Tampereen yliopisto

Teemu Salminen

KUBERNETES K8S JA K3S JAKELUIDEN EVALUOINTI

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Joulukuu 2022

TIIVISTELMÄ

Teemu Salminen: Kubernetes K8s ja K3s jakeluiden evaluointi
Diplomityö
Tampereen yliopisto
Tietotekniikka, DI
Tarkastajat: Professori Kari Systä, Tenure track -professori David Hästbacka
Joulukuu 2022

Konttien orkestroinnin jatkuvasti kehittyvässä maailmassa Kubernetes on noussut keskeiseksi alustaksi, jolla voidaan hallita ja automatisoida konttisovellusten käyttöönottoa, skaalautumista ja toimintaa useiden klustereiden välillä. Kubernetes voi kuitenkin osoittautua raskaaksi ja kompleksiseksi, erityisesti resurssirajoitteisissa ympäristöissä tai pienemmän mittakaavan sovelluksissa. Tästä syystä Rancher Labs kehitti K3s:n, kevyen, helposti asennettavan ja tuotantokelpoisen Kubernetes-jakelun, joka on suunniteltu yksinkertaisempaa Kubernetes-käyttöympäristöä etsiville kehittäjille.

Tässä työssä tarkastellaan kahta suosittua Kubernetes-jakelua: alkuperäistä, täysin varusteltua Kubernetesia (K8s) ja sen kevyttä vastinetta, K3s:ää. Työn tavoitteena on arvioida niiden keskeisiä näkökohtia, kuten arkkitehtuuria, asennuksen ja käyttöönoton monimutkaisuutta, resurssien käyttöä, skaalautuvuutta, suorituskykyä, tietoturvaominaisuuksia ja yhteisön tukea.

Selvityksessä nähdään, että K8s, alkuperäinen ja täysin varusteltu Kubernetes, tarjoaa erittäin laajennettavissa olevan alustan, joka voidaan räätälöidä käytännössä kaikkiin yritystason sovellusten käyttöönottotarpeisiin. Kattava valikoima ominaisuuksia, korkea skaalautuvuus ja yksityiskohtainen konfiguroitavuus tekevät siitä sopivan suurten ja monimutkaisten klustereiden hallintaan. K3s puolestaan tarjoaa tehokkaan, kompaktin ja yksinkertaistetun lähestymistavan Kubernetesiin. Se vähentää Kubernetesin monimutkaisuutta ja resurssivaatimuksia uhraamatta sen keskeisiä turvallisuus- ja hallintatoimintoja. K3s:n suoraviivainen käyttöönotto ja nopea käynnistymisaika tekevät siitä erityisen hyödyllisen tapauksissa, joissa tarvitaan nopeaa ja ketterää klusterin pystyttämistä.

Avainsanat: Kubernetes, konttien orkestrointi, konttiteknologia, pilvi, mikropalvelu.

ABSTRACT

Teemu Salminen: Evaluation of Kubernetes K8s and K3s distributions
Master's thesis
Tampere University
Information Technology, MSc
Examiners: Professor Kari Systä, Associate Professor David Hästbacka
December 2022

In the evolving world of container orchestration, Kubernetes has emerged as a critical platform for managing and automating deployments, scaling, and operations of application containers across clusters of hosts. However, Kubernetes can sometimes be overbearing, especially in resource-constrained environments or for smaller-scale applications. Recognizing this gap, Rancher Labs developed K3s, a lightweight, easy-to-install, and production-ready Kubernetes distribution. It is designed to cater to developers seeking a simplified Kubernetes experience and for applications in edge computing, Internet of Things (IoT), and Continuous Integration (CI) environments.

This thesis examines two popular distributions of Kubernetes: the original, fully-featured Kubernetes (K8s) and its lightweight counterpart, K3s. The goal of the study is to evaluate their key aspects including architecture, installation and deployment complexity, resource usage, scalability, performance, security features and community support.

The study finds that K8s, the original and fully-featured Kubernetes, offers a highly extensible platform that can be tailored to practically all enterprise-level application deployment needs. Its comprehensive range of features, high scalability, and granular configurability make it suitable for managing large and complex clusters. On the other hand, K3s offers an efficient, compact, and simplified approach to Kubernetes. It reduces the complexity and resource requirements of Kubernetes without sacrificing its core security and management functionalities. The straightforward deployment and quick startup time of K3s make it particularly useful in scenarios where rapid and agile cluster deployment is crucial.

Keywords: Kubernetes, container orchestration, containers, cloud, microservice

ALKUSANAT

Haluan esittää kiitokseni Twoday Oy:lle, joka tarjosi tämän kiinnostavan aiheen diplomityötäni varten. Olen kiitollinen siitä joustavuudesta, että sain käyttää vapaasti aikaa tämän työn tekemiseen, jonka ansiosta pystyin löytämään hyvän tasapainon työelämän ja diplomityön tekemisen välillä. Haluan myös kiittää työn ohjaajaa, professori Kari Systä, kommenteista ja ohjauksesta työn tekemisen aikana.

Tampereella, 01.12.2022

Teemu Salminen

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

API	Ohjelmointirajapinta (engl. Application Programming Interface); ohjelmointiliitäntä, jonka avulla eri sovellukset voivat kommunikoida keskenään.
CI	Jatkuva integraatio (engl. Continuous Integration); ohjelmistotuotannon menetelmä, jossa kehittäjät yhdistävät säännöllisesti lähdekoodimuutoksia yhteiseen päätietovarastoon, jossa automatisoidut toiminnot suorittavat koonnin (build) ja testauksen.
CIS	Center for Internet Security; CIS Kubernetes Benchmark on joukko suosituksia Kubernetesin konfiguroimiseksi vahvan tietoturvan varmistamiseksi.
CNCF	Cloud Native Computing Foundation; Kubernetesia ja monia muita avoimen lähdekoodin hankkeita hallinnoiva avoimen lähdekoodin ohjelmistosäätiö, joka auttaa rakentamaan kestäviä ekosysteemejä ja tukemaan yhteisöä sellaisten hankkeiden ympärille, jotka orkestroivat kontteja osana mikropalveluarkkitehtuuria.
CNI	Container Network Interface; Kehys konttien verkkoresurssien dynaamiseen konfigurointiin, joka määrittää rajapinnan verkon konfigurointia ja IP-osoitteiden provisiointia varten.
CRI	Container Runtime Interface; Konttien ajoalustan rajapinta, jolla mahdollistetaan Kubelet-komponentille riippumattomuus konttien ajoalustasta.
DNS	Domain Name System; Nimipalvelujärjestelmä, joka muuntaa verkkotunnukset (domain name) numeerisiksi IP-osoitteiksi.
IaaS	Infrastructure as a service; pilvipalvelumalli, jossa vuokrataan suoraan laitteistoresursseja.
IoT	Esineiden internet (engl. Internet of Things); yhdistettyjen älylaitteiden verkko.
K8s	Kubernetes; avoimen lähdekoodin konttien orkestrointialusta.
K3s	Sertifioitu Kubernetesin kevyempi jakelu.
OCI	Open Container Initiative; joukko standardeja, joiden tarkoitus on yhtenäistää konttien pakkaus ja konttien ajoalustat.
PaaS	Platform as a service; pilvipalvelumalli, jossa vuokrataan jotakin alustaa sovellusten kehittämistä ja käyttöönottoa varten.
SaaS	Software as a service; pilvipalvelumalli, jossa vuokrataan suoraan ohjelmistosovelluksia internetin kautta.
VM	Virtual Machine; virtuaalikone on laitteistovirtualisointitekniikka, jolla emuloidaan kokonaista tietokonejärjestelmää.

1. JOHDANTO

Pilvilaskenta on muuttanut tapaa, jolla ohjelmistotuotannon alan yritykset kehittävät ohjelmistojaan ja tarjoavat niitä asiakkailleen. Pilvilaskennan yleistymisen myötä, ohjelmistoja ei enää tarvitse asentaa paikallisesti tietokoneille, vaan käyttäjät voivat käyttää näitä palveluja suoraan "pilvestä" eli asiakkaan ja palvelimen välisestä kerroksesta, joka tekee asiakaspuolen isännöinnistä tarpeetonta ja näin ollen myös takaen ohjelmiston alustariippumattomuuden. Sen sijaan, että organisaatiot hankkisivat fyysisiä palvelemia sovellustensa suorittamista varten, ne yhä useammin hankkivat datakeskuksesta provisioidun virtuaalikoneen. Tätä sovellusten ja resurssien jakamismallia internetin välityksellä kutsutaan pilvipalveluksi. Yksi sen tärkeimmistä eduista on joustavuus ja skaalautuvuus, jossa sovelluksilla on kyky varata ja vapauttaa resursseja tarpeen mukaan. Jotta sovelluksia voidaan käyttää vaihtelevan kuormituksen palvelemiseen, ne on usein pystytettävä ja skaalattava useille eri palvelimille.

Pilvessä suoritettavat ohjelmistosovellukset voivat hyödyntää pilvilaskennan etuja noudattamalla mikropalveluarkkitehtuuria. Sen sijaan, että ohjelmistosovellus toimisi yhtenä suurena monoliittisena prosessina, mikropalveluarkkitehtuurisessa lähestymistavassa sovellus jaetaan pienempiin itsenäisiin prosesseihin, joissa kullakin osalla on oma vastuualueensa, ja jotka kommunikoivat keskenään tarkasti määriteltyjen ohjelmointirajapintojen (API) kautta. Yksittäisen käyttäjän pyynnön palvelemiseksi mikropalveluihin perustuva sovellus voi kutsua monia sisäisiä mikropalveluja vastauksen kokoamiseksi. Sen sijaan, että koko sovellus otettaisiin uudelleen käyttöön sen arkkitehtuurin muuttuessa, mikropalveluja voidaan ottaa käyttöön ja kehittää itsenäisesti, koska ne toimivat itsenäisenä prosessina. Mikropalveluarkkitehtuuri mahdollistaa myös sovelluksen jokaisen komponentin skaalaamisen erikseen, mikä johtaa tehokkaampaan resurssien käyttöön.

Mikropalveluiden ajaminen fyysisellä bare metal -palvelimella yhden käyttöjärjestelmän alaisuudessa ei ole houkutteleva vaihtoehto, koska riskinä on, että kirjastojen ja sovelluskomponenttien versiot voivat olla ristiriidassa keskenään, puhumattakaan siitä, että yhden mikropalvelun vikaantuminen saattaa vaikuttaa muiden saatavuuteen. Yksi ilmeinen vaihtoehto on jakaa fyysinen palvelin useisiin virtuaalipalvelimiin (eli virtuaalikoneisiin), jolloin yhdellä palvelimella voi olla useita eristettyjä suoritusympäristöjä. Virtualisointi on kypsä ja vakiintunut teknologia, useimmat yritykset ovat jo investoineet virtuaaliseen infrastruktuuriin, ja useimmat pilvipalveluntarjoajat käyttävät virtuaalikoneita IaaS-tarjontansa (infrastructure-as-a-service) perustana. Suuren monoliittisen sovelluksen isännöinti voidaan toteuttaa erillisessä virtuaalikoneessa, mutta mikropalvelupohjaisen sovelluksen tapauksessa, joka koostuu useista pienistä itsenäisistä komponenteista, on tehokkaampaa ajaa niitä konteissa. Kontit (engl. container) ovat kevyempi vaihtoehto virtuaalikoneille, ja ne on tehnyt tunnetuksi Docker – yritys, josta on tullut synonyymi sanalle kontti.

Konttivirtualisoinnista on viime vuosien aikana tullut suosittu tekniikka, jolla sovelluksia paketoidaan ja ajetaan pilviympäristöissä. Nykyään useat teollisuudenalat ovat omaksuneet konttien käsitteen virtuaalikoneiden sijaan. Tämä johtuu konttien alhaisesta resurssien yleiskustannuksesta, siirrettävyydestä ja nopeista käynnistymisajoista verrattuna perinteisiin virtuaalikoneisiin [1]. Konttien avulla yksittäiset sovellukset ja niiden riippuvuudet voidaan paketoida yhteen ajoympäristöön, eristäen ajettavat sovellukset alla olevasta käyttöjärjestelmästä.

Mikropalveluiden kasvaessa, konttien manuaalisesta käynnistämisestä ja hallinnasta voi tulla hyvin työlästä, koska kontit, joissa mikropalveluita suoritetaan toimivat itsenäisesti erillisinä kokonaisuuksina, eivätkä ole tietoisia toisistaan. Konttiorkestraattorit tarjoavat automatisoidun hallinnan konttisovelluksille, erityisesti silloin, kun useita kontteja suoritetaan useilla eri palvelimilla. Konttien orkestrointityökalulla voidaan automatisoida konttien käyttöönotto, hallinta, viestintä ja vikasietoisuus. Se voi olla erityisen hyödyllinen monimutkaisten mikropalvelupohjaisten sovellusten käyttöönotossa ja hallinnassa niiden koko elinkaaren ajan. Vaikka orkestrointityökaluja on useita, Kubernetes (tunnetaan myös yleisesti ni-

mellä K8s) on ylivoimaisesti suosituin alusta konttipohjaisten sovellusten orkestrointiin. Kubernetes on avoimen lähdekoodiin perustuva konttiorkestrointialusta, joka muodostaa viestintälinjan konttien välille ja automatisoi konttipohjaisten sovellusten (mikropalveluiden) käyttöönoton, hallinnan ja skaalauksen [2].

Kubernetes voi kuitenkin osoittautua liian ylimitoitetuksi ja raskaaksi, varsinkin kun otetaan huomioon paikallinen kehitys ja yksinkertaiset sovellukset. Kubernetes vaatii toimiakseen huomattavan määrän resursseja, mikä voi merkittävästi rajoittaa sen hyödyntämistä ympäristöissä, joissa laskentaresurssien määrä on hyvin rajallinen. Näin ollen Kubernetesin toiminnan takaamiseksi rajallisten resurssien oloissa on luotu kevyitä kubernetes-distribuutioita, kuten MicroK8s ja K3s. Niiden tavoitteena on yksinkertaistaa käyttöönoton konfigurointia, käynnistystä ja ylläpitoa. Kevyet Kubernetes-distribuutiot ovat kasvavassa suosiossa paikallisessa kehityksessä, edge/IoT-konttien hallinnassa ja yksinkertaisten mikropalvelusovellusten käyttöönotoissa.

