Konttiteknologian hyödyt verrattuna virtuaalikoneisiin

Riku Tenhunen

Haaga-Helia ammattikorkeakoulu

Tietojenkäsittely

Amk-opinnäytetyö

2022

Tiivistelmä

Tekijä(t)
Riku Tenhunen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Konttiteknologian hyödyt verrattuna virtuaalikoneisiin

Sivu- ja liitesivumäärä
31

Viime vuosikymmeninä tietoteknisiä palveluita on rakennettu ja ylläpidetty virtualisointialustoilla toimivilla virtuaalipalvelimilla. Hiljattain näitä ratkaisuja täydentämään on noussut konttiteknologiaa edustavat tuotteet, kuten Dockers ja Kubernetes. Nämä ohjelmistotuotteet ovat nousseet houkuttelevaksi vaihtoehdoksi palveluiden rakentamisessa silloin, kun on haluttu vähentää laitteistoresurssien käyttöä, helpottaa sovellustestausta, ja vähentää manuaalista asennustyötä.

Opinnäytetyössä vertaillaan konttiteknologian ja virtuaalipalvelimien eroavaisuuksia resurssien, käyttöönoton, kehitystyön ja siirrettävyyden näkökulmista. Opinnäytetyö on luonteeltaan tutkimuksellinen. Resurssinäkökulmaan haetaan vastausta suunnittelemalla samaa palvelua tuottava järjestelmä molempia teknologioita käyttäen ja vertailemalla niitä keskenään. Muihin tutkimuskysymyksiin haetaan vastauksia kirjallisista lähteistä.

Tietoperusta on koottu ohjelmistotuotteiden valmistajien dokumentaatioista, suurien yritysten julkaisuista, asiantuntijoiden blogeista, sekä kirjoista. Tietoperustassa käsitellään vertailtavia aiheita eli konttiteknlogiaa ja virtualisointiteknologiaa.

Tutkimusten tuloksista ilmeni, että konttiteknologialla toteutettu järjestelmä käyttäisi hieman vähemmän laitteistoresursseja, sekä toisi useita hyötyjä käyttöönoton, kehitystyön ja siirrettävyyden saralla.

Asiasanat
kontti, mikropalveluarkkitehtuuri, konesali, tietojärjestelmä, ohjelmisto, virtualisointi

Sisällys

1 Johdanto

Viime vuosikymmeninä tietoteknisiä palveluita on rakennettu ja ylläpidetty virtualisoiduilla palvelimilla, jotka ovat toimeet erilasilla tuotteilla kuten KVM (Kernel-based Virtual Machine) ja OVM (Oracle Virtual Machine). Hiljattain näitä ratkaisuja täydentämään on noussut ns. ”konttiteknologiaa” edustavat tuotteet (eng. container), kuten Dockers ja Kubernetes. Nämä ohjelmistotuotteet ovat nousseet houkuttelevaksi vaihtoehdoksi palveluiden rakentamisessa silloin, kun on haluttu vähentää laitteistoresurssien käyttöä, helpottaa sovellustestausta, ja vähentää manuaalista asennustyötä.

Kontista tulee ensimmäiseksi mieleen merikontti, joka onkin hyvä vertauskuva tälle teknologialle. Merikonttia voidaan siirtää helposti esimerkiksi eri kuljetusvälineiden, kuten laivan, kuorma-auton tai junan välillä. Myös tietotekniikassa kontti on suunniteltu siten, että se on helposti siirrettävä erilaisten alustojen välillä kuten työaseman, pilven ja konesalin Kontti on kevyt kokonaisuus, johon sovellus on pakattu riippuvuuksineen. Konteissa voidaan suorittaa esimerkiksi sovelluksia ja tietokantoja. Tämä mahdollistaa sovellusten nopean käyttöönoton, testaamisen ja kehityksen.

Opinnäytetyössä tutkitaan perinteisinä pidettyjä virtuaalikoneratkaisuja ja uusien konttiratkaisujen erovaisuuksia. Työssä etsitään syitä ja hyötyjä sille, miksi konttiteknologialla toimivat palvelut voisivat olla parempi valinta organisaatiolle kuin perinteiset virutaalikoneratkaisut. Tarkastelen tätä kolmesta näkökulmasta, joihin kuuluvat laitteistoresurssien käyttö, sovellusten käyttöönotto ja kehitys, sekä järjestelmän siirtämisen helppous eri laitteistojen välillä. Eri teknologioilla toteutettujen järjestelmien suorituskykyä ja resurssitehokkuutta vertaillaan esimerkkiarkkitehtuurin avulla, etsien vastauksia seuraaviin kysymyksiin:

- Käyttävätkö teknologiat saman verran laitteistoresursseja, vai käyttääkö toinen teknologioista vähemmän resursseja?
- Millaisia eroja eri konttiteknologiat tuovat verrattuna virtualisointiteknlogiaan kehityksen, käyttöönoton ja siirrettävyyden kannalta?

Kontti- ja virtuaalikoneratkaisujen kehityksen, käyttöönoton ja siirrettävyyden eroja ja hyötyjä selvitetään tietoperustassa kirjallisia lähteitä käyttäen. Empiirisessä osuudessa tutkitaan kahta erilaista tapaa toteuttaa esimerkkijärjestelmä. Lopputuloksena syntyy vertailu siitä, mitä etuja konttiteknologiaan perustuvilla ratkaisuilla on perinteisenä pidettyyn virtualisointitekniikkaan verrattuna. Tämä uusi tieto hyödyttää organisaatioita, mikäli he pohtivat tietoteknisten palveluiden arkkitehtuuriratkaisuvalintoja.

