. Komponenttien kannattaa olla tilattomia aina kuin mahdollista. [1, 6]

Tilallisten komponenttien skaalauksessa joudutaan valitsemaan johdonmukaisuuden (consistency) ja saavutettavuuden (availability) välillä. Komponentin saavutettavuutta voidaan parantaa skaalaamalla sitä horisontaalisesti. Kun komponentista lisätään uusia ilmentymiä, joudutaan samaa tilallista tietoa säilyttämään useammassa ilmentymässä samanaikaisesti. Jos dataa muokataan yhdessä ilmentymässä, täytyy muokkausten heijastua *lopulta* muihinkin ilmentymiin. Tätä mallia, jossa eri ilmentymät ovat eri tiloissa ja lopulta samassa tilassa, kutsutaan Basically Available, Soft state, Eventual consistency (BASE) -malliksi. Vastaavasti täytyy uhrata osia saavutettavuudesta, jos johdonmukaisuudesta halutaan pitää kiinni. Pilviympäristöt mahdollistavat helpon resurssien lisäämisen yksittäiselle instanssille, mikä luonnollisesti parantaa saavutettavuutta. Vertikaalinen skaalaaminen pilviympäristössä on joustavaa, kun tilallisen komponentin resursseja lisätään ja vähennetään tarpeen mukaan automaattisesti. Saavutettavuuteen kuuluu myös vikasietoisuus. Jos tilallinen tieto on yksittäisessä tilallisessa komponentissa, muodostaa kyseinen komponentti vikapisteen (single point of failure) järjestelmään ja sen pettäessä järjestelmä on vikatilassa, eikä pysty toimimaan halutulla tavalla. [1, 39, 40]

Skaalautumisessa täytyy ottaa myös huomioon komponenttien väliset riippuvuussuhteet. Mitä löyhemmin kytkettyjä komponentit ovat toisiinsa (loosely coupled), sitä helpompi nii-

tä on skaalata erillään toisistaan. Löyhästi toistaan erillään olevat komponentit voivat olla toteutettuja eri teknologioilla, olla ajossa eri käyttöjärjestelmien päällä, ja pilviympäristöille tyypillisesti olla ajossa erillisissä konteissa. Se mitä löyhä tai tiukka kytkentä tarkoittaa voi riippua kontekstista. Monoliitissa tiukka kytkentä voi näkyä esimerkiksi yksittäisen koodimuutoksen heijastumisena useampaan paikkaan. Pilvinatiivissa arkkitehtuurissa, jossa komponentit ovat hajautettu eri kontteihin voi tiukka kytkentä ilmetä esimerkiksi siten, että palvelu käyttää aktiivisesti toisen palvelun dataa sen sijaan, että keskittyisi käsittelemään omaa dataansa. [1, 41]

2.6.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuri on yleisin ja tunnetuin pilvinatiiviarkkitehtuuri. Nimensä mukaisesti mikropalveluarkkitehtuurin mukaan rakennettu sovellus koostuu pienistä, erillisitä palveluista, jotka toimiessaan yhdessä muodostavat sovelluskokonaisuuden. Mikropalveluarkkitehtuurissa on tavoitteena, että jokainen yksittäinen mikropalvelu on itsenäinen, tarkkaan rajattu ja löyhästi toisiin mikropalveluihin kytketty. Mikropalvelut kommunikoivat keskenään viestipohjaisilla protokollilla kuten HTTP. Mikropalvelulle ei ole mielekästä asettaa rajoja esimerkiksi koodirivien perusteella, vaan edellä mainittujen tekijöiden toteutuminen on tärkeämpää, kuin yksittäiset määrälliset mittarit. [42]

Se mitä mikropalveluarkkitehtuurilla yleensä tavoitellaan voidaan tiivistää seuraavasti: Korkea muutosvauhti uhraamatta järjestelmän vakautta. Muutosvauhti ja järjestelmän vakaus ovat karkeasti toisistaan käänteisestiriippuvaisia. Kun muutosvauhti kiihtyy on yhä vaikeampaa varmistaa, että järjestelmä toimii oikein kaikissa mahdollisissa tilanteissa. Vastaavasti, jos vakaus asetetaan ykkösprioriteetiksi, kestää jokaisen yksittäisen muutoksen elinkaari määrittelystä tuotantoon pitkään. Myös mikropalveluiden kanssa joudutaan tasapainoilemaan näiden kahden prioriteetin välillä. Mikropalveluarkkitehtuurin tavoitteena on saavuttaa perinteisiin monoliittiehin verrattuna tasapainon taso huomattavasti korkeamalla muutosnopeudella. Vakauteen liittyy myös kyky sietää ennakoitua suurempaa kuormaa, mikä on toinen tapa sanoa, että järjestelmän pitää skaalautua. [30]

Automaatiolla on merkittävä rooli muutosvauhdin ja vakauden tasapainon saavuttamisessa. Tämän vuoksi luvussa 2.5 kuvattujen DevOps tekniikoiden hyödyntäminen on oleellinen osa mikropalvelujen tehokasta kehittämistä. Mikropalvelujen rajaaminen itsenäisiksi, toiminnaltaan rajatuiksi komponenteiksi, myötävaikuttaa automaation hyödyntämistä. Testien rakentaminen pienelle komponentille on nopeampaa ja testejä on helpompi hallita komponentin kehittyessä. Jokaisen komponentin kohdalla käydään aina läpi integraatiotestaus koko järjestelmän kanssa. Pienen, itsenäisen ja riippumattoman komponentin jalkauttaminen on myös nopeampi ja yksinkertaisempi prosessi, kuin monoliitisessa sovelluksessa. Monoliitissa järjestelmän pienen osan muuttaminen saattaa vaatia koko järjestelmän kääntämistä ja jalkauttamista uudelleen. [30, 42]

Mikropalveluarkkitehtuuri antaa ylläpidettävyydelle myös uuden muodon, korvattavuuden. Jos yksittäinen mikropalvelu on onnistuttu kehittämään siten, että se on selkeästi

rajattu ja itsenäinen, se on myös helppo korvata tarvittaessa. Komponenttia ei tarvitse päivittää ad hoc -ratkaisuilla, jotta uusiin tarpeisiin voidaan vastata. Sen sijaan komponentti voidaan suunnitella uudelleen siten, että uudet vaatimukset on huomioitu. Tämä on samankaltaista ajattelua, kuin johdannossa mainittu lemmikit ja karja -analogia. Taibi et al. havaitsivat [43] haastatellessaan joukkoa mikropalveluihin siirtyneitä toimijoita, että nimenomaan ylläpidettävyys oli kaikkien haastateltavien mielestä tärkeä motivaattori mikropalveluihin siirtymiselle ja sen tuomat hyödyt havaittiin myös selkeästi. [30, 42]

Mikropalvelut mahdollistavat sen, että yksittäisillä kehitystiimeillä on vahva autonomia omiin komponentteihinsa. He voivat valita teknologiat ja komponentin sisäisen arkkitehtuurin vapaasti. Itsenäiset tiimit pystyvät nopeampaan kehitykseen keskitetyn päätöksenteon pullonkalojen puuttuessa, mikä parantaa muutosnopeutta. Hajautus ei kuitenkaan vähennä johtamisen tarvetta, vaan päinvastoin lisää sitä. Mikropalveluista koostuva järjestelmä on arkkitehtuuritasolla monimutkaisempi, kuin vastaava järjestelmä olisi monoliittina toteutettuna, jolloin kokonaisuuden hahmottaminen ja ohjailu on hankalaa. Eli vaikka autonomiset tiimit tekevät päätöksiä miten yksittäinen palvelu toteutetaan, tarvitaan korkeamman tason päätöksiä siitä, mitä palveluja toteutetaan, ja miten komponentit integroidaan toisiinsa. [30, 43]

Mikropalveluarkkitehtuuria harkitessa on pidettävä mielessä se, että se soveltuu parhaiten suurille järjestelmille, jotka ovat jatkuvassa kehityksessä ja tarvitsevat mikropalveluiden tuomaa nopeuden ja vakauden tasapainoa [30]. Taibi et al. kyselyiden [43] vastaajat ilmoittivat, että mikropalveluina toteutetulla järjestelmällä on monoliittiin verrattuna korkeammat yleiskustannukset *läpi koko järjestelmän elinkaaren*. Tällöin saavutettavien etujen täytyy kompensoida kohonneet yleiskustannukset koko järjestelmän elinkaaren ajalta, jotta mikropalveluarkkitehtuuri on taloudellisesti oikeutettu. Yleiskustannukset ovat kustannuksia, jotka syntyvät työstä, joka ei suoraan liity järjestelmän kehittämiseen, vaan on esimerkiksi tässä luvussa mainittua kompleksisen arkkitehtuurin hahmottamisesta syntyvää työtä. Saman kyselytutkimuksen vastaajat ilmoittivat myös, että mikropalvelut ovat halvempia ja omaavat paremman tuottoprosentin kuin monoliitit. Kustannusedut saavutettiin nimenomaan pienempinä ylläpitokustannuksina. [43]

2.6.2 Palveliton arkkitehtuuri

Palvelittomassa (serverless) arkkitehtuurissa pyritään sovellus toteuttamaan ilman sovelluskehittäjien ylläpitämää palvelinpuolta. Palvelinpuolen toiminnalisuus pyritään joko ostamaan Backend-as-a-Service (BaaS) -palveluina tai toteuttamaan tilattomina FaaS-funktioina. Myös FaaS-funktioiden ajoympäristö ostetaan palveluna. Esimerkkejä tyypillisimmistä ostettavista palvelinpuolen palveluista ovat tietokannat ja autentikointi. Mallin idea on, että kehittäjät pystyvät keskittymään applikaatiologiikkaan ja toteuttamaan sen asiakasohjelmassa (client-side), kuten esimerkiksi selaimessa. [44, 45]

FaaS:ssa ajoympäristöön jalkautetaan applikaatiokontin kuva, joka käynnistetään vain, kun ympäristö vastaanottaa sen suorituksen käynnistävän tapahtuman. FaaS-funktiot so-

veltuvat parhaiten tapahtumapohjaiseen (event-driven) malliin. Tämä on suuri ero verrattuna mikropalveluihin, jotka toimivat kutsuttavien rajapintojen kautta. FaaS-funktio suoritetaan tarpeen mukaan ja ennen suoritusta laukaisevaa tapahtumaa siitä ei ole ajoympäristössä ajossa olevaa ilmentymää. Ilmentymä on lyhimmillään olemassa vain yhden ajon verran, jonka jälkeen se terminoidaan. Tämän vuoksi onkin tärkeää, että funktion käynnistymisaika (warmup-time) on mahdollisimman lyhyt, jotta järjestelmään ei synny viivettä. Funktiot ovat täysin tilattomia ja sen vuoksi helppoja skaalata. FaaS:sa skaalautumislogiikka on palveluntarjoajan vastuulla. [44, 45]

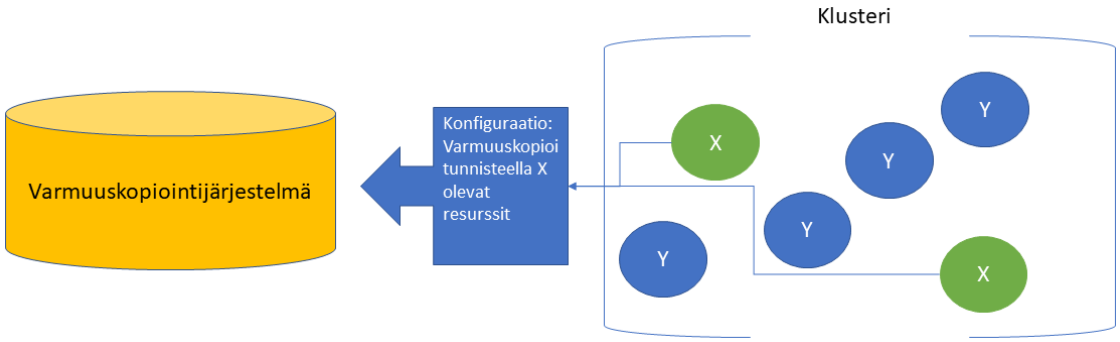
FaaS-funktoita kannattaa hyödyntää, kun sovellukselta vaaditaan automaattista skaalaamista ja suorituskulut halutaan minimoida. FaaS-funktioiden hyödyntämisessä edellytyksenä on, että ne eivät ole pitkäkestoisia, ja että järjestelmä voidaan toteuttaa tapahtumavetoisena. FaaS-funktiot tarjoavat myös mikropalveluja hienojakoisemman kontrollin jalkautuksiin, kun jalkautettavat komponentit ovat yksittäisiä funktioita. Mikropalvelut soveltuvat paremmin järjestelmiin, joissa on pitkäkestoista ja resurssi-intensiivistä prosessointia. [44]

2.7 Varmuuskopiointi pilviympäristössä

Pilviympäristöön suunnitelluilla varmuuskopiointiratkaisuilla pyritään siihen, että varmuuskopiointijärjestelmä reagoi muuttuvaan ympäristöön. Jos ylläpitotiimi tai sovelluksesta vastuussa oleva tiimi joutuu erikseen lisäämään uudet palvelut varmuuskopioinnin piiriin, on olemassa riski, että järjestelmä ei ole ajan tasalla [26]. Varmuuskopiointi-infrastruktuuria suunniteltaessa pitää ottaa myös huomioon järjestelmäkohtaiset aikamääreet: kuinka nopeasti järjestelmän pitää palautua ja kuinka pitkältä ajalta dataa voidaan menettää? Jos aikamääreet ovat tiukat, sekunteja tai minuutteja, täytyy palautuksen tapahtua kokonaan automaattisesti virheen jälkeen [46]. Tiukan sääntelyn piirissä olevien järjestelmien pitää ottaa myös huomioon sääntelyn asettamat vaatimukset. Esimerkiksi Suomessa potilastietojen käsittelystä syntyvien lokitietojen säilytysaika on 12 vuotta [47].

Pilvinatiivi sovellus, jota ajetaan Kuberneteksessa tai Kuberneteksen kaltaisessa ympäristössä, palautuu nopeasti haluttuun tilaan yksittäisten työntekijäpalvelinten tai isäntäpalvelimen kaaduttua. Itse sovellus on varastoitu konttikuvan muodossa erilliseen rekisteriin. Sovelluksen ympäristön kuvaavat oliomanifestit ovat versionhallinnassa ja lisäksi replikoidussa etcd-tietokannassa. Kubernetes luo uuden ilmentymän terveelle työntekijäpalvelimelle itsenäisesti [26]. Tämän kaltaisissa ympäristöissä tietovarantojen ja tiedostojen varmuuskopioinnin rooli korostuu. Tietovarannot ja tiedostot voivat korruptoitua sovelluksen virheellisen toiminnan, käyttäjävirheen tai hyökkäyksen takia. Jos korruption syynä on jokin muu kuin laiterikko, virhe leviää läpi hajautetun järjestelmän tietyllä aikavälillä, joka on usein johdonmukaisuusvaatimusten takia niin pieni, että ihminen ei ehdi siihen reagoida [46].

Siitä huolimatta, että lähdekoodi, konfiguraatiot ja konttikuvat ovat omissa varastoissaan, on klusterin tilan varmuuskopiointi järkevää, jos käytössä olevat työkalut mahdollista-



Kuva 2.4. *Varmuuskopioinnin kohdistaminen tiettyihin resursseihin on edellytys tiimien itsenäisyydelle. Resurssit voivat sijaita samassa klusterissa ja samassa nimiavaruudessa.*

vat sen. Vaikka paras käytäntö on, että kaikki muutokset Kubernetes-olioihin tehdään versionhallinnassa olevien oliomanifestien kautta, niin varsinkin kehitysvaiheessa olevilla palveluilla osa konfiguraatioista saattaa olla pelkästään klusterissa. Tällöin katastrofitilanteessa, jossa klusteri menetetään kokonaan, klusterin varmuukopioista on hyötyä. Testiympäristöjen luominen ja hallinta on myös helppoa, jos haluttu tila voidaan palauttaa nopeasti varmuuskopioista. Versiopäivityksen peruminen on myös helmpompaa, jos koko sovellus, tietovarannot, tiedostot ja itse sovellus voidaan palauttaa yhdestä koherentista lähteestä siihen tilaan, jossa se oli ennen päivityksen tekemistä. Ylläpidollinen erityistapaus, jossa Kubernetes-olioiden ja klusterin tilan varmuuskopionti tuo etua on migraatio klusterista toiseen. Siirtymä tapahtuu helpoimmilaan vaihtamalla varmuuskopioinnin kohde ja palauttamalla klusterin tila uusimmasta varmuuskopiosta. [26, 46]

Varmuuskopiointi pitää myös pystyä kohdistamaan tarkasti tiettyihin resursseihin, jotta vain järjestelmän yksittäistä osaa kehittävillä pienemmillä tiimeillä säilyy itsenäisyys. Kuvassa 2.4 on havainnollistettu resurssien varmuuskopiointia tunnisteiden perusteella. Jos kehitystiimi menettää itsenäisyytensä varmuuskopioinnin suhteen, se hidastaa kaikkea dataan liittyvää kehitystyötä. Kehitystiimi saattaa haluta säilyttää dataa eri muodossa, eri teknologialla tai eri sijainnissa, jolloin heidän pitää itse pystyä tekemään palautuksia ja muutoksia varmuuskopioinnin konfiguraatioon. Jos varmuuskopiointi tehdään klusteritasolla, täytyy koko klusteri palauttaa tai luoda uudelleen kerralla. Tällöin hukataan sekä laskentaresursseja että aikaa, kun koko klusteri joudutaan luomaan uudelleen. Pienempää osakokonaisuutta kehittävä tiimi ei välttämättä pysty tekemään itsenäisesti koko klusterin palautusta, koska klusterin palautus tarvitsee laajat oikeudet eri järjestelmiin. [26, 48]

Hyödyntämällä ylläpidettyjä palveluita kehitystiimi pystyy siirtämään vastuun varmuuskopioinnista palveluntarjoajalle, jolloin kehitystiimi voi keskittyä itse sovelluksen kehittämiseen. Palveluntarjoajat, joiden varmuuskopiointiratkaisut on integroitu suuriin julkisiin pilviin saavat myös varmuuskopioinnissa yleiset suurten alustojen hyödyt: lähes loputon skaalautuminen, hinnoittelu käytön mukaan, kattava globaali maantieteellinen hajautus ja lukuisat lisäarvoa tuovat palvelut liittyen analytiikkaan ja tietoturvaan. Suurilla julkisen pilven palveluntarjoajilla on luonnollisesti omat varmuuskopiointiratkaisunsa, jotka

on integroitu suoraan dataan liittyviin palveluihin kuten Amazonin Redshift [33, Section: RedShift]. Varmuuskopiointipalvelun käyttö voi myös edesauttaa toimittajaloukkua, jos varmuuskopiot voi natiivisti palauttaa vain yhteen ympäristöön esimerkiksi Azureen tai AWS:ään. [6, 49]

Varmuuskopioinnin voi hankkia myös erillisenä palveluna. On tärkeää huomata ero pilvisäilytysjärjestelmien, jotka hyödyntävät pilveä ainoastaan varastona varmuuskopioille, ja pilvinatiivien järjestelmien välillä [50]. Pilvisäilytysjärjestelmät eivät usein pysty samaan hienojakoisuuteen, kuin pilvinatiivit järjestelmät. Palautettavat objektit ovat esimerkiksi suuria tiedostojoukkoja tai kokonaisia virtuaalikoneita, kun taas pilvinatiivit järjestelmät pystyvät toimimaan esimerkiksi Kubernetes-resurssien tasolla [48]. Pilvisäilytysjärjestelmissä, palautukset ovat usein myös raskaita, koska palautusta ei pystytä tekemään suoraan, vaan varmuuskopio siirretään ensin järjestelmään palautuksen tekevään järjestelmään, mikä huonontaa suorituskykyä [50]. Pilvinatiivi järjestelmä pystyy palauttamaan varmuuskopiot suoraan niiden ajoympäristöön, kuten esimerkiksi Kubernetekseen [48]. Pilvisäilytysjärjestelmät kärsivät suurpiirteisyytensä takia suuremmista kuluista, koska pilvialustat veloittavat datan siirrosta. Suuremmat kulut ja huonompi suorituskyky voi johtaa siihen, että palautukset tehdään vain pakon edessä, jolloin osa hyödyistä, kuten testiympäristöjen tilojen palautukset osana normaalia testausrutiinia jäävät hyödyntämättä [49].

AWS:n objektivarastoon (Object Storage) Simple Storage Service (S3) tallennettu data on automaattisesti replikoitu yksittäisen maantieteellisen alueen sisällä useampaan datakeskukseen. Amazon lupaa S3-palvelussa olevalle datalle 99,999999999% eheyden ja 99,99% saavutettavuuden vuoden aikajaksolla. Replikoinnin avulla datan pitäisi olla saavutettavissa tilanteessa, jossa samalla alueella kaksi datakeskusta on alhaalla samanaikaisesti. Useat muut AWS:n datansäilytyspalvelut, kuten Elastic Block Store, DynamoDB ja Elastic File System saa keskitetyn varmuuskopioinnin piiriin AWS Backup palvelulla. AWS Backup pystyy varmuuskopioimaan myös virtuaalikoneiden tilan levykuvina. [33, Section: Data protection in Amazon S3, AWS Backup features]

AWS ei tällä hetkellä tarjoa mahdollisuutta varmuuskopioida EKS-klustereita natiivisti AWS:n omilla työkaluilla. Klusterien *pysyväislevyistä* saa kuitenkin otettua tilannekuvia, jotka tallennetaan S3 palveluun. Klusterin varmuuskopiointi onnistuu erillisillä työkaluilla kuten Velerolla, jota käsitellään tarkemmin luvussa 4.2.4.

2.8 Sovelluslokien käsittely pilviympäristössä

Pilviympäristö itsessään ei vaikuta yksittäisessä virtuaalikoneessa ajettavan monoliittisen sovelluksen lokituksen tarpeisiin. Lokitus virtuaalikoneella, jota ylläpidetään ja jota ei ole tarkoitus poistaa ellei sovellusta poisteta käytöstä, toimii samoilla periaatteilla kuin perinteisellä palvelimella. Jos sovellus on skaalautuva ja virtuaalikoneet tai kontit ovat väliaikaisia, hetkittäisen tarpeen mukaan luotuja, lokit menetetään aina kun ilmentymä poistetaan. Lokit voidaan tallentaa erilliseen tiedostojärjestelmään, jota ei pyyhitä, kun ilmentymiä poistetaan. Tämänkaltainen järjestelmä osoittautuu kuitenkin nopeasti köm-

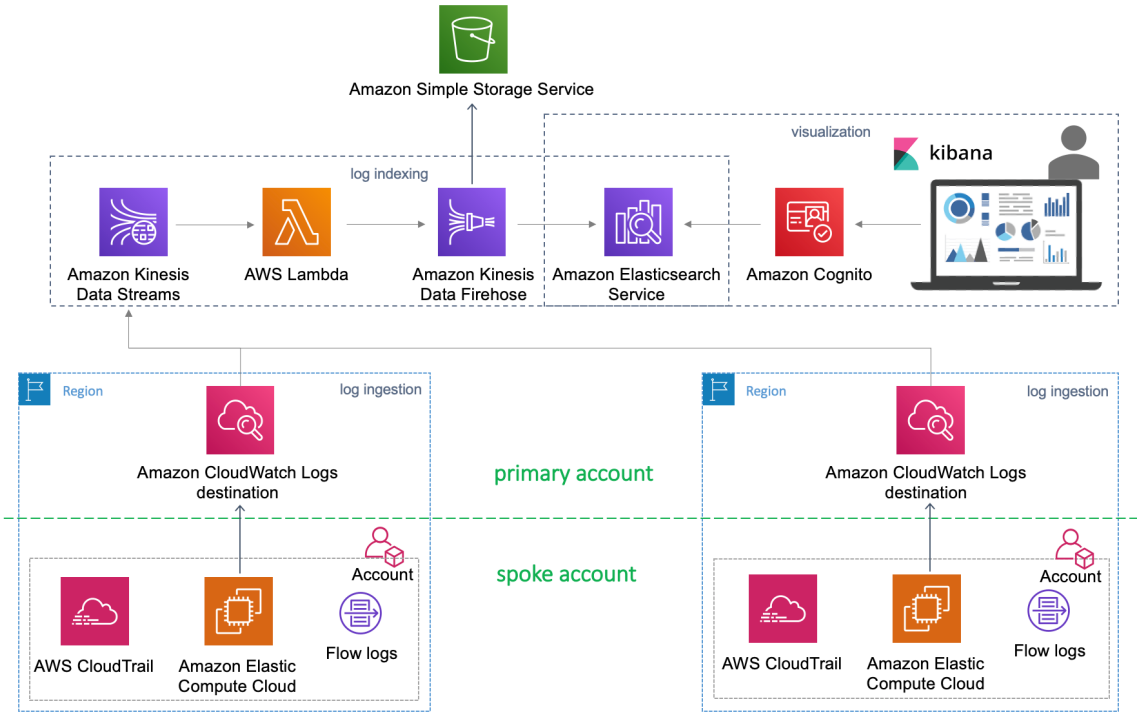
pelöksi, kun virhetilanteessa, joudutaan käsin etsimään, missä ilmentymässä virhe on tapahtunut. Lokien kerääminen keskitettyyn lokivarastoon, joka on haettavissa ja visualisoitavissa on järkevämpi vaihtoehto skaalautuvassa järjestelmässä. Keskityksen tärkeys korostuu varsinkin hajautetussa mikropalveluarkkitehtuurilla toteutetussa järjestelmässä, jossa yksittäinen toiminto asiakassovelluksesta aiheuttaa pyyntöjen ketjun, joka kattaa useita mikropalveluita. [1, 6]