1.1 Työn tavoitteet ja rajaus

Tässä diplomityössä esitellään perinteinen Kubernetes (K8s) ja sen kevyempi K3s jakelu. Esittely suoritetaan ensin olemassa olevan kirjallisuuden ja teknisen dokumentaation pohjalta ja lopussa jakeluja vertaillaan eri evaluointikriteerien avulla. Tämän diplomityön tavoitteena on selvittää K3s version käyttöönoton sopivuutta K8s version sijasta. Tavoitteena ei ole ainoastaan vertailla perinteisen ja kevyen Kubernetes-jakeluiden eroavaisuuksia, vaan tutustua myös tarkemmin Kubernetekseen ja siihen, miten Kubernetesin kanssa työskentelevä organisaatio voisi hyötyä K3s käyttöönotosta.

1.2 Työn rakenne

Työn loppuosa on jaoteltu seuraavasti. Työn toisessa osiossa keskitytään taustoittamaan työtä avaamalla lukijalle pilviohjelmiston teoriaa, jossa tutustutaan pilvilaskennan ja mikropalveluiden käsitteisiin. Kolmannessa osiossa käydään läpi konttiteknologiat. Osiossa esitetään konttiteknologioiden yleiskuvaus ja niiden

ero perinteisistä virtuaalikoneista sekä niiden hyödyt. Lukujen 2 ja 3 tarkoitus on antaa valmiudet luvussa 4 läpi käytäville asioille. Luvussa 4 esitellään Kubernetes yleisesti, jakeluiden arkkitehtuurit ja tärkeimmät komponentit, sekä niiden hyödyt ja haasteet. Luvussa 5 tehdään vertailu perinteisen Kuberneteksen ja K3s välillä niiden dokumentaation ja olemassa olevien tutkimuksien perusteella. Lopuksi esitetään lyhyt yhteenveto, jossa vedetään yhteen koko työ, vertailun tulokset ja esitetään valintakriteeristö.

2. PILVIOHJELMISTOT

Pilvilaskenta on ollut yksi eniten muutosta aikaansaavista teknologioista viimeisen vuosikymmenen aikana. Vaikka pilvilaskenta sai alkunsa jo ennen vuotta 2010, teknologia sai sen käyttöönoton helpottumisen myötä vauhtia 2010-luvun alussa ja puolivälissä. Tämän hetken suosituimpien pilviteknologioiden Dockerin (2013) ja Kubernetesin (2014) julkaisujen jälkeen, siirtyminen pilvipalveluihin on helpottunut ja pilviteknologioihin liittyvät epävarmuustekijät ovat vähentyneet merkittävästi. Nykyisin yhä suurempi osa maailman tietojenkäsittelystä suoritetaan pilvipalvelujen kautta.

Tämän luvun ensimmäisessä osiossa käsitellään pilvilaskennan määritelmää ja ominaisuuspiirteitä sekä sen eri toteutustapoja ja palvelumalleja. Luvun toisessa osiossa esitetään mikropalveluarkkitehtuuri, sen erot perinteiseen monoliittiarkkitehtuuriin, sen tuomat edut, sekä miten mikropalveluarkkitehtuuri voi auttaa pilvilaskennan etujen hyödyntämisessä.

2.1 Pilvilaskenta

Pilvilaskenta on tietojenkäsittelymalli, jossa palvelimet, verkot, tallennustilat, kehitystyökalut ja sovellukset ovat käytettävissä internetin kautta. Yleisesti käytetty määritelmä pilvilaskennan palveluille on peräisin Yhdysvaltain standardisointi- ja teknologiainstituutti (National Institute of Standards and Technology, NIST) -organisaatiolta. Määritelmässä luetellaan viisi pilvipalvelun toisiaan täydentävää ominaisuuspiirrettä, jotka ovat: itsepalvelullisuus, päätelaiteriippumattomuus, resurssien yhteiskäyttö, nopea joustavuus, ja tarkka resurssien käyttö ja valvonta [3, 4].

1) Pilviresurssien itsepalvelullisuudella, edellyttämättä ihmisen välistä vuorovaikutusta palveluntarjoajan kanssa, pilvipalveluidenkäyttäjät voivat varata pilviresursseja käyttöönsä milloin tahansa tarpeen mukaisesti. Tämä mahdollistaa hel-

pon skaalautuvuuden, mikä puolestaan johtaa kustannussäästöihin. Koska ihmisten välistä vuorovaikutusta ei tarvita, resurssien skaalaus on mahdollista automatisoida [3, 4].

2) Päätelaiteriippumattomuudella tarkoitetaan pilvipalveluresurssien saatavuutta verkoissa standardiprotokollien avulla. Tämä minimoi sijaintirajoitukset ja vähentää erikoistuneiden laitteiden tai ohjelmistojen tarvetta palveluja käytettäessä, mahdollistaen palveluiden saatavuuden erilaisilla laitteilla, kuten tietokoneilla, älypuhelimilla ja tableteilla. Kattavien verkkoyhteyksien ansiosta palvelut ovat helpommin erilaisten käyttäjien saatavilla [3, 4].

3) Resurssien yhteiskäytön ansiosta pilvipalvelujen tarjoajat voivat hallita ja suorittaa palveluja tehokkaasti. Pilvipalvelut ovat tyypillisesti monivuokralaispalveluita, joissa laitteistoresurssit, kuten laskentateho, tallennustila ja verkon kaistanleveys jaetaan useiden eri käyttäjien kesken siten, että kunkin käyttäjän resurssit on eristetty muista käyttäjistä [3, 4]. Resurssien yhteiskäyttö ja vuokraus toteutetaan virtualisointitekniikoiden avulla, joita käsitellään tarkemmin luvussa 3. Näitä yhteiskäyttöisiä resursseja jaetaan dynaamisesti käyttäjille kysynnän mukaisesti.

4) Nopea joustavuus on virtualisointiin ja automaatioon perustuva ominaisuus, joka on myös sidoksissa kustannussäästöihin. Jouston ansiosta pilvipalvelun kuluttaja voi nopeasti skaalata pilviresursseja horisontaalisesti muuttuvan kysynnän mukaan. Horisontaalisella skaalautumisella tarkoitetaan mahdollisuutta lisätä ja vähentää saman resurssin instanssien määrää. Joustavuus mahdollistaa esimerkiksi sen, että liikennehuipun aikana voidaan automaattisesti varata lisää verkkopalvelimia, jotta lisääntynyt kuormitus voidaan käsitellä [3, 4].

5) Pilvilaskennassa resurssien käyttöä mitataan, ja palvelun kustannukset suhteutetaan käytön mukaan. Sen sijaan, että pilvipalvelun käyttäjä joutuisi varautumaan etukäteen laskentaresurssien käyttötarpeeseen, hän voi reaaliaikaisesti varata ja vapauttaa resursseja tarpeen mukaan. Tällöin pilvipalveluresurssien käyttökustannukset ovat suhteessa niiden käyttöaikaan, jolloin käyttäjät maksavat ainoastaan käyttämistään resursseista. Palveluntarjoajat voivat käyttää käyt-

töasteen mittauksia myös sisäiseen palveluoptimointiin ja -valvontaan. Raportointi- ja seurantaominaisuuksia voidaan tarjota sekä pilvipalveluntarjoajalle että niiden käyttäjille [3, 4].

Edellä esitettyjen keskeisien ominaisuuksien lisäksi NIST on pilvipalvelujen määritelmässä esittänyt neljä erilaista pilvi-infrastruktuurin toteutustapaa ja kolme erilaista pilvipalvelumallia, joita tarkastellaan seuraavissa osioissa.

2.1.1 Pilvipalvelun toteutustavat

Pilven toteutustavat edustavat tietynlaista pilviympäristöä, jotka eroavat toisistaan omistuksen, koon ja käyttöoikeuden perusteella. NIST on määritellyt neljä pilvipalvelun toteutustapaa: julkinen, yksityinen, hybridi ja yhteisöpilvi. Toteutustapojen erot perustuvat pääasiassa siihen, kuka on pilviresurssien omistaja ja kenelle resursseja tarjotaan.

Julkiset pilvipalvelut ovat kolmannen osapuolen pilvipalveluntarjoajien omistamia ja ylläpitämiä, ja ne ovat yleisesti saatavilla internetin kautta. Yksi julkisen pilven merkittävistä eduista on mahdollisuus skaalata palvelua nopeasti kysynnän mukaan. Julkisia pilvipalveluja käytetään yleensä sovelluksissa, jotka eivät ole arkaluonteisia tai säänneltyjä, ja jotka vaativat vaihtelevan määrän resursseja. Julkisen pilven käyttäjät voivat olla joko organisaatioita tai yksittäisiä henkilöitä [3, 4]. Esimerkkejä nykyisistä julkisista pilvipalveluntarjoajista ovat Amazon AWS, Microsoft Azure ja Google Cloud Platform.

Yksityiset pilvipalvelut ovat yhden organisaation omistuksessa ja käytössä, eivätkä ne ole suuren yleisön saatavilla, ja ovat yleensä käytettävissä sisäisen verkon kautta Internetin sijaan. Yksityisiä pilvipalveluja käytetään tyypillisesti sovelluksissa, jotka edellyttävät korkeaa tietoturvaa ja tiettyjen vaatimusten noudattamista. Yksityisten pilvien etuna on parempi hallinta ja turvallisuus, sillä organisaatio voi mukauttaa infrastruktuurin vastaamaan omia tarpeitaan. Yksityisiä pilvipalveluja voidaan rakentaa ja hallinnoida organisaation sisällä tai ne voidaan ulkois-

taa ulkopuoliselle palveluntarjoajalle [3, 4]. Tyypillinen esimerkki yksityisestä pilvestä on suuri yritys, joka tarjoaa pilvipalveluja pienemmille organisaatioille ja tiimeille yrityksen sisällä.

Hybridipilvet ovat julkisten ja yksityisten pilvien yhdistelmä, jossa osa sovelluksista ja työkuormista suoritetaan julkisessa pilvessä ja osa yksityisessä pilvessä. Hybridipilvet tarjoavat sekä julkisten että yksityisten pilvien edut, jolloin organisaatiot voivat hyödyntää julkisen pilven mittakaavaetuja ja joustavuutta, ja samalla säilyttäen yksityisen pilven hallinnan ja tietoturvan [3, 4].

Yhteisöpilvi infrastruktuuri on tyypillisesti varattu yksinomaan tietyn käyttäjäyhteisön käyttöön organisaatioiden välillä, jotka jakavat yhteiset intressit (esim. tietoturvavaatimukset, toimintaperiaatteet ja vaatimustenmukaisuutta koskevat näkökohdat) [3, 4]. Esimerkkejä yhteisöllisten pilvien käyttäjistä ovat mm. yliopistot, jotka tekevät yhteistyötä tietyillä tutkimusaloilla, tai poliisilaitokset, jotka käyttävät yhteisiä resursseja. Pääsy yhteisön pilviympäristöön on yleensä rajoitettu yhteisön jäsenille, joiden välillä on enemmän luottamusta kuin julkisessa pilvessä, jossa samaa ympäristöä voi käyttää kuka tahansa.

2.1.2 Palvelumallit

Pilvilaskennan monet paradigmat voidaan jakaa kolmeen eri palvelumalliluokitukseen. NIST-instituutin pilvilaskennan määritelmässä nämä pilvipalvelumallit ovat: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) ja SaaS (Software as a Service). Pilvipalvelumalli kuvaa palveluntarjoajan tarjoamien palvelujen abstraktiotasoa. Korkeamman tason palvelumallit sisältävät alemman tason mallit tai ovat riippuvaisia niistä.

IaaS on pilvipalvelumalli, jossa käyttäjät saavat käyttöönsä tietotekniikan infrastruktuuria, kuten palvelimia, tallennustilaa ja verkkolaitteita, joita voidaan käyttää ja määrittää verkkoportaalin tai API:n kautta. Käyttäjät voivat vuokrata näitä resursseja pay-as-you-go -periaatteella sen sijaan, että heidän tarvitsisi ostaa ja ylläpitää omaa fyysistä infrastruktuuria. IaaS-palvelussa palveluntarjoaja vastaa

konesaleista, fyysisistä palvelimista ja virtualisointikerroksesta, kun taas käyttäjä vastaa virtuaalipalvelimista ja käyttöjärjestelmistä sekä kaikista palveluun liittyvistä toiminnoista [3, 4]. Pohjimmiltaan IaaS edustaa pilvipalvelun alinta abstraktiotasoa, jota teknologiayritykset käyttävät saadakseen käyttöönsä raakaa laskentatehoa omien IT-järjestelmien ja ohjelmistojensa ajamiseen. IaaS tarjoaa näistä kolmesta eniten joustavuutta ja hyvän siirrettävyyden, mutta käyttäjä on vastuussa infrastruktuurin päällä ajettavien alustojen ja sovellusten ylläpidosta.

PaaS pilvipalvelumalli tarjoaa käyttäjille alustan sovellusten kehittämistä, testausta ja käyttöönottoa varten. PaaS-palveluntarjoajat tarjoavat tyypillisesti erilaisia työkaluja ja palveluja, kuten tietokantoja, kehitysympäristöjä ja ajoympäristöjä, joita käyttäjät voivat käyttää sovellustensa kehittämiseen ja ajamiseen. PaaS-palveluntarjoajat huolehtivat taustalla olevasta infrastruktuurista, kuten palvelimista ja tallennustiloista, jolloin käyttäjät voivat keskittyä sovellustensa kehittämiseen ja käyttöönottoon [3, 4]. Toisin kuin IaaS:ssa, käyttäjällä ei ole suoraa hallintaa infrastruktuurin päällä olevista alustoista, jolloin sovellukset on kehitettävä tukemaan palveluntarjoajan tarjoamaa alustaa, heikentäen sen siirrettävyyttä.

SaaS-palvelu tarjoaa käyttäjille pääsyn ohjelmistosovelluksiin internetin kautta. SaaS-palveluntarjoajat tyypillisesti isännöivät ja ylläpitävät ohjelmistoja ja antavat ne käyttäjien käyttöön verkkoselaimen tai sovellusohjelmointirajapinnan (API) kautta [3, 4]. Käyttäjiltä peritään yleensä tilausmaksu ohjelmiston käytöstä sen sijaan, että heidän tarvitsisi ostaa ja ylläpitää omia kopioita ohjelmistosta.

Sovelluksen vaihtelevan kysyntä tarpeen täyttäminen ja siten pilvipalvelujen tehokas hyödyntäminen asettaa tiettyjä vaatimuksia sovelluskehitykselle. Tärkeintä on, että sovelluksen horisontaalinen skaalautuminen onnistuisi vaivattomasti ja automaattisesti. Tämä vaatimus voidaan täyttää mikropalveluarkkitehtuurilla, jota käsitellään seuraavassa osiossa.