Käsitteet

Kontti (eng. container) = Ohjelmallisesti erotettu kokonaisuus, johon sovellus on pakattu riippuvuuksineen, konfiguraatiotiedostoineen ja ohjelmistokirjatoineen.

Kirjasto, luokkakirjasto, ohjelmakirjasto = Tiedosto, joka sisältää yleiskäyttöön tarkoitettuja valmiita aliohjelmia ja ohjelmia, joita kutsutaan ja ajetaan suoritettavien ohjelmien apuna.

Konttialusta (container engine) = Alusta, jolla kontteja ajetaan. Lisäksi konttialusta sisältää erilaisia työkaluja konttien hallintaan.

Orkestrointitykalu (eng. orchestration tool) = Orkestrointityökalu on osa konttialustan ominaisuuksia. Sillä voidaan luoda ja hallinnoida useasta kontista koostuvaa tietojärjestelmää,

Klusterointi = Klusterointi tarkoittaa usean saman palvelun asentamista toimimaan rinnakkain. Klusteroitavia palveluita voi olla esimerkiksi erilaiset sovellukset ja tietokannat.

Linux-kernel tai käyttöjärjestelmän ydin = Käyttöjärjestelmänydin on käyttöjärjestelmän keskeinen ohjelma. Käyttöjärjestelmän ydin toimii ohjelmien ja laitteiston välillä järjestelemällä ja ohjaamalla laitteiston resursseja.

Virtuaalikone = Ohjelmistolla toteutettu näennäiskone, joka emuloi fyysisen tietokoneen toiminnallisuuksia.

Virtualisointialusta, virtualisointiympäristö tai hyperviisori (eng. hypervisor) = tietokoneohjelma tai käyttöjärjestelmä, joka tarjoaa virtuaalikoneille alustan toimia.

Laitteisto, laitteistoalusta, laitteistoresurssit (eng. hardware) = Pilvipalvelun, konesalin tai työaseman fyysiset laitteistoresurssit, kuten keskusmuisti, prosessori, kovalevy ja muiden komponenttien muodostama kokonaisuus.

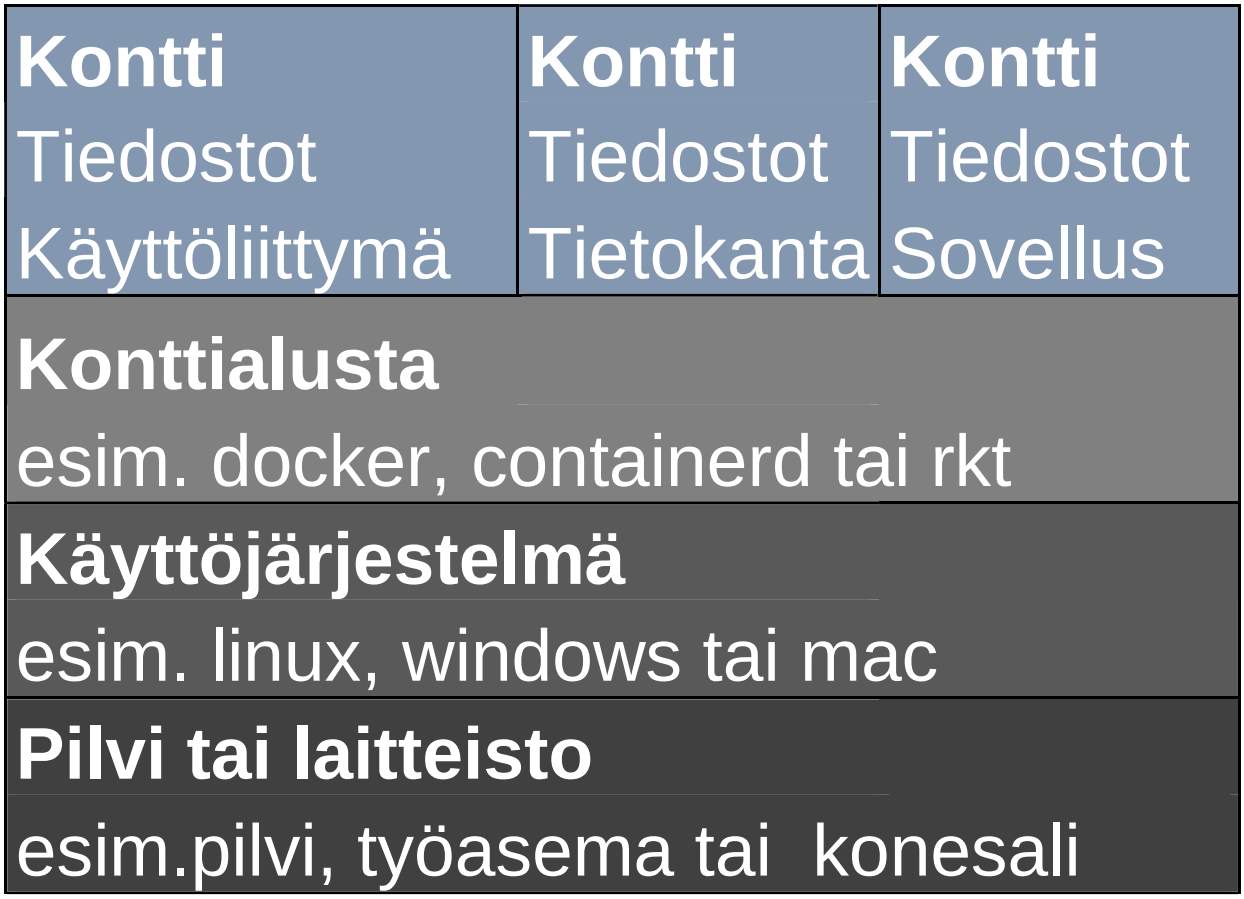
Löyhä kytkentä (eng. loose coupling) = Löyhä kytkentä on suunnitteluperiaate, jota noudattamalla palvelujen väliset riippuvuudet vähenevät ja vikasietoisuus lisääntyy. Erilaiset palvelut tai sovellukset laitetaan kommunikoimaan API-rajapintojen välityksellä.