Pilvinatiivien sovellusten perusperiaate on, että ne lokittavat standarditulosteeseen ja standardivirheeseen. Standardivirtoja hyödyntämällä lokitus saadaan yhtenäistettyä helposti eri komponenttien välillä ja tehtyä riippumattomaksi ympäristöstä. Skaalautuvan järjestelmän lokiviesteissä tarvitaan konteksti-informaatiota ilmentymistä, jotta lokien tutkiminen on mielekästä. Kontekstista on löydyttävä ainakin palvelun nimi, ilmentymän tunniste, konttikuvan versio, jos sovellusta ajetaan kontissa tai sovelluksen versio, jos sovellusta ajetaan virtuaalikoneessa. Maantieteellisesti hajautetun sovelluksen tapauksessa lokituksessa kannattaa käyttää yhtenäistä aikavyöhykettä, jotta lokirivien kronologinen järjestys eri ilmentymien välillä on helppo hahmottaa. Hajautetun järjestelmän lokeissa kannattaa olla jäljityksen (tracing) helpottamiseksi korrelaatiotunniste (correlation ID), joka on sama kaikissa samaa pyyntöä käsittelevissä palveluissa. Jäljittämisen käsittely tarkemmin on tämän työn laajuuden ulkopuolella. [1, 6, 26]

Lokien vastaanotossa keskitettyyn järjestelmään kannattaa ennakoida suurta volyymia. Lokiviestien yleinen ongelma on, että niissä on huono signaali-kohina-suhde, eli suurin osa lokiriveistä on turhia. Rivit ovat turhia, koska ne eivät anna mitään hyödyllistä informaatiota järjestelmän toimiessa odotetusti. Vikatilanteessa samat rivit saattavat antaa usein hyödyllistä tietoa järjestelmän tilasta ennen vikatilannetta ja ongelman selvittäjä saa riveistä kuvan järjestelmän normaalista toiminnasta. Lokituslogiikkaa on kuitenkin hankala rakentaa siten, että vain hyödyllinen määrä normaalia toimintaa lokitettaisiin. Huonon signaali-kohina-suhteen takia lokitus myös skaalautuu huonosti. Jokainen pyyntö tuottaa useita lokirivejä, joista suurin osa on turhia, jotka pitää kaikki siirtää keskitettyyn lokijärjestelmään. Koko prosessi vie resursseja lokiviestien muotoilemisesta keskitettyyn järjestelmään vientiin. Sovelluskehittäjän on siis hyvä olla tietoinen lokituksen rajoista, yrittäessään ennustaa, mitä viestejä lokitetaan. [6, 51]

Vastaanoton on hyvä skaalautua tarvittaessa ja sisältää puskureita, koska sovellusten kuormituspiikit aiheuttavat piikin myös lokeissa. Jotta lokiviestien suuri volyymi olisi paremmin hallittavissa, lokit kannattaa kirjoittaa rakenteellisessa muodossa. Keskitetyn lokijärjestelmän on helpompi parsia ja indeksoida rakenteelliset viestit tietokantaan, josta niitä voidaan hakea. Järjestelmässä on hyvä olla myös kyky vastaanottaa vapaamuotoisia lokeja, sillä kehittäjät eivät usein voi hallita 3. osapuolien kirjastojen tekemiä lokiviestejä. Lokijärjestelmässä olisi myös hyvä olla mahdollisuus tehdä hälytyksiä tietyistä viesteistä, jotka liittyvät esimerkiksi virheisiin ja jotka vaativat ylläpidolta välittömästi huomiota. Hälytyksiä ei kannata tehdä liikaa sillä ylläpitäjät väsyvät niihin nopeasti ja lopettavat reagoimasta, jos järjestelmä hälyttää asioista jotka eivät vaadi nopeasti huomiota.[1, 6, 26]

AWS ei toteuta keskitettyä lokitusta yhtenä palveluna, vaan useammasta erillisestä pal-



Kuva 2.5. *AWS;n keskitetyn lokitusratkaisun arkkitehtuuri. [33, Section: Centralized Logging]*

velusta koostuvana kokonaisuutena. Kokonaisuuden voi ottaa helposti käyttöön AWS:n virallisella CloudFormation-sapluunalla. Kokonaisuuden arkkitehtuuri on havainnollistettu kuvassa 2.5. CloudFormation-sapluunat, ovat JSON- tai YAML-syntaksilla kirjoitettuja konfiguraatiotiedostoja, jotka sisältävät tiedot siitä luotavista AWS-palveluista ja niiden konfiguraatioista. Keskitetyn lokituksen sapluuna koostuu kolmesta osasta: sisäänotto (ingestion), indeksointi ja visualisointi. [33, Section: Centralized Logging, AWS CloudFormation template formats]

Sisäänottovaiheessa Amazon CloudWatch Logs -palvelu vastaanottaa kaikki sovelluksilta ja palveluilta tulevat lokit. CloudWatch-ilmentymiä on yksi jokaista järjestelmään kuuluvaa Region-aluetta kohti. CloudWatch on agenttipohjainen järjestelmä. Elastic Compute Cloud (EC2) -virtuaalikoneille agentti asennetaan kuten mikä tahansa paketti. ECS -ympäristössä ajossa olevilla konteilla toiminnallisuus on osa ECS Container Agent kokonaisuutta, joka kytkee yksittäiset kontit osaksi laajempaa kokonaisuutta. EKS -ympäristöihin agentti konfiguroidaan osana Container Insights -kokonaisuutta, joka kerää laajasti metriikkatietoja, Kuberneteksen omia lokeja ja sovelluslokeja EKS:stä. Sisäänottoon kuuluvat myös AWS Cloud Trail -palvelun tuottamat lokit, jotka sisältävät AWS-tasolla tapahtuvat toimenpiteet kuten erilaiset API-kutsut, joilla luodaan palveluiden ilmentymiä. [13, Section: Centralized Logging, Amazon CloudWatch, Amazon ECS CloudWatch metrics, Logging and Monitoring in Amazon Elastic Container Service, Set Up the CloudWatch Agent to Collect Cluster Metrics] [33, Section: AWS CloudTrail]

CloudWatchista lokit ohjataan eteenpäin indeksoitaviksi. Amazon Kinesis Data Stream -palvelu vastaanottaa eri CloudWatch ilmentymiltä tulevat lokit. Kinesis Data Streams

on datan liikuttamiseen ja prosessointiin tarkoitettu palvelu. Palvelu skaalautuu nopeasti erikokoisille datamäärille realiaikaisen tarpeen mukaan. Prosessoinnin voi toteuttaa Amazonin Big Data palveluilla tai omilla palvelittomilla AWS Lambda -funktioilla tai perinteisillä EC2-virtuaalikoneilla suorittavilla ohjelmilla. Keskitetyn lokituksen CloudWatch-sapluunassa lokit prosessoidaan AWS Lambda -funktioilla. Lokit muokataan muotoon, jota ElasticSearch voi indeksoida. ElasticSearch on hakukone, jota käytetään usein lokitietojen hakemiseen osana ELK-pinoa (Elasticsearch-Logstash-Kibana). Amazon ElasticSearch -palvelun ja Lambda-funktoiden välissä on Amazon Kinesis Firehose, jonka rooli on syöttää dataa eteenpäin Kinesis-putkesta. Kinesis Data Firehose pakkaa, puskuroi ja salaa lähetettävät tiedot, jotta datan käsittely olisi mahdollisimman helppoa vastaanottavalla palvelulle. [33, Sectiom: Centralized Logging, Amazon Kinesis Data Streams, AWS Lambda, Amazon Kinesis Data Firehose, Amazon Elasticsearch Service] [26]

Visualisointiin käytetään Kibanaa, joka ElasticSearchin tavoin on osa ELK-pinoa. Kibanaan tehtävät visualisoinnit ja haut käyttävät ElasticSearchia oikeiden lokien löytämiseen. Osana keskitettyä lokitusta Kibana suljetaan omaan yksityisverkkoonsa Amazon Virtual Private Cloud -palvelulla, johon pääsyä hallitaan Amazon Cogntio identiteetin tarjoajalla (identity provider). Näin Kibanaan pääsevät käsiksi ne henkilöt, joilla ei ole tarvetta päästä käsiksi muihin AWS palveluihin. Jos CloudWatchia olisi käytetty visualisointiin ja hakuihin vastaavan pääsynhallinan toteuttaminen olisi vaikeampaa. [33, Amazon Elasticsearch Service, Amazon Virtual Private Cloud] [13, Section: Amazon Cognito Documentation] [26]

3 KOHDEORGANISAAATION VAATIMUKSET

Tämän työn tavoitteena on suunnitella, osittain toteuttaa ja arvioida ratkaisut, joilla työn kohdeorganisaatio pääsee siirtymään nykyisestä virtuaalikoneisiin nojaavasta ympäristöstään modernimpaan sovelluskontteja hyödyntävään ympäristöön. Kohdeorganisaatio on päättänyt jatkaa yhteistyötä nykyisen konesalipalvelutarjoajansa kanssa. Palveluntarjoajan tarjoama ympäristö on Nutanixin Kubernetes-tuote Karbon, joten sen käyttö tulee työn näkökulmasta määrättynä. Tässä luvussa esitellään ne vaatimukset, jotka kohdeorganisaatio on tunnistanut Karbon-ympäristön saamiseksi tuotantokelpoiseksi. Kaikille vaatimuksille yhteistä on se, että kohdeorganisaatio pyrkii välttämään toimittajaloukkua (vendor lock-in) ja valittujen teknologioiden halutaan olevan ylläpidettyjä keskipitkällä aikavälillä eli 3 vuoden ajan.

3.1 Reititys

Nykyinen reititys on toteutettu joukolla käänteisproxyja. Proxyja on ylläpidetty käsin ja niiden konfiguraatioita on muokattu suoraan palvelimille sitä mukaa, kuin niihin on ollut tarvetta tehdä muutoksia. Liikenteen salaamisessa käytettyjä TLS-varmenteita on hallinnoitu käsin. Niiden luominen, toimitus oikeaan paikkaan ja päivittäminen on manuaalista työtä, joka ylläpidon pitää muistaa tehdä.

Karbon-ympäristössä palveluita on tarkoitus julkaista vanhaan tapaan, sekä yrityksen sisäverkossa että julkiverkossa. Julkiverkkoon julkaistaan yrityksen itselleen ja asiakkailleen ylläpitämiä palveluja ja sisäverkkoon palveluja, jotka liittyvät lähinnä infrastruktuuriin kuten monitorointiin. Julkiverkon palveluiden reitityksestä halutaan mahdollisimman automaattista. Valitun reititysteknologian pitäisi pystyä tunnistamaan Kubernetes-klusterissa ajossa olevat palvelut, luomaan reitit ja tarvittavat varmenteet.

Tällä hetkellä kohdeorganisaatiolla ei ole itsellään ylläpidossa palveluita, jotka kykenisivät arkkitehtuurinsa puolesta tehokkaaseen horisontaaliseen taakanjakoon. Mahdollisuus taakanjakoon on kuitenkin oltava suunnitellussa reititysratkaisussa, jotta myöhemmin ei jouduta tekemään rinnakkaista ratkaisua yksittäistä sovellusta varten.

3.2 Klusterissa olevan datan varmuuskopiointi

Osa sovelluksista tallentaa levylle tiedostoja, joita sovellus generoi itsenäisesti, tai joita käyttäjät lisäävät sinne. Tällaisia tiedostoja ovat esimerkiksi kuvat, PDF-tiedostot ja STL-

formaatissa olevat 3D-mallit. Vaatimuksena on, että nämä tiedostot saadaan varmuuskopioitua klusterin ulkopuolelle, josta ne ovat tarvittaessa palautettavissa. Varmuuskopioinnin täytyy olla automaattista ja olla ajastettavissa tapahtumaan halutulla aikataululla. Varmuuskopioinnin pitää olla myös kohdennettavissa tuotanto- ja testiympäristöihin erikseen.

3.3 Sovelluslokien tallettaminen klusterin ulkopuolelle

Useimmissa sovelluksissa lokeja on säilytetty sovellusten virtuaalikoneiden levyllä. Virhetilanteissa kehittäjät ovat ottaneet SSH-yhteyden virtuaalikoneelle ja lukenee lokit suoraan levyltä. Joidenkin palveluiden kohdalla on päädytty keräämään lokit keskitetylle lokipalvelimelle Filebeat lokiagenteilla. Keräimien data on indeksoitu Elasticsearchiin ja visualisoitu Kibanalla. [52, Section: ElasticSearch, Filebeat, Kibana]. Filebeatin, Elasticsearchin ja Kibanan toiminta on kuvattu tarkemmin aliluvussa 4.3.

Tämän työn tavoitteena on, että kehittäjien ei tarvitsisi erikseen ottaa yhteyttä sovellusilmentymään lukeakseen sen lokeja, vaan, että lokit kerättäisiin keskitetysti yhteen paikkaan. Tavoitteena on myös, että lokien kerääminen olisi mahdollisimman yhtenäistä eri sovellusten välillä. Ensijaisesti lokien esittämisessä on tarkoitus käyttää olemassa olevaa Elastic-pinoa (ElasticSearch ja Kibana), mutta tarvittaessa se voidaan korvata paremmin soveltuvilla teknologioilla.

4 YLLÄPIDOLLISET RATKAISUT

Tässä työssä ratkaistiin kolme tärkeintä ylläpidollista ongelmaa liittyen Karbonin käyttöönottoon tuotantoympäristönä. Koska Karbon on hoidettu Kubernetes-ratkaisu työssä ei tarvinnut keskittyä klusterin itsensä pystyttämiseen.

4.1 Reititys

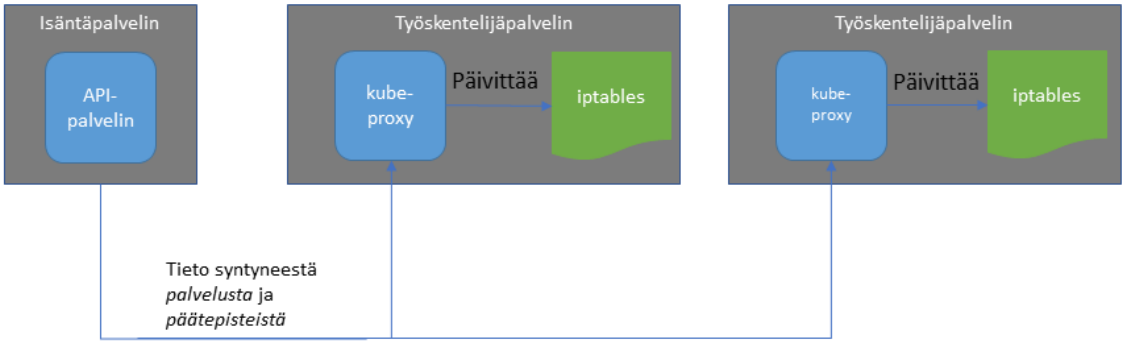
Reitityksellä tarkoitetaan tämän työn kontekstissa klusteriin tulevien HTTP/HTTPS-pyyntöjen ohjaamista oikeille sovelluksille. Sovellukset ovat ajossa Kubernetes-klusterissa, joka on kohdeorganisaation sisäverkossa ja niihin täytyy saada yhteys ulkoverkosta. Valitun reititysteknologian pitää mahdollistaa pyynnön tekeminen ulkoverkosta ja pyynnön ohjautuminen lopulta sovellusilmentymälle, joka on ajossa *kapselissa*.

4.1.1 Klusterin sisäinen liikenne

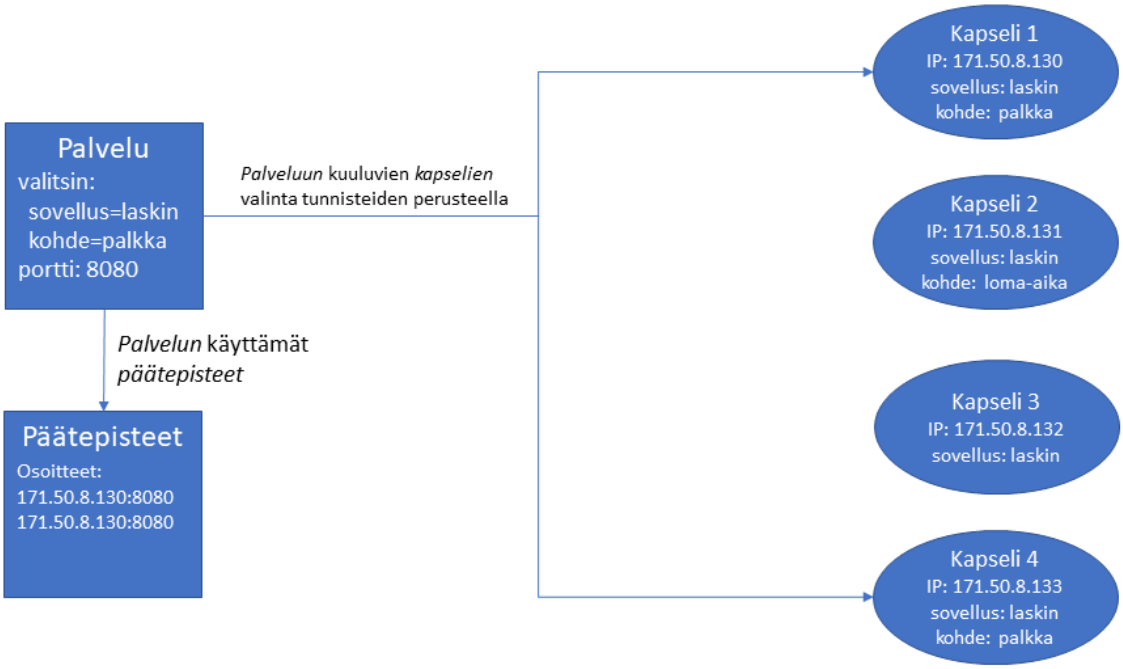
Kuberneteksessa palveluhaku (service-discovery) on valmiiksi sisäänrakennettuna. Kuberneteksen sisäinen reititys vastaa hyvin kohdeorganisaation vaatimukseen mahdollisimman automaattisesta reitityksestä. Kuberneteksen palveluhaun keskiössä ovat *palvelu* (service) ja *päätepisteet* (Endpoints) -oliot. Palveluhaku on hyödyntää myös kube-proxya ja useimmissa klustereissa olevaa DNS-lisäosaa. Tässä luvussa käsitellään *palveluita* klusterin sisäisen liikenteen näkökulmasta ja klusterin ulkopuolinen liikenne käsitellään aliluvussa 4.1.2. [1, 7]

Palvelu on Kuberneteksen versio palvelurekisteristä, jonka kautta asiakassovellukset (client) voivat olla yhteydessä sovelluksiin. Asiakassovellukset voivat olla muita samassa klusterissa olevia sovelluksia tai klusterin ulkopuolisia sovelluksia. *Palvelu*-oliolle määritellään portti ja Kubernetes antaa *palvelulle* klusterin sisäisen IP-osoitteen välittömästi kun se luodaan. Asiakassovellukset ottavat yhteyttä tähän IP-osoite-portti-pariin, josta liikenne uudelleenohjataan sovelluksen ilmentymille, jotka sijaitsevat *kapseleissa* (Pod). [24, Section: Service] [1]

Kuvassa 4.1 Kuberneteksen API-palvelin välittää tiedot luoduista *palveluista* ja *päätepisteistä* kaikille klusterin kube-proxy ilmentymille, jotka sijaitsevat klusterin työskentelijäpalvelimilla. Kube-proxyt muokkaavat palvelimiensa iptables-sääntöjä, siten että *palveluiden* IP-osoite-portti-parit yhdistetään *päätepisteiden* IP-osoite-portti-pareihin. Näiden iptables-sääntöjen perusteella kutsut, jotka tehdään *palveluille* reitetään uudelleen



Kuva 4.1. *Palveluiden ja päätepisteiden tiedot kulkevat API-palvelimen kautta työntekijäpalvelinten kube-proxyille.*



Kuva 4.2. *Palvelu tunnistaa sille kuuluvat kapselit tunnisteiden perusteella. Palvelulle tuleva liikenne ohjataan päätepisteissä oleviin osoitteisiin.*

kernel-tasolla. Yksinkertaistuksena tässä työssä puhutaan jatkossa, että *palvelu* ohjaa pyynnöt kapseleille. [7][24, Section: kube-proxy]

Palvelu tunnistaa sen vastuulla olevat *kapselit* sen määrittelyssä olevien *valitsimien* (selector) perusteella. *Valitsimet* ovat mielivaltaisia avain-arvo-pareja, jotka vastaavat *palvelun* kohteena olevalle *kapselille* määriteltyjä tunnisteita (label). Valitsimien perusteella *palveluille* luodaan *päätepisteet*, joka sisältää valitsimeen osuvien *kapselien* IP-osoitteet ja portit joita ne kuuntelevat. Tämä mekanismi on havainnollistettu kuvassa 4.2. *Päätepisteiden* tiedot päivittyvät aina kun *palvelun* vastuulla oleva *kapseli* luodaan tai tuhotaan. Tarvittaessa *palveluille* voidaan määritellä kohdeportti, joka vastaa sitä porttia, jota sovellus kuuntelee. *Kapseleiden* IP-osoitteet ovat aina oletuksena klusterin sisäisen verkon IP-osoitteita. [7] [24, Section: Service, Labels and Selectors]

Kubernetes injektoi *kapselin* ympäristömuuttujiin tiedot kaikista luontihetkellä olevista *palveluiden* IP-osoite-portti-pareista. Tämä on yksinkertaisin tapa klusterissa oleville sovelluksille löytää muut klusterissa olevat sovellukset ilman manuaalista konfigurointia. Tämä tapa on aina käytössä kaikissa klustereissa. Kuberneteksessa ei ole toteutusta nimipalvelulle, mutta käytännössä kaikissa kolmannen osapuolen hoidetuissa (managed) Kubernetes-klustereissa on nimipalvelu asennettuna [1, 7]. Kuberneteksen kehittäjät ovat luoneet määrittelyn, jonka tarkoituksena on pitää käyttäjälle näkyvä toiminnallisuus yhdenmukaisena eri toteutusten välillä [53]. Yleisin Kuberneteksessa käytetty nimipalvelutoteutus on CoreDNS, joka korvasi vuonna 2018 Kube-DNS:n. Karbonissa on kirjoitushetkellä käytössä Kube-DNS, sen erot CoreDNS:ään ovat suorituskyvyssä. CoreDNS:n ja Kube-DNS:n toimintaperiaatteet ovat samat [54].