2.2 Mikropalvelut

Mikropalvelut ovat arkkitehtuurinen ja organisatorinen lähestymistapa pilvinatiiviseen ohjelmistokehitykseen [5]. Nimensä mukaisesti mikropalveluarkkitehtuuri jakaa sovelluksen pienempiin, itsenäisiin palveluihin, jotka yhdistettynä muodostavat kokonaisen sovelluskokonaisuuden. Mikropalvelut liitetään usein pilvilaskentaan ja pilvinatiivi -termiin, koska ne soveltuvat pilvilaskennan hyötyjen täysimääräiseen hyödyntämiseen.

Perinteisissä monoliittisissa arkkitehtuureissa kaikki prosessit ovat kytketty tiukasti toisiinsa ja ne toimivat yhtenä kokonaisena palveluna. Näin ollen, jos jonkin sovelluksen prosessin kysyntä kasvaa, koko arkkitehtuuri on uudelleen skaalattava [6]. Myös ominaisuuksien lisääminen tai muuttaminen monimutkaistuu koodikannan kasvaessa, joka vaikeuttaa uusien ideoiden toteuttamista. Monoliittiset arkkitehtuurit lisäävät riskiä sovelluksen saatavuudelle, koska monet toisistaan riippuvaiset ja tiukasti kytketyt prosessit lisäävät yksittäisen prosessin vikaantumisen mahdollisuutta.

Mikropalveluarkkitehtuurissa sovellus rakennetaan itsenäisistä komponenteista, jotka suorittavat kutakin sovellusprosessia palveluna. Kukin näistä palveluista vastaa erillisestä tehtävästä ja kommunikoi muiden palveluiden kanssa viestipohjaisilla protokollilla kuten HTTP. Ne ovat hajautettuja ja löyhästi kytkettyjä, jolloin yhden mikropalvelun päivittäminen tai vikaantuminen ei estä muiden mikropalveluiden saatavuutta. Lisäksi, jos jonkin palvelun instanssi vikaantuu, palvelusta voidaan automaattisesti luoda muita instansseja, jotka ottavat vikaantuneen palvelun vastuut. Jokaisella mikropalvelulla on oma elinkaarensa, ja sitä voidaan hallita ja skaalata itsenäisesti. Näin ollen, kutakin palvelua voidaan skaalata itsenäisesti sekä horisontaalisesti että vertikaalisesti resurssien optimaalisen käytön saavuttamiseksi. Usein käytetyn mikropalvelun useampia instansseja voidaan ajaa samanaikaisesti, kun taas harvemmin käytetty mikropalvelu saattaa tarvita vain yhden instanssin. Tätä horisontaalista skaalautumista voidaan hallita automaattisesti kysynnän ja palvelun kuormituksen mukaan. Tämä voi tehostaa re-

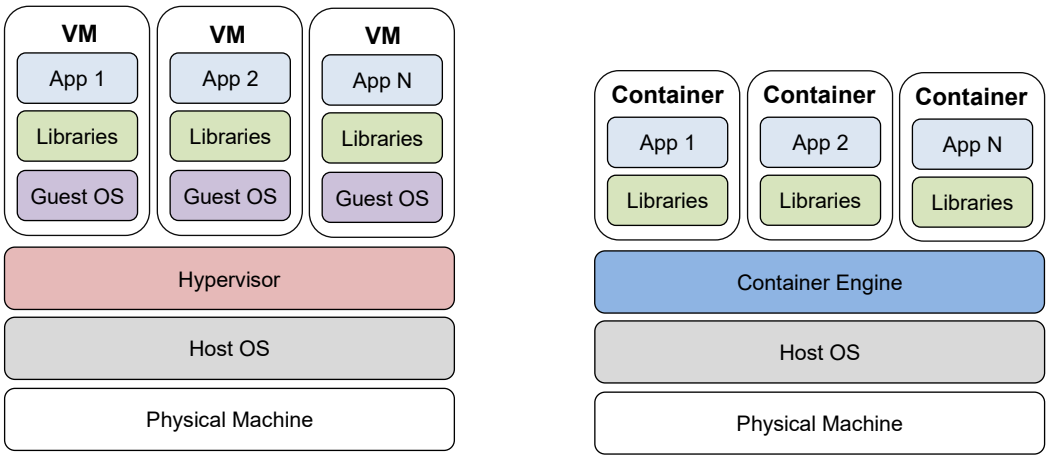
surssien optimaalista käyttöä, vähentäen sekä pullonkauloja että resurssien ylijakoa. Mikropalveluiden hyödyt koostuvat pääasiassa niiden riippumattomuudesta, joka helpottaa niiden kehittämistä, käyttöönottoa ja skaalausta [6][7].

Koska mikropalvelut ovat itsenäisiä, niitä on mahdollista ajaa omissa eristetyissä ympäristöissä, joka tekee palveluista suojatumpia ja helpommin hallittavia [6]. Palvelujen ajaminen erillisissä ympäristöissä parantaa myös palveluiden siirrettävyyttä ja suoraviivaistaa skaalausta ja käyttöönottoa. Palveluiden eristys voidaan toteuttaa virtualisoinnilla, mutta perinteisen virtualisoinnin resurssitarve on usein tarpeettoman suuri pienien mikropalveluiden ajamiseen. Konttivirtualisoinnit tarjoavat tähän optimaalisen ratkaisun, sillä ne tarjoavat eristettyjä suoritusympäristöjä pienemmällä resurssi vaatimuksilla ja paremmalla kustannustehokkuudella.

3. KONTTITEKNOLOGIAT

Konttivirtualisoinnista on tullut pilvipalvelujen yleistymisen myötä yleistynyt teknologia ohjelmistojen kehityksessä ja käyttöönotossa. Konttien tavoitteena on pääasiassa tarjota parempi tapa kehittää, paketoida ja käyttöönottaa ohjelmistoja erilaisissa ympäristöissä ennakoitavalla ja helposti hallittavissa olevalla tavalla. Konttiteknologiat tarjoavat standardoidun ympäristön ohjelmistojen ajamiselle [8].

Konttivirtualisointi on käyttöjärjestelmätason virtualisointia, jossa sovelluksia ajetaan eristetyissä ympäristöissä – konteissa – saman käyttöjärjestelmän alaisuudessa. Yksi konttivirtualisoinnin tärkeimmistä eduista on sen kevyt luonne. Toisin kuin perinteiset virtuaalikoneet (VM), jotka sisältävät kokonaisen käyttöjärjestelmän ja kaikki sen päällä pyörivät ohjelmistot, kontit jakavat isännän käyttöjärjestelmän ja virtualisointi tapahtuu ytimen (kernel) tasolla. Tämä mahdollistaa useiden erillisten sovellusten tehokkaan suorittamisen yhdellä palvelimella ilman useiden käyttöjärjestelmien käyttämisestä aiheutuvia resurssien ylikuormitusta [6]. Koska käyttöjärjestelmää ei tarvitse käynnistää konttien käynnistämisen yhteydessä, ne ovat huomattavasti nopeampi käynnistää ja pysäyttää kuin perinteiset virtuaalikoneet, joten ne sopivat hyvin sovelluksiin, joiden on skaalauduttava nopeasti [8]. Perinteisten virtuaalikoneiden ja konttien välistä eroa havainnollistetaan kuvassa 1.



Kuva 1. *Virtuaalikoneilla (vasen) on omat käyttöjärjestelmänsä, kun taas kontit jakavat isäntäkäyttöjärjestelmän palvelut Container Engine:n kautta [1][9][10].*

Kontit paketoidaan kontti-imageiksi, jotka sisältävät sovellusten ohella myös kaikki niiden riippuvuudet, kuten ohjelmointi kirjastot ja kehykset. Virtuaalikoneiden tavoin, sovellukset toimitetaan omien kirjasto- ja kehysversioiden kanssa, ja useat samalla palvelimella toimivat kontit voivat sisältää ja käyttää eri kirjastoversioita ilman ristiriitoja muiden konttisovellusten kanssa. Kontti-image on käyttövalmis ohjelmistopaketti, joka sisältää vain olennaiset komponentit, mitä tarvitaan sovelluksen suorittamiseen. Tämä tekee konteista erittäin omavaraisia ja siirrettäviä [8]. Koska kaikki tarvittavat kirjastot ja ohjelmistot toimitetaan sovelluskontissa, niitä voidaan helposti siirtää eri ympäristöjen välillä, esimerkiksi kehityksestä tuotantoon. Näin sovellusten käyttöönottoa ja hallintaa on helppo automatisoida, mikä voi auttaa vähentämään tuotantoympäristön ylläpitoon tarvittavaa aikaa ja vaivaa.

Konttimoottori (Container engine) on ohjelmistoalusta, jonka avulla isäntäkäyttöjärjestelmä voi toimia konttien isäntänä. Konttimoottorin tehtävänä on ottaa vastaan käyttäjän CLI komentoja konttien luomiseksi, käynnistämiseksi ja hallitsemiseksi, ja se tarjoaa myös API:n, jonka avulla ulkoiset ohjelmat, kuten kontti orkestrointialustat, voivat tehdä vastaavia pyyntöjä. Konttimoottorin toiminnallisuuden keskeisimmät toiminnot suorittaa kuitenkin sen ydinkomponentti, jota kutsutaan kontin ajoalustaksi (Container runtime), joka suorittaa kontin asennuksen ja toimii yhdessä isäntäkäyttöjärjestelmän ytimen kanssa konttiprosessin käynnistämiseksi ja ylläpitämiseksi. Tyypillisiä konttimoottoreita ovat Docker, CRI-O, Containerd [11][12].

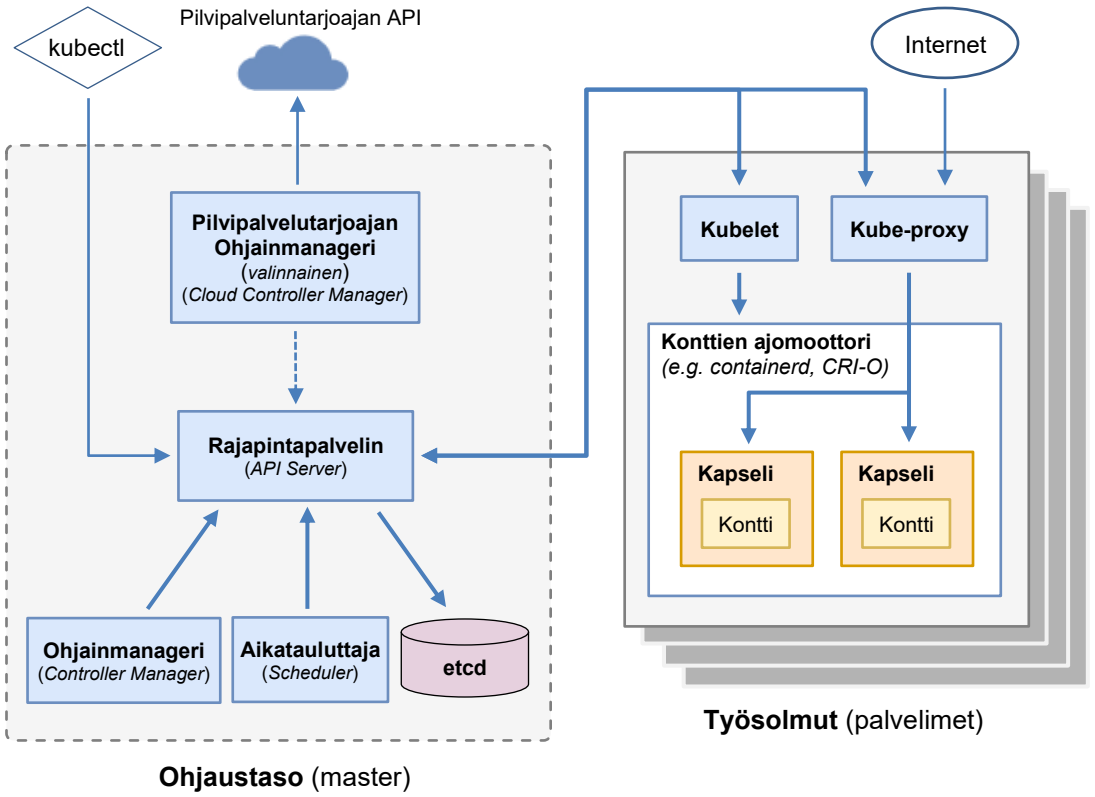
Konttivirtualisointi on kustannustehokas menetelmä sovellusten paketoimiseen ja ajamiseen. Suurten konttimäärien käyttäminen aiheuttaa kuitenkin merkittäviä hallinnointikustannuksia. Erityisesti tuotantoympäristössä on hallittava sovelluksia pyörittäviä kontteja ja varmistettava, ettei käyttökatkoksia pääse syntymään. Esimerkiksi jos jokin kontti kaatuu, toisen kontin on käynnistyttävä. Konttijärjestelmien hallinnan helpottamiseksi on kehitetty erilaisia konttien orkestrointialustoja. Yksi näistä alustoista on Kubernetes, jota tarkastellaan seuraavaksi.

4. KUBERNETES

Kubernetes on avoimen lähdekoodin konttien orkestrointijärjestelmä, jolla automatisoidaan konttipohjaisten sovellusten käyttöönottoa, skaalausta ja hallintaa. Nimi Kubernetes on peräisin kreikan kielestä, joka tarkoittaa ruorimiestä, henkilöä, joka ohjaa laivaa, kuten konttialusta. Sen kehitti alun perin Google, ja projekti julkaistiin avoimena lähdekoodina vuonna 2014, ja sitä ylläpitää nykyään Cloud Native Computing Foundation (CNCF). Kubernetes yhdistää Googlen yli 15 vuoden kokemuksen suuren mittakaavan tuotantotyökuormien ajamisesta ja avoimen lähdekoodi yhteisön parhaat ideat ja käytännöt [9]. Kubernetes tarjoaa tehokkaan ja joustavan alustan konttien hallintaan ja skaalaamiseen, ja sen suosio on kasvanut viime vuosina, kun organisaatiot ovat alkaneet siirtyä mikropalvelupohjaisiin arkkitehtuureihin.

4.1 K8s

Korkean tason Kubernetes-arkkitehtuuri koostuu keskusyksiköstä, jota kutsutaan pääsolmuksi (Master Node) tai ohjaustasoksi (Control Plane), ja yhdestä tai useammasta työntekijäsolmusta (Worker Nodes). Ohjaustaso toimii Kubernetes-klusterin orkestroijana, ja se vastaa klusterin tilan hallinnasta sekä konttien käyttöönoton ja skaalauksen koordinoinnista. Ohjaustasoa käytetään myös tarvittavien tietojen arkistona sovelluksen tilan ylläpitämiseen ja palauttamiseen työntekijäsolmujen vikaantuessa, ja se voidaan replikoida ja hajauttaa korkean saatavuuden ja vikasietoisuuden varmistamiseksi. Ohjaustason replikaatiot ovat toistensa kopiota ja pitävät sisällään samat tiedot. Ohjaustasossa ja työntekijäsolmuissa toimivat komponentit on esitetty kuvassa 2.