Järjestelmäkutsu (eng. system call) = Järjestelmäkutsuilla sovellukset pyytävät käyttöjärjestelmän ytimeltä palvelua suorittaakseen prosessin laitteistotasolla.

Korkea saatavuus, HA (eng. High Availability) = Perjaate jolla tietojärjestelmä on rakennettu. Korkean saatavuuden järjestelmissä kaikki on vähintään kahdennettu siten, että järjestelmä pystyy palvelemaan, vaikka yksi palvelin kaatuisi.

2 Konttiteknologia

Kontit, eng. container, ovat pieniä kokonaisuuksia, joihin sovellus on pakattu riippuvuuksineen ja kirjastoineen siten, että ne voidaan ajaa missä tahansa vähäisillä konfiguraatiomuutoksilla tai kokonaan ilman muutoksia. Kuvasta 1 voidaan nähdä konttiratkaisun perusrakenne. Sovelluskontteja ajetaan konttialustan päällä eristetysti. Eristetysti ajaminen tarkoittaa sitä, että sovelluskontit eivät pysty lukemaan, kirjottamaan tai näkemään toistensa prosesseja tai tiedostoja, ellei niin erikseen aseteta näkemään. (Wallenius 2019)



Kuva 1 konttiratkaisun perusrakenne

Maailmanlaajuinen konttialan liiketoiminta on ollut vuonna 2020 lähes puolen miljardin dollarin kokoluokkaa. On ennustettu, että liiketoiminta kasvaa noin miljardiin vuoteen 2024 mennessä. (Gartner 2020) Konttiteknologian hyödyntäminen on kasvamaan päin ja teknologian hyödyt on tullut esiin vasta tällä vuosikymmenellä, kun yritykset ovat siirtyneet käyttämään erilaisia pilvipalveluita.

Vuodesta 2020 vuoteen 2021 yritysten pilvipalveluiden käyttö kasvoi Euroopan unionin alueella 36 prosentista 41 prosenttiin. (Eurostat 2021) Kasvu on ollut huimaa vuodesta 2014, jolloin vastaava prosentti oli 19 prosenttiin. (Eurostat 2014)

CNCF:n vuoden 2020 raportin kyselyyn vastanneista yrityksistä ilmoitti, että heillä on käytössään tai suunnittelevat yli 5000 konttia. Kasvua oli tullut 300 prosenttia verrattuna vuoden 2016 tehtyyn kyselyyn. (Cloud Native Computing Foundation 2020)

2.1 Konttialustat ja orkestrointityökalut

Kontit tarvitsevat toimiakseen alustan. Konttialusta (eng. Container Engine) mahdollistaa konttien ajamisen käyttöjärjestelmän päällä. Konttialusta tarjoaa konteille mahdolliset verkkoyhteydet, levytilan, ohjelmointirajapinnat, konttien hallinointityökalut. Konttialustan perustyökalut riittävät sovelluskehittäjälle kehittämään sovelluskontteja. Kuva 2 esittää konttialustan ominaisuudet eriteltynä Orkestrointityökaluihin ja konttityökaluihin.

Konttialusta

Orkestrointityökalut	Konttityökalut
Klusterointi ominaisuudet	Tietoliikenneasetukset
Sertifikaatit	Levyjärjestelmät
Kuormanjako	Rajapinnat
Reititys ominasuudet	Paketointityökalut
Klusterin tietovarasto	Ajoympäristö