Kaikki klusterin *kapselit* konfiguroidaan käyttämään klusteriin asennettua nimipalvelua automaattisesti. Klusterin sisäinen nimipalvelu sisältää ajantasaisen tiedon kaikista klusterissa olevista *palveluista* ja niille muodostetaan verkkotunnukset. Verkkotunnukset ovat tyyppiä <*palvelun* nimi>.<*palvelun* nimiavaruus>.<klusterin pääte>. *Palvelun* nimiavaruus tarkoittaa Kubernetes-klusterin sisäistä nimiavaruutta ja klusterin pääte on konfiguroitavissa oleva loppuosa, jolla voidaan erotella klusterit toisistaan. Kun viittaukset tapahtuvat saman klusterin sisällä on loppuosa oletuksena svc.cluster.local. Klusterin sisällä tapahtuvissa kutsuissa loppuosaa ei tarvita ja nimiavaruuden voi jättää pois nimiavaruuden sisällä tapahtuvissa kutsuissa. Nimipalvelua hyödyntämällä kehittäjien ei tarvitse välittää siitä, missä järjestyksessä *palvelut* ja *kapselit* luodaan ja luodaanko *palvelu* uudelleen, jolloin sen IP-osoite vaihtuu. Verkkotunnukset myös helpottavat sovelluksen toiminnan ymmärtämistä ja ylläpitoa, sillä ne ovat informatiivisempia ja selkeämpiä kuin IP-osoitteiden numerosarjat. Tarvittaessa käytetty nimipalvelu voidaan määritellä *kapseleiden* oliomanifesteissa. Jokainen *kapselissa* tapahtuva DNS-kysely ohjataan konfiguroituun nimipalveluun. [7]

Palvelun täytyy olla jatkuvasti tietoinen sen takana olevan sovelluksien ilmentymien tilasta, jotta se ei ohjaa liikennettä vikatilassa tai kuormituksen alaisena oleville ilmentymille. Kuberneteksessa on mahdollista määritellä *eloluotain* (liveness probe), *käynnistysluotain* (startup probe) ja *valmiusluotain* (readiness probe) näitä tehtäviä varten. Luotaimet määritellään konttikohtaisesti, sillä *kapseli* voi sisältää useamman erityyppisen kontin. Jokaiselle luotaintyypille on olemassa kolme vaihtoehtoista tapaa luodata konttia: HTTP GET -luotain, TCP-pistokeluotain ja Exec-luotain. Kaikille luotaimille voidaan määrittää kuinka usein luotaus suoritetaan, aikakatkaisu (timeout) ja kuinka monta kertaa luotain saa epäonnistua. [1, 7] [24, Section: Configure Liveness, Readiness and Startup Probes]

Jos *eloluotain* saa jonkin muun vastauksen kuin, mikä sille on konfiguroitu, Kubernetes katsoo, että luodattu kontti on vikatilassa ja käynnistää kontin uudelleen. *Käynnistysluotain* kuuluu määritellä suorittamaan sama tarkastus kuin *eloluotain*, mutta sille määritellään korkeampi sallittu epäonnistumisten määrä. Kertomalla epäonnistumisten määrä luotaustiheydellä saadaan käynnistymisaika. Jos kontti käynnisty määritellyssä ajassa se tuhotaan ja luodaan uudelleen. Jos *valmiusluotain* palauttaa jonkin muun vastauksen,

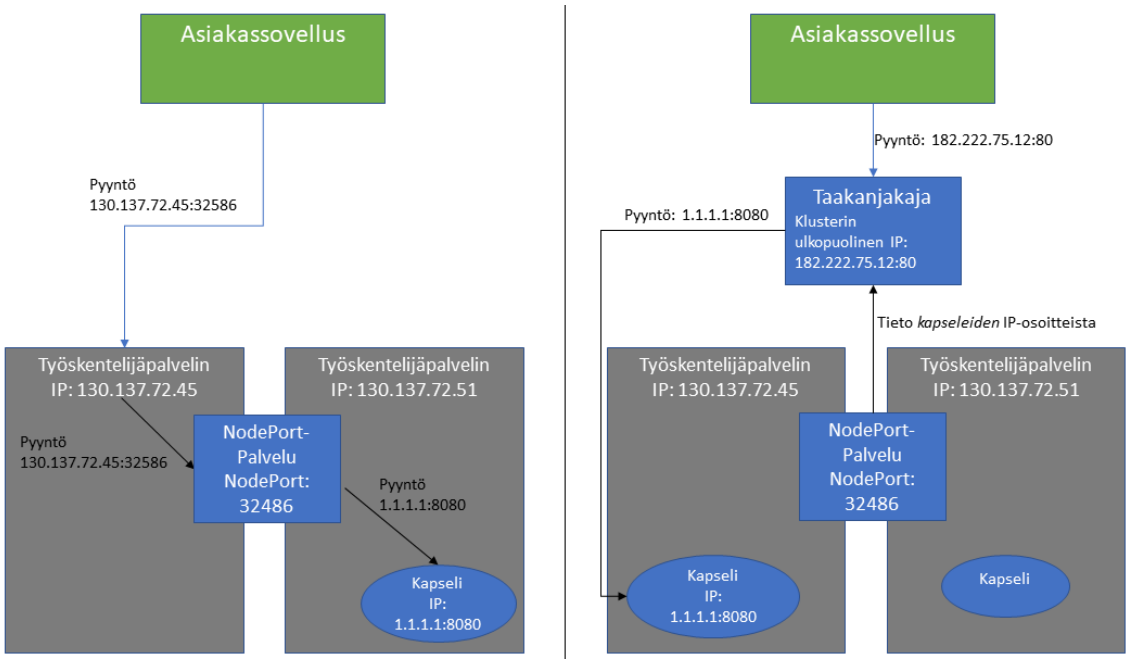
kuin mikä sille on konfiguroitu, Kubernetes katsoo, että kyseinen kontti ja sitä kautta *kapseli* ei ole valmis vastaanottamaan liikennettä. Kubernetes ei ohjaa *kapselille* liikennettä ennen kuin valmiusluotain palauttaa halutun tuloksen. Se mikä katsotaan vikatilaksi ja valmiudeksi on aina sovelluskohtaista, joten on sovelluksen kehittäjien vastuulla toteuttaa mielekkäät kohteet luotaimille. [1, 7] [24, Section: Configure Liveness, Readiness and Startup]

4.1.2 Klusterin ulkopuolinen liikenne

Palveluille voidaan asettaa tyyppi sen mukaan minne sen halutaan näkyvän. Oletustyyppi on ClusterIP, jonka toiminta on kuvattu aliluvussa 4.1.1. Muita tyyppejä ovat NodePort ja LoadBalancer. NodePort-*palvelu* saa Kubernetekselta portin jokaiselta klusterin muodostavalta työntekijä- ja isäntäpalvelimelta. Portti on sama kaikilla klusterin palvelimilla. *Palvelun* takana oleviin sovelluksiin saa yhteyden minkä tahansa klusterin palvelimen IP-osoitteella ja *palvelulle* annetulla portilla. Portin voi määrittää itse oliomanifestissa tai jättää määrittelemättä, jolloin Kubernetes varaa satunnaisen portin. Pyynnöt ohjataan lopulta luvussa 4.1.1 kuvatuilla iptables-säännöillä oikeille *kapseleille*. *Palvelu* saa myös normaalin klusterin sisäisen IP-osoitteen, jolloin se on löydettävissä verkkotunnuksella. [7] [24, Section: Service]

LoadBalancer-*palvelu* vaatii, että pilvi-infrastruktuuri, jossa Kubernetes-klusteri on ajossa, tarjoaa toteutuksen taakanjaolle. Kun infrastruktuuri tukee taakanjakoa klusterin ulkopuoliselle liikenteelle, LoadBalancer-*palvelu* saa klusterin ulkopuolisen IP-osoitteen siitä verkosta johon taakanjako on konfiguroitu. Kun *palvelu* luodaan, Kubernetes luo samalla NodePort-palvelun, jota taakanjakaja hyödyntää. Taakanjakaja ohjaa liikenteen *kapseleille*. NodePort-*palvelun* ja LoadBalancer-*palvelun* toiminta on kuvattu kuvassa 4.3. Vasemmalla on kuvattu pyynnön ohjautuminen asiakassovellukselta *kapsellille*. Pyyntö menee ensimmäisenä palvelimelle, jolla haluttu *kapseli* ei sijaitse. NodePort-*palvelu* ohjaa pyynnön oikealle *kapselille*. Oikealla on kuvattu asiakassovelluksen tekemä pyyntö taakanjakajalle, jolla on klusterin ulkopuolisen verkon IP-osoite. Taakanjakaja ohjaa pyynnön oikealle *kapselille*. Taakanjakopalveluiden tarjoaminen on hyvin tyypillistä julkisina palveluina tarjottaville Kubernetes-klustereille. Näillä pilvialustoilla julkisen IP-osoitteen saaminen *palvelulle* on hyvin helppoa. Jotkut palveluntarjoajat tarjoavat mahdollisuuden asettaa IP-osoitteen *palvelun* loadBalancerIP-kentän kautta. Jos tätä mahdollisuutta ei ole niin loadBalancerIP-kenttä ohitetaan. Taakanjaossa käytetty algoritmi riippuu alustan tarjomasta toteutuksesta. Kuberneteksen tarjoama mahdollisuus LoadBalancer-tyyppisten *palveluiden* kautta vastaa vaatimukseen mahdollisuudesta taakanjakoon.[7][24, Section: Service]

Palveluilla yhden klusterin ulkopuolisen IP-osoitteen saa kytkettyä haluttuihin palveluihin. *Palvelut* kuitenkin toimivat OSI-mallin [55] kuljetuskerroksella (taso 4), eivätkä pysty tekemään reitityspäätöksiä esimerkiksi pyynnön host otsikkotiedon perusteella. Jos reititys on rakennettu puhtaasti *palveluiden* varaan, jokaista palvelukokonaisuutta varten



Kuva 4.3. *NodePort-palvelun ja LoadBalancer-palvelun toiminta Kuberneteksessa.*

täytyy varata oma IP-osoitteensa [7, 56]. Kuberneteksessa on monipuolisempaa reitittämistä varten OSI-mallin sovelluskerroksella (taso 7) toimiva *sisääntulo* (Ingress). *Sisääntulo* vaatii toimiakseen *sisääntulo-ohjaimen* (Ingress Controller), joka vastaa IP-osoitteen hankkimisesta *sisääntulolle* [24, Section: Ingress]. Useimmissa hoidetuissa Kubernetes-klustereissa on valmiina *sisääntulo-ohjain*. Karbonissa *sisääntulo-ohjaimen* toteutuksen valitseminen ja käyttöönotto on jätetty klusterin käyttäjän vastuulle [7, 56].

Sisääntulo pystyy toimimaan porttina usealle eri palvelulle ja jakamaan liikennettä palveluille HTTP-sääntöjen perusteella. Liikennettä voidaan jakaa pyynnöön host otsikkotiedon ja polun perusteella. Host otsikkotieto voi olla tarkka tai sisältää jokerimerkin, jolloin riittää, että host otsikkotiedon *loppuosa* täsmää reitityssäännön kanssa. Polkupohjaiset säännöt voivat olla tarkkoja osapolkuja, jotka täsmäävät polun alkuosaan tai ohjainkohtaisia (Implementation specific). Ohjainkohtaisissa poluissa täsmäyssäännöt ovat riippuvaisia *sisääntulon* toteutuksesta. [7][24, Section: Ingress]

Sisääntulo voi myös vastaanottaa TLS-liikennettä (Transport Layer Security), sillä oletuksella, että liikenne tulee portin 443 kautta. Salaus päättyy *sisäntuloon* ja liikenne siitä eteenpäin on salaamatonta. *Sisääntulo* ei kirjoitushetkellä tue jokerimerkkisiä varmenteita, joten host-otsikkotietoon pohjautuva reititys pitää tapahtua tarkkana täsmäyksenä. Kuberneteksen oman dokumentaation mukaan TLS-ominaisuuksissa voi olla eroja eri *sisääntulo-ohjain* toteutusten välillä. [24, Section: Ingress]

Reititys sisäverkossa

Kohdeorganisaation tarpeet sisäverkon palveluille ovat hyvin kevyitä ja sisäverkossa tapahtuva reititys haluttiin pitää mahdollisimman yksinkertaisena. Kubernetes-kontekstissa,

tämä tarkoittaa sitä, että halutulle sovellukselle luodaan LoadBalancer-*palvelu*, joka saa sisäverkon ip-osoitteen. Näin haluttiin varmistaa, että sisäverkon palveluiden konfigurointi noudattaa samoja periaatteita, kuin silloin kuin palvelut ovat virtuaalikoneilla sisäverkossa.

Tässä työssä tehdyssä selvityksessä valittiin sisäverkon reitiystä varten Apache 2.0 lisensoitu MetalLB taakanjakaja, joka on suunniteltu raudalla (bare metal) toimivien klustereiden käyttöön. Raudalla tarkoitetaan tässä kontekstissa itse pystytettyjä ja ylläpidettyjä Kubernetes-klustereita. Tällaiset klusterit eivät hyödy Kuberneteksen LoadBalancer-*palveluiden* toteutukesta, koska se on suunniteltu integroitumaan suurten pilvialustojen kuten Amazonin Elastic Kubernetes Servicen ja Microsoftin Azure Kubernetes Servicen kanssa. MetalLB:n tekijöiden tavoitteena on tuoda rautaklustereihin sama mahdollisuus hyödyntää LoadBalancer-*palveluita* kuin, mikä on suurilla pilvipalvelujen tarjoajilla. [57, Section: MetalLB]

Organisaation reititysvaatimukset koskivat ulkoverkon palveluita, joten MetalLB:n kohdalla ei ole kriittistä, että se ei tarjoa automaattista varmenteiden hallintaa. MetalLB kuitenkin tarjoaa mahdollisuuden hyödyntää LoadBalancer-*palveluita* ja sitä kautta taakanjakoa sekä klusterin sisäistä automaattista reititystä. Vaihtoehtoisia taakanjaon toteuttavia teknologioita olisivat PureLB ja OpenELB. OpenELB projektin entinen nimi on Porter [58]. Kohdeorganisaation kannalta merkittävin ero MetalLB:n ja PureLB:n ja OpenELB:n välillä on se, että PureLB:n ja OpenELB:n voi konfiguroida niiden omilla *omaolioilla* kun taas MetalLB pitää konfiguroida Kuberneteksen geneerisillä konfiguraatio-olioilla. *Omaolioiden* etu on, että ne sisältävät nimetyt kentät ja pystyvät suorittamaan konfiguraatioiden oikeellisuden tarkistuksen jo luontihetkellä. Geneerisen konfiguraatio-olion sisältämän konfiguraation oikeellisuus tulee ilmi vasta kun sitä käyttävä sovellus käynnistyy. Muut erot kuten PureLB:n integraatio Linuxin verkkojärjestelmiin ja IPv6 tuki eivät ole kohdeorganisaatiolle tärkeitä [59]. MetalLB:n selkeä etu oli, että sen käyttöönotolle Karbonissa oli olemassa Nutanixin työntekijän tekemät ohjeet [60]. Karbonin julkinen dokumentaatio on hyvin suppea, joten suoraan ei ole varmaa toimisivatko muut taakanjakajat Karbonissa tai mitä konfiguraatiomuutoksia Karboniin pitäisi tehdä, jotta ne toimisivat.

MetalLB asennetaan palveluna klusteriin osaksi klusterin infrastruktuuria. Seuraavien vaatimusten pitää täyttyä, jotta MetalLB:n käyttö on mahdollista:

- Kubernetes-klusteri, jonka versio on suurempi kuin 1.13.0 ja jossa ei ole toista taakanjako järjestelmää.
- MetalLB:n kanssa yhteensopiva Container Network Interface (CNI) -lisäosa.
- Alue verkosta, josta MetalLB voi jakaa IPv4 osoitteita.
- Liikenne klusterin verkon portissa 7946 pitää olla sallittua.

Karbon tukee kirjoitushetkellä Kuberneteksen versioita 1.17.0 asti, joten ensimmäinen vaatimus on täytetty. Kuberneteksessa on vaatimukset sille, miten klusterin verkon pitää toimia *kapselien* näkökulmasta. Karkeasti jokaisen *kapselin* pitää saada oma uniikki IP-osoite ja pystyä kommunikoimaan kaikkien muiden klusterin *kapselien* kanssa näiden

IP-osoitteilla. Kommunikoinnin pitää olla mahdollista riippumatta siitä, millä palvelimilla *kapselit* ovat. Lisäksi verkon täytyy olla yksitasoinen (flat network) ja vailla osoitteenmuunnoksia (NAT-less). Verkko on helpointa toteuttaa ohjelmallisesti määriteltynä verkkona (Software Defined Network), jonka avulla fyysinen verkkoinfrastruktuuri voidaan abstraktoida ja palvelimet näkyvät toisilleen ikään kuin ne olisivat samassa verkossa. Kuberneteksessa yksi vaihtoehto tähän on käyttää CNI-lisäosaa, joka täyttää Cloud Native Computing Foundationin CNI-projektin spesifikaation konttien välisestä kommunikoinnista [61]. Karbonilla pystytetyt klusterit käyttävät Flannel-lisäosaa, joka on todettu toimivaksi MetalLB kehittäjien toimesta, joten myös toinen vaatimus täytyy. Kolmas ja neljäs vaatimus ovat vaatimuksia verkolle, johon klusteri on asennettu ja ovat siten kohdeorganisaation omissa käsissä ja täytettävissä. [7] [24, Section: Cluster Networking]

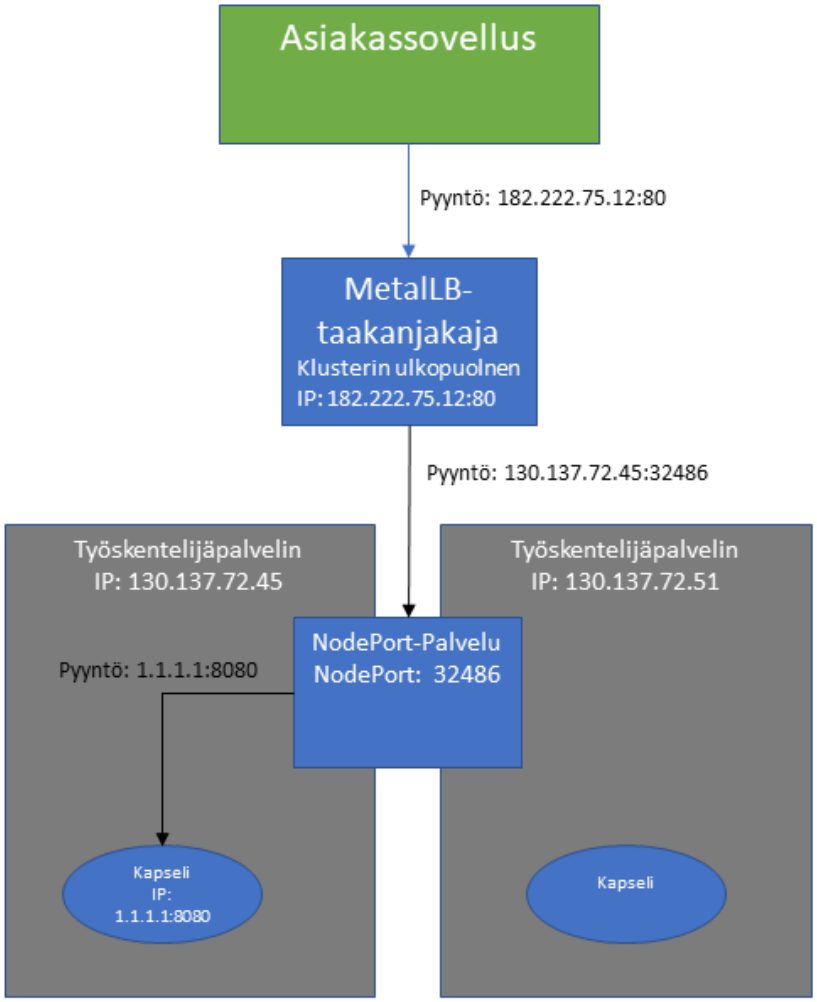
MetalLB antaa *palveluille* osoitteet siltä väliltä, joka sille on konfiguroitu. Karbonin tapauksessa pitää huomioida, että tämä alue ei mene päällekkäin sen osoitealueen kanssa, joka on allokoitu Karbonille. Se miten MetalLB ilmoittaa verkkolle, että jokin IP on otettu käyttöön ja löydettävissä klusterista, riippuu valitusta toimintamoodista. MetalLB voidaan konfiguroida joko layer 2 tai BGP-moodiin. Layer 2 on näistä yksinkertaisempi verrattuna BGP:hen, joka edellyttää verkon reitittimiltä tukea sille. Organisaatio valitsi Layer 2 -moodin sen helppokäyttöisyyden ja tointarvarmuuden vuoksi. [57, Section: Concepts, MetalLB in BGP mode, MetalLB in layer 2 mode]

Layer 2 -moodissa MetalLB ilmoitta ARP:lla (Address Resolution Protocol), mitkä IPv4 osoitteet se on jakanut sille annetulta alueelta. Verkon näkökulmasta osoite liitetään yhteen klusterin palvelimeen ja tällä palvelimella voi olla useita IP-osoitteita. Kaikki liikenne kulkee tälle yhdelle palvelimelle, josta kube-proxy ohjaa liikenteen *kapseleille* kuten aliluvussa 4.1.1 on kuvattu. MetalLB ei toteuta layer 2 -moodissa taakanjakoa, vaan antaa *palvelun* hoitaa tämän tehtävän. Tämä on havainnollistettu kuvassa 4.4. Layer 2 -moodissa palvelin, johon *palvelun* IP-osoite on liitetty, on järjestelmän pullonkaula, koska kaikki *palvelun* takana olevan sovelluksen liikenne kulkee ensin tälle yksittäiselle palvelimelle. [57, Section: MetalLB in layer 2 mode]

Jos palvelin, joka toimii *palvelun* isäntänä menee vikatilaan tai poistuu klusterista, MetalLB viestii verkkoon muille koneille, että IP-osoitteeseen liittyvä MAC-osoite (Media Access Control) on muuttunut ja liittää IP-osoitteen toiseen klusterin palvelimeen. Vaihdon piituus riippuu siitä, miten verkon muiden palvelimien käyttöjärjestelmät reagoivat MetalLB:n ilmoitukseen vaihtuneesta MAC-osoitteesta. Jos käyttöjärjestelmiin on toteutettu MetalLB:n käyttämien gratuitous-pakettien käsittely, vaihdos tapahtuu sekunneissa. Jos käsittelyä ei ole toteutettu vaihdos kestää niin pitkään, kunnes palvelimet päivittävät MAC-osoitteet sisältävät välimuistinsa. [57, Section: MetalLB in layer 2 mode]

Reititys ulkoverkossa

Kohdeorganisaation tärkeimmät valintakriteerit varsinaisten ylläpidiettävien sovellusten liikenteen reitittämiseen olivat reittien automaattinen tunnistaminen ja TLS-varmenteiden



Kuva 4.4. *MetalLB ei reititä itse pyyntöjä suoraan kapseleille, vaan ohjaa pyynnöt palvelulle. Palvelu käyttää kube-proxya pyynnön reitittämisessä valitulle kapselille.*

automaattinen luominen. Taakanjako ei ole huolenaihe tällä hetkellä ylläpidossa olevissa sovelluksissa, mutta reititysratkaisun haluttiin silti olevan konfiguroitavissa taakanjakoa varten, jos tarve ilmenee. Reititys päätettiin lopulta toteuttaa käänteisen proxyn muodossa, koska vastaava malli on ollut organisaatiossa aiemmin käytössä ja siihen liittyvät tietoturvaperiaatteet tunnettiin hyvin ja Kubernetes-klusteria ei haluttu kytkeä suoraan ulkoverkkoon.

Tässä työssä tehdyn arvioinnin tuloksena käänteiseksi proxyksi valittiin Traefik, joka on avoimen lähdekoodin reititysratkaisu, jota ylläpitää Traefik Labs. Traefikista on myös tarjolla maksullinen Traefik Enterprise Edition, jonka tuomaa lisätukea ja lisäpalveluita ei katsottu tarpeellisiksi. Traefik toimii OSI-mallin [55] tasoilla neljä ja seitsemän eli kuljetuskerroksella ja sovellustasolla. Taakanjako on mahdollista molemmilla tasoilla. Traefik tukee automaattista varmenteiden luontia ja osaa reitittää liikennettä palveluihin automaattisesti useissa eri ympäristöissä Kuberneteksen lisäksi. Kirjoitushetkellä useamman Traefik-*kapselin* ajaminen rinnakkain riikkoo automaattisen TLS-varmenteiden hankkimisen, koska eri Traefik-ilmentymä saattaa vastata haasteeseen kuin varmenteen vahvis-

tusta pyytänyt Traefik-ilmentymä, mikä myös puolsi päätöstä asentaa Traefik käänteisproxynä erilliselle palvelimelle. [32]

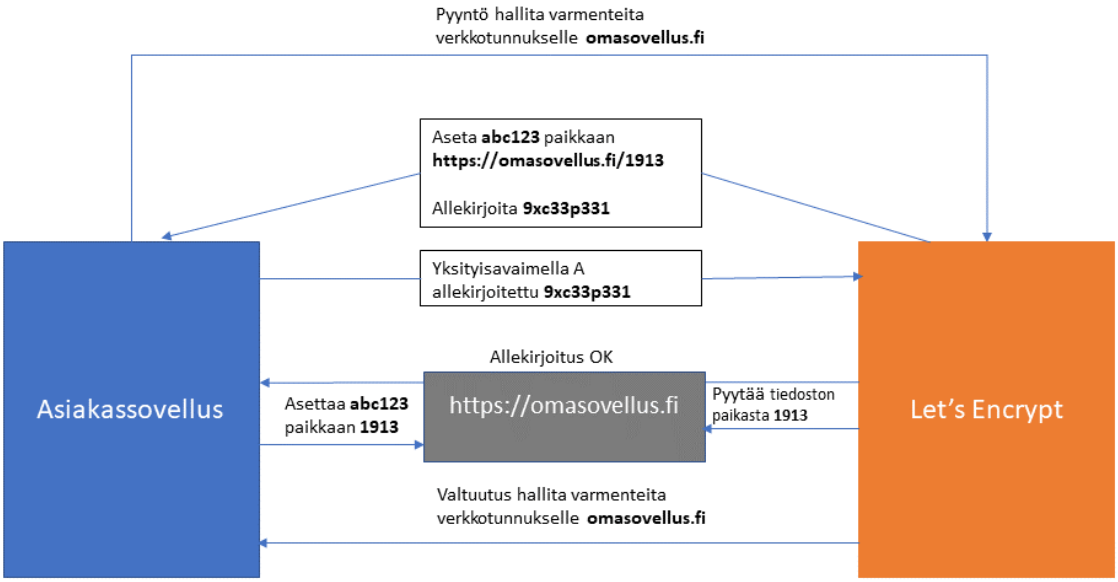
Vaihtoehtoina Traefikille tutkittiin Nginxiä, Envoyta ja Caddya, jotka ovat kaikki moderneja proxyjä. Nginxissä ja Envoyssa ei kuitenkaan ole automaattista varmenteiden luontia valmiina. Nginxin kanssa pitää käyttää esimerkiksi certbottia ja Envoy vaatii, että käytössä on ulkopuolinen Secret Discovery Service, joka hallitsee varmenteita [62, 63]. Caddyssä on automaattinen varmenteiden hallinta, mutta se ei integroidu Kuberneteksen kanssa samalla tavalla kuin Traefik. Se ei pysty esimeriksi automaattisesti tunnistamaan klusterissa olevia *kapseleita* ja reitittämään niille liikennettä. Kirjoitushetkellä Caddystä on vasta kehitteillä *sisääntulo-ohjain* implementaatio Kubernetekseen [64].