Kuva 2. *Korkean tason Kubernetes klusterin arkkitehtuuri ja sen komponentit [13][14].*

4.1.1 Ohjaustaso

Ohjaustaso vastaa klusterin työsolmujen ja niissä ajettavien konttisovellusten yleisestä hallinnasta ja koordinoinnista sekä klusterin tilan valvonnasta. Kaikki komponenttien väliset toiminnot ja viestintä sekä ulkoiset käyttäjäkomennot tapahtuu ohjaustason kautta Kubernetesin API:n avulla. Käyttäjät voivat olla vuorovaikutuksessa klusterin ohjaustason kanssa Kubernetesin komentorivi-konfigurointityökalun (*kubectl*) avulla. Ohjaustaso koostuu toisiinsa löyhästi yhteydessä olevista komponenteista, jotka toimivat yhdessä varmistaakseen, että klusterin tila vastaa käyttäjän odotuksia. Nämä ohjaustason komponentit ovat: *Rajapintapalvelin* (API Server), *Aikatauluttaja* (Scheduler), *Ohjainmanageri* (Controller Manager), *Pilvipalvelutarjoajan ohjainmanageri* (Cloud Control Manager) ja *etcd-tietovarasto [14]*.

Rajapintapalvelin on ohjaustason keskeisin komponentti, joka toimii klusterin frontendina, ja jonka kautta käyttäjät ovat vuorovaikutuksessa Kubernetes-klusterinsa kanssa kubectl-komentorivityökalun avulla. API-palvelin toteuttaa RESTful API:n HTTP:n kautta, validoi ja käsittelee kaikki API-operaatiot sekä vastaa klusterin tilan tallentamisesta etcd:hen ja sen päivittämisestä tarpeen mukaan. Pohjimmiltaan API-palvelin on käyttöliittymä, jota käytetään Kubernetes-klusterien hallintaan, luomiseen ja konfigurointiin. Sen avulla käyttäjät, ulkoiset komponentit ja klusterin osat kommunikoivat keskenään. Se on myös ainoa Kubernetes-komponentti, joka on yhteydessä etcd:hen, kaikkien muiden komponenttien on tehtävä pyyntö API-palvelimelle voidakseen työskennellä klusterin tilan kanssa. API-palvelin toteuttaa myös tarkkailuominaisuuden, jonka avulla komponentit voivat tarkkailla etcd:hen tehtyjä muutoksia. Kun jotain etcd avainarvoa muutetaan, sen "tarkkailijoille" ilmoitetaan asiasta. Tämä mahdollistaa sen, että komponentit, kuten Aikatauluttaja ja Ohjainmanageri, voivat olla vuorovaikutuksessa API-palvelimen kanssa löyhästi kytketyllä tavalla. API-palvelin on suunniteltu skaalautumaan horisontaalisesti – eli lisäämällä instanssien määrää, jolloin liikennettä voidaan tasapainottaa eri instanssien välillä [14][15].

etcd on avoimen lähdekoodin hyvin vikasietoinen ja hajautettu avainarvotietokanta, jota Kubernetesin ohjaustaso käyttää klusterin hallintaan tarvittavien tilatietojen tallentamiseen. Etcd:n tallennetaan tietoa klusterin konfiguraatiosta ja tilasta sekä tietoa yksittäisistä konteista, palveluista ja muista klusterin objekteista. Etcd-tietovarasto tarjoaa yhtenäisen ja johdonmukaisen totuuden lähteen klusterin tilasta, kuten kaikista sen kapseleista (Pod) ja niiden sisältämistä konteista, minä tahansa ajankohtana. Nimi etcd tulee Linuxin hakemistorakenteen nimeämiskäytännöstä, jossa kaikki yksittäisen järjestelmän konfigurointitiedostot ovat kansiossa nimeltä "/etc", ja "d" tulee sanasta "distributed". Näin ollen etcd-tietovaraston suunnittelun perustana on ollut hajautetun etc-hakemiston tuottaminen [14][16].

Aikatauluttaja vastaa luotujen konttien jakamisesta sopiviin työläissolmuihin niiden tarvitsemien resurssien ja työläissolmuissa käytettävissä olevien resurssien perusteella. Tarkemmin sanottuna se jakaa kapselit, jotka edustavat yhden tai

useamman samassa suoritusympäristössä toimivan kontin ryhmää. Aikatauluttaja ottaa huomioon resurssivaatimukset, laitteisto- ja ohjelmistorajoitukset, yhteenkuuluvuus määrittelyt (affinity), datan sijainnin, työkuormien väliset häiriöt ja elinkaaren ennen kapselin solmun määrittämistä. Aikatauluttaja käyttää erilaisia algoritmeja ja heuristiikkoja näiden päätösten tekemiseen ja sen varmistamiseen, että kontit otetaan käyttöön tavalla, joka optimoi resurssien käytön ja minimoi käyttökatkokset [14][17].

Ohjainmanageri pyörittää klusterin hallintaprosesseja, jotka ovat vastuussa klusterin toivotun tilan (desired state) ylläpitämisestä. Se on käytännössä valvontasilmukka, joka tarkkailee etcd:hen tallennettujen työkuormia ajavien palvelimien tilaa API–palvelimen kautta ja tekee tarvittavat muutokset, jos nykyinen tila (current state) on poikennut toivotusta tilasta. Ohjainmanageri koostuu käytännössä useasta erillisestä ohjausprosessista, mutta monimutkaisuuden vähentämiseksi, kaikki ohjaimet on niputettu yhdeksi binääriksi, jota ajetaan yhtenä prosessina. Näihin ohjausprosesseihin lukeutuu mm. node-controller, joka vastaa solmujen kaatumisen havaitsemisesta ja siihen reagoimisesta, sekä replikaohjain, joka varmistaa, että tietty määrä kapselireplikaatioita on kerrallaan käynnissä [14][18].

Pilvipalvelutarjoajien ohjainmanageri on plugin-tyyppinen komponentti, joka upottaa pilvikohtaisen ohjauslogiikan ohjaustasoon. Tämä mahdollistaa klusterin yhdistämisen pilvipalveluntarjoajan API:in, ja palvelutarjoajakohtaisten komponenttien erottamisen klusterin sisäisistä komponenteista. Pilviohjain irrottaa osan Kuberneteksen ohjainmanagerin toiminnoista ja ajaa sitä erillisenä prosessina. Erityisesti se irrottaa ohjainmanagerista ne ohjaimet, jotka ovat pilvipalvelusta riippuvaisia. Pilviohjain ajaa pelkästään pilvipalveluntarjoajakohtaisia ohjaimia, jotka hallitsevat ulkoisia pilviresursseja, kuten virtuaalikoneita, tallennustiloja ja kuormantasaajia. Esimerkiksi, kun ohjainmanagerin pitää tarkistaa taustalla olevan pilvipalveluntarjoajan infrastruktuurista onko työläissolmu terminoitu onnistuneesti, tai asettaa reititykset, tallennustilat tai kuormantasaajat, niin ne hoidetaan pilven ohjainmanagerilla. Ohjainmanagerin tavoin, pilviohjainmanageri yhdistää useita loogisesti riippumattomia valvontasilmukoita yhdeksi binääriksi, jota ajetaan yhtenä prosessina [14][19].

4.1.2 Työsolmut

Työntekijäsolmut isännöivät klusterissa ajettavia työkuormia, eli Kubernetesissa suoritettavia konttisovelluksia. Niitä ohjaa ensisijaisesti ohjaustaso, mutta joitakin komentoja voidaan suorittaa manuaalisesti. Solmut ovat joko virtuaalikoneita tai fyysisiä palvelimia, jotka suorittavat sovelluksen suorittamiseen konfiguroituja kontteja. Solmun komponentteja ovat: *kubelet*, *konttien ajomoottori* ja *kube-välityspalvelin* (kube-proxy).

Kubelet on työläissolmun agentti, joka sijaitsee ohjaustason API–palvelimen ja konttien ajomoottorin välissä, ja jonka tehtävänä on hallinnoida kapselissa olevia kontteja ja varmistaa, että ne ovat käynnissä ja toimintakuntoisia. Kubelet tarkkailee API–palvelinta uusien työtehtävien varalta ja kun se havaitsee sille osoitetun uuden kapselin, se hakee API–palvelimelta kuvaukset kapselin toivotusta tilasta (PodSpecs) ja delegoi kapselissa olevien konttien luonnin konttiajomoottorille. Konttien luonnin jälkeen, kubelet valvoo käynnissä olevan kapselin tilaa konttien ajomoottorin kautta ja raportoi mahdolliset tila muutokset takaisin API–palvelimelle. Kubelet hallinnoi ainoastaan Kubernetesin luomia kontteja [14][20].

Kube-välityspalvelin toimii klusterin jokaisella työläissolmulla, ja se toteuttaa osan Kubernetesin palvelukonseptista (service concept). Se toimii verkon välityspalvelimena ja kuormantasaajana solmussa käynnissä oleville kapseleille. Välityspalvelin ylläpitää solmun verkkomäärityksiä ja -sääntöjä, jotka mahdollistavat verkkoviestinnän kapseleihin klusterin sisäisistä tai ulkopuolisista verkkoistunnoista [14]. Verkon pitämiseksi ajan tasalla, välityspalvelin tarkkailee API-palvelinta kapseli- ja palvelumuutosten varalta. Välityspalvelin käytännössä julkaisee kapselissa pyörivän sovelluksen klusterin ulkopuolelle.

Jokaisessa työläissolmussa ajetaan *konttimoottoria*, kuten containerd tai CRI-O:ta, joka vastaa konttisovellusten käynnistämisestä, ajamisesta ja terminoinnista ohjaustason ohjeiden mukaisesti. Tähän on olemassa useita mahdollisia ajomoottoreita, mutta Kubernetesin 1.24 version jälkeen, ajomoottorin on sovellettava Open Container Initiative (OCI) projektin standardoimaa Container Run-

time Interface (CRI) -rajapintaa. Ennen CRI:n käyttöönottoa, Kubernetes integroitui tiiviisti Dockerin kanssa, joka toimi oletusarvoisena konttien ajomoottorina. Tämä tarkoitti sitä, että kaikki Dockerin API muutokset vaativat myös päivityksiä Kubernetesiin. CRI:n käyttöönoton myötä Kubernetesin kubelet -komponentti voi kommunikoida minkä tahansa CRI-määrittelyn toteuttavan konttimoottorin kanssa, mikä helpottaa työläissolmun konttien ajomoottorin vaihtamista [21][22]. Kuten aiemmin mainittiin, kubelet on suoraan vuorovaikutuksessa konttien ajomoottorin kanssa konttien elinkaaritoimintojen hallitsemiseksi.

4.1.3 Kapselit ja niiden hallinta

Kapselit ovat Kubernetes-arkkitehtuurin pienimpiä aikataulutettavia yksiköitä. Kapseli koostuu yhdestä tai useammasta tiukasti toisiinsa kytketystä kontista ja niiden konfiguraatiosta, metatiedoista, sekä konttien yhteisistä verkko- ja varastoresursseista. Kapselin avulla sen sisältämät kontit voivat toimia ikään kuin ne toimisivat samassa eristetyssä virtuaalikoneessa, jossa ne jakavat kapselille osoitetun IP-osoitteen ja verkkoportit, sekä tallennustilat. Tällä tavoin voidaan ottaa käyttöön useita saman sovelluksen instansseja tai eri sovellusten eri instansseja samassa solmussa tai eri solmuissa ilman, että niiden konfiguraatiota tarvitsisi muuttaa. Kukin kapseli on tarkoitettu suorittamaan yhtä instanssia tietystä sovelluksesta [23][24].

Klusterin kapseleiden haluttua toimintatapaa ja tilaa voidaan hallita *Deployment* -manifestin avulla. Deployment on Kubernetes-resurssiobjekti, joka kuvaa tietyn sovelluskomponentin halutun tilan kapselimallina [24]. Deployment manifestissa (YAML-muotoinen konfiguraatiotiedosto) kuvataan kapselin sisältämien konttien kuvaukset, laitteistorajoitukset (esim. RAM-muistin käyttörajoitus), korkean saavutettavuuden edellyttämien kapseli replikaatioiden määrän (*ReplicaSet*), käytettävät verkkoportit, asetettavat ympäristömuuttujat, mahdolliset tallennustilat ja useita muita vapaaehtoisia asetuskenttiä. Manifesti on deklaratiivinen, eli siinä kuvataan lopullinen tila, jonka järjestelmän tulisi saavuttaa, sen sijasta, että ilmoitettaisiin miten nykyistä tilaa, tulisi muuttaa (imperatiivinen). Järjestelmänvalvojat määrittelevät Deployment-manifestin avulla mallin siitä, miten sovelluksen tulisi

toimia Kubernetes-klusterissa, ja Kubernetesin tehtävänä on suorittaa kaikki tarvittavat toimenpiteet sovelluksen kapseleiden toivotun tilan saavuttamiseksi hallitulla tavalla, sekä toivotun tilan ylläpitämiseksi.