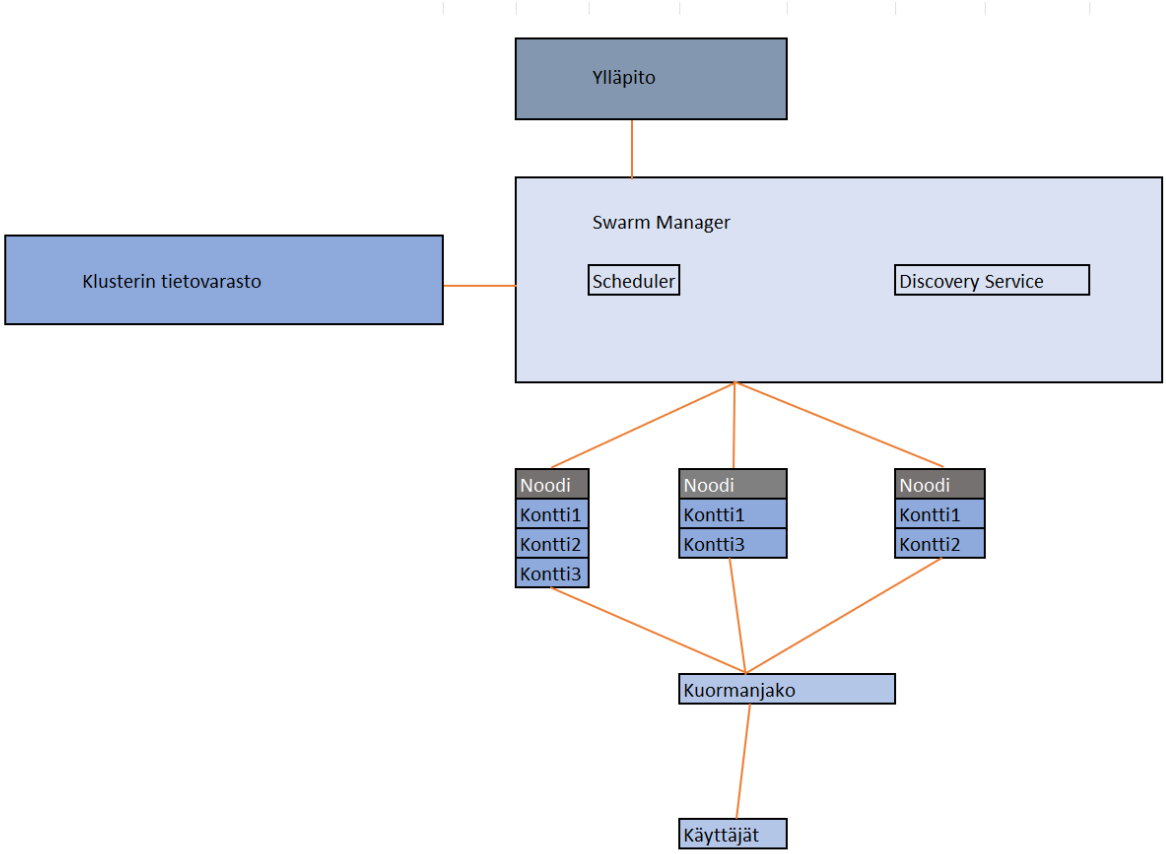
Kuva 2 Konttialustan ominaisuudet

Orkestrointityökalut tuovat konteissa toimiville palveluille ominaisuuksia, jotka ovat tärkeitä korkean käytettävyyden palveluille, sekä suuria työkuormia käsitteleville sovelluskonteille. Orkestrointityökalut tarjoavat konteille klusterointia, älyä, sekä kuormanjakoa. (Avinetworks 2021)

Orkestroinnin tärkeimpiä ominaisuuksia on klusterointi. Klusterointi tarkoittaa useampaa samaa sovellusta ajavan kontin asentamista toimimaan rinnakkain saman orkestroinnin alle. Klusteroinnin keskeisiä hyötyjä on korkean käytettävyyden lisääminen. Tilanteessa, jossa esimerkiksi sovelluskonttia päivitetään tai sovelluskontti ei ole käytettävissä kaatumisen takia pystyy korkean käytettävyyden järjestelmä ohjaamaan liikennettä toimiville sovelluskonteille. Lisäksi orkestrointityökaluun on voitu rakentaa älyä, joka nostaa sovelluskontit ylös itsestään, jos ne sattuvat kaatumaan.

Klusteri tarvitsee lisäksi tietovaraston. Tietovarastoon klusteri tallentaa avaintietoja tilastaan reaaliaikaisesti, jotta klusterin eri palvelut pystyvät toimimaan synergiassa. (Kubernetes 2021)

Kuva 3 näyttää Docker swarm tuotteen klusteroinnilla toimivan konttiratkaisun rakenteen.



Kuva 3 Docker swarm tuotteen esimerkkirakenne

Docker swarm kostuu komponenteista kuten Swarm Manager. Swarm Manager sisältää solmuja tai noodeja ohjaavia komponentteja kuten ajoittaja eng. scheduler. Ajoittaja on toiminnallisuus, joka kerää tietoa koko klusterin laajuudelta siitä, että kuinka kovalla rasituksella tietyn palvelun sovelluskontit ovat ja tämän perusteella lisää tai poistaa sovelluskontteja klusterin solmuilta. Ajoittajaa voidaan konfiguroida toimimaan myös ylläpitäjän toiveiden mukaisesti. (Digitalocean 2014)