Traefikin konfigurointi jaetaan kahteen eri osaan: staattiseen ja dynaamiseen. Dynaaminen konfigurointi päivittyy ajonaikana ilman käyttökatkoja järjestelmässä, kun taas vastaavasti staattisen konfiguraation päivittäminen vaatii Traefikin uudelleenkäynnistämisen. Traefikin kehityksessä on pyritty siihen, että kaikki sovelluksiin ajonaikana tapahtuvat muutokset olisivat dynaamisen konfiguraation piirissä ja staattista konfiguraatiota tarvitsisi päivittää mahdollisimman harvoin. Traefikin termein staattisen konfiguraation piiriin kuuluvat EntryPoint ja Provider-oliot ja dynaamisen konfiguraation piiriin kuuluvat Router-, Middleware- ja Service-oliot.[32] [65, Section: Concepts, Configuration Introduction]

Staattinen konfiguraatio voidaan määritellä ympäristömuuttujilla, komentoriviparametreillä tai konfiguraatiotiedostoilla. Konfiguraatiotiedostot voidaan kirjoittaa joko YAML- tai TOML-syntaksilla. Tässä työssä esimerkit annetaan YAML-muodossa, koska se on myös Kuberneteksen oliomanifestien syntaksi. Dynaaminen konfiguraatio määritellään Providerien kautta. [65, Section: Concepts, Configuration Introduction] [32]

EntryPointeilla määritellään mistä Traefik vastaanottaa pyyntöjä ja miten pyyntöjä käsitellään matallalla tasolla. EntryPointin address-kenttään määritetään portti, jota Traefik kuuntelee ja tarvittaessa EntryPointtia voi tarkentaa IP-osoitteettella ja protokollalla. Oletuksena käytetty protokolla on TCP. Liikennettä voidaan ohjata ja muokata sisääntulossa sallimalla tai estämällä X-forwarded-for otsikkotietodot, asettamalla aikakatkaisuja ja käytetty proxyprotokolla (proxy protocol). Korkeammalla tasolla toimivalle HTTP-liikenteelle voidaan asettaa enemmän sääntöjä, kuten uudelleenohjaus toiseen EntryPointtiin, Middlewaret ja TLS-konfiguraatio. Middlewaret määritellään dynaamisessa konfiguraatiossa, mutta konfiguroimalla ne EntryPointtiin, ne ovat oletuksena käytössä EntryPointtia käyttävissä Routereissa. TLS-konfiguraatio toimii samalla periaatteella kuin Middleware. Routerit, jotka käyttävät sisääntuloa ja joille on määritelty TLS-konfiguraatio, eivät käsittele salaamatonta liikennettä. [32] [65, Section: EntryPoints]

Traefik tukee automaattista TLS-varmenteiden luomista ja käyttöönottoa mistä tahansa palvelusta, joka tukee Automatic Certificate Management Environment (ACME) -standardia. ACME-standardi määrittelee protokollan, jossa asiakassovellus voi hankkia itselleen varmenteen JSON-viesteillä Hypertext Transfer Protocol Secure (HTTPS) -yhteydellä. Omistajuus osoitetaan täyttämällä jonkin ACME-standardissa määritellyn haasteen (challen-

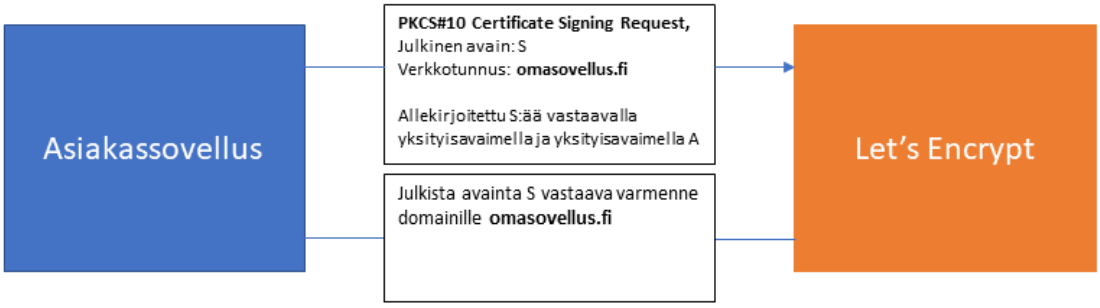


Kuva 4.5. *Traefik pyytää valtuutusta hallita varmenteita verkkotunnukselle omasovellus.fi. Saadakseen valtuutuksen Traefik laittaa tiedoston ennalta sovittuun paikkaan verkkotunnuksen alle ja allekirjoittaa yksityisavaimella A Let's Encryptin antaman satunnaisen luvun. Yksityisavain A on luotu Traefikille, kun se on ottanut ensimmäisen kerran yhteyttä Let's Encryptiin. Kun valtuutus on valmis, Traefik voi hakea varmenetta verkkotunnukselle. [68, Section: How It Works]*

ges) ehdot. Kohdeorganisaatiossa päädyttiin HTTP-01 haasteeseen, joka on haasteista yleisin ja yksinkertaisin. ACME-palveluntarjoajana päädyttiin käyttämään Let's Encryptiä. [32, 66] [65, Section: Let's Encrypt]

Asiakassovellus luo itselleen julkisen ja yksityisen avainparin kun se on ensimmäistä kertaa yhteydessä Let's Encryptiin. Osana vahvistusprosessia asikassovellus lähettää yksityisellä avaimella allekirjoitetun viestin, joka sisältää Let's Encryptin antaman satunnaisen luvun (nonce). Let's Encryptin HTTP-01 toteutus antaa asiakassovellukselle kertakäyttöisen tunnisteen (token), ja asiakassovellus laittaa ennalta sovittuun polkuun tiedoston. Let's Encryptin tapauksessa polku on muotoa https://<verkkotunnus, jolle haetaan varmennetta>/well-known/acme-challenge/<tunniste>. Tiedosto sisältää tunnisteen ja asiakassovelluksen julkisen avaimen tiivisteen (key thumbprint). Kun tiedosto on paikoillaan, asiakassovellus ilmoittaa tästä Let's Encryptille ja Let's Encrypt yrittää hakea tiedoston ennalta sovitusta polusta. Jos tiedoston haku onnistuu ja Let's Encrypt hyväksyy yksityisellä avaimella kirjoitetun viestin, Let's Encrypt antaa asiakassovellukselle oikeuden hallinnoida varmenteita kyseiselle verkkotunnukselle. HTTP-01-haaste tehdään aina portin 80 kautta, ja liikenteen voi uudelleenohjata ainoastaan porttiin 443. Varmenteiden hallintaoikeuden haku on havainnollistettu kuvassa 4.5. [67, Section: Challenge Types, How It Works]

Varmenteen hakeminen tapahtuu erityisellä PKCS#10 Certificate Signing Request -pyynnöllä, joka sisältää julkisen avaimen, jota vastaan Let's Encrypt luo varmenteen, ja julkisen avaimen yksityisellä parilla tehdyn allekirjoituksen. Pyyntö allekirjoitetaan lisäksi asiakasso-



Kuva 4.6. *Traefik lähettää PKCS#10 -viestin, joka sisältää julkisen avaimen S, verkkotunnusta omasovellus.fi varten. Viesti on allekirjoitettu yksityisavaimella A ja S:ää vastaavalla yksityisavaimella. Jos Let's Encrypt hyväksyy viestin, se palauttaa sen halutulle verkkotunnukselle. [68, Section: How It Works]*

velluksen yksityisellä avaimella, joka on luotu, kun asiakassovellus on ollut ensimmäisen kerran yhteydessä Let's Encryptiin. Varmenteen hakeminen on visualisoitu kuvassa 4.6. [68, Section: How It Works]

ACME-palveluntarjoajan käyttöönottaminen varmenteiden automaattiseen generointiin tapahtuu määrittelemällä Traefikin staattiseen konfiguraatioon certificatesResolvers-kentän alle tarvittavat tiedot. Ohjelmassa 4.1 on esitetty miten pienellä konfiguraatiolla käyttöönotto onnistuu. Rivillä 2 oleva myResolver on nimi, johon viitataan Routereissa, kun halutaan käyttää varmenteiden automaattista generointia. Rivillä 4 annettu sähköpostiosoite on tieto, mitä käytetään asiakassovelluksen rekisteröintiin ACME-palveluntarjoajalle. Rivillä 5 annettu polku määrittää paikan, johon Traefik tallentaa generoitujen varmenteiden tiedot. Jos tiedostoa ei ole määritelty tai Traefikilla ei ole siihen oikeuksia, niin varmenteet luodaan jokaisella käynnistyskerralla uudelleen, kunnes Let's Encryptin verkkotunnuskohtaiset rajat tulevat vastaan. Rivillä 7 annetaan sen EntryPointin nimi, jota käytetään HTTP-01-haasteessa. Let's Encrypt on oletus palveluntarjoaja, muut palveluntarjoajat, esimerkiksi Let's Encryptin testivarmenteiden luonti, konfiguroidaan caServer-kentällä. [67, Section: Challenge Types, Rate Limits] [32][65, Section: Let's Encrypt]

```
1 certificatesResolvers:
2   myresolver:
3     acme:
4       email: esimerkki@esimerkki.fi
5       storage: "/polku/sailoon/acme.json"
6       httpChallenge:
7         entryPoint: web
```

Ohjelma 4.1. *Traefik-konfiguraatio, jolla otetaan käyttöön Let's Encrypt ACME-palveluntarjoajana.*

Ajon aikana päivittyvä dynaaminen konfiguraatio luodaan Providereilta haetuilla tiedoilla. Traefik luo Providerilta saamillaan tiedoilla Routerit, Middlewaret ja Servicet. Providereita voi olla esimerkiksi konttiteknologia (container engine), Kubernetes tai konfiguraatiotiedosto. [65, Section: Providers Overview]

Tämän työn kannalta oleellisin Provider on KubernetesCRD (Custom Resource Definition), jonka käyttäminen vaatii Kuberneteksen laajentamista Traefikin *omaolioilla*. *Omaoliot* vastaavat dynaamisen konfiguraation piirissä olevia Traefik-olioita. Tässä luvussa esitellyt oliot ja niitä vastaavat *omaoliot* on kerätty taulukkoon 4.1. [32][65, Section: Traefik & Kubernetes]

Trafik-käsite	Vastaava *omaolio*
Router	IngressRoute, IngressRouteUDP, IngressRouteTCP
Service	TraefikService
Middleware	Middleware

Taulukko 4.1. *Traefik-käsitteet ja niitä vastaavat omaoliot*

Kubernetesta voi käyttää Providerina myös ilman *omaolioita*. Tällöin Providerin tyyppi on kubernetesIngress. Jos Traefik asennetaan suoraan Kubernetes-klusteriin, se voi toimia klusterin *sisääntulo-ohjaimena*. Routerit voidaan konfiguroida Kuberneteksen *sisääntuloilla* ja Servicet *palveluilla*, sen sijaan Middlewaret vaativat *omaolioiden* käyttöä. Traefikin tarjoamien ominaisuuksien käyttöönotto Kuberneteksen natiiviresursseissa on kuitenkin vaivalloista, sillä ne vaativat runsasta annotaatioiden käyttöä verrattuna *omaolioihin*, joissa Traefikin tarjoamille ominaisuuksille on omat kentät. [65, Section: Traefik & Kubernetes]

Routerit ohjaavat liikennettä Serviceille ja käyttävät tarvittaessa Middlewareja pyyntöjen muokkaamiseen. Routerit kuuntelevat niihin määritellyn EntryPointin kautta kulkevaa liikennettä ja ohjaavat sitä eteenpäin, mikä vastaa Routeriin asetettuja reitityssääntöjä. Käytettävissä olevat reitityssäännöt riippuvat Routerin tyypistä. UDP-liikennettä ohjaavalle Routerille ei voida asettaa lainkaan sääntöjä, joten se ohjaa kaiken liikenteen eteenpäin, mikä tulee sen kuuntelemaan EntryPointtiin. TCP-liikennettä ohjaavalle Routerille voidaan määritellä Host Server Name Indication (HostSNI) -sääntöjä, jotka ohjaavat liikenteen eteenpäin verkkotunnuksen perusteella. HostSNI-sääntöjä voidaan määritellä ainoastaan TLS-liikennettä käsitteleville Routereille, sillä HostSNI on jatke TLS-protokollaan, ei TCP:hen. UDP- ja TCP-liikennettä ohjaavien Routereiden rajoittuneisuus johtuu siitä, että ne toimivat OSI-mallin [55] reititystasolla, jossa liikenteen monipuoliseen ohjaamiseen tarvittavia tietoja ei ole olemassa. Sovellustasolla toimivat HTTP-liikennettä ohjaavat Routerit mahdollistavat huomattavasti monipuolisempien reitityssääntöjen määrittämisen. Sääntöjä voidaan määritellä otsikkotietojen, HTTP-metodin, pyynnön polun ja parametrien perusteella (Query Parameters). Otsikkotietoihin ja polkuun kohdistuvat säännöt voidaan määritellä myös säännöllisinä lausekkeina. Traefik käyttää säännöllisissä lausekkeissa implementointikielensä GOn syntaksia. [32] [65, Section: Routers]

Middlewareilla voidaan vaikuttaa Routereilta lähteviin pyyntöihin ennen kuin ne menevät Serviceille, tai ennen kuin Serviceiltä tuleva vastaus lähetetään asiakassovellukselle. Middlewareilla voidaan muokata, uudelleen reitittää, autentikoida pyyntöjä ja muokata vastauksia. Middlewareja voidaan ketjuttaa halutun vaikutuksen saamiseksi Chain-tyyppisellä Middlewarella. Ohjelmassa 4.2 reititetään example.com ja www.example.com

URL:eihin tulevaa liikennettä. Ennen kuin liikenne ohjataan Serviceille, siihen tehdään uudelleenohjaus, joka varmistaa, että URL alkaa www-etuliittellä ja ensimmäinen osa polkua verkkotunnuksen jälkeen on en_US. Rivien 3 ja 10 avaimet ovat reittimen ja Middlewaren nimet, jotka voivat olla mielivaltaisia. [32] [65, Section: Middlewares]

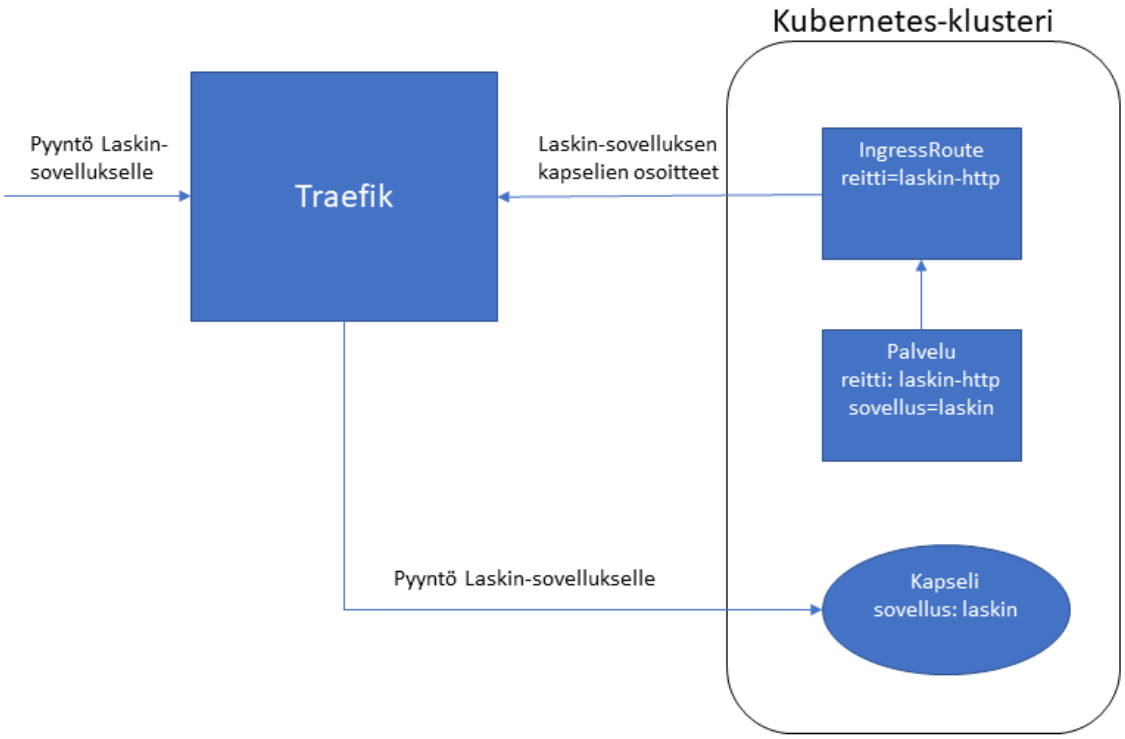
```
1  http:
2    routers:
3      router1:
4        service: myService
5        middlewares:
6          - "www-redirectregex"
7        rule: "Host(`example.com`) || Host(`www.example.com`)"
8
9    middlewares:
10     www-redirectregex:
11       redirectRegex:
12         regex: "^https://(www\\.)?example\\.com/?$"
13         replacement: "https://www.example.com/en_US"
```

Ohjelma 4.2. *Middlewaren ja Router konfiguraatio, joka uudelleenohjaa liikennettä säännöllisen lausekkeen perusteella.*

Traefikin Serviceillä on hyvin samanlainen tehtävä, kuin Kuberneteksen *palveluilla*. Niiden tehtävä on ylläpitää tietoa siitä, missä sovellusten konkreettiset ilmentymät ovat. Ilmentymät voidaan konfiguroida Serviceihin kiinteillä IP-osoitteilla, mutta pilviympäristöissä, joissa ilmentymisen IP-osoitteet elävät jatkuvasti, on järkevämpi käyttää prosessin automatisoivaa Provideria. Traefik hakee jatkuvasti Providerin rajapinnan kautta tietoja siitä missä osoitteissa ilmentymät ovat. Jos Kubernetes toimii Providerina Traefikille, Traefik hyödyntää Kuberneteksen *palveluita* selvittääkseen *kapseleiden* IP-osoitteet klusterin sisällä. Kuva 4.7 havainnollistaa reititysprosessin. Kun Traefik saa *kapseleiden* IP-osoitteet Kuberneteksen *palveluilta*, se lisää osoitteet omiin *palveluihinsa*. Traefik reitittää liikenteen suoraan *kapseleille*, eikä tällöin käytä Kuberneteksen iptables-sääntöihin perustuvia kernel-tason uudelleenohjauksia. [65, Section: Provider Overview, Services][7, 32]

Traefikissa taakanjako tapahtuu palvelutasolla. Serviceihin voidaan konfiguroida, mitä taakanjakoa käytetään. Kirjoitushetkellä Traefik ei tarjoa kuin kiertovuorottelun (round robin) ja sen painotetun variantin (weighted round robin). Painotettu kiertovuorottelu vaatii, että palvelimet jaetaan eri Serviceille, ja että painot annetaan näille palveluille palvelimien sijaan. Painot ovat siis staattisia ja eivät elä sovellusilmentymien reaaliaikaisen kuormituksen mukaan. Traefikiin voidaan konfiguroida myös terveystarkastuksia (Health Check), jotka ylläpitävät tietoa siitä, mitkä sovellusilmentymät ovat valmiita vastaanottamaan liikennettä. Niiden toimintaperiaatteet ovat samat kuin aliluvussa 4.1.1 kuvatuilla Kuberneteksen luotaimilla (probe). Traefikin terveystarkastukset eivät pysty vaikuttamaan ilmentymien tilaan esimerkiksi uudelleenkäynnistämällä niitä. [65, Section: Services] [32]

Traefik tukee myös Sticky-tyyppisiä sessioita, joiden avulla voidaan varmistaa, että sama sovelluksen ilmentymä vastaa istunnon ajan kaikkiin asiakassovelluksen pyyntöihin. Näin



Kuva 4.7. *Traefik saa IngressRoute-olioilta kapselien IP-osoitteet klusterin sisällä ja ohjaa pyynnöt suoraan niille. IngressRoute hakee IP-osoitteet palvelulta, joka valitaan samalla valitsin-tunniste-parilla kuin palvelujen ja kapselien tapauksessa.*

varmistetaan, että ne sovellukset, jotka säilyttävät tilallista tietoa istuntotasolla omassa muistissaan, toimivat oikein. Traefik ohjaa pyynnöt samalle ilmentymälle evästeen perusteella, joten Sticky sessiot toimivat vain HTTP-liikennettä ohjaavilla Routereilla. Jos ilmentymä, johon eväste osoittaa on tippunut pois, Traefik ohjaa pyynnön uudelle ilmentymälle ja päivittää evästeen. Tämä edellyttää, että terveystarkastukset on konfiguroitu Serviceille. [32]

4.2 Varmuuskopionti

Kohdeorganisaation varmuuskopiointijärjestelmän toimitti sama palveluntarjoaja, joka vastaa kohdeorganisaation palvelinympäristön ylläpidosta. Näin toimittiin aikataulullisista syistä ja sen vuoksi, että varmuuskopiointi on kriittinen osa tuotantoympäristöä. Kohdeorganisaatiossa katsottiin, että palveluntoimittajalla on paremmat edellytykset toimittaa suoraan Karbon ympäristöön sopiva varmuuskopiointiratkaisu. Tässä luvussa kuvataan varmuuskopiointijärjestelmän teknologiat ja miten niistä on rakennettu kohdeorganisaation varmuuskopiointijärjestelmä.

Varmuuskopiointijärjestelmä koostuu kolmesta eri komponentista: Velero, Restic ja MinIO. Velero on pilvinatiivi suoraan klusteriin asennettava Kubernetes-olioiden ja *pysyväislevyjen* varmuuskopiointiteknologia [69, Section: About Velero]. Restic mahdollistaa muidenkin kuin suurten pilvipalvelujen *pysyväislevyjen* varmuuskopioinnin [70, Section:

Introduction]. MinIO on objektivarasto (Object Storage), joka säilöö varsinaiset varmuuskopiot [71, Section: Introduction].

4.2.1 Tiedostojen pysyvä säilytys Kuberneteksessa

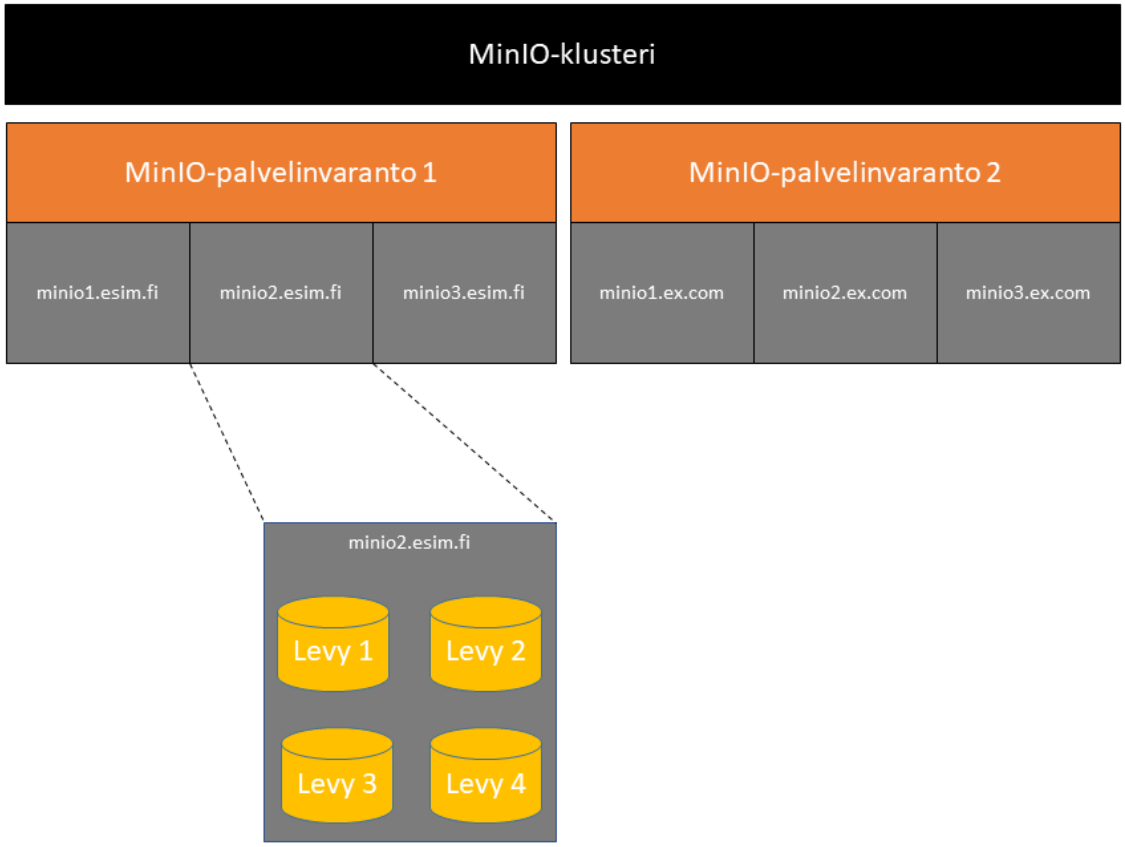
Konteilla on omat eristetyt tiedostojärjestelmänsä, jotka ovat yhtä lyhytikäisiä kuin kontit. Niitä sovelluksia varten, jotka tarvitsevat kontin elinkaaren yli säilyviä tiedostoja, Kuberneteksessa voidaan käyttää *kapselitasolla* levyjä (volume). Yksittäisessä *kapselissa* voi olla useita eri levyjä ja yhteen konttiin voi olla liitettynä useampia eri levyjä. Levy voi olla myös liitettynä useampaan konttiin *kapselin* sisällä, jolloin kontit jakavat tuon tiedostojärjestelmän osan. Kirjoitus- ja lukuoikudet voi konfiguroida konttikohtaisesti. [7] [24, Section: Volumes]

Kapseliin kuuluva levy voi olla myös kytketty *kapselin* ulkopuoliseen pysyvään tiedostojärjestelmään, esimerkiksi verkkolevyyn. Näin tiedostot saadaan säilymään *kapselien* poiston jälkeenkin ja ne ovat käytettävissä vanhan *kapselin* korvaavissa *kapseleissa*. *Kapselin* ulkopuolinen säilytys mahdollistaa tiedostojen jakamisen myös useamman *kapselin* välillä. Kuberneteksessa on tuki suoraan Amazonin, Googlen ja Microsoftin pilvialustojen käyttämille säilytysteknologoille. Kubernetes tukee myös useita muita verkkolevyteknologioita kuten cephfs ja cinder. Levyt voidaan luoda erikseen valmiiksi ulkopuolisiin järjestelmiin tai luoda dynaamisesti *pysyväislevypyynnöillä* (PersistentVolumeClaim). Säilytysteknologiasta riippuen ulkopuolinen levy voi olla liitettynä yhteen tai useampaan työskentelijäpalvelimeen kirjoitusoikeuksilla. Usein kuitenkin pilvialustojen tapauksessa kirjoitus on sallittu vain yhdelle palvelimelle. [7][24, Section: Volumes]