Tätä Kubernetes-komponenttien keskinäistä toimintaa toivotun tilan saavuttamiseksi voidaan havainnollistaa seuraavan esimerkin kautta, jossa klusterin ylläpitäjä lisää uuden Deployment-objektin Kuberneteksen komentorivityökalun *kubectl* kautta:

1. **API-palvelimen validointi**: API-palvelin suorittaa ensin lähetetylle Deployment-manifestille (.yaml) perustarkistukset, kuten skeeman oikeanmukaisuuden. API-palvelin tarkistaa myös onko käyttäjällä tarvittavat oikeudet Deploymentin luomiseen.
2. **Etcd Varastointi**: Valtuutus ja validointi tarkastuksien läpäistyä API-palvelin tallentaa Deployment-objektin klusterin hajautettuun avainarvosäiliöön ja palauttaa vastauksen kubectl:lle.
3. **Replikaatiojoukon luonti**: Ohjainmanagerin ohjaimet, kuten Deployment-ohjain, tarkkailevat jatkuvasti API-palvelinta hallinnoimiensa resurssien muutosten varalta. Deployment-ohjain havaitsee uuden Deployment-objektin etcd:ssä ja luo replikaatiojoukon (ReplicaSet) Deployment-manifestin määrittelyn mukaisesti. Replikaatiojoukkoa käytetään vakaan kapseli-replikaatio joukon ylläpitämisessä [25]. Tämä resurssi lähetetään API-palvelimelle, joka tallentaa sen etcd-tietovarastoon.
4. **Kapselien resurssien luonti**: Edellisen vaiheen tavoin kaikki tarkkailijat saavat ilmoituksen niitä koskevasta API-palvelimessa tehdystä muutoksesta, joista tällä kertaa Ohjainmanagerin Replikaatio-ohjain havaitsee äskettäin luodun replikaatiojoukon. Replikaatio-ohjain vastaa replikaatiojoukkojen elinkaaren hallinnasta. Se huolehtii siitä, että kapseli replikaatioiden todellinen tila vastaa replikaatiojoukko-objektin toivottua tilaa. Replikaatio-ohjain luo kapselimallista toivotun määrän replikaatioita (yksi tai useampi) replikaatiojoukko-objektin mukaisesti ja lähettää luodut kapseliresurssit API-palvelimelle, joka tallentaa ne etcd-tietovarastoon.
5. **Kapselien aikataulutus**: Etcd:ssä on nyt kaikki tarvittavat tiedot konkreettisien kapselien luomiseen, lukuun ottamatta sitä, millä solmulla kapseleita

pitäisi ajaa. Aikatauluttaja tarkkailee kapseliresursseja, joille ei ole vielä määritetty solmua, ja aloittaa kaikkien solmujen suodattamisen ja luokittelun, jotta se voi valita parhaan solmun, jossa kapselia voidaan ajaa. Kun solmu on valittu, tieto lisätään kapselin määrittelyyn, ja lähetetään takaisin API-palvelimelle, joka päivittää tiedon etcd-tietovarastoon.

6. **Kapselien alustaminen**: Aikatauluttajan valitsemassa solmussa toimiva Kubelet saa ilmoituksen sen solmuun sidottujen kapselien muutoksesta ja delegoi saamansa kapselimäärityksessä (PodSpecs) olevien konttien luonnin konttiajomoottorille. Konttien ajomoottori alustaa kapselit käynnistämällä kapselimäärittelyssä määritellyt kontit. Tähän sisältyy mm. konttikuvien lataus konttirekisteristä, verkkoasetusten määrittäminen, tallennustilojen liittäminen ja mahdollisten alustuskomentojen suorittaminen. Konttimoottorin luomat kontit ovat etcd:ssä kuvattujen kapseleiden konkreettisia ilmentymiä.
7. **Kapselien tilan päivitys**: Kun kapselit ovat toimintakunnossa, Kubelet raportoi niiden tilan API-palvelimelle, joka päivittää kapselien metatiedot ja tilan etcd:hen. Tämä sisältää tietoja, kuten kapselin IP-osoite, olosuhteet ja valmiustilan.
8. **Seuranta ja terveystarkastukset**: Kubelet valvoo kapselien kuntoa ja suorittaa terveystarkastuksia ja raportoi mahdolliset tila muutokset takaisin API-palvelimelle. Jos kapseli ei läpäise tarkastuksia, se voidaan käynnistää uudelleen tai korvata määritetyn uudelleenkäynnistyskäytännön mukaisesti [26].

4.1.4 Hyödyt ja haasteet

Kubernetes on lyhyessä ajassa noussut yhdeksi nopeimmin kasvavaksi avoimen lähdekoodin projektiksi ja yleisimmin käytetyksi konttien orkestrointialustaksi [27]. Syynä Kubernetesin laajalle levinneelle suosiolle voidaan pitää muun muassa sen seuraavia etuja:

- Johdonmukaisuus
- Skaalautuvuus
- Siirrettävyys

- Häiriösietoisuus
- Resurssien optimointi

Kubernetesin muuttumattomat (immutable) resurssit mahdollistavat johdonmukaisen käyttöönoton koko kehitysprosessin läpi. Kubernetesin muuttumattomuus saavutetaan konttien ja deklaratiivisen konfiguroinnin avulla. Muuttumattomissa järjestelmissä artefakteja ei voida muuttaa niiden luomisen jälkeen. Kontit ilmentävät muuttumattoman infrastruktuurin periaatetta niputtamalla yhteen kaikki sovelluksen suorittamiseen tarvittavat riippuvuudet ja määritykset. Muuttumattomuus ulottuu klusterissa toimivien konttien lisäksi myös tapaan, jolla konttisovellukset on kuvattu Kubernetesille. Jokainen Kubernetesin resurssi on deklaratiivinen konfigurointiobjekti, joka edustaa järjestelmän toivottua tilaa. Kubernetesin tehtävänä on varmistaa, että todellinen tila vastaa tätä toivottua tilaa. Koska kontit ja deklaratiiviset konfiguraatiotiedostot ovat muuttumattomia, kaikkien niihin tehtävien päivitysten on kuljettava versionhallintajärjestelmän kautta. Tämä mahdollistaa muutosten jäljitettävyyden ja muutoksen peruuttamisen viallisen julkaisun sattuessa. Koska kontit luodaan aina samalla tavalla, kontti voidaan poistaa ja käynnistää uudelleen milloin tahansa vaarantamatta järjestelmän luotettavuutta. Johdonmukaisuuden ansiosta kehittäjät voivat käyttää vähemmän aikaa vianetsintään ja enemmän aikaa uusien ominaisuuksien kehittämiseen ja käyttöönottoon [28].

Kubernetes-klusterin palveluiden horisontaalinen skaalaus on hyvin suoraviivaista sen muuttumattoman ja deklaratiivisen arkkitehtuurin ansiosta. Kubernetesin avulla sovelluksesta voidaan helposti luoda useita replikaatioita, ja Kubernetes tasapainottaa automaattisesti kuormituksen niiden välillä. Näin sovelluksia on helppo skaalata niiden kysynnän mukaisesti. Tietyn palvelun instanssien lukumäärä määritellään numerona deklaratiivisessa konfiguraatiossa, ja numeron muuttamisen jälkeen Kubernetes vapauttaa tai varaa tarvittavat resurssit klusterista. Jos klusterissa ei ole riittävästi resursseja uusien instanssien allokointiin, instanssien luonti jää odottamaan, kunnes resursseja on riittävästi. Myös itse klusterin skaalaus on suoraviivaista, koska klusterin solmut ovat identtisiä keskenään. Kubernetes tarjoaa myös ominaisuuksia, kuten kapselien horisontaalisen

autoskaalauksen (Horizontal Pod Autoscaling), jonka avulla kapselien replikaatioiden määrää voidaan skaalata automaattisesti eri indikaattorien perusteella, kuten suorittimen tai muistin käyttöasteen perusteella [29].

Kubernetes parantaa myös sovellusten saatavuutta tarjoamalla ominaisuuksia, kuten klusterin jatkuvan tilaseurannan, konttien automaattisen uudelleenkäynnistyksen ja versio päivitysten asteittaisen käyttöönoton. Kubernetes on itsestään korjaantuva järjestelmä, joka seuraa jatkuvasti klusterin todellista tilaa ja ryhtyy toimenpiteisiin, jos se havaitsee poikkeamia toivotusta tilasta. Tämä tarkoittaa sitä, että Kubernetes ei ainoastaan alusta klusteria, vaan se myös aktiivisesti vartioi sitä kaikilta vioilta tai häiriöiltä, jotka saattavat horjuttaa klusterin vakautta ja vaikuttaa sen luotettavuuteen [28]. Jos kapselit tai kontit vikaantuvat tai lakkaavat toimimasta, Kubernetes pyrkii automaattisesti palauttamaan klusterin toivotun tilan luomalla konttisovellukset automaattisesti uudelleen toivottujen määritysten mukaisesti. Työläissolmun vikaantuessa, Kubernetes pyrkii välttämään käyttökatkokset sijoittamalla kaikki kapselit automaattisesti klusterin terveisiin työläissolmuihin, kunnes ongelma on korjattu [23].

Kubernetesin dynaamisesta ja älykkäästä konttien hallinnasta voi syntyä myös konkreettista taloudellista hyötyä. Kubernetesin avulla kapselissa oleville konteille voidaan määrittää minimiresurssivaatimukset, jotka ne tarvitsevat aina, kuten suorittimen ja muistin tarve, sekä resurssikiintiön, jonka avulla voidaan rajoittaa kontin käyttämien resurssien määrää. Kubernetesin aikatauluttaja jakaa automaattisesti kapselit klusterin työläissolmuille näiden resurssien varaustietojen perusteella, jolloin klusterin suorituskyky ja tehokkuus saadaan optimoitua [17]. Koska resurssien allokointi mukautuu automaattisesti sovelluksen reaaliaikaisiin tarpeisiin, Kubernetes maksimoi resurssien tehokkaan hyödyntämisen, hallitsee infrastruktuurikustannuksia ja ratkaisee kysyntä- ja skaalautuvuushaasteet. Automaattiset ominaisuudet, kuten automaattinen skaalautuminen ja itsekorjautuvuus, minimoivat myös manuaalisiin järjestelmänvalvonta tehtäviin kuluvaa aikaa [28].

Kubernetes tarjoaa myös standardoidun tavan sovellusten käyttöönottoon ja hallintaan, jolloin sovelluksia on helppo siirtää eri ympäristöjen ja pilvipalveluntarjoajien välillä. Kubernetes voi toimia minkä tahansa CRI-määrittelyn toteuttavan konttien ajomoottorin kanssa, ja se on myös helposti siirrettävissä paikalliseen, julkiseen tai yksityisessä pilvessä sijaitsevaan infrastruktuuriin. Tämä voi auttaa vähentämään palveluntarjoajiin sitoutumista ja lisätä joustavuutta sovellusten käyttöönotossa ja hallinnassa.

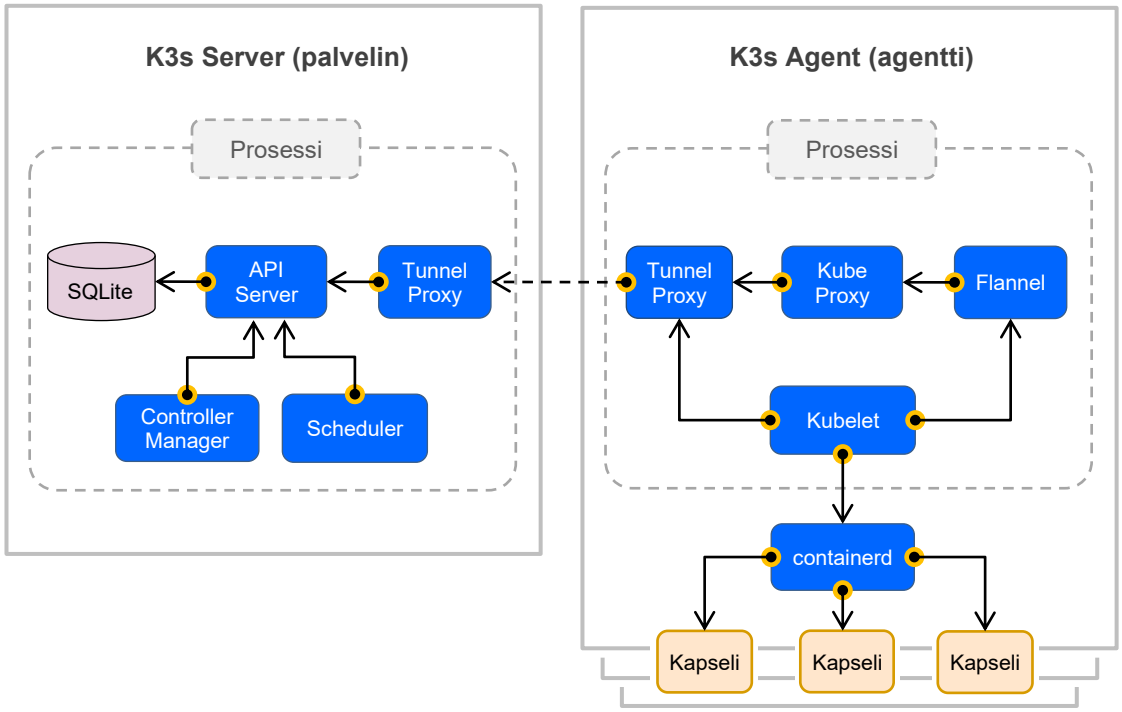
Vaikka Kubernetes (K8s) on tehokas alusta konttisovellusten käyttöönoton, skaalauksen ja hallinnan automatisointiin, sillä on myös joitakin mahdollisia puutteita ja rajoitteita. Yksi Kubernetesin käytön suurimmista haasteista on sen monimutkaisuus. Alusta sisältää suuren määrän komponentteja, ja sitä voi olla vaikea ymmärtää ja konfiguroida, erityisesti organisaatioille, jotka ovat uusia konttien ja orkestroinnin suhteen. Erityisesti tuotantokäyttöön tarkoitetun Kubernetes-ympäristön perustaminen ja hallinta vaatii paljon asiantuntemusta ja resursseja, joka voi vaikeuttaa organisaatioiden Kubernetesin käyttöönoton aloittamista.

Kubernetesin toinen haittapuoli on, että se voi olla hyvin resurssi-intensiivinen. Alusta vaatii huomattavan määrän laskenta- ja muistiresursseja toimiakseen tehokkaasti, mikä voi vaikuttaa Kubernetes-klusterin suorituskykyyn ja kustannuksiin. Kubernetes toimii hyvin monimutkaisten arkkitehtuurien kanssa, mutta pienemmissä sovelluksissa Kubernetesin käyttö voi tuottaa enemmän kustannuksia kuin säästöjä, jonka vuoksi se on usein ylimitoitettu ratkaisu monille pienemmille organisaatioille.

4.2 K3s

K3s on Rancher Labs-yhtiön (nykyisin osa SUSEa) kehittämä avoimeen lähdekoodiin perustuva kevennetty Kubernetes-jakelu, jonka tavoitteena on yksinkertaistaa Kubernetes-käyttöönottoja. Se on CNCF:n (Cloud Native Computing Foundation) sertifioima Kubernetes-yhteensopiva jakelu, mikä tarkoittaa, että se on täysin yhteensopiva perinteisen Kubernetesin API:n ja työkalujen kanssa, joten se on helppo integroida olemassa oleviin Kubernetes-ympäristöihin [30] [31].