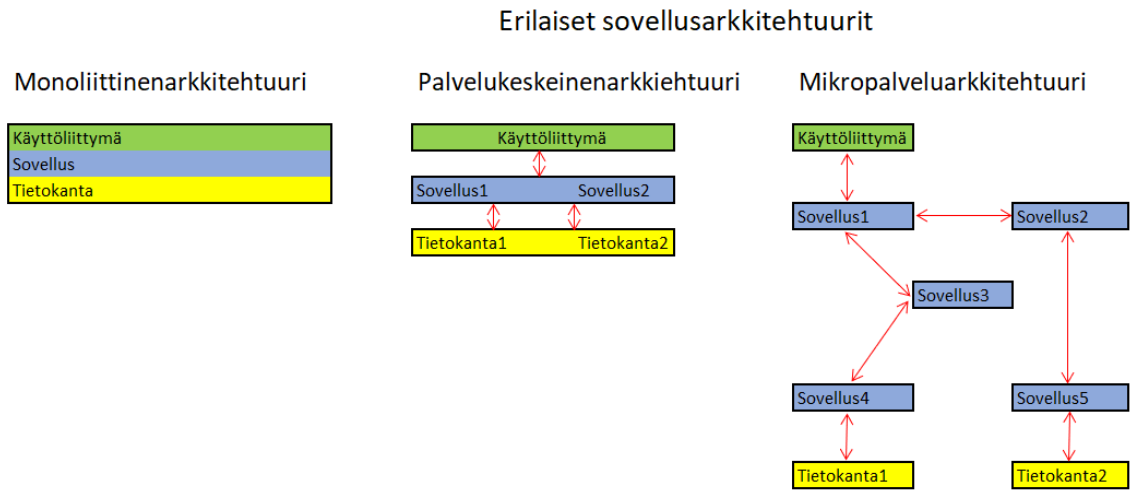
Palvelunetsintä eng. Service Discovery on Swarm Manager toiminallisuus, joka reitittää ja jakaa kuormaa toisillensa kommunikoivien sovelluskonttien kesken. Toiminnallisuus on tärkeä esimerkiksi silloin, kun klusterissa toimii järjestelmä, jonka toiminnallisuudet on pilkottu useampaan sovelluskonttiin. Palvelunetsintä mahdollistaa konttien juttelemisen toisillensa klusterin sisällä. (Docker 2021)

2.2 Mikropalveluarkkitehtuuri

Esimerkkejä konteissa ajettavista sovelluksista ja toiminnallisuuksista on verkkosivut, verkkokaupat, tietokannat, käyttöjärjestelmät ja paljon muuta. On mahdollista, että kaikki järjestelmän toiminnallisuudet ovat yhdessä kontissa, mutta on suositeltavaa, että konteissa ajettavat sovellukset käyttäisivät mikropalveluarkkitehtuuria.

Perinteisesti sovellukset ovat olleet monoliittisia tai palvelukeskeisiä, eli sovelluksen kaikki toiminnot ovat toimineet yhdessä instanssissa tai muutamassa instanssissa. Kontti ja pilvipalveluihin siirryttäessä tulisi uudet järjestelmät kehittää ja päivittää käyttämään mikropalveluarkkitehtuuria. Mikropalveluarkkitehtuuria käytettäessä saadaan täysi hyöty konttiteknologiasta. (IBM 2020)

Kuvasta 4 voidaan havainnollistaa eri sovellusarkkitehtuureiden ero. Eri sovellusarkkitehtuurien ero



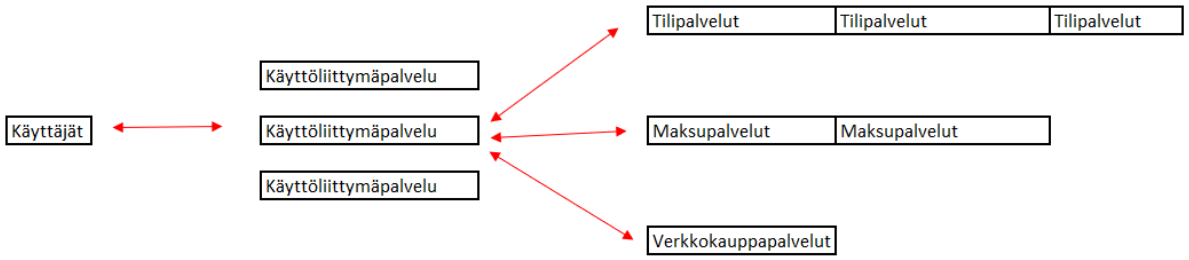
Kuva 4 Sovellusarkkitehtuurien erot

Mikropalveluarkkitehtuurista on hyötyä erityisesti sovelluskehittäjille. Kehittäjät voivat työskennellä pienempien kokonaisuuksien kanssa kerrallaan ja käyttää erilaisia koodikieliä, teknologioita ja tietokantoja palvelu- tai konttikohtaisesti. Lisäksi kehittäjien vastuut tarkentuvat sovelluskehittäjä tai ryhmä voidaan asettaa vastuuseen esimerkiksi yhdestä sovelluskontista. Mikropalveluarkkitehtuuri tuo myös vikasietoisuutta konttialusta automaattisten konttien uudelleenkäynnistysten kautta, sekä yksi virhe ei aiheuta koko palvelun kaatumista, koska eri palvelut ovat kytketty toisiinsa löyhällä kytkennällä eng. loose coupling (IBM 2020)

Löyhällä kytkennällä tarkoitetaan, sitä että palvelut ovat kytketty toisiinsa siten, että muutokset tai ongelmat yhdessä palvelussa vaikuta muihin palveluihin. Löyhässä kytkennässä palvelut kommunikoivat erilaisten API-rajapintojen kautta, jossa varmistetaan, että tieto pysyy samassa muodossa,

vaikka itse palveluun tehtäisiin muutoksia. Löyhä kytkentä on suunnitteluperiaate, jota noudattamalla palvelujen väliset riippuvuudet vähenevät ja vikasietoisuus lisääntyy. (Capitalone 2020)

Lisäetua tuo myös mikropalveluiden skaalautuvuus. Mikropalveluita voidaan skaalata palvelu kerrallaan kuvan 5 mukaisesti. Skaalaaminen mikropalvelukerrallaan säästää konesaliresursseja. Monoliittisessa järjestelmässä täytyisi skaalata sovellus kerrallaan.