4.2.2 MinIO

MinIO on Kubernetes-natiivi S3 yhteensopiva avoimenlähdekoodin objektivarasto. S3-yhteensopivuus tarkoittaa sitä, että MinIO toteuttaa AWS:n S3-objektivaraston API:n ja osaa kommunikoida sen kanssa. MinIOsta on olemassa perinteisillä palvelimilla (bare metal) ja Kubernetekseen asennettava toteutus. Kubernetes-toteutus on Kubernetes-natiivi eli se toimii MinIOn *omaolioilla* ja *omaohjaimilla*. Tässä työssä keskitytään palvelinversioon, sillä se on kohdeorganisaatiossa käytössä. Palvelin- ja Kubernetes-versiot kuitenkin vastaavat ominaisuuksiltaan toisiaan. Niiden erot ovat ylläpidollisia. MinIO:oon kuuluu palvelin- (MinIO server) ja asiakassovellus (MinIO client). Palvelin jalkautetaan haluttuun ympäristöön ja se rakentaa varsinaisen objektivaraston. Asiakassovellus kommunikoi MinIO-palvelimen tai minkä tahansa muun S3-API:n toteuttavan objektivaraston kanssa. Asiakassovelluksella pystyy halutessaan tarkastelemaan myös tavallisessa tiedostojärjestelmässä olevien tiedostojen kanssa. MinIO-palvelin täyttää kohdeorganisaation vaatimuksen siltä osin, että varmuuskopiot säilytetään klusterin ulkopuolella. [71, Section: Introduction, MinIO Client, MinIO Server]

Yksittäisellä palvelin (node) koostuu MinIO-palvelinsovelluksesta ja siihen liitetyistä kiinto-

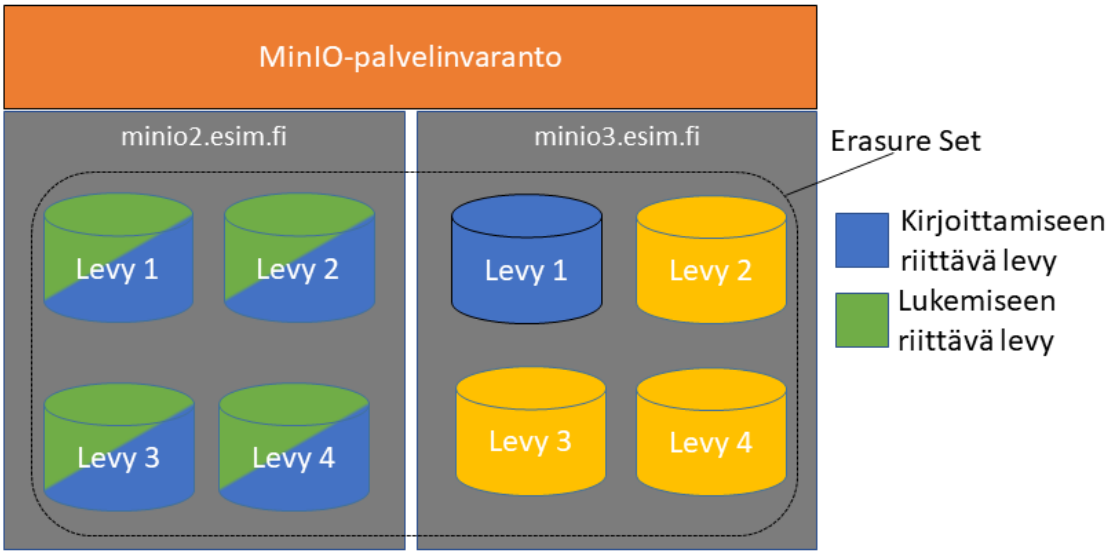


Kuva 4.8. *Varannossa jokaisella palvelimella on oltava järjestysnumerollinen isäntänimi, jotta MinIO-palvelin pystyy käynnistyessään liittymään osaksi varantoa isäntänimensä perusteella. Klusteria voi skaalata horisontaalisesti liittämällä siihen uusia palvelinvarantoja.*

levyistä. MinIOn kanssa suositellaan käytettävän paikallisia levyjä eikä verkkolevyjä. Levyjä ei myöskään kannata osioida. Paikallisilla levyillä ja yhdellä osiolla pyritään varmistamaan suorituskyky ja toimintavarmuus. MinIO-klusteri koostuu yksittäisistä palvelimista (node), jotka muodostavat palvelinvarantoja (server pool). MinIO-klusteriin voi kuulua yksi tai useampia varantoja, tämä on havainnollistettu kuvassa 4.8. Varannon palvelimet muodostavat loogisen kokonaisuuden, joka yhdistää palvelinten tallennuskapasiteetin ja resurssit asiakassovellusten pyyntöjen käsittelyyn. Jokainen objekti, sen eri versiot ja palauttamiseen tarvittava pariteettidata tallennetaan vain yhteen varantoon klusterissa. [71, Section: Installation, MinIO server]

MinIOn voi ottaa käyttöön yksittäisenä palvelimena. Tuotantokäyttöön suositellaan kuitenkin vain hajautettua klusteria luotettavuuden takaamiseksi. Tuotantoklusterin suositellaan myös koostuvan vähintään neljästä erillisestä palvelimesta (yksi palvelinvaranto), joilla on kullakin vähintään neljä kiintolevyä. Näin pyritään takaamaan korkea saavutettavuus. [71, Section: installation]

MinIO jakaa palvelinvarannon kiintolevyt Erasure Set -joukoiksi. Joukko kostuu 4–16 kiintolevystä. MinIO pyrkii jakamaan levyt mahdollisimman suuriin joukkoihin, jotta joukkojen määrä pysyisi mahdollisimman pienenä. Suuret Erasure Set -joukot mahdollistavat sen, että yksittäisestä joukosta voi olla suurempi määrä palvelimia tai levyjä poissa käytöstä



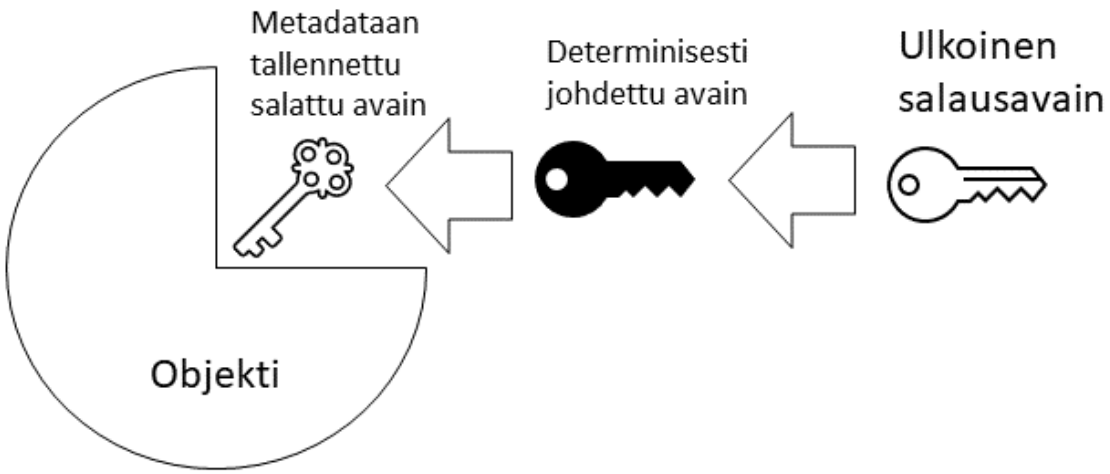
Kuva 4.9. *Erasure Set -joukko koostuu 8 levystä ja pariteetti on korkein mahdollinen EC:4, niin 4 kiintolevyä riittää lukemiseen ja 5 kirjoittamiseen.*

ilman datan menetystä tai toiminnan häiriintymistä. MinIO määrittää joukon levyjen määrän, palvelinvarannon palvelinten määrän ja varannon levyjen määrän suurimman yhteisen tekijän mukaan. Jos palvelimia on parillinen määrä MinIO käyttää suoraan suurinta yhteistä tekijää joukon levyjen määränä. Jos palvelimia on pariton määrä MinIO valitsee mahdollisimman suuren levyjen määrän, joka johtaa parittomaan määrään joukkoja. Pariton määrä joukkoja mahdollistaa, että jokaisella palvelimella on mahdollisimman tasainen määrä levyjä jokaista Erasure Set -joukkoa kohden. [71, Section: Erasure Coding]

MiniIOn datan pilkkomistekniikka on nimeltään Erasure Coding (EC). EC jakaa yksittäisen objektin datalohkoihin (block) ja pariteettilohkoihin, joka jaetaan saman Erasure Set -joukon kiintolevyille. Data jaetaan järjestelmällisesti kiintolevyjen välille niin, että yksittäisellä levyllä on vain yksi lohko samasta objektista. Myös saman objektin eri versiot tallennetaan siten, että samat levyt toimivat sekä datan että pariteettidatan säilytyspaikkoina. Pariteettilohkojen määrä määrittää suoraan sen, kuinka monta levyä voi olla samanaikaisesti poissa käytöstä ilman käytön häiriintymistä. Pariteettilohkoja käytetään puuttuvan datan paikkaamisen ja eheyttämään korruptoituneita datalohkoja.[71, Section: Erasure Coding]

MinIOn dokumentaatiossa käytetään EC:n N-notaatiota, missä N on pariteettilohkohjen määrä Erasure Set -joukossa. Lukutoiminnallisuus vaatii, että kiintolevyistä on toiminnassa [Kiintoleyvjen kokonaismäärä] - EC:N kappaletta levyjä. Kirjoitustoiminnallisuus vaatii, että toiminnassa on [Kiintolevyjen kokonaismäärä] - EC:N + 1 kappaletta kiintolevyjä. Kuvassa 4.9 on havainnollistettu levyjen jakautuminen 8 levystä koostuvalle Erasure Set -joukolle. Pariteetilohkojen määrän voi valita myös objektikohtaisesti Storage Class -attribuutilla. Storage Classien parieettiarvot voi asettaa ympäristömuuttujilla tai MinIOn asikassovelluksen admin-komennoilla. [71, Section: Erasure Coding]

Käyttäjä tunnistautuu MinIOon tunnuksella ja salasanalla. Tunnukseen on liitetty tiettyjä



Kuva 4.10. *Objekti salataan objektikohtaisella avaimella, joka tallennetaan salattuna osaksi objektin metadataa. Objektikohtaisen avaimen käyttöön tarvitaan erillinen ulkoisesta salausavaimesta deterministisesti luotu salausavain. Ulkoinen salausavain voi olla joko asiakassovelluksen toimittama tai salausavainhallintajärjestelmästä tuleva avain (Key Management System).*

valtuutuksia (policy), jotka oikeuttevat toimintoihin järjestelmässä yleisesti tai tiettyihin resursseihin. Valtuudet määritellään joko käyttäjäkohtaisesti tai ryhmätasolla. Valtuutukset noudattavat AWS Identity and Access Management määrittelyjä. MinIO toteuttaa osan Identity and Access Management -valtuutuskomennoista, jos S3:a varten kehitetty valtuutus käyttää MinIOn toteuttamaa osaa valtuutuskomennoista se toimii MinIOssa sellaisenaan. Oletuksena kaikki toiminnot ovat kiellettyjä elleivät käyttäjän valtuutukset eksplisiittisesti salli niitä. MinIO tallentaa kaikki objektit salattuina levylle. Palvelimella tehtävä salaus tapahtuu kolmessa tasossa, mikä on kuvattu kuvassa 4.10. Objektit myös siirretään salattuina verkon yli TLS:n avulla. [71, Section: Policies, Server-Side Encryption][72, Section: MinIO Security Overview]

MinIOssa objektit voi tarvittaessa versioida. Versioinnin ollessa käytössä MinIO ei ylikirjoita objektia, joka on jo varastossa vaan lisää koriin uuden version objektista. MinIO ei deduplikoi objekteja vaan kaikki versiot tallennetaan kokonaisina. Uusimpaan versioon liitetään latest-tunniste, ja jos asiakassovellus ei spesifioi haettavan objektin versiota, niin MinIO palauttaa uusimman version. Versioinnin ollessa käytössä poisto-operaatio poistaa uusimman version objektista ja merkkaa objektin poistetuksi. Objektin viimeisintä versiota hakevat käyttäjät saavat tiedon, että objekti on poistettu. Objektin vanhemmat versiot ovat yhä tallessa ja noudettavissa versiotiedolla. Vanhan version voi myös merkata uusimmaksi, jolloin MinIO lisää sille latest-tunnisteen. Objekti ja kaikki sen versiot on mahdollista poistaa antamalla poistokomennolle versions-parametri. [71, Section: Bucket Versioning]

Versioinnin käyttö vaatii, että MinIOlla on käytössään vähintään 4 kiintolevyä, sillä versiointi on riippuvainen EC:stä. Versiointi otetaan käyttöön korikohtaisesti **mc version enable <versioitava kori>** -komennolla. Korissa valmiiksi oleville objekteille ei generoida versiotunnisteita. Versioinnin voi myös tarvittaessa keskeyttää, jolloin versioimattomat

objektit voidaan ylikirjoittaa, mutta MinIO korissa jo olevat versioidut objektit säilyttävät vanhat versiot. Versiotunnisteet ovat 128-bittisiä Universally Unique IDentifier (UUID) -tunnisteita. [71, Section: Bucket Versioning]

Objektien säilytystä voidaan hallita ottamalla käyttöön objektien lukitus (Object Locking). Lukitus otetaan käyttöön korissa ja se asettaa kaikille korissa oleville objekteille säilytystilan (Retention Mode) ja säilytysajan (Retention Length). Lukituksen käyttö edellyttää, että kori on versioitu. Säilytystilat ovat AWS:n S3:n ominaisuus MinIO hyödyntää samoja tiloja. Korin oletuslukon voi ylikirjoittaa asettamalla objektille sen luontihetkellä säilytystilan ja säilytysajan. Säilytystilan mahdolliset arvot ovat Governance-tila ja Compliance-tila. Governance-tilassa objektiin ei voi tehdä muutoksia ilman erillisiä oikeuksia. Compliance-tilassa objektiin ei voi tehdä lainkaan muutoksia. Säilytysaika määrittää kuinka pitkään objekti on lukittu. Koritasolla säilytysaika määritellään siten, että se on N päivää tai vuotta objektin luontihetkestä. Kun oletuslukko ylikirjoitetaan objektia luotaessa, lukon määritellään kestävän tiettyyn päivämäärään asti. Säilytystila ja säilytysaika voivat olla eri objektin eri versioilla. Lukitusta käyttävässä korissa olevan objektin tietty versio voidaan asettaa pitoon (legal hold). Pito estää objektiversion poistamisen tai ylikirjoittamisen ja se on toistaiseksi voimassa. Objekteille voidaan asettaa myös poistopäivämäärä, jolloin MinIO poistaa objektin automaattisesti. Poistopäivämäärää sovelletaan vasta kun objektin säilytystila, säilytysaika ja pito sallivat sen. [72, Section: Object Lock and Immutablity Guide, Bucket Lifecycle Configuration Quickstart Guide][13, Section: How S3 Object Lock works]

MinIO tukee objektien replikointia korien välillä. Replikointi voi olla joko yhdensuuntaista tai kahdensuuntaista. Yhdensuuntaisessa replikoinnissa aktiivinen kori kopioidaan passiiviseen koriin, mutta passiivisen korin muutokset eivät heijastu aktiiviseen koriin. Kahdensuuntaisessa replikoinnissa molemmat korit ovat aktiivisia ja muutokset toisessa heijastuvat aina toiseen. Myös metadatan muutokset kuten objektilukot replikoidaan. Replikointi voi tapahtua saman MinIO-klusterin sisällä tai kahden eri klusterin välillä, mutta molempien korien täytyy olla klusterissa. MinIO tarkkailee replikoitavan korin objektien tilaa ja asettaa X-Amz-Replication-Status-kenttään tiedon siitä, missä vaiheessa replikointisykliä objekti on. Replikointi tapahtuu oletuksena asynkronisesti, jolloin objektit kirjoitetaan loppuun asti ensimmäiseen koriin ja lisätään sen jälkeen replikointijonoon. Synkronisessa replikoinnissa kirjoitusoperaatio ei pääty ennen kuin MinIO on yrittänyt replikoida objektin. Synkroninen replikointi ei siis takaa, että objekti päätyy kohdekoriin ennen kuin kirjoitusoperaatio katsotaan onnistuneeksi. Replikoinnin voi tehdä myös asiakassovelluksen avulla, jolloin asiakassovellus huolehtii objektien replikoinnista kahden eri korin välillä. [71, Section: Bucket Replication]

4.2.3 Restic

Restic on Go-ohjelmointikielellä toteutettu avoimen lähdekoodin varmuuskopiointityökalu. Restic pystyy varmuuskopioimaan, tiedostoja ja hakemistorakenteita Linux, BSD, Mac ja Windows ympäristöistä ja tukee useaa erityyppistä säilöä varmuuskopioille. Restic salaa

ja deduplikoi kaikki käsittelelmänsä tiedostot. [70]

Resticissä varmuuskopiot tallennetaan säilöön (repository), joka voi sijaita paikallisesti laitteella, josta otetaan varmuuskopioita tai toisella ulkopuolisella palvelimella. Säilö luodaan komennolla **restic init –repo /polku/säilöön**. Osana säilön luontia käyttäjä määrittää salasanan, jolla säilön käyttöä hallitaan. Varmuuskopiot luodaan komennolla **restic –repo /polku/säilöön backup /polku/varmuuskopion/lähteeseen**. Varmuuskopion lähde voi olla hakemisto tai yksittäinen tiedosto. Varmuuskopiota voi rajata määrittelemällä poissuljettavat tiedostot tai tiedostonimijokerit (glob patterns). Vastaavasti varmuuskopion voi rajata määrittelemällä sisällytettävät tiedostot tai tiedostonimijokerit. Tiedostonimijokerit käyttävät Go-ohjelmointikielen syntaksia. Varmuuskopion voi myös luoda suoraan standardisyötteestä (stdin) putkittamalla syötteen Resticille ja antamalla backup-komennolle –stdin option. Varmuuskopiot palautetaan restore-komennolla. Palautuksen voi rajata koskemaan vain tiettyjä tiedostoja samaan tapaan kuin varmuuskopion. Palautettava varmuuskopio valitaan antamalla varmuuskopioon liittyvän tilannekuvan (snapshot) tunniste, joka on SHA-256 tiivisteen muodossa. Vaihtoehtoisesti voi palauttaa viimeisimmän varmuuskopion, jonka voi vielä rajata tiettyyn lähteeseen polun tai palvelimen (host) perusteella. [70, Section: Preparing a new repository, Backing up, Restoring from backup]

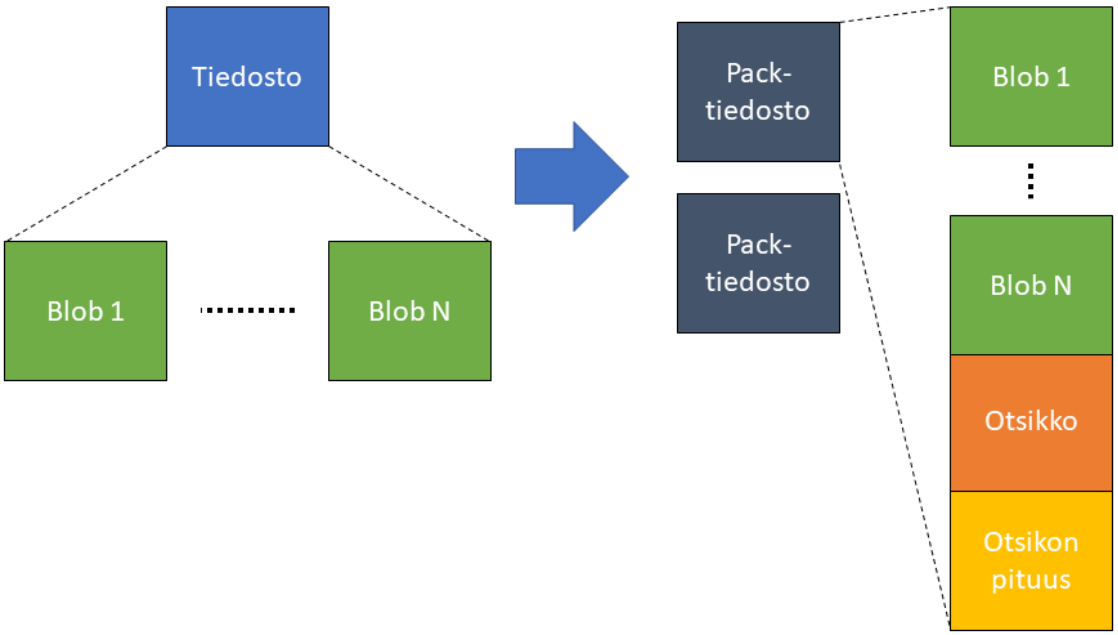
Erityyppiset säilöt käyttävät eri URL-skeemoja, jotka pitää lisätä säilön polkuun. Esimerkiksi SFTP-säilön URL on muotoa kayttaja@palvelin:/polku/säilöön. Paikallinen ja SFTP-yhteyden yli toimiva säilö hyödyntävät minkä tahansa palvelimen tavallista tiedostojärjestelmää ilman ylimääräisiä komponentteja. Resticillä on oma REST-palvelintoteutus, joka mahdollistaa tietojen siirron HTTP:n tai HTTPS:n yli. REST-palvelin toimii samanaikaisesti paikallisena säilönä sillä palvelimella, jolle se on asennettu. [70, Section: Preparing a new repository]

Restic tukee S3:n lisäksi Alibaba Cloud Object Storage System, OpenStack Swift, Backblaze B2, Microsoft Azure Blob Storage, Google Cloud Storage objektivarastoja. Kaikki konfiguroidaan S3:n tapaan vaihtamalla URL-skeema ja asettamalla oikeat ympäristömuuttujat. Lisäksi Restic tukee objektivarastojen käyttöön tarkoitettua Rclone-komentorivityökalua, joka mahdollistaa sen tukemien objektivarastojen hyödyntämisen. [70, Section: Preparing a new repository]

Jokainen säilö koostuu seuraavista säilön juuressa olevista hakemistoista:

- data
- index
- keys
- locks
- snapshots

Keys-hakemistoon on tallennettu säilön avaimet JSON-formaatissa. Avaintiedosto sisältää parametrit ja salatun datakentän, jotka yhdistämällä käyttäjän määrittämään salasa-



Kuva 4.11. *Pack-tiedostot koostuvat salatuista blobeista, salatusta otsikosta ja sen pituuden kertovasta kentästä. Blobeja voi olla Pack-tiedostossa n-kappaletta. Salattu otsikko sisältää tiedot Pack-tiedoston sisältämistä blobeista muodossa blobin tyyppi, salatun blobin pituus ja salaamattoman blobin tiiviste.*

naan säilön salaus voidaan purkaa. Säilöllä voi olla samanaikaisesti useita avaintiedostoja, joihin liittyy eri salasana. Salasanan voi vaihtaa ilman tiedostojen uudelleen salaamista. Restic käyttää 256 bittistä Advanced Encryption Standard (AES) salausta kaiken käsittelemänsä Datan salaamiseen ja Poly1305-AES todennuskoodia sisällön varmentamiseen. [73] [70, Section: References]

Restic kartoittaa varmuuskopioitavaksi valitun hakemiston ja poimii sieltä hakemistot ja tiedostot. Tiedostot jaetaan erimittaisiin blobeihin 64-bittisellä liukuvalla ikkunalla. Restic pyrkii siihen, että blobit ovat kooltaan välillä 512 KiB - 8 MiB ja keskimäärin 1 MiB. Blobit tallennetaan Pack-tiedostoihin, jotka sijaitsevat datahakemistossa. Pack-tiedostot on salattu samalla tavalla kuin muutkin Resticin käsittelemät tiedostot. Tiedostojen pilkkominen on havainnollistettu kuvassa 4.11. [70, Section: References]

Blobin tyyppi voi olla joko data tai puu (tree). Jokainen puu sisältää n-kappaletta solmuja, jotka voivat olla hakemistoja tai tiedostoja. Hakemistosolmut sisältävät subtree-kentän. Kentän arvo on tiiviste, joka osoittaa toiseen puu-blobiin, joka sisältää hakemiston tiedostot ja alihakemistot. Tiedostosolmut sisältävät content-kentän, jossa on lista tiivisteistä, jotka osoittavat niihin data-blobeihin, jotka muodostavat tiedoston. [70, Section: References]

Restic käyttää deduplikointia, eli se ei tallenna samaa tiedostoa kahdesti vaan tallennetaan pelkästään muuttuneet tiedostot uudelleen. Restic tunnistaa muuttuneet tiedostot käymällä niiden sisällön läpi uudelleen ja vertaamalla sitä säilössä olevaan tiedostoon. Läpikäynti on kuitenkin resurssi-intensiivinen toimenpide, joten Restic käyttää sääntöjä,

joilla se päättää mitkä tiedostot se käy läpi. Unix-järjestelmissä tiedosto käydään läpi, jos sen koko, i-node, muutosaikaleima (mtime) tai metadata-aikaleima (ctime) on muuttunut. Windowsilla Restic katsoo tiedoston mahdollisesti muuttuneen, jos sen koko, polku tai muutosaikaleima on muuttunut. Jos jokin hakemisto on uudelleen nimetty siitä polusta, missä tiedosto on vaikka itse tiedosto ei olisi , Restic katsoo tiedoston mahdollisesti muuttuneen ja käy sen läpi. [70, Section: Backing up]