K3s tarjoaa monia samoja etuja kuin perinteinen Kubernetes, vähentäen samalla resurssien tarvetta ja kompleksisuutta. K3s on pakattu yhdeksi binääriksi, joka on helppo asentaa ja konfiguroida, vähentäen riippuvuuksia ja vaiheita, joita tarvitaan tuotantokelpoisen Kubernetes-klusterin asentamiseen ja ajamiseen. Binaarin koko on julkaisusta riippuen 50–100 Mt, ja se sisältää kaikki tarvittavat komponentit täyden Kubernetesin API:n implementoimiseksi, ilman ulkoisia riippuvuuksia. Pienen binäärikoon saavuttamiseksi Kubernetesista on poistettu ydintoiminnallisuuteen vaikuttamattomat toiminnot, kuten pilvipalveluntarjoaja- ja varastointiliitännäiset, jotka ovat riippuvaisia kolmannen osapuolen pilvi- tai datakeskusteknologioista/-palveluista, joita ei välttämättä tarvita paikallisissa (On-Premise) käyttötapauksissa. Tästä huolimatta K3s voidaan kuitenkin integroida pilvipalveluntarjoajan, kuten AWS:n tai GCP:n, kanssa liitännäisten avulla [30][32]. Näistä muutoksista huolimatta on hyvä huomioida, että K3s ei ole Kubernetes projektin haarautuma (fork), sillä se ei muuta Kubernetesin ydintoiminnallisuutta ja se pyrkii pysymään mahdollisimman lähellä alkuperäistä (upstream) Kubernetesia [33].



Kuva 3. *Usean solmun K3s-klusterin arkkitehtuuri, jossa on yksittäinen K3s master-palvelin upotetulla SQLite-tietokannalla [30].*

K3s:n arkkitehtuuri koostuu palvelin prosessista ja klusteriin liitettävistä agentti prosesseista, jotka vastaavat K8s:n ohjaustasoa ja työläissolmuja. Palvelinprosessi ajaa Kubernetesin ohjaustason peruskomponentteja (Rajapintapalvelinta, Ohjainmanageria ja Aikatauluttajaa) sekä sisäänrakennettua SQLite-tietokantaa klusterin tilan tallentamista varten. Agenttiprosessi toimii jokaisessa työläissolmussa ja rekisteröi itsensä palvelimelle käänteisen tunnelivälityspalvelimen (Tunnel Proxy) avulla. Sekä palvelimessa että agentissa kaikki komponentit on niputettu yhtenäiseksi prosessiksi, mikä tekee siitä todella kevyen toisin kuin K8s:ssä, jossa jokainen komponentti toimii erillisenä prosessina.

Palvelinta ja agenttia on mahdollista ajaa yhden solmun kokoonpanossa. Tämä tarkoittaa, että palvelinsolmu voi toimia samanaikaisesti agenttisolmuna, joka huolehtii ohjaustason (palvelin) tehtävien lisäksi myös agentti prosessin tehtävistä (Kubelet, Flannel, Kube proxy), sekä varsinaisten työkuormien ajamisesta. Tämä mahdollistaa klusterin entistä nopeamman käynnistämisen, koska sekä palvelin että agentti voivat toimia yhtenä prosessina samassa solmussa. Yhden solmun K3s-klusteri on erityisen hyödyllinen pienissä projekteissa, kehitysympäristöissä, testauksessa tai jopa tuotantokäytössä, joissa sovelluksen vaatimukset ovat vaatimattomat eikä korkea saatavuus ole ensisijainen prioriteetti [32].

K3s käyttää kevyttä ja suorituskykyistä *containerd* konttiajoalustaa oletusarvoisena CRI:nä kapselien elinkaaren hallintaan. Se on CNCF ylläpitämä avoimen lähdekoodin projekti, joka tarjoaa tehokkaan, johdonmukaisen ja luotettavan alustan konttien ajamiseen tuotantoympäristöissä. Koska K3s erottaa agentti prosessin ja työkuormat toisistaan, ylläpitäjät voivat pysäyttää ja käynnistää K3s:n vaikuttamatta käynnissä oleviin konttisovelluksiin. Tämän ansiosta K3s on helppo päivittää vaihtamalla binääri ja käynnistämällä prosessi uudelleen tai uudelleen konfiguroida muuttamalla käynnistystiedoston lippuja [34]. Kubernetesin ydinkomponenttien lisäksi K3s sisältää käyttöönoton ja toimintamallin yksinkertaistamiseksi myös muita yhteisöltä hankittuja komponentteja, kuten *Flannel*, *Tunnel Proxy* ja *SQLite.*

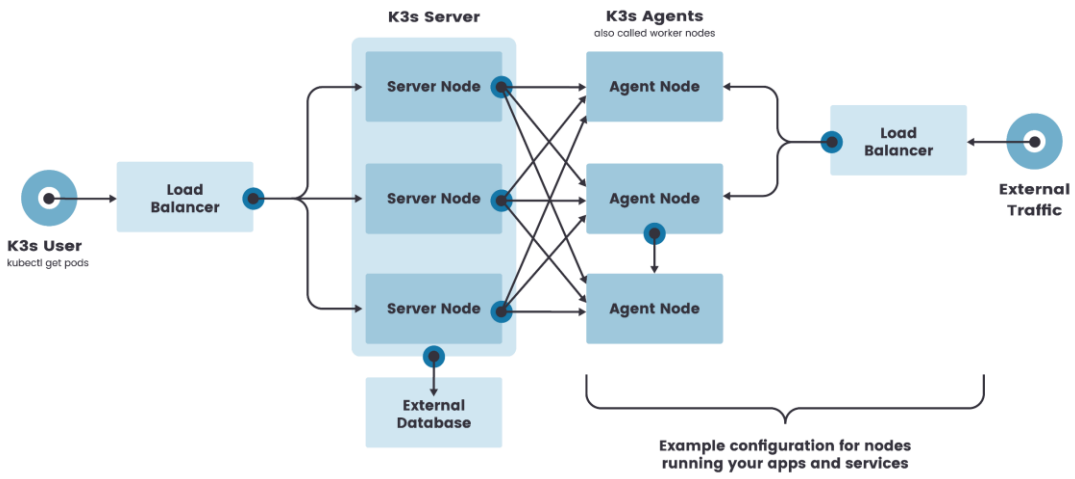
Flannel on avoimen lähdekoodin virtuaaliverkkoprojekti, joka tarjoaa yksinkertaisen ja helpon tavan konfiguroida 3. kerroksen IPv4-verkon klusterin solmujen välille. Se toteuttaa Kubernetes Container Network Interface (CNI) -standardin, joka määrittää kehyksen konttien verkkoresurssien dynaamiseen konfigurointiin. Kun uusi kapseli luodaan solmulle, kubelet kutsuu konfiguroitua CNI-liitännäistä kapselin verkon määrittämiseksi. K3s käyttää oletusarvoisesti Flannelia CNI:nä [30], kun taas K8s jättää verkkoliitännän valinnan ja konfiguroinnin käyttäjän tehtäväksi. Flannel määrittää kullekin solmulle aliverkon, joka toimii solmulla toimivien kapselien IP-osoitepoolina. Tästä aliverkosta jaetaan kullekin luodulle kapselille yksilöllinen IP-osoite. Kapselit voivat kommunikoida keskenään tämän IP-osoitteen avulla riippumatta siitä, millä solmussa ne toimivat. Flannelin integrointi K3s:ään tarjoaa käyttövalmiin ja helppokäyttöisen verkkoliitännän, minkä ansiosta kompleksisia verkkoasetuksia ei tarvitse määrittää manuaalisesti [35].

Tunnel Proxy tarjoaa suojatun WebSocket -viestintäkanavan palvelimen ja agenttisolmujen välille [30]. Sen ensisijainen tarkoitus on yksinkertaistaa solmujen välisten yhteyksien muodostamista. Tunnel Proxy -komponentti toimii asiakas-palvelin-mallilla, jossa agentit ylläpitävät asiakkaina pysyvää yhteyttä palvelimeen. Tämän mallin ansiosta palvelin voi tehokkaasti kommunikoida minkä tahansa agentin kanssa ilman, että sen tarvitsee tietää agentin sijaintia. Yleensä Kube Proxy käyttää useita portteja saadakseen yhteyden API-palvelimeen. Mutta K3s:n tapauksessa Kube Proxy saa yhteyden API-palvelimeen Tunnel Proxyn avulla, joka luo yksisuuntaisen yhteyden API-palvelimeen. Kun yhteys on muodostettu, otetaan käyttöön kaksisuuntainen viestintä, joka muodostaa turvallisemman yhteyden käyttämällä yksittäistä porttia kommunikointiin. Tämän jälkeen palvelin voi kommunikoida agentin kanssa tämän tunnelin kautta, lähettää sille ohjeita ja vastaanottaa tilapäivityksiä.

Näiden lisäysten lisäksi yksi keskeinen ero on tapa, jolla klusterin tilaa hallitaan. Perinteinen Kubernetes (K8s) käyttää koko klusterin tilan tallentamisessa etcd:tä – hajautettua avain-arvotietokantaa. K3s korvaa etcd:n oletusarvoisesti kevyellä SQLite-tietokannalla, joka on yleisesti käytetty tietokanta sulautetuissa käyttöympäristöissä. Se on palvelimeton, itsenäinen ja konfigurointivapaa transaktionaalinen SQL-tietokantajärjestelmä. Toisin kuin perinteiset SQL-tietokannat, SQLite

ei vaadi erillistä palvelinprosessia, vaan se mahdollistaa suoran pääsyn isäntäkoneelle tallennettavaan tietokantaan [30][36].

Yksittäisistä palvelimista koostuvat klusterit soveltuvat monenlaisiin käyttötapauksiin. Kuitenkin ympäristöissä, joissa Kubernetesin ohjaustason saatavuus on kriittinen, K3s:sta voidaan ajaa korkean saatavuuden (HA) kokoonpanossa. HA-K3s-klusteri koostuu kahdesta tai useammasta palvelinsolmusta, jotka palvelevat Kubernetes API:ta ja ylläpitävät agenttisolmuja sekä ulkoista tietovarastoa [30]. K8s:n ohjaustason korkea saatavuus voidaan saavuttaa käyttämällä etcd:tä vähintään kolmessa master-solmussa. SQLite ei kuitenkaan ole hajautettu tietokanta, jonka vuoksi siitä tulee ketjun heikoin lenkki. Näin ollen ohjaustason korkean saatavuuden mahdollistamiseksi K3s-palvelimet voidaan osoittaa ulkoiseen tietokantapäätteeseen, upotetun SQLite-tietovaraston sijasta. Tuettuja ulkoisia tietokantoja ovat etcd3, MySQL ja PostgreSQL [33]. Delegoimalla tilan ulkoiselle tietokannalle, ohjaustasosta voidaan luoda useita instansseja, mikä mahdollistaa klusterin korkean saatavuuden.



Kuva 4. *Korkean saatavuuden HA-K3s klusteri usealla master-palvelimella ja ulkoisella tietokannalla [30].*

Kokonaisuudessaan K3s-arkkitehtuuri on suunniteltu kevyeksi ja kustannustehokkaaksi, joka tarjoaa yksinkertaisemman vaihtoehdon K8s:lle. Pohjimmiltaan K3s käyttää samoja sovellusrajapintoja kuin K8s, joten se on täysin yhteensopiva K8s-työkalujen kanssa. K3s:n kevyt rakenne ja helpompi hallittavuus tekee siitä

varteenotettavan vaihtoehdon organisaatioille, joilla on rajalliset resurssit tai jotka etsivät yksinkertaisempaa vaihtoehtoa K8s:lle. K3s soveltuu hyvin edge laskentaan, IoT-laitteisiin, CI-putkiin, kehitysympäristöihin, ARM-arkkitehtuureihin tai tilanteisiin, joissa täysiverisen Kubernetesin käyttö ei ole kannattavaa [30]. Se tarjoaa helpon tavan asentaa ja hallita Kubernetes-klustereita mahdollisimman pienellä yleiskustannuksella ja kompleksisuudella.

5. VERTAILU

Kubernetesin K8s ja K3s jakelut tarjoavat klusterin käyttäjille saman toiminnallisen rajapinnan. Tämä tarkoittaa, että Kubernetesin YAML-manifesti voidaan ottaa käyttöön sellaisenaan kummallakin ratkaisulla. Missä jakelut pääosin erottautuvat toisistaan, on niiden paketointitapa ja niiden sisältämissä komponenteissa. Tässä osiossa esitetään jakeluiden välisiä keskeisimpiä eroavaisuuksia, jotka ovat muodostettu erilaisten lähteiden, sekä kehityksestä opittujen asioiden pohjalta.

5.1 Käyttöönotto

Merkittävin ero Kubernetesin (K8s) ja K3s:n välillä on asennuksen ja käyttöönoton kompleksisuus. Kubernetes on hyvin monipuolinen ja laajennettavissa oleva konttien orkestrointialusta, joka tarjoaa monipuolisia konfigurointimahdollisuuksia. Tämän vuoksi sen käyttöönotto on hyvin kompleksinen ja aikaa vievä prosessi. Siihen kuuluu useita vaiheita, kuten ohjaustason, työsolmujen ja verkkoasetusten määrittäminen. Kubernetesin asentamiseen on saatavilla useita työkaluja, kuten kubeadm [37] tavanomaisiin asennuksiin, kOps [38] AWS:lle (Amazon Web Services) tai GCP:lle (Google Cloud Platform) ja Rancher [39] yritystason asennuksiin. Vaikka asennustyökalut tekevät asennuksesta suoraviivaisempaa, tuotantokelpoisen Kubernetes-klusterin pystyttäminen vaatii kuitenkin huomattavasti kokemusta ja asiantuntemusta.

Kubernetes-projekti tarjoaa erilliset binaarit jokaiselle komponentille, ja käyttäjän tehtävänä on ottaa kukin komponentti onnistuneesti käyttöön ohjaustason ja klusteriin liitettävien työsolmujen luomiseksi. Klusterin luomista voidaan suoraviivaistaa käyttämällä kubeadm-asennustyökalua, joka luo parhaiden käytäntöjen mukaisen minimaalisen toimintakykyisen K8s klusterin [37]. Tästä huolimatta käyttäjälle jää vielä tehtäväksi useita manuaalisia asennus ja konfiguraatio vaiheita, kuten kuormantasauksen määritys, työsolmujen kubelet ja CRI-määritykset, sekä kapselien verkkoasetukset (CNI).