Kuva 5 Esimerkki skaalautuvasta järjestelmästä

Mikropalveluarkkitehtuurin käyttö ei ole aina helppoa. Vianetsintä muuttuu erilaiseksi, kun kokonaisuuden toiminnallisuudet toimii usean toisiensa kanssa kommunikoivan sovelluskonttien kanssa. Lisäksi konttialusta, joka sisältää paljon toisillensa kommunikoivia kontteja vaatii paljon työtä ja asiantuntemusta. (IBM 2020)

2.3 Tuotteet ja teknologiat

Konttiteknologia on melko vanha keksintö IT-alalla. Ensimmäinen konttitekniikkaan viittaava Chroot-ohjelma keksittiin vuonna 1979. Chroot-ohjelma kehitettiin unix-käyttöjärjestelmälle ja Chroot ohjelmistolla pystytään rajaamaan esimerkiksi sovelluksen ajoa varten tehdyn käyttäjän

hakemistopuuta. Hakemistopuun rajaamisella sovelluksen prosessit eivät pääse käsiksi kuin sovelluskäyttäjän tiedostohakemistoon. (Manpages 2021)

Tämän jälkeen on kehitetty erilaisia teknologioita, jotka ovat mahdollistaneet konttiteknologian nykyisessä muodossaan.

Kuva 6 näyttää konttiteknologian alla toimivat erilaiset Linux-kernelin toiminnallisuudet, kuten Namespaces, Cgroups, SELinux ja SECCOMP. Lisäksi, jos kontteja ajetaan jonkin muun käyttöjärjestelmän päällä kuin Linuxin, niin vaatii käyttöjärjestelmä kyvyn virtualisoida Linuxia. (Opesource.com 2021) Esimerkiksi Windows käyttöjärjestelmissä täytyy olla toiminnallisuus Hyper-V käytössä, jotta siinä voidaan ajaa kontteja.

<table>
<tr><td>Kontti
Tiedostot
Käyttöliittymä</td><td>Kontti
Tiedostot
Tietokanta</td><td>Kontti
Tiedostot
Sovellus</td></tr>
<tr><td colspan="3">SELINUX / APPARMOR
SECCOMP
CGROUPS</td></tr>
<tr><td colspan="3">NAMESPACES
Hyper-V + windows / xhyve + mac / linux</td></tr>
<tr><td colspan="3">Pilvi tai laitteisto
esim.azure cloud, työasema, konesali</td></tr>
</table>

Kuva 6 Konttiteknologian alla toimivat Linux-kernelin toiminnallisuudet

Kontin ajoympäristö ja konttialustaa luullaan monesti samaksi asiaksi, mutta kontin ajoympäristö on konttialustan ominaisuus. Kun uusi kontti tallennetaan ja käynnistetään ajoympäristössä, niin silloin ajoympäristö järjestelee uudelle kontille ympäristön toimia näillä linux-kernelin toiminnallisuuksilla.

Suosituimpia ajoympäristöjä on esimerkiksi Containerd, jota käytetään yhdessä suosituimmassa konttialustassa Dockerissa. Containerd toimii siten, että kun Dockers-ohjelmassa käynnistetään kontti, toimii Containerd-ohjelma vuorovaikutuksessa Dockerssin kanssa. Containerdiin latautuu kopio kontista ja Containerd järjestelee kontille verkkoyhteydet sekä tallennustilan. Containerd toimii lisäksi vuorovaikutuksessa toisen Containerdin mukana tulevan komponentin Runc:in kanssa. (Iximiuz 2021)

Runc taas toimii vuorovaikutuksessa linux-kernelin kanssa ja järjestelee kernelin ominaisuuksilla kontille eristetyn tilan toimia tietokoneella. Runc-komponenttin prosessi poistuu, kun se on järjestellyt kontille tilan toimia containerd-shim- tallennustilassa. Containerd-shim on paikka, jossa kontti on tallennettuna. (Iximiuz 2022)

Cgroups eli control group on linux-kernelin ominasuus, jolla voi luoda erilaisia ryhmiä. Ryhmillä voidaan allokoida resursseja eri prosesseja. Resursseja joita cgroups hallinnoi on esimerkiksi suoritusaika, keskusmuisti ja verkon kaistan leveys. (Opesource.com 2021)

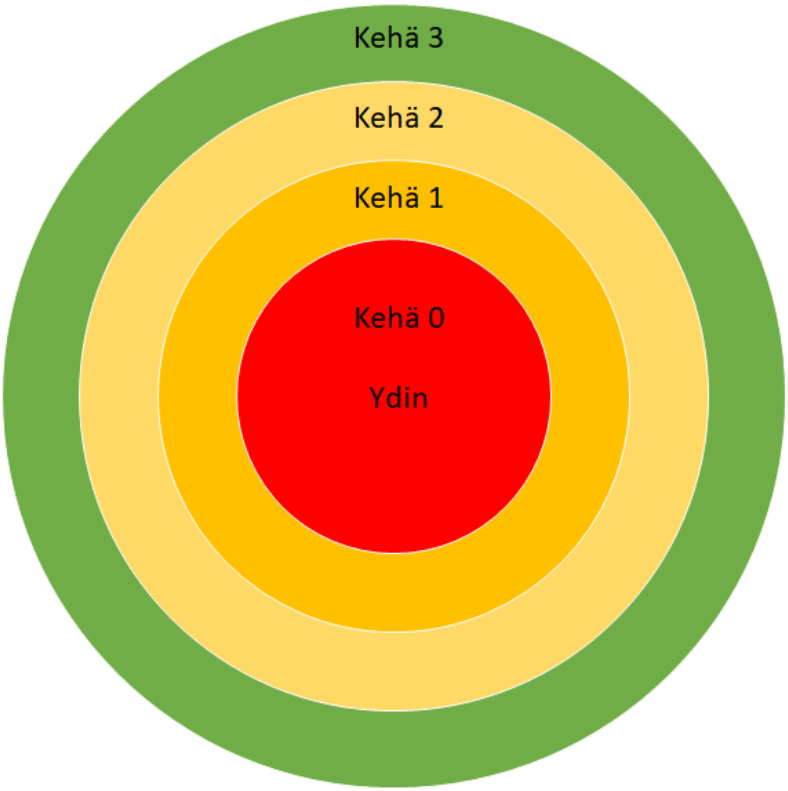
Konttia voidaan siirtää eri alustojen välillä, koska kontit ovat suunniteltu siten, että niillä on todella vähän riippuvuuksia tai ei ollenkaan. Eri alustoja on esimerkiksi yrityksen konesalissa toimivat palvelimet tai pilvipalvelutarjoajan pilvessä toimiva virtuaalipalvelin.