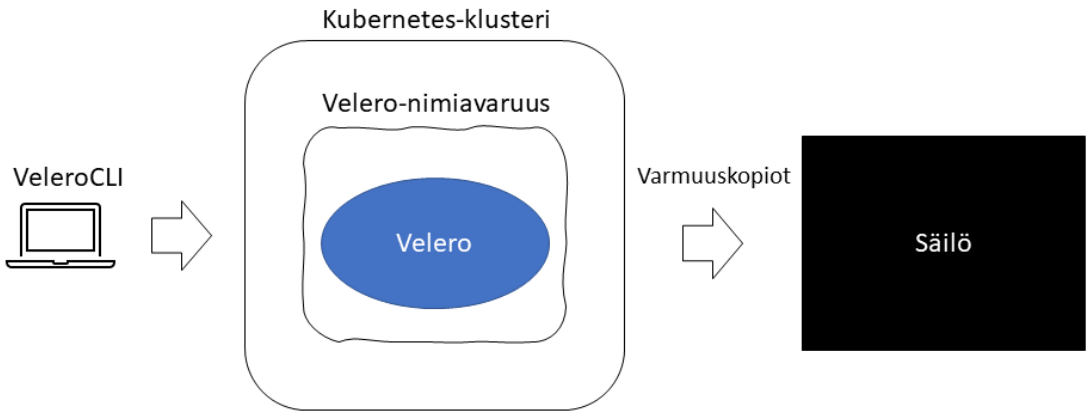
Index-hakemisto sisältää indeksitiedostot, joista käy ilmi missä pack-tiedostossa ja missä kohtaa pack-tiedostoa mikäkin blob ja puu sijaitsee. Restic tallentaa indeksin välimuistiin ja säilössä olevaa indeksiä käytetään vain tarvittaessa. Indeksitiedostot salataan samaan tapaan kuin muutkin Resticin käsittelemät tiedostot. [70, Section: References]

Jokaista varmuuskopiota kohden on olemassa tilannekuva, jota säilytetään snapshots-hakemistossa. Tilannekuva sisältää tiedot siitä, mistä polusta varmuuskopio on otettu, varmuuskopion tekohetken, puun tiivisteen ja varmuuskopion ottaneen käyttäjän tiedot. Puun tiiviste osoittaa varmuuskopion "juureen", josta loppu varmuuskopio rakentuu. Tilannekuville voidaan lisäksi antaa varmuuskopiointivaiheessa tai sen jälkeen tunnisteita (tags), joiden avulla varmuuskopioon voi kohdentaa toimenpiteitä kuten poiston. Tunnisteiden muuttuessa tilannekuva salataan ja tallennetaan uudelleen ja se saa uuden tiivisteen. Tilannekuvan metatiedoissa on kuitenkin viite alkuperäiseen tilannekuvaan. Jos säilöstä loppuu tila kesken varmuuskopioinnin, säilöön siirtyy uutta dataa, mutta tilannekuvaa ei luoda. Tilannekuva syntyy vasta onnistuneen varmuuskopioinnin lopussa. Tilannekuvan ja kaiken siihen liittyvän datan pystyy siirtämään copy-komennolla kahden eri säilön välillä [70, Section: References, Backing up]

Useampi Restic-instanssi voi kirjoittaa samaan säilöön samanaikaisesti. Jotkin Resticin toiminnot kuitenkin vaativat, että vain yhdellä instanssilla on käyttöoikeus säilöön, kunnes toiminto on suoritettu. Restic käyttää säilön käytönhallinnassa lukkoja, joita se säilyttää JSON-tiedostoina locks-hakemistossa. Lukkoja on kahta tyyppiä eksklusiivinen ja ei-eksklusiivinen. Ei-eksklusiivisia lukkoja voi olla useita voimassa samaan aikaan. Ei-eksklusiivinen lukko edellyttää, että se on ainut lukko, joka on säilössä voimassa. Jokainen Restic-prosessi luo lukon käyttäessään säilöä. Lukko sisältää tiedon siitä onko se eksklusiivinen, milloin lukko on luotu, lukon luoneen prosessin käyttäjänimen ja palvelimennimen. Luontivaiheessa Restic käy läpi säilössä olevat lukot ja poistaa vanhentuneet lukot. Yleisesti 30 minuuttia vanhemmat lukot katsotaan vanhentuneiksi. Restic katsoo, että myös uudemmat lukot, jotka eivät vastaa sen lähettämään signaaliin ja ovat samalta palvelimelta kuin uutta lukkoa pyytävä prosessi, ovat vanhentuneita. Jos säilössä ei ole vanhentuneiden lukkojen poiston jälkeen voimassa olevia uuden lukon kanssa konfliktissa olevia lukkoja, Restic luo lukon. [70, Section: References]

4.2.4 Velero

Velero on avoimen lähdekoodin varmuuskopiointityökalu, joka koostuu kahdesta komponentista, komentorivityökalusta ja Kubernetekseen jalkautettavasta palvelimesta. Velero



Kuva 4.12. *Veleroon ollaan yhteydessä VeleroCLI komentorivityökalulla. Velero asennetaan oletuksena Velero nimiseen nimiavaruuteen. Nimiavaruuden nimen voi vaihtaa tarvittaessa. Kohdeorganisaation tapauksessa varmuuskopioiden säilytyspaikkana on MinIO objektivarasto.*

on ohjelmoitu Go-ohjelmointikielellä ja hyödyntää Kuberneteksen virallista Go-kirjastoa (client library) kommunikaatiossa Kuberneteksen API-palvelimen kanssa. Velerolla käyttäjä pystyy varmuuskopioimaan ja palauttamaan kaikki Kuberneteksen natiiviresurssit, klusteriin määritellyt *omaoliot* ja *kapseleihin* liitetyt levyt. Velerolla käyttäjä pystyy automatisoimaan varmuuskopioinnin asettamalla kopioinnille aikataulun ja määrittämän kopioiden säilytysajat. Velerolla on myös mahdollista määrittää tietyt toimenpiteet, jotka suoritetaan aina ennen tai jälkeen varmuuskopioiden tekemisen. Veleron asennus on havainnollistettu kuvassa 4.12 [69, Section: About Velero, How Velero Works, Run in a non-default namespace][74]

Kun käyttäjä aloittaa varmuuskopioinnin tai ajastettu varmuuskopiointi alkaa, Kubernetesklusterissa oleva Veleron *BackupController*-ohjain tunnistaa klusterissa olevat resurssit Kuberneteksen API-palvelimen kautta. Yksinkertaistettuna Velero pyrkii varmuuskopioimaan etcd-tietokannan varmuuskopiolle relevantit osat ja pysyväislevyjen datan. Resurssit varmuuskopioidaan Kuberneteksen oliomanifesteihin JSON-formaatissa. Liitteessä B on esimerkki Veleron varmuuskopioimasta *palvelusta*. Ohjain kerää resurssit yhteen ja muodostaa niistä tar-tiedoston, joka pakataan gzipillä ja tallennetaan konfiguroituun objektivarastoon. Varmuuskopiointiprosessi ei ole täysin atominen. Jos klusterissa tapahtuu muutoksia varmuuskopioinnin aikana ne eivät välttämättä ole mukana varmuuskopiossa. Tämä on kuitenkin Veleron kehittäjien mukaan epätodennäköistä. [69, Section: How Velero Works, Output File Format][74]

Velero hyödyntää omia *omaolioitaan* ja *omaohjaimiaan* toiminnassaan. Jokaista varmuuskopiota vastaa klusterissa oleva Backup-olio. *Backup-olion* tiedoissa on, mitkä nimiavarauudet ja resurssit (Kubernetes-oliot ja levyt) kuuluvat varmuuskopioon. Varmuuskopion kohdistaminen tiettyihin nimiavaruuksiin täyttää kohdeorganisaation vaatimuksen siitä, että testi- ja tuotantoympäristöjä on voitava käsitellä erikseen. Veleron toimintaperiaatteena on ja oletuksena on, että kaikki resurssit ja nimiavaruudet varmuuskopioidaan. Käyttäjä

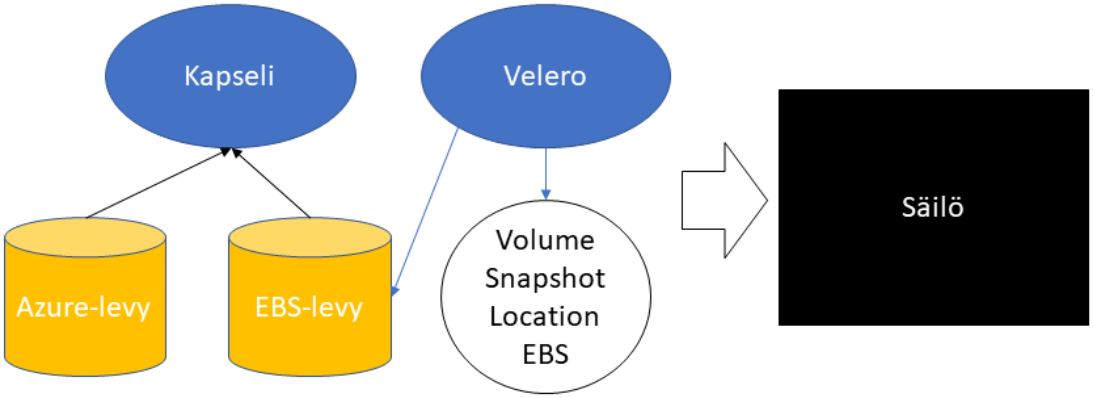
voi kuitenkin halutessaan valita varmuuskopioitavat nimiavaruudet ja resurssit erikseen. Käyttäjä voi myös valita sen, mitä resursseja ja nimiavaruuksia ei varmuuskopioida. Seuraava komento luo varmuuskopion ainoastaan omaymparisto ja testiymparisto nimiavaruuksista ja jättää varmuuskopiosta pois *palvelut* ja *päätepisteet*:

```
velero backup create testivarmuuskopio \
  --include-namespaces omaymparisto, testiymparisto \
  --exclude-resources services,endpoints
```

Klusteritason resurssit, kuten *klusteriroolit* (ClusterRole) varmuuskopioidaan oletuksena, jos kaikki nimiavaruudet varmuuskopioidaan, mutta jos nimiavaruuksia rajoitetaan klusteritason resursseja ei varmuuskopioida. Varmuuskopiointi voidaan myös kohdistaa Kubernetes-olioiden tunnisteisiin. Resurssi voidaan myös merkata sellaiseksi, että sitä ei koskaan varmuuskopioida asettamalla velero.io/exclude-from-backup -tunnisteen arvo todeksi. [69, Section: Resource filtering, Backup API Type, Backup Reference]

Objektivarasto on Veleron näkökulmasta lopullinen totuuden lähde. Velero tarkastaa säännöllisesti Kuberneteksen API:n kautta, että ovatko objektivaraston tilaa vastaavat Backup-oliot olemassa klusterissa. Jos oliot puuttuvat Velero luo ne automaattisesti klusteriin. Jos klusterissa on Backup-olio, jolle ei löydy vastaavaa varmuuskopiota objektivarastosta, Velero poistaa olion, jotta klusterin tila vastaa objektivaraston tilaa. [69, Section: How Velero Works]

Kubernetes-olioiden varmuuskopioiden sijoituspaikka määritetään BackupStorageLocation-oliolla. Klusterissa pitää olla vähintään yksi BackupStorageLocation-olio ja Velero odottaa, että olion nimi on default. Oletusnimen voi kuitenkin halutessaan vaihtaa. Olioon konfiguroidaan kori (bucket), jossa varmuuskopioita säilytetään. Varmuuskopioiden sijaintia korissa voi tarkentaa etuliitteellä, jolloin Velero toimii vain määritellyn polun alla. Eri säilöillä on vielä omat mahdolliset konfiguraatioarvonsa, jotka riippuvat käytetystä liitännäisestä. Esimerkiksi AWS:n S3 objektivarastoa käytettäessä voi määrittää maantieteellisen alueen (region), jolle varmuuskopiot sijoitetaan. Levyjen varmuuskopioiden sijainnit konfiguroidaan VolumeSnapshotLocation-oliolla, jonka kaikki attribuutit riippuvat käytetystä liitännäisestä ja levyteknologiasta. Molempia olioita voi olla käytössä useampia samanaikaisesti. Yksittäinen varmuuskopio voi hyödyntää yhtä BackupStorageLocation-oliota kerrallaan, mutta varmuuskopiot voidaan määrittää käyttämään eri sijainteja, jolloin esimerkiksi päivittäiset varmuuskopiot voidaan säilöä eri AWS-alueille. Jos käytettyä sijaintia ei ole määritelty, Velero käytää oletussijaintia. Varmuuskopion pitää käyttää jokaista varmuuskopioitavaa levytyyppiä kohden sitä vastaavaa VolumeSnapshotLocation-oliota, mikä on havainnollistettu kuvassa 4.13. Oletuksena Velero käyttää cloud-credentials nimiseen salaisuuteen tallennettuja tunnuksia tunnistautuakseen säilytyspaikkoihin. Tarvittaessa Veleroon voi konfiguroida useampia eri tunnuksia BackupStorageLocation-olioille, jos käytetty liitännäinen sitä tukee. Tällöin BackupStorageLocation-olioon voidaan konfiguroida käytetty tunnus oletustunnusten sijaan. VolumeSnapshotLocation-oliot käyttävät ainoastaan asennusvaiheessa määriteltyjä tunnuksia, jotka oletuksena löytyvät cloud-credentials-*salaisuudesta*. [69, Section: Backup Storage Locations and Volume Snaps-



Kuva 4.13. *Varmuuskopio sisältää, sekä AWS:n Elastic Block Store (EBS)-levyjä että Azuren Managed Disks -levyjä. Vain EBS-levyille on määritetty VolumeSnapshotLocation-olio, joten vain EBS-levyt varmuuskopioidaan.*

hot Locations, Velero Backup Storage Locations, Velero Volume Snapshot Location]

Kun käyttäjä aloittaa palautuksen varmuuskopiosta, klusteriin luodaan sitä vastaava Restore-olio. Restore-olion tehtävä on varmistaa, että klusteriin luodaan ne asiat, jotka palautuskomennossa määritellään. Palautuskomennossa voi palauttaa vain osan varmuuskopion sisältämistä resursseista. Logiikka on sama kuin varmuuskopioinnissa eli oletuksena kaikki nimiavaruudet, oliot ja levyt palautetaan. Tarkentamalla käyttäjä voi määrittää, mitkä resurssit ja nimiavaruudet palautetaan ja mitkä jätetään palauttamatta. Seuraava komento palauttaa testivarmuuskopio nimisestä varmuuskopiosta kaikki muut nimiavaruudet paitsi testiympäristön ja palauttaa ainoastaan *jalkautukset*:

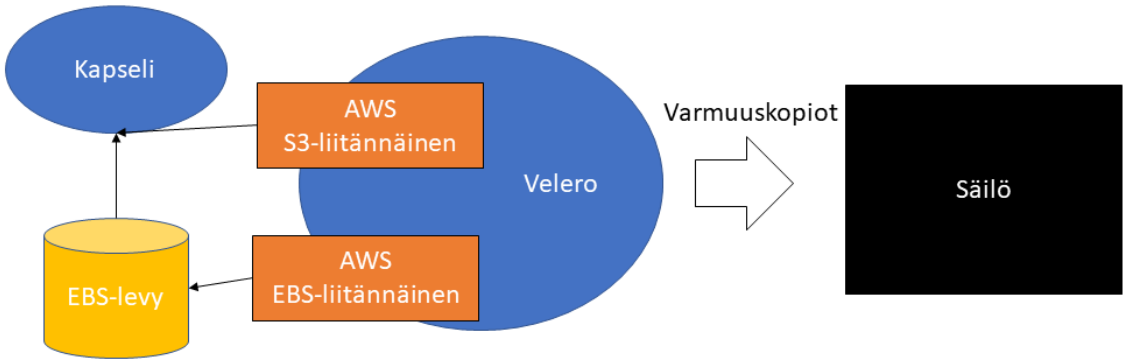
```
velero restore create omapalautus \
  --from-backup testivarmuuskopio \
  --exclude-namespaces testiymparisto \
  --include-resources deployments
```

Palautuksessa voi palauttaa tietyt nimiavaruudet uusilla nimillä. Seuraava komento tekee saman kuin edellinen komento, mutta omaympäristö-nimiavaruus palautetaan tuotantoympäristö nimellä:

```
velero restore create omapalautus \
  --from-backup testivarmuuskopio  \
  --exclude-namespaces testiymparisto \
  --include-resources deployments \
  --namespace-mappings omaymparisto:tuotantoymparisto
```

Palautusvaiheessa on myös mahdollista vaihtaa levyjen ominaisuudet määrittävä *säilytysluokka* (StorageClass) ja *pysyväislevypyynnöissä* määritettävä kohdepalvelin (selected-node). [69, Section: Resource Filterin, Restore Reference, Restore API Type]

Veleroon voi tarvittaessa määritellä varmuuskopiokytkentöjä (Backup hook), jotka suoritetaan osana varmuuskopiointia. Kytkennät määritellään *kapselien* tasolla tai Backup-

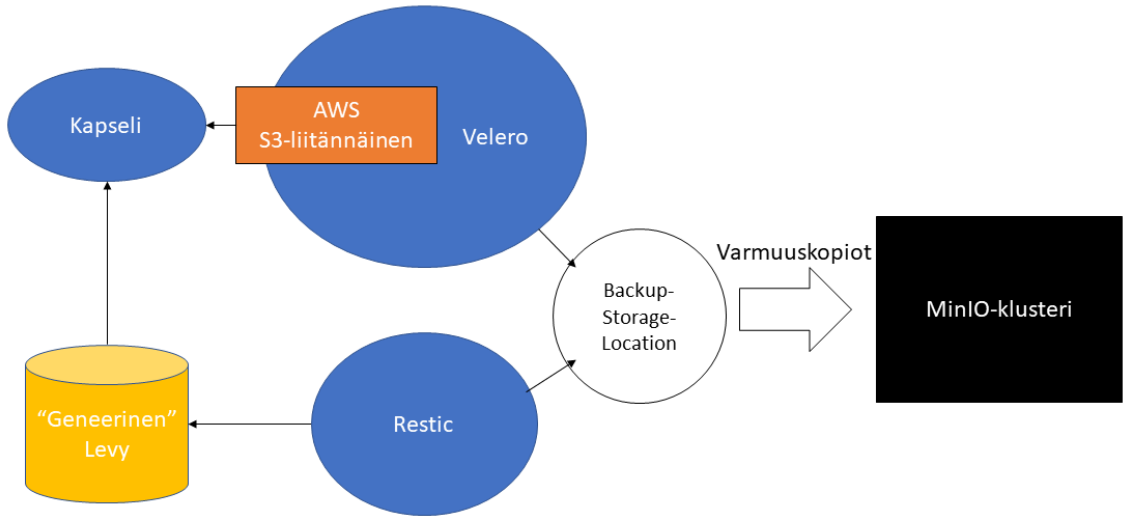


Kuva 4.14. *Velero hyödyntää AWS-liitännäisiä kopioidakseen sekä Kapselin, että siihen liitetyn pysyväislevyn.*

olioon. Kytkentä voidaan suorittaa ennen tai jälkeen varmuuskopioinnin. Oletuksena komento suoritetaan *kapselin* ensimmäisessä kontissa. Ensimmäinen kontti tarkoittaa *kapselin* manifestissa ensimmäisenä määriteltyä konttia. Backup-olioon määriteltävät kytkennät toimivat samoilla parametreilla. Backup-olioon pystyy lisäksi rajamaan nimiavaruustasolla mihin kytkennät kohdistuvat. [69, Section: Backup Hooks, Backup API Type]

Palautuskytkentöjä (Restore Hooks) voidaan suorittaa ennen *kapselin* konttien käynnistymistä tai käynnistymisen jälkeen osana palautusta. Palautuskytkennät voiaan määritellä annotoimalla kapseleita tai osana Restore-oliota. Ennen kontin käynnistymistä suoritettavat kytkennät määritellään InitContainer-kytkentöinä (InitContainer Restore Hook), jotka lisäävät *kapseliin* InitContainer-kontin osana palautusprosessia. Kuberneteksessa InitContainer-kontteja voi määrittää osaksi *kapseleita* niiden oliomanifesteissa initContainers-kentän alle ja ne suoritetaan siinä järjestyksessä kuin ne on määritelty. Veleron lisäämä InitContainer-kontti suoritetaan ensimmäisenä. InitContainer-kontit suoritetaan aina ennen kapselin pääkonttia ja ne toimivat esiehtona pääkontin käynnistymiselle. Restore-olioon määriteltävät InitContainer-kytkennät toimivat samoilla parametreilla ja ne voi palautuskytkentöjen tapaan rajata nimiavaruustasolla. Suorituskytkennät (Exec Restore Hooks) suoritetaan kontin käynnistyksen jälkeen ja ne määritellään samaan tapaan, kuin varmuuskopioinnin post-tyyppiset kytkennät, joko annotaatioilla tai Restore-olioon. [69, Section: Restore Hook, Restore API Type]

Velero hyödyntää liitännäisiä (plugins) integroituakseen eri säilytysteknologioihin. Liitännäisiä tarvitaan, koska jokaisella säilytysteknologialla on oma rajapintansa, jota Veleron pitää osata hydödyntää, jotta varmuuskopiointi ja palautus onnistuvat. Objektivarastot ja levyteknologiat tarvitsevat omat liitännäisensä. Velerossa on valmiiksi virallinen tuki AWS, GCP, Azuren VMwaren vSpeheren ja säilytysteknologioille. Virallinen tuki löytyy myös Container Storage Interface (CSI) spesifikaation implementoiville teknoloigioille. Virallisen tuen lisäksi Veleron käyttäjät ovat luoneet liitännäiset kuudelle alustalle. Veleron AWS-liitännäinen toimii usean objektivarastoteknologian kanssa, jotka hyödyntävät AWS:n S3:n rajapintaa. Velerossa on beta-tasolla tuki Resticille, joka mahdollistaa useimpien eri levytyyppien varmuuskopioinnin. [69, Section: Velero plugin system, Providers]



Kuva 4.15. *Kubernetes-oliot varmuuskopioidaan S3-liitännäisellä. BackUpStorageLocation-olioon on konfiguroitu S3-APIn täyttävään säilöön, joka on kohdeorganisaation tapauksessa MinIO. Veleron BackupController suorittaa tarvittavat Restic komennot. Kapselin nimiavaruuteen liittyvään ResticRepository-olioon on konfiguroitu, mitä BackupStorageLocation oliota Resticin tekemille varmuuskopioille käytetään. Kohdeorganisaation tapauksessa säilönä on sama MinIO-klusteri kuin mitä Kubernetes-olioille käytetään.*

Kohdeorganisaation tapauksessa *pysyväislevypyynnöllä* dynaamisesti luodut levyt lohkotaan palveluntarjoajan levypinnalta, joten niiden varmuuskopiointiin ei löydy virallista tai yhteisön tarjoamaa liitännäistä. Varmuuskopiointi onnistuu kuitenkin Resticillä, joka on asennettuna suoraan Kubernetes-klusteriin. Resticin ollessa käytössä varmuuskopioitavat levyt voidaan valita joko niin, että kaikki varmuuskopioitavat levyt pitää annotoida tai ne levyt, joita ei varmuuskopioida annotoidaan. Hyödyntämällä Resticiä Velero täyttää kohdeorganisaation vaatimuksen *pysyväislevyjen* sisällön kopioimisesta. [69, Section: Providers, Restic Integration]

Velero luo ResticRepository-olion Kubernetes-nimiavaruuteen, kun ensimmäinen levy nimiavaruudesta varmuuskopioidaan. Jokaista ResticRepository-oliota vastaa erillinen Restic-säilö. Resticin toiminta kohdeorganisaation klusterin kontekstissa on havainnollistettu kuvassa 4.15. Jokaista varmuuskopioitavaa levyä kohti, Velero luo PodVolumeBackup-olion. Varmuuskopioinnista vastaa ohjain, joka suorittaa tarvittavat Restic komennot ja niiden onnistumisen perusteella päivittää PodVolumeBackup-olion tilan. Kun kaikkien PodVolumeBackup-oliot ovat valmiita, Velero tietää, että varmuuskopiointiprosessi on suoritettu onnistuneesti loppuun. [69, Section: Restic Integration]

Palautuksessa Velero tarkastaa palautettavan varmuuskopion sisältämät PodVolumeBackup-oliot ja luo niitä vastaavat PodVolumeRestore-oliot ja varmistaa, että tarvittavat ResticRepository-oliot ovat olemassa. Palautusprosessista vastaava ohjain ajaa tarvittavat Resticin palautuskomennot ja kirjoittaa palautetulle levylle .velero nimiseen hakemistoon varmistustiedoston, joka merkkaa palautuksen onnistuneen. Ohjain päivittää PodVolumeRestore-olioiden tilan onnistuneeksi, kun se on kirjoittanut varmistustiedoston. Velero käyttää Res-

ticistä tehtävissä palautuksissa InitCointaner-kytkentää, joka varmistaa, että kaikki *kapselin* levyt on palautettu onnistuneesti tarkastamalla että varmistustiedosto löytyy *kapselin* levyltä. InitCointaner-kytkennän tarkoitus on varmistaa, että *kapseli* ei siirry suorittamaan muita alustustoimenpiteitä ennen kuin levy on palautettu. Velero tarkastaa palautuksen onnistumisen PodVolumeRestore-olioiden tiloista. [69, Section: Restic Integration]

4.3 Lokien kerääminen ja visualisointi

Koska sovelluksien uusi ajoympäristö on Kubernetes, jossa kontit ovat ohimeneviä, tarvitaan lokien pysyvämpää säilytystä varten keskitetty lokipalvelu. Keskitetty lokipalvelu palvelee samalla myös mahdollisia skaalautuvia sovelluksia. Tässä työssä oletetaan, että sovellukset lokittavat standarditulosteeseen ja standardivirheeseen sekä lisäävät sovelluksen tiedossa olevat kontekstiedot lokiriveille. Kohdeorganisaatiossa päädyttiin käyttämään Fluent Bittiä lokien siirtoon, hakumoottorina Elasticsearchia ja visualisointiin Kibanaa. Elasticsearch ja Kibana ovat osa Elastic-pinoa, joka on yleinen vaihtoehto lokien keräämiseen, tallentamiseen ja visualisointiin [75]. Kohdeorganisaatiolla oli jo ennestään käytössä Elastic-pinon asennus, joka vaikutti vahvasti Elasticsearchin ja Kibanan valintaan hakumoottoriksi ja visualisointityökaluksi. Valitut teknologiat muodostavat kokonaisuuden, joka vastaa kohdeorganisaation vaatimukseen siitä, että lokit halutaan säilöä Kuberneteksen ulkopuolelle ja niiden tarkastelu tapahtuu keskitetysti yhdestä paikasta.