K3s puolestaan tarjoaa yksinkertaistetun käyttöönottokokemuksen. K3s on yksittäinen binääripaketti, joka minimoi vaiheita ja konfigurointia, joita tarvitaan tuotantokelpoisen Kubernetes-klusterin asentamiseen [30]. Binääritiedosto voidaan asentaa yksinkertaisella yhden rivin komennolla ja se sisältää kaikki tarvittavat riippuvuudet ja komponentit Kubernetesin asentamiseen ohjaustasolle ja työsolmuille, mikä tekee asennusprosessista hyvin suoraviivaisen. K3s tekee käyttöönoton helpottamiseksi monia konfiguraatio arkkitehtuurisia päätöksiä valmiiksi, jotka perustuvat kaikkein yleisimpiin standardeihin. Näin ollen Kubernetes-jakeluita voidaan käsitteellisesti verrata Linux-käyttöjärjestelmiin, jossa K3s on Kubernetes-jakelu samalla tavalla kuin Ubuntu on Linux-jakelu. K3s:n sisältämiä oletusarvoisia riippuvuuksia ja konfiguraatioita tarkastellaan tarkemmin seuraavassa alaluvussa.

5.2 Oletusarvoiset komponentit

Kubeadm työkalu alustaa minimaalisen K8s klusterin lataamalla ja asentamalla klusterin tietovaraston (etcd) ja Kubernetesin ydinkomponentit. K3s niputtaa ydinkomponentit ja kaikki tarvittavat ulkoiset riippuvuudet samaan binääritiedostoon. K3s tekee myös joitakin arkkitehtuurisia yksinkertaistuksia ja esikonfiguraatioita kevyen, yksinkertaisen ja yhtenäisen Kubernetes-jakelun tarjoamiseksi.

Yksi suurimmista muutoksista liittyy ohjaustasossa käytettävään tietovarastoon: Kubernetes käyttää etcd:tä, kun taas K3s käyttää upotettua SQLite-tietokantaa. Tämä parantaa yleisesti suorituskykyä ja pienentää binäärikokoa, mutta ei välttämättä sovellu suurille klustereille. K3s voi tarvittaessa muodostaa yhteyden ulkoiseen etcd-tietovarastoon sekä muihin SQL-pohjaisiin tietokantoihin, kuten MySQL:ään ja PostgreSQL:ään.

Koska K3s pyrkii tarjoamaan yksinkertaisemman käyttökokemuksen, se on esikonfiguroitu seuraavien avoimen lähdekoodin yhteisöstä hankittujen riippuvuuksien kanssa:

- *containerd*: CNCF-standardin mukainen kevyt konttien ajoympäristö, joka toteuttaa konttien ajoalustan rajapinta (CRI) määrityksen [40].

- *Flannel*: Kapselien verkostointikomponentti, joka toteuttaa konttien verkkorajapinta (CNI) määrityksen.
- *ServiceLB*: Yksinkertainen isäntäporttipohjainen palveluiden kuormantasaaja.
- *CoreDNS*: Nimipalvelujärjestelmä, joka toimii palveluiden ja kapselien hakutyökaluna, jonka avulla niihin voidaan ottaa yhteyttä muuttuvien IP-osoitteiden sijasta pysyvillä DNS-nimillä. Palveluhaku helpottaa kapseleiden ja palveluiden löytämistä ja keskinäistä viestintää niiden muuttuvista IP-osoitteista huolimatta [41].
- *Traefik*: Ingressiohjain, jonka avulla voidaan määritellä, miten saapuva liikenne ohjataan klusterissa oleviin palveluihin esimerkiksi URL-polun tai HTTP-otsikon host-kentän perusteella. Ingressiohjain on keskeinen ominaisuus palveluille, joiden on oltava saavutettavissa Kubernetes-klusterin ulkopuolelta [42].

Näiden oletusarvoisten riippuvuuksien lisäksi, K3s niputtaa myös mukaan suositun Helm aputyökalun [43], joka auttaa hallitsemaan (asentamaan, päivittämään, poistamaan) Kubernetes-klusterin Helm-kaavioiksi pakattuja sovelluksia. Helm työkalu yksinkertaistaa sovellusten asentamista ja hallintaa Kubernetesissa paketoimalla nämä sovellukset ja niiden määritykset kätevään, uudelleenkäytettävään muotoon.

K3s tarjoamat oletusarvoiset riippuvuudet voidaan tarvittaessa korvata muilla ominaisuuksilla. Esimerkiksi Flannel voidaan korvata Calico tai Canal CNI-liitännäisellä ja Traefik voidaan korvata Nginx ingressiohjaimella [30]. K8s klusterin luonti vaatii näiden riippuvuuksien ja Helm työkalun manuaalisen määrittämisen, kun taas K3s tarjoaa ns. ”paristot mukana, mutta vaihdettavissa” lähestymistavan.

5.3 Skaalautuvuus

K3s tukee Kubernetesin keskeisiä skaalautuvuuden edellyttämiä ominaisuuksia, kuten kapselien horisontaalista autoskaalausta, mutta se ei sisällä kaikkia Kuber-

netesista löytyviä skaalautumisominaisuuksia. Vaikka K3s käyttää samoja ydinkomponentteja kuin K8s, ne on pakattu tavalla, joka vähentää niiden modulaarisuutta. Tämä tekee K3s:stä yksinkertaisemman ja resurssitehokkaamman, mutta se tarkoittaa myös sitä, että jotkin hienojakoiset skaalaus- ja konfigurointivaihtoehdot eivät ole käytettävissä, kuten rajapintapalvelimen horisontaalinen skaalaus. Vaikka K3s kykenee hallitsemaan tuotannon työkuormia ja tarjoamaan skaalautuvuutta ja korkean saavutettavuuden, se on suunniteltu ensisijaisesti resurssirajoitteisiin ympäristöihin, jonka vuoksi se ei välttämättä sovellu hyvin suurille ja monimutkaisille klustereille. K8s puolestaan pystyy hallitsemaan tuhansia solmuja yhdessä klusterissa (Kubernetes v1.27 tukee jopa 5 000 solmun klustereita [44]), kun taas K3s pärjää tyypillisesti paremmin muutaman sadan solmun kanssa. K3s:n yksinkertaisuus saattaa osoittautua liian rajoittavaksi suurissa klustereissa, joissa halutaan täysi kontrolli yksittäisiin ohjaustasojen komponentteihin.

5.4 Resurssivaatimukset

Resurssien käytön osalta Kubernetes (K8s) vaatii kattavien ominaisuuksiensa ja laajennettavuuden vuoksi myös huomattavan määrän laskentaresursseja, mikä voi aiheuttaa lisäkustannuksia, jos klusteri sijoitetaan pilvipalveluun. Kubernetes v1.27 kubeadm:lla luodun yksittäisen ohjaustasosolmun vähimmäisvaatimukset ovat 2 suorittimen ydintä ja 2 Gt RAM-muistia [37]. Todellinen resurssikulutus voi olla tuotantoympäristössä paljon suurempi riippuen työmääristä ja klusterin koosta. Lisäksi Kubernetesin käyttämä etcd-tietovarasto voi vaatia huomattavia levyresursseja erityisesti suurissa käyttöönotoissa.

K3s:n vähimmäisvaatimukset ovat huomattavasti alhaisemmat kuin Kubernetesin. K3s:n dokumentaation mukaan palvelinsolmu vaatii minimissään vain yhden suoritinytimen ja 512 Mt RAM-muistia, tosin suositus on 1 Gt [30]. Resurssien tarvetta on saatu pienennettyä ensisijaisesti ajamalla Kubernetesin komponentteja yhden prosessin sisällä [33]. K3s:n vaatimukset vaihtelevat työmäärän mukaan, mutta yleensä ne ovat huomattavasti pienemmät kuin K8s:n.

5.5 Nopeus

K8s-klusteri ja K3s-klusteri, jotka on otettu käyttöön vastaavalla laitteistolla ja konttien ajoalustalla, pitäisi ajaa kontteja vertailukelpoisella suorituskyvyllä. K3s on kuitenkin niin kevyt, että se asentaa ja käynnistää ohjaustason huomattavasti nopeammin kuin K8s. Kubernetesin (K8s) käynnistäminen voi kestää useita minuutteja, kun taas K3s on yleensä käyttövalmis alle minuutissa. Tämän vuoksi K3s soveltuu paljon paremmin tilapäisiin klustereihin, kuten paikallisiin kehitys- ja testausympäristöihin. K3s nopeuttaa ja helpottaa myös monia klusterin hallinnan osa-alueita sen virtaviivaistetun rakenteen ansiosta, kuten kapselien luontia, solmujen lisäämisen ja skaalautumisen kaltaisia toimintoja, erityisesti pienemmissä tai yksinkertaisemmissa käyttötapauksissa.

5.6 Turvallisuus

Sekä Kubernetes että K3s tarjoavat vahvat tietoturvaominaisuudet. Kubernetes tarjoaa kattavine ominaisuuksineen kehittyneitä, hienojakoisia tietoturvakontrolleja, mutta sen monimutkaisuus voi johtaa mahdollisiin virheellisiin konfiguraatioihin. K3s on oletusarvoisesti tietoturvallinen, ja sen ulkoisten riippuvuuksien minimointi tekee myös tietoturva-aukkojen muodostumisesta epätodennäköisempää [30].

Yksinkertaistetusta rakenteestaan huolimatta K3s ei kompromisoi tietoturvan suhteen. K3s sisältää Kubernetesin keskeiset tietoturvaominaisuudet, kuten RBAC:n (Role-Based Access Control), joka mahdollistaa yksityiskohtaisen hallinnan siitä, että klusterin käyttäjillä ja työkuormilla on oikeudet käyttää vain niiden rooliensa suorittamiseen tarvittavia resursseja. K3s tukee myös verkkokäytäntöjä (Network Policies), joilla voidaan rajoittaa sitä, mitkä kapselit voivat kommunikoida keskenään ja muiden verkkopäätteiden kanssa, sekä miltä kapseleilta ne voivat vastaanottaa liikennettä. K3s soveltaa oletusarvoisesti useita tietoturvakäytäntöjä, ja se läpäisee useimmat Kubernetes Center for Internet Security (CIS) -kyberturvallisuusstandardit ilman muutoksia. Täydellisen CIS -standardien täyttämiseksi on tehtävä joitakin manuaalisia toimenpiteitä, kuten isäntäkone-tason tietoturvamääritykset, sekä kapselien verkkokäytäntöjen soveltaminen [45].

5.7 Yhteisön tuki

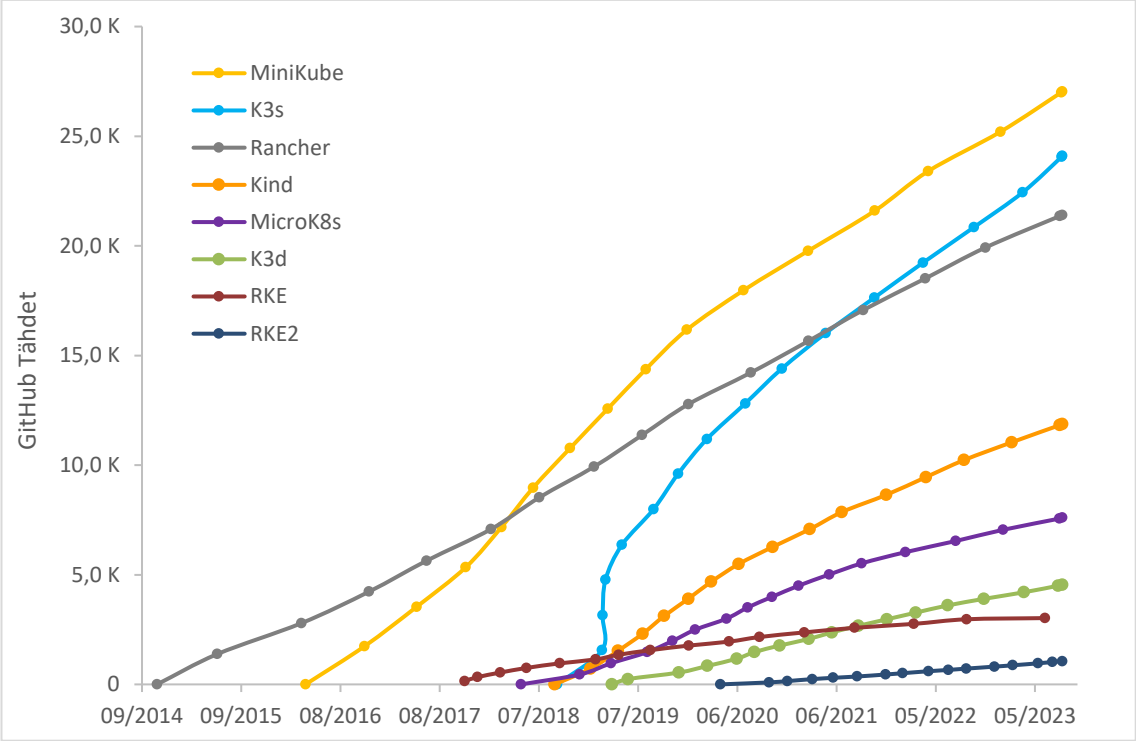
Kubernetes-ekosysteemi ja -yhteisö ovat merkittäviä tekijöitä Kubernetes-jakelua valittaessa. Alkuperäisenä Kubernetes-projektina K8s:llä on laaja ekosysteemi laajennuksia ja lisäosia, suuri aktiivinen yhteisö ja sitä tukevat useat pilvipalveluntarjoajat, mikä tekee siitä monipuolisen resurssin kehittäjille ja ylläpitäjille [9]. Se on saavuttanut suuren suosion sen kesäkuun 2014 julkaisun jälkeen, ja sen GitHub-tähtien määrä on elokuun 2023 mennessä noussut 101 000:een [46].

GitHub-tähdet ovat tapa osoittaa arvostusta ja tukea GitHub-projekteille. Ne ovat myös mittari projektin suosiosta ja merkityksestä GitHub-yhteisössä. GitHub-tähtien tarkoituksena on auttaa kehittäjiä löytämään, seuraamaan ja osallistumaan merkittäviin ja korkeatasoisiin projekteihin. GitHub projektin tähtien määrää käytetään yleisesti yhtenä isoimpana mittarina projektin tärkeyttä mitatessa.

GitHub-tähtien jatkuva kasvu osoittaa, että Kubernetesia käyttävä yhteisö ei ainoastaan käytä Kubernetesia, vaan myös osallistuu aktiivisesti sen parantamiseen. Tämä valtaisa tuki on johtanut siihen, että Kubernetesin ympärille on syntynyt laaja ekosysteemi, joka koostuu työkaluista, laajennuksista, liitännäisistä ja integraatioista. Tällainen tuki varmistaa, että kaikkiin raportoituihin ongelmiin tai vikoihin puututaan nopeasti, uusia ominaisuuksia lisätään säännöllisesti ja oppimiseen ja vianmääritykseen on saatavilla runsaasti resursseja.