3 Virtuaalikoneet ja virtualisointitekniikka

Virtuaalikoneet on tietokoneohjelmalla toteutettuja näennäiskoneita, jotka emuloivat fyysisen tietokoneen toiminnallisuuksia. Tietokoneohjelmaa, joka toteuttaa virtualisointia kutsutaan virtualisointiympäristöksi tai virtualisointialustaksi (eng.hypervisor). Tietokonetta, jolla virtuaalikonetta ajetaan, kutsutaan isäntäkoneeksi. Virtuaaliympäristössä toimivaa virtuaalikonetta voidaan kutsua joko instanssiksi tai vieraskoneeksi (eng. guest machine). (Termipankki 2017)Virtualisoinnilla voidaan jakaa yksi fyysinen laitteisto useampaan käyttöjärjestelmäympäristöön. Virtualisointiympäristö mahdollistaa fyysisen koneen laitteistoresurssien käytön virtuaalikoneille. Fyysisen koneen laitteistoresursseja ovat suoritinaika, keskusmuisti, verkkoresurssit, sekä tallennustila. Seuraavissa kappaleissa käsitellään erilaisia tapoja, joilla laitteistoresurssien käyttö voidaan virtualisoida. Näihin kuuluvat paravirtualisointi, laitteistoavusteinen täysvirtualisointi, sekä ohjelmistoavustei**n**en täysvirtualisointi.

3.1 Kehämalli

Virtualisointitekniikkaan liittyy x86 prosessoriteknologiaan perustuvan laitteiston suoritustasot ja kehämalli. Kehämalli on tapa kuvata laitteiston päällä toimivien ohjelmien tärkeyttä ja oikeuksia suorittaa ohjelmakoodia laitteistossa. Kehämallissa suoritustasoja on neljä, mutta olennaisimmat näistä ovat 0 ja 3. Kehällä 0 toimii käyttöjärjestelmän ydin ja kehällä 3 toimii sovellukset. Käyttöjärjestelmän ydin mahdollistaa laitteistoresurssien hyödyntämisen käyttöjärjestelmälle ja muilla tasoilla toimiville sovelluksille. Kehällä 0 toimivalla ohjelmistoilla on eniten oikeuksia ja kehällä 3 vähiten. Kuvassa 7 voidaan nähdä prosessorinkehämalli. (Dittner & Rule. 2019, alaluku OS Relationships with the CPU Architecture)



Kuva 7 x86 prosessori arkkitehtuurin kehämalli

3.2 Virtualisointialustat

Virtualisointialustat jaetaan kahteen tyyppiin riippuen siitä ovatko ne käyttöjärjestelmiä vai käyttöjärjestelmän päällä toimivia virtualisointiohjelmistoja.

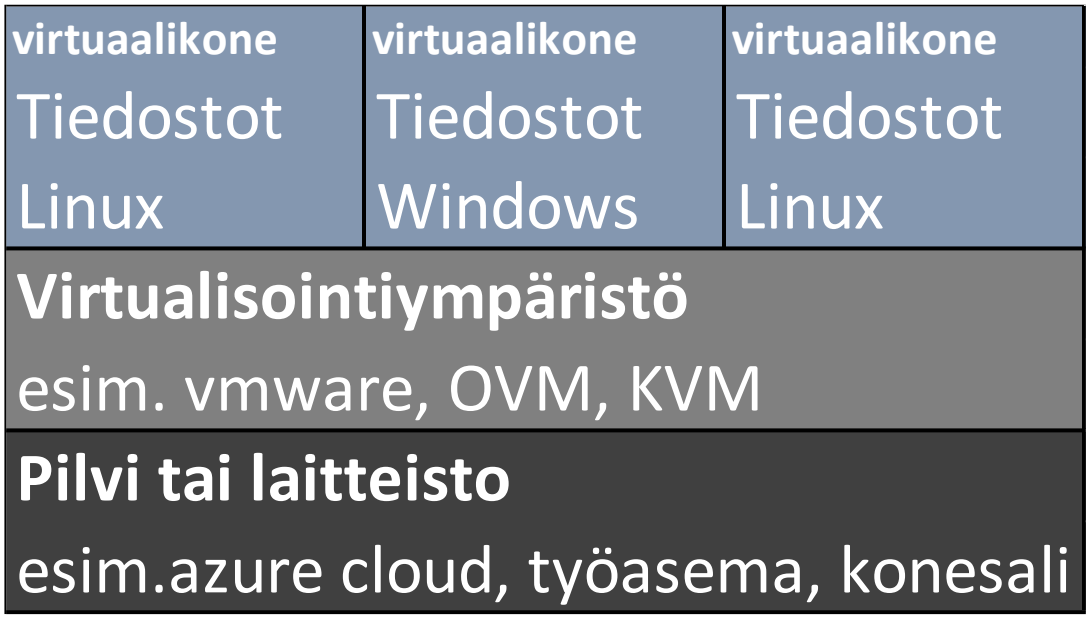
Virtualisointialustat tarjoavat virtuaalikoneille ympäristön toimia. Virtuaalialustat mahdollistavat useiden virtualisoitujen vieraskoneiden käyttää laitteistoresursseja samanaikaisesti.