Keskitetyissä lokipalveluissa lokien siirtoon käytetään sovelluksia, jotka erikoistuvat lokien keräämiseen ja eteenpäin lähettämiseen [76]. Yleensä Kuberneteksessa jokaisen klusterissa ajossa olevan kontin standardivirtaan tulevat lokit ovat luettavissa työskentelijäpalvelimilla olevista tiedostoista. Niiden formaatti ja sijainti riippuu käytetystä konttiteknologiasta, mikä on kohdeorganisaation tapauksessa Docker [24, Section: Logging Architecture]. Kohdeorganisaatiossa lokien siirtoon valittiin Fluent Bit. Karbon-klusterissa on valmiiksi asennettuna Fluent Bit -keräimiä, mutta ne tarkkailevat pelkästään Karbonin Kubernetes ja Nutanix-nimiavaruuksia.

Fluent Bit perustuu eri liitännäisiin, joilla luetaan, muokataan, filtteröidään ja lähetetään dataa eteenpäin. Fluent Bit kommunikoi Kuberneteksen API-palvelimen kanssa ja saa siltä metatietoja, kuten mistä *kapselista* ja miltä palvelimelta lokiviesti on. Lokeja voidaan myös hakea ja filtteröidä metatietojen perusteella, mikä nopeuttaa haluttujen viestien löytymistä. Metadatasta filtteröidään turhat tiedot pois ja uudelleen nimetään kenttiä tarpeen mukaan. Lopulta Fluent Bit muokkaa ulostuloliitännäisillä datan siihen muotoon, jota lokien säilytykseen valittu tietovarasto ymmärtää. Kohdeorganisaation tapauksessa Elasticsearch-liitännäinen muokkaa lokiviestin sellaiseen muotoon, että Elasticsearch pystyy sen vastaanottamaan. [77, Section: Kubernetes, Key Concepts]

Muita vaihtoehtoja keräimiksi olivat Fluentd ja Filebeat. Fluentd ja Fluent Bit ovat saman yhtiön, Treasure Datan, kehittämiä keräimiä. Fluent Bit on kevyempi ja arkkitehtuuriltaan minimalistisempi. Fluent Bitillä ei ole ulkoisia riippuvuksia toisin kuin Fluentd'llä, joka on riippuvainen Ruby-kirjastoista. Fluentd on näistä vanhempi ja sille on enemmän liitännäi-

siä, ja näin ollen laajempi tuki kuin Fluent Bitille [78]. Kohdeorganisaatiolla ei kuitenkaan ollut tarvetta Flunetd'en laajimmille ominaisuuksille. Kohdeorganisaatiossa haluttiin valita mahdollisimman kevyt keräin, joka ratkaisi Fluent Bitin valinnan. Tarvittaessa Fluent Bit -keräimet voivat syöttää datansa Fluentd'lle, jos jotakin sen ominaisuutta tarvitaan. Keveys mahdollistaa myös Fluent Bitin lokienkerääjänä tarvittaessa sivuvaunukontissa. Sivuvaunukontissa joita ajetaan konttikohtaisesti sen sijaan, että koko työskentelijäpalvelimen lokeja tarkkailee yksi agentti [24, Section: Logging Architecture].

Elastic-pinon oletusvalinta sovelluslokien ja muun datan keräämiseen käytetään beat-kerääjiä ja niistä Filebeattia käytetään lokien keräämiseen tiedostoista [76]. Filebeat on myös hyvin kevyt ja integroituu Kuberneteksen API-palvelimeen [79, Section: Filebeat overview, Add Kubernetes metadata]. Filebeatilla on jonkin verran vähemmän ulostuloliitännäisiä kuin Fluent Bitillä, mutta se tukee silti yleisimpiä datavarastoja [79, Section: Configure the output]. Filebeat ei eroa kohdeorganisaatiolle tärkeiltä ominaisuuksilta, keveys ja Kuberetes-integraatio, merkittävästi Fluent Bitistä. Fluent Bit saatiin toimimaan nopeasti Karbon ympäristössä, joten Filebeattia ei harkittu sen tilalle.

Lokit täytyy säilöä, johonkin, jotta niitä voidaan lukea ja analysoida tarpeen mukaan senkin jälkeen, kun sovelluskontti, josta ne ovat peräisin ei ole enää olemassa. Elasticsearch on Apache Lucenen päälle rakennettu hakukone ja analytiikkatyökalu, joka on suunniteltu skaalautuvaksi ja vikasietoiseksi. Elasticsearch tallentaa vastaanottamansa tiedon JSON-dokumentteina. Elasticsearch indeksoi jokaisen dokumentin kentän sille sopivalla tietorakenteella. Elasticsearch osaa indeksoida dokumentit myös ilman ennalta annettua skeemaa, lisätä dynaamisesti kentät hakuindeksiin ja päättellä niille sopivan tietotyypin. Elasticsearch tukee myös hybridimallia, jossa osalle kentistä on määritelty tietotyyppi ennakkoon. [76][79, What is Elasticsearch?, Data in: documents and indices]

Elasticsearchissa on REST-API ylläpitoa, dokumenttien lisäystä ja hakuja varten. Haut voi kohdistaa joko suoraan dokumenttien kenttiin rakenteellisella haulla (structured query) tai tekstihaulla, joka kohdistaa haun indeksoituihin tekstikenttiin, joista on luotu käänteinen indeksi. Käänteinen indeksi pitää sisällään tiedon kaikista dokumenteissa esiintyvistä termeistä ja tiedon siitä, missä dokumenteissa ne esiintyvät. Haut voivat olla yhdistelmä tekstihakuja ja rakenteisia hakuja. [76][79, Information out: search and analyze]

Elasticsearch on suunniteltu sulavasti skaalautuvaksi ja sitä voidaankin käyttää joko yhdellä palvelimella tai useammasta palvelimesta koostuvana klusterina. Elasticsearch jakaa dokumentit Shardeihin, joita yhdistelemällä Elasticsearch luo hakuindeksin. Shardeja on kahta tyyppiä, primäärisiä ja replikoita. Jokainen dokumentti kuuluu ensisijaisesti primärishardiin, josta luodaan replikoita redundanssin ja suorituskyvyn parantamiseksi. Kun Elasticsearch-klusterin koko muuttuu, Elasticsearch tasapainottaa klusteria lisäämällä, vähentämällä ja siirtämällä shardeja. Klusterin muodostavien palvelimien on oltava samassa verkossa suorituskyvyn takia. Maantieteellisen, useamman datakeskuksen hajautuksen saavuttamiseksi täytyy käyttää klusterien välistä replikointia (Cross-Cluster replication). Replikointi on yksisuuntaista eli yksi klusteri toimii pääklusterina, jonka muutokset heijastuvat muihin klustereihin [76][79, Scalability and resilience: clusters,

nodes, and shards]Elasticsearchissa on REST-API ylläpitoa, dokumenttien lisäystä ja hakuja varten. Haut voi kohdistaa joko suoraan dokumenttien kenttiin rakenteellisella haulla (structured query) tai tekstihaulla, joka kohdistaa haun indeksoituihin tekstikenttiin, joista on luotu käänteinen indeksi. Käänteinen indeksi pitää sisällään tiedon kaikista dokumenteissa esiintyvistä termeistä ja tiedon siitä, missä dokumenteissa ne esiintyvät. Haut voivat olla yhdistelmä tekstihakuja ja rakenteisia hakuja. [76][79, Information out: search and analyze]

Tässä työssä harkittiin Elasticsearchin vaihtoehdoksi Logglyä, Lokia ja InfluxDB:tä. Loggly on SolarWinds Worldwide -yhtiön tuote ja siitä on olemassa vain Software as a Service -versio, eikä sitä voi asentaa omalle palvelimelle. Kohdeorganisaatiossa haluttiin säilyttää lokidata omilla palvelimilla, joten Loggly ei ollut vaihtoehto [80]. InfluxDB avoimen lähdekoodin hakumoottori, jota kehittää InfluxData-yhtiö. InfluxDB on suunniteltu pääasiallisesti aikasarjojen, kuten metriikkadatan säilyttämiseen, hakuun ja analysointiin. Elasticsearch on vahvempi dokumenttihaussa, jota lokidata on. Elasticsearch on myös parempi skaalautumaan kuin InfluxDB, joka on tärkeää koska kohdeorganisaation tahtotilana on siirtää kaikki sen itse ylläpitämät palvelut Karboniin [81]. Vaikka InfluxDB:en lisenssi sopii paremmin kohdeorganisaatiolle kuin Elasticsearchin nykymuotoinen lisenssi, valittiin Elasticsearchin siitä huolimatta, sen paremman soveltuvuuden ja suorituskyvyn vuoksi [82, 83].

Loki on Grafana Labsin kehittämä hakumoottori, joka on pyritty pitämään pelkistettynä ja yksinkertaisena [84]. Lokin yhteydessä käytetään yleensä sille suunniteltua Promtail-keräintä, mutta myös Fluent Bit pystyy lähettämään lokeja Lokiin [84] [77, Section: Outputs]. Loki on pelkistetyn toiminnallisuutensa vuoksi resurssitehokkaampi, kuin Elasticsearch, mikä näkyy kapeampina hakumahdollisuuksina ja muina toiminnallisuuksina [84]. Testaamalla havaittiin, että jo toiminnassa ja klusterin ulkopuolella oleva Elasticsearch pystyy käsittelemään Kuberneteksesta tulevat lokit ja sen resurssikulutusta ei katsottu ongelmaksi. Jos Elasticsearch olisi haluttu korvat Lokilla olisi myös Kibana pitänyt vaihtaa toiseen visualisointikomponenttiin, minkä katsottiin olevan liian työlästä suhteessa Lokin tehoetuihin. Jos jokin yksittäinen sovellus tarvitsee tulevaisuudessa ympäristön, johon ei tule muiden sovellusten lokeja niin tällöin Loki on varteenotettava vaihtoehto. Myöskään Lokin avoimemman lisenssin ei katsottu olevan riittävän merkittävä etu [85].

Elasticsearch toimii hakumoottorina ja tarjoaa hakupalvelunsa REST-rajapinnan yli, eikä toteuta hakukäyttöliittymää tai muita visualisointeja. Elastic-pinossa visualisontikomponenttina toimii Kibana [76]. Kibanalla pystyy lokien lukemisen ja hakemisen lisäksi esimerkiksi piirtämään erilaisia kaavioita, analysoimaan lokidataa yhdistettynä paikkatietoihin. Kibanaan pystyy rakentamaan erilaisia näkymiä, joihin voi rakentaa sovellus- ja tapauskohtaisesti tarvittavan informaation näkyville. Kibana myös integroituu vahvasti Elasticsearchiin ja sitä voi käyttää Elasticsearch-klusterin hallinointiin. Esimerkiksi Elasticsearchin-hakuindeksien kierron ja säilytysajan pystyy konfiguroimaan suoraan Kibanasta [76][52, Section: What is Kibana]. Sen jälkeen, kun kohdeorganisaatiossa tehtiin päätös Elasticsearchin käytön jatkamisesta tässä työssä tehdyn selvityksen perusteella, tehtiin samalla

päätös Kibanan käytön jatkamisesta. Kibanan katsottiin täyttävän visualisoinnin ja hakemisen tarpeet. Lisäksi kohdenorganisaatiossa oltiin jo totuttu Kibanan hakutyökaluihin ja hallitsemaan Elasticsearchia Kibanan kautta.

5 ARVIOINTI JA JATKOKEHITYS

Tässä luvussa arvioidaan kuinka hyvin valitut teknologiat ja toteutetut ratkaisut vastaavat luvussa 3 esitettyjä vaatimuksia. Jatkokehitystä käsittelevissä aliluvuissa on avattu tämän työn ulkopuolelle jääneitä ominaisuuksia. Koska työ liittyy luonteeltaan operatiivisiin järjestelmiin, on luonnollista, että uusia vaatimuksia syntyi myös työn kirjoituksen aikana havaituista tarpeista.

5.1 Reititys

Reititys toteutettiin luvussa 4.1.2 kuvatulla Traefik käänteisproxyllä. Vaatimuksista tärkeimmäksi noussut vaatimus oli, että TLS-varmenteita ei tarvitsisi enää ylläpitää käsin. Ensimmäisessä asennuksessa oli ongelmia varmenteiden tallentamisessa, tämän jälkeen Traefik on toiminut ongelmitta. Vaatimus automaattisista varmenteista saatiin siis toteutettua. Sisäverkossa ei ollut vastaavaa vaatimusta automaattisista varmenteista. Myös Traefikin automaattinen reititys on toiminut ilman ongelmia koko sen käytön ajan. MetalLB:n käyttö on ollut vähäistä, sillä ulkoverkon asiakaspalveluiden siirto on ollut korkeammalla prioriteetilla.

Molemmissa sekä sisä- että ulkoverkon tapauksessa haluttiin, että valitut teknologiat kykenevät taakanjakoon tarvittaessa. Tämä vaatimus toteutuu sekä ulkoverkon Traefikilla, että sisäverkossa käytössä olevalla MetalLB:llä. Traefik jakaa liikenteen suoraan *kapseleille*, kun taas MetalLB ohjaa liikenteen *palveluille*, jotka ohjaavat sen oikeille *kapseleille*. Kumpikaan valituista teknologioista ei aiheuta toimittajaloukkua, sillä ne eivät ole mitenkään sidottuja Karboniin tai tiettyyn palveluntarjoajaan. Trefik vaatii vain sen, että sen endpoint-kenttään konfiguroidaan IP-osoite, joka ohjaa Kubernetes-klusterin isäntäpalvelimelle. Traefik voi käyttää myös muidenkin kuin Kuberneteksen yhteydessä, joten Traefik ei sido kohdeorganisaatiota Kubernetekseen. MetalLB on luotu yksityispilviä varten, jotka eivät hyödynnä julkisten pilvipalvelujen teknologioita IP-osoitteiden jakamiseen. Sen käyttö edellyttää ainoastaan, että ympäristössä ei ole valmiiksi taakanjakajaa, joka jakaa IP-osoitteita.

On hankala arvioida miten Traefikin ylläpito jatkuu keskipitkällä aikavälillä. Traefik on kuitenkin laajasti käytössä: se on ylittänyt 2 miljardin latauksen rajan ja Traefik labs on kerännyt 16 miljoonaa dollaria rahoitusta pääomasijoittajilta. V1.x versioille taattiin aluksi 1 vuoden ylläpito, joka pidennettiin kestämään 2 vuoteen. Kohdeorganisaatiossa käytössä olevalle V2.x versiolle ei ole vielä annettu vastaavaa ylläpitolupausta, koska se on aktiivi-

sessa kehityksessä. Vaatimus keskipitkän aikavälin ylläpidosta toteutuu todennäköisesti, sillä Traefik v3 ei ole ainakaan julkisesti vielä edes suunnitteluasteella. [86, 87, 88, 89]

MetalLB:n dokumentaatiossa sanotaan suoraan, että se on vielä nuori ja beta-vaiheessa. MetalLB:llä ei ole myöskään takanaan yritystä, joten sen kehitys riippuu kokonaan kehittäjien käytössä olevasta ajasta. MetalLB:lle ei ole taetta, että se saa tietoturvapäivityksiä tai että se päivittyy uusien Kubernetes-versioiden mukana [57, Section: Project Maturity]. Näin ollen keskipitkän aikavälin tuen ei voida olettaa toteutuvan.

Palvelimet, joille Traefik on asennettu, pitää yhä pystyttää ja konfiguroida käsin, joten esimerkiksi vikatilanteessa uuden Traefik-palvelimen pystyttäminen vie aikaa. Traefik-palvelimen pystytyksen automatisointi on selkeä jatkokehityskohde. Traefik pitää myös käynnistää uudelleen, kun se konfiguroidaan tarkkailemaan jotakin tiettyä Kuberneteksen nimiavaruutta, mikä aiheuttaa lyhyen palvelukatkoksen. Palvelukatkoksesta eroon pääsy on toinen selkeä jatkokehityskohde.

Koska MetalLB:n ylläpito on niin epävarmaa, selkeä jatkokehityskohde on etsiä sille varmemmin tuettu vaihtoehto. Kunkin vaihtoehdon kohdalla pitää aina selvittää yhteensopivuus Karbonin kanssa, jos sen vaatimukset poikkeavat MetalLB:n vastaavista. Koska tässä työssä tutkitut suorat vaihtoehdot eivät tarjonneet varmistettua parempaa ylläpidettävyyttä, on yksi mahdollisuus tutkia Traefikin käyttöä myös sisäverkossa.

5.2 Varmuuskopiointi

Varmuuskopioinnin vaatimukseksi asetettiin, että *pysyväislevyt* saadaan varmuuskopioitua ja palautettua tarvittaessa joustavasti. Velerolla, MinIOlla ja Resticillä rakennettu varmuuskopiointijärjestelmä toteuttaa tämän vaatimuksen selkeästi. Järjestelmä varmuuskopioi levyjen lisäksi Kubernetes-oliot, jolloin yksittäisten nimiavaruuksien palauttaminen vikatilanteissa tai esimerkiksi testausta varten on helppoa. Myös koko klusterin palauttaminen tai siirto on mahdollista. MinIOn EC datan pilkkomistekniikka takaa hyvän suojan laiterikoilta ja datan korruptoitumiselta. Resticin ja MinIOn vahvat salaukset takaavat teknologian tasolla riittävän tietoturvan. Varmuuskopiointiin ollaan oltu kohdeorganisaatiossa tyytyväisiä. Varmuuskopiointi kohdistuu koko klusteriin, ilman että se vie liikaa säilytysresursseja. Näin varmuuskopioinnin konfiguraatio on saatu pidettyä yksinkertaisena ja uudet sovellukset astuvat varmuuskopioinnin piiriin automaattisesti.

Veleroa ei ole sidottu mihinkään yksittäiseen Kubernetes-ympäristöön, vaan se pystyy lukuisten liitännäistensä avulla toimimaan useimmissa julkisissa ja yksityisissä pilvissä, joten se ei aiheuta toimittajaloukkua. MinIO sitoo kohdeorganisaatiota S3-API:in, sillä MinIOsta pystyy suoraan siirtämään dataa vain toisiin S3-API:a hyödyntäviin varastoihin. Datan voi kuitenkin palauuttaa Velerolla Kubernetekseen ja siirtää toiseen loppusäilöön tarvittaessa. Restic ei sido kohdeorganisaatiota vaan on ratkaisu, jolla geneeriset levyt saadaan varmuuskopioitua Kubernetes ympäristössä. Jos tulevaisuudessa toimitaan jossakin toisessa Kubernetes-ympäristössä kuin Karbonissa, voidaan Resticiä käyttää todennäköisesti uudelleen tai Velero tukee alustan levyteknologioita suoraan.

Kohdeorganisaation käyttäjällä MinIOn versiolla ei ole varsinaisesti ylläpitoa, vaan virallinen ohje on aina päivittää uusimpaan versioon. Jos ilmenee pakottava tarve saada varmasti tuki käytössä olevalle versiolle, voi kohdeorganisaatio siirtyä joko standard-lisenssin 1 vuoden tuen piiriin tai enterprise-lisenssin 5 vuoden tuen piiriin. Resticillä ei ole taustalla yrityssponsoria. Sen kehitys ja ylläpito on sen kehittäjäyhteisön varassa, joten ylläpidon pituutta on hankala arvioida ja sen toteutumista ei voi sanoa varmaksi. Resticistä ei myöskään ole vielä 1.0 versiota, joten se voi muuttua vielä paljon. Veleron taustalla on VMWare, joka on suuri pörssiyhtiö, jonka liikevaihto liikkuu miljardeissa. VMWare ei kuitenkaan ole antanut eksplisiittistä sitoumusta Veleron kehityksen tukemiselle. Velero 2.0 löytyy Veleron kehityssuunnitelmista. Tieto siitä, kuinka pitkään 1.x versiota tuetaan, tulee vasta lähempänä 2.0 julkaisua. Keskipitkän aikavälin tukea ei voida siis Veleron kohdalla katsoa varmaksi.

Kaikilla kehitystiimeillä olisi hyvä olla oikeudet projektinsa varmuuskopioihin. Tällöin kehitystiimi voi palauttaa järjestelmän tilan varmuuskopiosta ongelmatilanteissa ja hyödyntää Veleron tarjoamaa mahdollisuutta pystyttää testiympäristöjä nopeasti palauttamalla varmuuskopio uudelleen nimettyyn Kuberneteksen nimiavaruuteen. Nyt palautukseen tarvittavat oikeudet ovat pelkästään ylläpitotiimillä.

Tiimien olisi hyvä harjoitella säännöllisesti varmuuskopista palauttamista ongelmatilanteissa, varsinkin jos varmuuskopioita ei hyödynnetä aktiivisesti, esimerkiksi testauksessa. Säännöllisellä varmuuskopioinnilla varmistetaan, että palautus ylipäätään onnistuu ja että kaikki tarvittava varmuuskopioidaan. Ylläpitotiimin olisi myös hyvä harjoitella koko klusterin palauttamista varmuuskopiosta uuteen klusteriin.

5.3 Lokitus

Lokien tarkastelun tärkeimpänä tavoitteena oli, että kehittäjien ei enää tarvitsisi ottaa erikseen yhteyttä ilmentymiin ssh-yhteyden tai Kuberneteksen lokirajapinnan kautta. Tämä tavoite saavutettiin ohjaamalla klusterissa olevien sovellusten lokit Fluent Bitillä Elasticsearhille. Fluent Bittiin keräimenä ja Elastisearch hakumoottorina ovat soveltuneet yrityksen käyttöön hyvin. Aiemmin Kibana oli ollut pienemmän kehittäjoukon käytössä ja Kuberneteksen myötä koko organisaation pitäisi käyttää sitä. Laajempi käyttäjäkunta on pitänyt Kibanaa turhan monimutkaisena. Kibanassa ei myöskään ole vapaassa lisenssisä mahdollisuutta rajata käyttäjien pääsyä tiettyihin hakuindekseihin. Elasticsearchiin liittyvissä ylläpitotehtävissä Kibana on osoittautunut yhä toimivaksi.

Valituista teknologioista Fluent Bit on helpoiten vaihdettavissa toiseen vastaavan toiminnallisuuden teknologiaan, joka pystyy lähettämään dataa Elasticsearchiin. Fluent Bit pystyy myös lähettämään dataa useihin eri tietovarastoihin ja se on vapaasti laajennettavissa tarvittaessa. Kibana pystyy vastaanottamaan dataa useista eri lähteistä, kuten esimerkiksi Kafka tai Traefik, mutta varsinaisista hakuindekseistä se tukee tällä hetkellä ainoastaan Elasticsearchia. [52, Section: Kibana Plugins]

Elastic-pinon kehityksestä vastaa Elasticsearch BV, jonka päätuote on palveluna ylläpi-

detty Elastic-pino. Elastic-pino on todella laajasti käytetty, joten sen ylläpidon voidaan odottaa jatkuvan keskipitkällä aikavälillä. Yksittäisten komponenttien ylläpito riippuu siitä, miten seuraavia versioita julkaistaan. Ylläpito koskee uusimman major-version (kirjoitushetkellä 7.X versiot) viimeisintä minor-versiota ja edellisen major-version viimeisintä minor-versiota. Kaikkia pinon osia koskee sama järjestelmä ja tällä hetkellä jokainen komponentti on major-versioltaan 7. Jos Major-versioita julkaistaan samalla syklillä kuin tähän asti, niin viimeisiä 7.X-versioita ylläpidetään 2023 asti, mikä on kohdeorganisaatiossa arvioitu kohtuulliseksi ajaksi. [52, Section: Elastic product end of life dates], [90]

Fluent bitin taustalla on yli 200 hengen kehittäjäyhteisö. Se on Fluentd:n alaprojekti, joten sillä on Fluentd:n kautta Cloud Native Computing Foundationin projektistatus. Fluent Bit on myös käytössä useilla isoilla toimijoilla muunmuassa Googlella, Amazonilla ja Microsoftilla. Ylläpidon säilyvyyttä voidaan pitää siis hyvin todennäköisenä keskipitkällä aikavälillä. [91, Section: Fluent Bit, Community]

Tässä työssä suunniteltiin pelkästään sovelluslokien kerääminen Karbon-ympäristöstä keskitettyyn säilytyspaikkaan. Työn ulkopuolelle jäi resurssien monitorointi ja siihen liittyvät hälytykset. Resurssien monitoroinnilla, esimerkiksi Prometheuksen avulla, saadaan monipuolista tietoa siitä, kuinka paljon ja mitä resursseja sovellukset käyttävät. Prometheuksen keräämistä aikasarjoista voidaan myös tarkkailla, miten resurssien käyttö muuttuu ajan kuluessa. [26]

Toinen työn ulkopuolelle jäänyt lokituksen ulottuvuus on jäljitys. Jäljitys lisää lokiviesteihin tunnisteen, joka linkittää esimerkiksi yhdestä asiakassovelluksen tekemästä pyynnöstä syntyvät lokirivit. Jäljitys ei ole toistaiseksi ajankohtainen, kun sovellukset ovat yksittäisiä monoliittejä. Monimutkaisemmissa järjestelmissä jäljitys on hyödyllinen elementti virheiden ja suorituskykyongelmien ratkaisussa. Traefikissa on valmiiksi tuki jäljitykselle ja se tukee kuutta eri jälijitysteknologiaa. Oletuksena Traefik käyttää jäljitykseen Jaegeria. [65, Section: Tracing] [6, 32]

6 YHTEENVETO

Tässä työssä työssä on selvitetty pilven ominaispiirteitä suoritusympäristönä ja miten kyseiset ominaisuudet on toteutettu Amazon Web Services (AWS) -alustalla. Selvitys tehtiin, jotta työn kohdeorganisaatioon pystyttäisiin toteuttamaan riittävät palvelut, jotka puuttuvat Nutanixin Karbon alustasta. Työssä on kuvattu valitut teknologiat tarkkaan ja niiden käyttöönotto kohdeorganisaatiossa siten, kuin niitä on työn toteutuksen aikana saatu tuotantokelpoisiksi.