K3s on uudempi ja pienempi, mutta sillä on nopeasti kasvava ekosysteemi ja yhteisö. Sertifioituna Kubernetes-jakeluna se on kasvattamassa arvostusta resurssien tehokkaasta käytöstä, yksinkertaisuudesta ja sen soveltuvuudesta reunalaskentaan. K3s:n yhteisö on kasvava ja aktiivinen, josta merkkinä on GitHub:ssa annettujen tähtien määrä, joissa se on ollut 2019 julkaisun jälkeen nopeimmin suosiota kasvattava vaihtoehtoinen Kubernetes-jakelu [47][48]. Eri yksinkertaisempien Kubernetes-jakeluiden GitHub-tähtihistoriaa on kuvattu kuvaajassa 5, joka osoittaa K3s:n nopean omaksumisvauhdin, mikä asemoi sen yhdeksi eturivin Kubernetes-jakelun joukkoon.

Vaikka K3s:n kehittäjäkunta on vielä pieni verrattuna Kubernetesiin, se perii ison osan maturiteettisen Kubernetesin ytimestä, jonka vuoksi sen julkaisutahti noudattelee pitkälti Kubernetes-julkaisujen tahtia. Tämän vuoksi myös K3s:n julkaisuversiointi heijastaa Kubernetesin versiota. Esimerkiksi K3s-julkaisu v1.27.4+k3s1 vastaa Kubernetes-julkaisua v1.27.4, joka päivittää ydintoiminnallisuuden Kubernetesin versioon v1.27.4 ja +k3s<numero> jälkiliite kuvaa lisäjulkaisuja, kuten viankorjauksia tai muita muutoksia [33].



Kuva 5. *Eri yksinkertaisempien Kubernetes-jakeluiden GitHub-tähtihistoria, joka seuraa projektien suosion kasvua ajan myötä [49].*

6. YHTEENVETO

Työssä tutkittiin kahta suosittua Kubernetes-jakelua: alkuperäistä, täysin varusteltua Kubernetesia (K8s) ja kevyttä, virtaviivaistettua K3s:ää. Arvioinnissa tarkisteltiin keskeisiä näkökohtia, kuten niiden arkkitehtuuria, asennuksen ja käyttöönoton kompleksisuutta, resurssien käyttöä, skaalautuvuutta, suorituskykyä, tietoturvaominaisuuksia ja yhteisön tukea. Arvioinnilla pyrittiin tuomaan esiin näiden kahden Kubernetes-jakelun yhtäläisyyksiä, eroja ja kompromisseja.

Arvioinnista ilmenee, että sekä Kubernetesilla (K8s) että K3s:llä on omat vahvuutensa ja ideaaliset käyttötapauksensa konttien orkestroinnissa. K8s, täysin varusteltu alkuperäinen Kubernetes, tarjoaa erittäin laajennettavissa olevan alustan, jota voidaan mukauttaa käytännössä kaikkiin yritystason sovellusten käyttöönottotarpeisiin. Sen kattava ominaisuusvalikoima, korkea skaalautuvuus ja hienojakoinen konfiguroitavuus tekevät siitä sopivan suurten ja kompleksisten klustereiden hallintaan. Sen asennus- ja käyttöönotto voi kuitenkin olla monitahoinen ja aikaa vievä prosessi, joka vaatii huolellista konfigurointia, minkä vuoksi se sopii paremmin organisaatioille, joilla on käytettävissä riittävästi resursseja ja asiantuntemusta.

K3s ratkaisee käyttöönoton haasteet tarjoamalla yksittäiseksi binääritiedostoksi pakatun CNCF-sertifioidun Kubernetes-jakelun. K3s vähentää Kubernetesin kompleksisuutta ja resurssien tarvetta uhraamatta sen keskeisimpiä turvallisuus- ja hallintatoiminnallisuuksia. Käyttöönoton yksinkertaistamiseksi K3s binääriin on sisällytetty keskeisten Kubernetes-komponenttien lisäksi valmiiksi konfiguroituja riippuvuuksia, kuten containerd, Flannel ja Traefik, jotka voidaan tarvittaessa korvata muilla vaihtoehdoilla. Yksinkertaisuudestaan huolimatta K3s tarjoaa K8s:ään verrattavissa olevia vahvoja tietoturvaominaisuuksia, ja se täyttää oletusarvoisesti useimmat CIS-kyberturvallisuusstandardit. Sen kevennetty arkkitehtuuri mahdollistaa myös huomattavasti nopeammat käynnistysajat K8s:ään verrattuna. K3s:n suoraviivainen asennusprosessi ja nopea käynnistys tekevät siitä erityisen hyödyllisen tapauksissa, joissa nopea ja ketterä klusterin käyttöönotto on olennaista.

Viime kädessä valinta K8s:n ja K3s:n välillä riippuu organisaation erityistarpeista, resursseista ja käyttötapausvaatimuksista. Suuremmat yritykset, joilla on monimutkaisia ympäristöjä ja vaativia skaalautuvuusvaatimuksia, saattavat pitää K8s:ää sopivampana, kun taas pienemmät organisaatiot, jotka etsivät virtaviivaista Kubernetes-kokemusta, voivat hyötyä enemmän K3s:n käyttöönotosta. Molemmat jakelut tarjoavat vankan ja luotettavan alustan konttisovellusten hallintaan. K8s soveltuu kattavien ominaisuuksiensa ja skaalautuvuutensa ansiosta hyvin suuriin kompleksisiin klustereihin. Sen sijaan K3s sopii sen kevyen rakenteen ja helpon käyttöönoton ansiosta resurssirajoitteisiin ympäristöihin, helppokäyttöisyyttä tai nopeaa käyttöönottoa vaativiin käyttötapauksiin, sekä tilanteisiin, joissa K8s:n käyttö ei välttämättä ole kustannustehokasta tai tarpeellista.

K3s kasvattaa jatkuvasti suosiotaan tehokkaan, kompaktin ja yksinkertaistetun Kubernetes-lähestymistapansa ansiosta. Vaikka se ei välttämättä korvaa täysimittaista kubernetesia kaikissa tilanteissa, K3s osoittaa, että suurempi ei ole aina parempi, kun kyse on tehokkaasta ja toimivasta konttien orkestroinnista. Tämän evaluoinnin kannalta on myös syytä huomioida, että K8s ja K3s ovat aktiivisen kehitystyön alla, ja uusia ominaisuuksia julkaistaan säännöllisesti. Tässä työssä tehty vertailu todennäköisesti vanhentuu ajan myötä, minkä vuoksi vastaavia vertailuja tulisi tehdä säännöllisesti.

LÄHTEET

[1] A. M. Potdar, D. G. Narayan, S. Kengond, M. M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine”, 2020. Saatavissa: https://www.sciencedirect.com/science/article/pii/S1877050920311315

[2] The Official Upstream Kubernetes Documentation. Saatavissa: https://kubernetes.io/ . [Viitattu: 15.11.2022].

[3] P. Mell, T. Grance, “SP 800-145 The NIST Definition of Cloud Computing”, 2011. Saatavissa: https://csrc.nist.gov/publications/detail/sp/800-145/final

[4] E. Simmon, “Evaluation of Cloud Computing Services Based on NIST SP 800-145”, 2018. Saatavissa: https://www.nist.gov/publications/evaluation-cloud-computing-services-based-nist-sp-800-145

[5] What are microservices. Saatavissa: https://www.ibm.com/topics/microservices [Viitattu 15.1.2023]

[6] V. Singh, S. K. Peddoju, “Container-based Microservice Architecture for Cloud Applications”, 2017. Saatavissa: https://ieeexplore.ieee.org/document/8229914

[7] G. Blinowski, A. Ojdowska and A. Przybyłek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”, 2022. Saatavissa: https://ieeexplore.ieee.org/document/9717259

[8] What Are Containers. Saatavissa: https://www.suse.com/c/rancher_blog/what-are-containers/ [Viitattu 18.1.2023]

[9] Overview of Kubernetes. Saatavissa: https://kubernetes.io/docs/concepts/overview/ [Viitattu 22.1.2023]

[10] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, Y. Al-Hammadi, “Performance comparison between container-based and VM-based services”, 2017. Saatavissa: https://ieeexplore.ieee.org/document/7899408

[11] Stephen J. Bigelow, “A breakdown of container runtimes for Kubernetes”, 2021. Saatavissa: https://www.techtarget.com/searchitoperations/tip/A-breakdown-of-container-runtimes-for-Kubernetes-and-Docker [Viitattu 28.1.2023]

[12] S. McCarty, “A Practical Introduction to Container Terminology”, 2018. Saatavissa: https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction [Viitattu 28.1.2023]

[13] “An Introduction to Kubernetes”. Saatavissa: https://www.redhat.com/en/topics/containers/what-is-kubernetes [Viitattu 8.2.2023]

[14] Kubernetes Documentation: Kubernetes Components, Saatavissa: https://kubernetes.io/docs/concepts/overview/components/ [Viitattu 12.2.2023]

[15] Kubernetes Documentation: Kubernetes API Overview, Saatavissa: https://kubernetes.io/docs/reference/using-api/ [Viitattu 13.2.2023]

[16] The official etcd documentation: A distributed, reliable key-value store for the most critical data of a distributed system, Saatavissa: https://etcd.io/ [Viitattu 14.2.2023]

[17] Kubernetes Documentation: Kubernetes Scheduler, Saatavissa: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ [Viitattu 16.2.2023]

[18] Kubernetes Documentation: Controllers, Saatavissa: https://kubernetes.io/docs/concepts/architecture/controller/ [Viitattu 18.2.2023]

[19] Kubernetes Documentation: Cloud Controller Manager, Saatavissa: https://kubernetes.io/docs/concepts/architecture/cloud-controller/ [Viitattu 18.2.2023]

[20] Kubernetes Documentation: kubelet, Saatavissa: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ [Viitattu 20.2.2023]

[21] Kubernetes Documentation: Container Runtime Interface, Saatavissa: https://kubernetes.io/docs/concepts/architecture/cri/ [Viitattu 22.2.2023]

[22] S. Kanzhelev, J. Angel, D. Srinivas, S. Kularathna, C. Short, D. Chen, “Kubernetes is Moving on From Dockershim: Commitments and Next Steps”, Saatavissa: https://kubernetes.io/blog/2022/01/07/kubernetes-is-moving-on-from-dockershim/ [Viitattu 22.2.2023]

[23] Kubernetes Documentation: Pods, Saatavissa: https://kubernetes.io/docs/concepts/workloads/pods/ [Viitattu 1.6.2023]

[24] D. Sullivan, “Official Google Cloud Certified Associate Cloud Engineer Study Guide”, 2019. ISBN: 9781119564393

[25] Kubernetes Documentation: ReplicaSet, Saatavissa: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/ [Viitattu 1.6.2023]

[26] Kubernetes Documentation: Pod Lifecycle, Saatavissa: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/ [Viitattu 1.6.2023]

[27] “Cloud Native Computing Foundation: Kubernetes Project Journey Report”, 2019. Saatavissa: https://www.cncf.io/reports/kubernetes-project-journey-report/

[28] B. Burns, J. Beda, K. Hightower, “Kubernetes: Up and Running, 2nd Edition”, 2019. ISBN: 9781492046530

[29] Kubernetes Documentation: Horizontal Pod Autoscaling, Saatavissa: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ [Viitattu 24.2.2023]

[30] K3s - The certified Lightweight Kubernetes distribution Official documentation, Saatavissa: https://docs.k3s.io/ [Viitattu 8.3.2023]

[31] S. Telenyk; O. Sopov; E. Zharikov; G. Nowakowski, “A Comparison of Kubernetes and Kubernetes-Compatible Platforms”, Saatavissa: https://ieeexplore.ieee.org/document/9660392

[32] M. Abrams, B. Patel, “Accelerating Edge Computing with Arm and Rancher k3s Lightweight Kubernetes”, Saatavissa: https://more.suse.com/rs/937-DCH-261/images/ARM-White-Paper-V3.pdf [Viitattu 16.03.2023]

[33] GitHub - k3s-io/k3s: Lightweight Kubernetes, Saatavissa: https://github.com/k3s-io/k3s [Viitattu 16.3.2023]

[34] Stopping and Upgrading K3s Cluster, Saatavissa: https://docs.k3s.io/upgrades [Viitattu 18.3.2023]

[35] Flannel – GitHub: Network fabric for containers, designed for Kubernetes, Saatavissa: https://github.com/flannel-io/flannel [Viitattu 20.3.2023]

[36] SQLite: Embedded SQL Database Engine, Saatavissa: https://www.sqlite.org/ [Viitattu 1.6.2023]

[37] Creating a cluster with kubeadm, Saatavissa: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/ [Viitattu 2.8.2023]

[38] kOps. Kubernetes Operations, Saatavissa: https://kops.sigs.k8s.io/ [Viitattu 2.8.2023]

[39] Rancher. Kubernetes management tool to deploy and run clusters, Saatavissa: https://ranchermanager.docs.rancher.com/ [Viitattu 2.8.2023]

[40] containerd: An open and reliable container runtime, Saatavissa: https://github.com/containerd/containerd

[41] CoreDNS: DNS and Service Discovery, Saatavissa: https://coredns.io/ [Viitattu 1.7.2023]

[42] Traefik Proxy Documentation, Saatavissa: https://doc.traefik.io/traefik/ [Viitattu 1.7.2023]

[43] Helm: The package manager for Kubernetes, Saatavissa: https://helm.sh/

[44] Considerations for large Kubernetes clusters, Saatavissa: https://kubernetes.io/docs/setup/best-practices/cluster-large/ [Viitattu 8.7.2023]

[45] K3s CIS Hardening Guide, Saatavissa: https://docs.k3s.io/security/hardening-guide [Viitattu 15.7.2023]

[46] Kubernetes (K8s) – GitHub, Saatavissa: https://github.com/kubernetes/kubernetes [Viitattu 15.8.2023]

[47] Kubernetes Tooling - Ranking, Saatavissa: https://ossinsight.io/collections/kubernetes-tooling [Viitattu 16.8.2023]

[48] K3s Analytics Overview, Saatavissa: https://ossinsight.io/analyze/k3s-io/k3s#overview [Viitattu 16.8.2023]

[49] GitHub Star History, Saatavissa: https://star-history.com [Viitattu 16.8.2023]

[50] A. Malviya, R. K. Dwivedi, “A Comparative Analysis of Container Orchestration Tools in Cloud Computing”, 2022. Saatavissa: https://ieeexplore.ieee.org/document/9763171