Kohdeorganisaatiossa tunnistettiin tärkeimmiksi kolme eri osa-aluetta, jotta Karbon voidaan ottaa käyttöön tuotantoympäristönä: reititys, varmuuskopiointi ja sovelluslokien kerääminen ja keskittäminen yhteen paikkaan. Jokaisesta osa-alueesta on ensin kartoitettu perusperiaatteet ja esitelty miten osa-alue on toteutettu AWS:ssä. AWS:n pidetään yhtenä pilvialustojen edelläkävijöistä, joten se valittiin esimerkiksi alan viimeisimmistä kehityksestä. Lisäksi jokaiselle osa-alueelle on esitelty kohdeorganisaatiossa spesifioidut vaatimukset.

Työssä esitettiin työn aihepiirin paremman ymmärtämisen vuoksi konttiteknologiat, Kubernetes ja pilvinatiivit sovellusarkkitehtuurit. Konttiteknologioiden ja Kuberneteksen ymmärtäminen on tärkeää, sillä ympäristö, jota vasten työn eri ratkaisuja on suunniteltu ja toteutettu on Nutanixin Kubernetes-ratkaisu Karbon. Pilvinatiivit arkkitehtuurit esiteltiin, jotta lukija ymmärtää minkälaisia vaatimuksia sovellusarkkitehtuurit asettavat valituille teknologioille.

Ulkoverkon reitityksessä teknologiaksi valittiin Traefik sovellusproxy, joka otettiin kohdeorganisaatiossa käyttöön omalle palvelimelleen asennettuna käänteisproxyna. Traefik valittiin sen vuoksi, että se on alusta asti kehitetty pilvinatiiviin ympäristöön ja sisälsi hyvän integraation Kubernetekseen Traefik *omaolioilla*. Erilliselle palvelimelle asentamiseen päädyttiin, koska Traefik haluttiin erilleen varsinaisesta klusterista ja klusteriin suoraan jalkautettu Traefik ei pysty automaattisesti hankkimaan varmenteita.

Sisäverkon reititykseen valittiin MetalLB-taakanjakaja, joka on tarkoitettu yksityispilviin. MetalLB tuo yksityispilviin mahdollisuuden hyödyntää Kuberneteksen LoadBalancer-*palveluita* samalla tavalla, kuin niitä voidaan hyödyntää isoilla julkisilla pilvialustoilla. MetalLB:n ja muiden vastaavien teknologioiden heikkous on niiden ylläpidon epävarmuus. Toistaiseksi MetalLB kuitenkin vastaa sille asetettuun vaatimukseen allokoida *palveluille* sisäverkon IP-osoitteita ja mahdollistaa taakanjako.

Muista työn osa-alueista poiketen varmuuskopioinnin toteutti sama taho, jolta kohdeor-

ganisaatio ostaa palvelinresurssinsa ja Karbon ympäristön. Työssä kuitenkin esiteltiin samalla tavalla pilviympäristöjen varmuuskopioinnin taustat ja AWS:n Kubernetes-ympäristön EKS:n varmuuskopiointi. Hankkimalla kunnollinnen ymmärrys varmuuskopioinnissa käytetyistä teknologioista kohdeorganisaatioon muodostui parempi kuva siitä, miten niitä voidaan hyödyntää ja mitkä niiden rajoitteet ovat. Ratkaisun erityispiirteeksi nousi Resticin hyödyntäminen *kapselien pysyväislevyjen* varmuuskopioinnissa. Varmuuskopiointi ylitti sille asetetut tavoitteet ja mahdollistaa nyt varmuuskopioiden aktiivisen hyödyntämisen sovelluskehityksessä eikä pelkästään tiedostojen palauttamisen, johon se alun perin kaavailtiin.

Sovelluslokituksessa päädyttiin käyttämään Elastic-pinoa ja Fluent Bittiä. Fluent Bit valittiin Kubernetes-integraationsa ja keveytensä vuoksi. Elasticsearch hakumoottori osoittautui kohderoganisaation tarpeisiin sopivimmaksi vaihtoehdoksi lokien säilytykseen ja hakuun. Elasticsearchin valinta ohjasi vahvasti Kibanan valinnan visualisointikomponentiksi. Kohdeorganisaatiossa oli myös aiempaa kokemusta Elastic-pinon hyödyntämisestä.

LÄHTEET

[1] Davis, C. *Cloud Native Patterns. Designing change-tolerant software*. Manning Publications, 2019.

[2] Laberis, B. *What Is the Cloud?* O'Reilly Media, Inc., 2019.

[3] Huang, H. W. D. *Mobile Cloud Computing*. Morgan Kaufmann, 2017.

[4] Bias, R. *The History of Pets vs Cattle and How to Use the Analogy Properly*. 13. toukokuuta 2020. URL: http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/?utm_source=thenewstack&utm_medium=website (viitattu 13. 05. 2020).

[5] Goasguen, S. *Docker cookbook*. eng. First edition. Sebastopol, CA: O'Reilly, 2015. ISBN: 1-4919-1970-1.

[6] Scholl, B., Swanson, T. ja Jausovec, P. *Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications*. eng. Sebastopol: O'Reilly Media, Incorporated, 2019. ISBN: 1492053821.

[7] Luksa, M. *Kubernetes in action*. eng. 1st edition. Shelter Island, NY: Manning Publications. ISBN: 1-61729-372-5.

[8] Nutanix, Inc. *Karbon. Kubernetes Management Made Simple*. URL: https://www.nutanix.com/products/karbon (viitattu 15. 02. 2021).

[9] Lukka, K. *Kari Lukka: Konstruktiivinen tutkimusote*. 19. toukokuuta 2014. URL: https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/ (viitattu 22. 05. 2020).

[10] Martikainen, A. *Tilallisuuden hallinta mikropalvelusovelluksessa*. fin. Informaatioteknologian ja viestinnän tiedekunta - Faculty of Information Technology ja Communication Sciences, 2020.

[11] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. et al. A view of cloud computing. *Communications of the ACM* 53.4 (2010), 50–58.

[12] Microsoft. *What is Computer Vision?* URL: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home (viitattu 28. 07. 2020).

[13] Amazon Web Services, Inc. *AWS Documentation*. URL: https://docs.aws.amazon.com/ (viitattu 06. 05. 2021).

[14] Flexera. *RightScale, STATE OF THE CLOUDREPORT from Flexera. As Cloud Use Grows, Organizations Focus on Cloud Costs and Governance*. Maaliskuu 2019. URL: https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf (viitattu 04. 02. 2021).

[15] Serhane, Y., Sekkaki, A., Benzidane, K. ja Abid, M. Cost Effective Cloud Storage Interoperability Between Public Cloud Platforms. eng. *International journal of com-*

munication networks and information security 12.3 (2020), 440–449. ISSN: 2073-607X.

[16] Kritikos, K., Skrzypek, P. ja Zahid, F. Are Cloud Platforms Ready for Multi-cloud? eng. *Service-Oriented and Cloud Computing*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, 56–73. ISBN: 9783030447687.

[17] Hildred, T. *The History of Containers*. 28. elokuuta 2015. URL: https://www.redhat.com/en/blog/history-containers (viitattu 31. 07. 2020).

[18] *Chapter 14 jails*. URL: https://www.freebsd.org/doc/handbook/jails.html (viitattu 31. 07. 2020).

[19] *Welcome to Linux-Vserver.org*. URL: http://wiki.linux-vserver.org/Welcome_to_Linux-VServer.org (viitattu 31. 07. 2020).

[20] Buchanan, S., Rangama, J. ja Bellavance, N. *Introducing Azure Kubernetes Service: A Practical Guide to Container Orchestration*. eng. 1. painos. Berkeley, CA: Apress, 2020. ISBN: 9781484255186.

[21] Rice, L. *Container Security*. eng. 1. painos. O'Reilly Media, Inc, 2020. ISBN: 9781492056706.

[22] Cloud Native Computing Foundation. *Cloud Native Computing Foundation receives $9 million cloud credit grant from Google Cloud to fund Kubernetes development, empower community*. URL: https://www.cncf.io/announcements/2018/08/29/cncf-receives-9-million-cloud-credit-grant-from-google/ (viitattu 05. 02. 2021).

[23] The Kubernetes Authors. *rktnetes brings rkt container engine to Kubernetes*. URL: https://kubernetes.io/blog/2016/07/rktnetes-brings-rkt-container-engine-to-kubernetes/ (viitattu 05. 02. 2021).

[24] The Kubernetes Authors. *Kubernetes Documentation*. URL: https://kubernetes.io/docs/home/ (viitattu 16. 04. 2021).

[25] Poniszewska-Marańda, A. ja Czechowska, E. Kubernetes Cluster for Automating Software Production Environment. eng. *Sensors (Basel, Switzerland)* 21.5 (2021), 1910–. ISSN: 1424-8220.

[26] Arundel, J. ja Domingus, J. *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*. eng. Sebastopol: O'Reilly Media, Incorporated, 2019. ISBN: 1492040762.

[27] Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H. ja Kim, S. Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. eng. *Sensors (Basel, Switzerland)* 20.16 (2020), 4621–. ISSN: 1424-8220.

[28] Richardson, C. *Microservices Patterns*. eng. 1. painos. Manning Publications, 2018. ISBN: 9781617294549.

[29] Macero García, M. *Learn Microservices with Spring Boot: A Practical Approach to RESTful Services Using an Event-Driven Architecture, Cloud-Native Patterns, and Containerization*. eng. Berkeley, CA: Apress L. P, 2020. ISBN: 9781484261309.

[30] Nadareishvili, I., Mitra, R., McLarty, M. ja Amundsen, M. *Microservice Architecture*. eng. 1. painos. O'Reilly Media, Inc, 2016. ISBN: 9781491956250.

[31] Imperva, Inc. *Load Balancing Algorithms*. URL: https://www.imperva.com/learn/availability/load-balancing-algorithms/ (viitattu 22. 02. 2021).

[32] Sharma, R. *Traefik API Gateway for Microservices With Java and Python Microservices Deployed in Kubernetes*. eng. Berkeley, CA, 2021.

[33] Amazon Web Services, Inc. *Start Building on AWS Today*. URL: https://aws.amazon.com/ (viitattu 08. 04. 2021).

[34] Fuller, A. *AWS Cloud Map: Easily create and maintain custom maps of your applications*. 28. marraskuuta 2018. URL: https://aws.amazon.com/blogs/aws/aws-cloud-map-easily-create-and-maintain-custom-maps-of-your-applications/ (viitattu 01. 06. 2021).

[35] Kim, G., Humble, J., Debois, P. ja Willis, J. *The DevOps Handbook*. eng. 1. painos. IT Revolution Press, 2016. ISBN: 1942788002.

[36] Russo, B., Jaatun, M. G., Abrahamsson, P., Botterweck, G., Ghanbari, H., Kettunen, P., Mikkonen, T. J., Mjeda, A., Münch, J., Duc, A. N. ja Wang, X. Towards a Secure DevOps Approach for Cyber-Physical Systems: An Industrial Perspective. eng. *International journal of systems and software security and protection* 11.2 (2020), 38–57. ISSN: 2640-4265.

[37] Laukkarinen, T., Kuusinen, K. ja Mikkonen, T. Regulated software meets DevOps. eng. *Information and software technology* 97 (2018), 176–178. ISSN: 0950-5849.

[38] Karamitsos, I., Albarhami, S. ja Apostolopoulos, C. Applying devops practices of continuous automation for machine learning. eng. *Information (Basel)* 11.7 (2020), 1–15. ISSN: 2078-2489.

[39] Behara, S. *Designing Highly Scalable Database Architectures*. 9. huhtikuuta 2019. URL: https://www.red-gate.com/simple-talk/cloud/cloud-data/designing-highly-scalable-database-architectures/ (viitattu 07. 02. 2021).

[40] Barua, H. *Scaling Stateful Services*. URL: https://www.infoq.com/news/2015/11/scaling-stateful-services/ (viitattu 07. 02. 2021).

[41] Taibi, D. ja Lenarduzzi, V. On the Definition of Microservice Bad Smells. eng. *IEEE software* 35.3 (2018), 56–62. ISSN: 0740-7459.

[42] Flowler, M. ja Lewis, J. *Microservices*. 25. maaliskuuta 2014. URL: https://martinfowler.com/articles/microservices.html (viitattu 29. 01. 2021).

[43] Taibi, D., Lenarduzzi, V. ja Pahl, C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. eng. *IEEE cloud computing* 4.5 (2017), 22–32. ISSN: 2325-6095.

[44] Taibi, D., Spillner, J. ja Wawruch, K. Serverless Computing-Where Are We Now, and Where Are We Heading? eng. *IEEE software* 38.1 (2021), 25–31. ISSN: 0740-7459.

[45] Nupponen, J. ja Taibi, D. Serverless: What it Is, What to Do and What Not to Do. eng. *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2020, 49–50. ISBN: 9781728174150.

[46] Patrocinio, E. ja Loup, X. *Back up cloud-native applications*. URL: https://www.ibm.com/garage/method/practices/manage/backup-cloudnative/ (viitattu 04. 04. 2021).

[47] Edita Publishing Oy. *Sosiaali- ja terveysministeriön asetus potilasasiakirjoista*. URL: https://finlex.fi/fi/laki/ajantasa/2009/20090298 (viitattu 07. 04. 2021).

[48] Karslioglu, M. *Cloud Native Backups, Disaster Recovery and Migrations on Kubernetes*. 28. heinäkuuta 2020. URL: https://thenewstack.io/cloud-native-backups-disaster-recovery-and-migrations-on-kubernetes/ (viitattu 06. 04. 2021).

[49] Evans, C. *Cloud-Native Data Protection*. 9. heinäkuuta 2019. URL: https://www.architecting.it/blog/cloud-native-data-protection/ (viitattu 08. 04. 2021).

[50] Preston, W. C. *Cloud-Out vs Cloud-Native Data Protection – What's The Difference?* 26. marraskuuta 2019. URL: https://www.druva.com/blog/cloud-out-vs-cloud-native-data-protection-whats-the-difference/ (viitattu 08. 04. 2021).

[51] Marron, M. Log++ logging for a cloud-native world. eng. *SIGPLAN notices* 53.8 (2020), 25–36. ISSN: 0362-1340.

[52] Elasticsearch B.V. *Elastic Stack Main Page*. URL: https://www.elastic.co/guide (viitattu 06. 05. 2021).

[53] The Kubernetes Authors. *Kubernetes DNS-Based Service Discovery*. URL: https://github.com/kubernetes/dns/blob/master/docs/specification.md (viitattu 28. 02. 2021).

[54] The CoreDNS Authors. *Cluster DNS: CoreDNS vs Kube-DNS*. URL: https://github.com/kubernetes/dns/blob/master/docs/specification.md (viitattu 28. 02. 2021).

[55] Cloudflare, Inc. *What is the OSI model?* URL: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/ (viitattu 11. 03. 2021).

[56] Wang, S. *Introduction to Kubernetes* Networking and Acceleration with DPDK*. URL: https://01.org/kubernetes/blogs/qwang10/2019/introduction-kubernetes-networking-and-acceleration-dpdk (viitattu 14. 03. 2021).

[57] The MetalLB Authors. *MetalLB*. URL: https://metallb.universe.tf/ (viitattu 02. 03. 2021).

[58] Feynman, Z. *rename porterlb to openelb*. 6. heinäkuuta 2021. URL: https://github.com/kubesphere/openelb/commit/2b490156950fccb8cd261b2eb1b4b0f0ecb22881 (viitattu 30. 10. 2021).

[59] Dunstan, A. *Comparing Open Source k8s Load Balancers*. 14. lokakuuta 2020. URL: https://medium.com/thermokline/comparing-k8s-load-balancers-2f5c76ea8f31 (viitattu 30. 10. 2021).

[60] Luciani, A. *Utilizing MetalLB to Provide LoadBalancer Services for Nutanix Karbon*. 20. kesäkuuta 2019. URL: https://next.nutanix.com/community-blog-154/utilizing-metallb-to-provide-loadbalancer-services-for-nutanix-karbon-32966 (viitattu 30. 10. 2021).

[61] Cloud Native Computing Foundation. *CNI - the Container Network Interface*. URL: https://github.com/containernetworking/cni (viitattu 02. 03. 2021).

[62] Rawdat, A. *Update: Using Free Let's Encrypt SSL/TLS Certificates with NGINX*. 28. tammikuuta 2021. URL: https://www.nginx.com/blog/using-free-ssltls-certificates-from-lets-encrypt-with-nginx/ (viitattu 31. 10. 2021).

[63] Envoy Project Authors. *Secret discovery service (SDS)*. URL: https://www.envoyproxy.io/docs/envoy/latest/configuration/security/secret#config-secret-discovery-service (viitattu 31. 10. 2021).

[64] Caddy Project Authors. *Caddy Ingress Controller*. URL: https://github.com/caddyserver/ingress (viitattu 31. 10. 2021).

[65] Traefik Labs. *Traefic Documentation*. URL: https://doc.traefik.io/traefik/ (viitattu 18. 03. 2021).

[66] Barnes, R., Hoffman-Andrews, J., McCarney, D. ja Kasten, J. *Automatic Certificate Management Environment (ACME)*. RFC 8555. Maaliskuu 2019. DOI: 10.17487/RFC8555. URL: https://rfc-editor.org/rfc/rfc8555.txt.

[67] Internet Security Research Group. *Documentation*. URL: https://letsencrypt.org/docs (viitattu 11. 03. 2021).

[68] Internet Security Research Group. *Let's Encrypt*. URL: https://letsencrypt.org/ (viitattu 28. 06. 2021).

[69] Velero Authors. *Velero*. URL: https://velero.io/docs/v1.5/ (viitattu 12. 04. 2021).

[70] restic authors. *Restic Documentation*. URL: https://restic.readthedocs.io/en/latest/ (viitattu 12. 04. 2021).

[71] MinIO, Inc. *MinIO High Performance Object Storage*. URL: https://docs.min.io/minio/baremetal/ (viitattu 12. 04. 2021).

[72] MinIO, Inc. *MinIO Legacy Documentation*. URL: https://docs.min.io/minio/baremetal/ (viitattu 05. 05. 2021).

[73] restic authors. *Rest Server*. URL: https://github.com/restic/rest-server (viitattu 24. 04. 2021).

[74] The New Stack. *Kubernetes Back Up, Restore and Migration with Velero*. 27. marraskuuta 2019. URL: https://www.youtube.com/watch?v=71NoY5CIcQ8 (viitattu 12. 04. 2021).

[75] Abueg, R. *Elasticsearch: What it is, How it works, and what it's used for*. 7. maaliskuuta 2020. URL: https://www.knowi.com/blog/what-is-elastic-search/ (viitattu 28. 10. 2021).

[76] Shukla, P. *Learning elastic stack 7. 0 : distributed search, analytics, and visualization using elasticsearch, logstash, beats, and kibana.* eng. Birmingham ; 2019.

[77] The Fluent Bit Authors. *Fluent Bit v1.8 Documentation*. URL: https://docs.fluentbit.io/manual (viitattu 15. 08. 2021).

[78] Berman, D. *Fluentd vs. Fluent Bit: Side by Side Comparison*. 25. maaliskuuta 2020. URL: https://logz.io/blog/fluentd-vs-fluent-bit/ (viitattu 27. 10. 2021).

[79] Elasticsearch B.V. *Elastic Stack and Product Documentation*. URL: https://www.elastic.co/guide (viitattu 06. 05. 2021).

[80] SolarWinds Worldwide, LLC. *Documentation for Loggly*. URL: https://documentation.solarwinds.com/en/success_center/loggly (viitattu 28. 10. 2021).

[81] Barman, D. *InfluxDB vs. Elasticsearch for Time Series Analysis*. 23. lokakuuta 2020. URL: https://logz.io/blog/influxdb-vs-elasticsearch/ (viitattu 28. 10. 2021).

[82] InfluxData. *InfluxDB Licence*. URL: https://github.com/influxdata/influxdb/blob/master/LICENSE (viitattu 28. 10. 2021).

[83] Elasticsearch B.V. *Elasticsearch Licence*. URL: https://github.com/influxdata/influxdb/blob/master/LICENSE (viitattu 28. 10. 2021).

[84] Fedak, V. *ELK VS Loki! How to gather logs from Kubernetes cluster and effectively navigate through them*. 1. maaliskuuta 2021. URL: https://itsvit.com/blog/elk-vs-loki-how-to-gather-logs-from-kubernetes-cluster-and-effectively-navigate-through-them/ (viitattu 30. 10. 2021).

[85] Dutt, R. *Grafana, Loki, and Tempo will be relicensed to AGPLv3*. URL: https://grafana.com/blog/2021/04/20/grafana-loki-tempo-relicensing-to-agplv3/ (viitattu 30. 10. 2021).

[86] Crunchbase Inc. *Traefik Labs*. URL: https://www.crunchbase.com/organization/containous (viitattu 27. 05. 2021).

[87] Traefik Labs. *Releases*. URL: https://github.com/traefik/traefik/releases?after=v1.1.0 (viitattu 27. 05. 2021).

[88] Miller, R. *Five years after creating Traefik application proxy, open-source project hits 2B downloads*. 23. syyskuuta 2020. URL: https://techcrunch.com/2020/09/23/five-years-after-creating-traefik-application-proxy-open-source-project-hits-2b-downloads/ (viitattu 27. 05. 2021).

[89] Croes, G. *Releases*. 12. joulukuuta 2019. URL: https://traefik.io/blog/traefik-2-1-in-the-wild/ (viitattu 27. 05. 2021).

[90] Elasticsearch B.V. *Releases*. URL: https://github.com/elastic/elasticsearch/releases (viitattu 15. 08. 2021).

[91] The Fluent Bit Authors. *Fluent Bit*. URL: https://fluentbit.io/ (viitattu 15. 08. 2021).

A SUOMENNETUT TERMIT

Englanninkielinen termi	Suomennos
Container engine	Konttiteknologia
Container runtime	Konttien suoritin
Work plane	Työpinta
Liveness probe	Eloluotain
Secret	Salaisuus
Controller	Ohjain
Endpoints	Päätepisteet
Startup probe	Käynnistysluotain
Readiness probe	Valmiusluotain
Ingress	Sisääntulo
Ingress Controller	Sisääntulo-ohjain
EntryPoints	Tulo
ClusterRole	Klusterirooli
StorageClass	Säilytysluokka

Taulukko A.1. *Tätä työtä varten tehdyt suomennokset*

B VELERON TALLENTAMA KUBERNETEKSEN *PALVELU*-OLIO


```
 {
"apiVersion":"v1",
"kind":"Service",
"metadata":{
   "annotations":{
      "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"v1\",\"
   },
   "creationTimestamp":"2021-04-13T06:58:30Z",
   "labels":{
      "app":"nginx"
   },
   "managedFields":[
      {
         "apiVersion":"v1",
         "fieldsType":"FieldsV1",
         "fieldsV1":{
            "f:metadata":{
               "f:annotations":{
                  ".":{

                  },
                  "f:kubectl.kubernetes.io/last-applied-configuration":{

                  }
               },
               "f:labels":{
                  ".":{

                  },
                  "f:app":{

                  }
```

```
      }
   },
   "f:spec":{
      "f:externalTrafficPolicy":{

      },
      "f:ports":{
         ".":{

         },
         "k:{\"port\":80,\"protocol\":\"TCP\"}":{
            ".":{

            },
            "f:port":{

            },
            "f:protocol":{

            },
            "f:targetPort":{

            }
         }
      },
      "f:selector":{
         ".":{

         },
         "f:app":{

         }
      },
      "f:sessionAffinity":{

      },
      "f:type":{

      }
   }
},
"manager":"kubectl",
```

```
            "operation":"Update",
            "time":"2021-04-13T06:58:30Z"
         }
      ],
      "name":"my-nginx",
      "namespace":"nginx-example",
      "resourceVersion":"1287",
      "uid":"afd6d0f9-71b2-4739-b714-5cc71506a185"
   },
   "spec":{
      "clusterIP":"10.111.247.188",
      "clusterIPs":[
         "10.111.247.188"
      ],
      "externalTrafficPolicy":"Cluster",
      "ports":[
         {
            "nodePort":30205,
            "port":80,
            "protocol":"TCP",
            "targetPort":80
         }
      ],
      "selector":{
         "app":"nginx"
      },
      "sessionAffinity":"None",
      "type":"LoadBalancer"
   },
   "status":{
      "loadBalancer":{

      }
   }
}
```