Virtualisointialustat muodostavat laitteistosta resurssipoolin virtuaalikoneille. Virtualisointialusta kykenee jakamaan resursseja eri virtuaalikoneiden kesken tarpeen mukaan, joka mahdollistaa laitteistoresurssien paremman käyttöasteen. (Redhat 2020)

3.2.1 Tyypin 1 virtualisointialusta

Kuvasta 8 voidaan havainnoida esimerkkirakenteen tyypin 1 virtualisointialustalla toimivasta ratkaisusta.

Tyypin 1 virtualisointiympäristö toimii suoraan laitteiston päällä eli kehämallissa alueella 0. Tyypin 1 virtualisointialusta toimii yksinkertaisena käyttöjärjestelmänä virtuaalikoneille, jolla voi vain hallinnoida virtuaalikoneita. (Dittner & Rule. 2019, alaluku The Virtual Machine Monitor and Ring-0 Presentation)

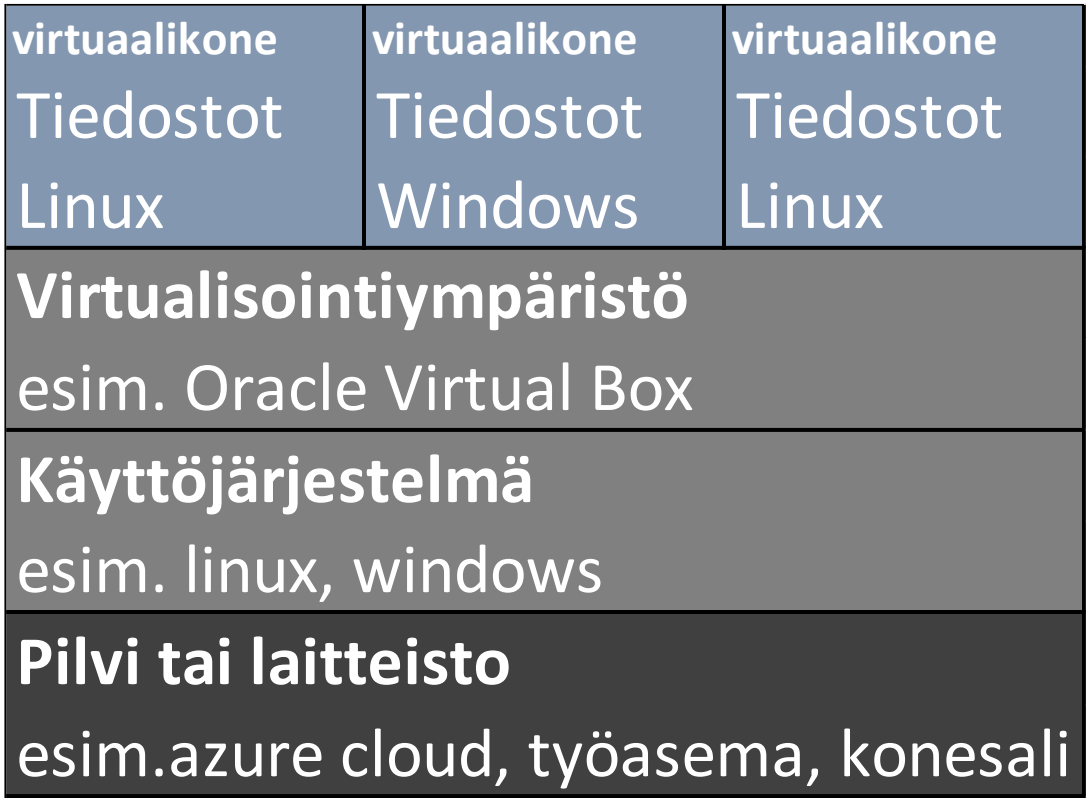


Kuva 8 Esimerkkirakenne tyypin 1 virtualisointialusta ratkaisusta

Esimerkkituotteita tyypin 1 virtualisointialustoista on VMware, Oracle Virtual Machine, Kernel-based Virtual Machine. Tyypin 1 virtualisointialustat ovat tarkoitettu yrityskäyttöön, koska ovat suorituskyvyltään parempia, sekä niissä on monia ylläpitoa helpottavia asioita. Lisäksi tyypin 1 virrutalisointialustat ovat usein maksullisia. (Phoenixnap 2019)

3.2.2 Tyypin 2 virtualisointialusta

Tyypin 2 virtualisointialustan esimerkkirakenteen voidaan nähdä kuvasta 9. Tyypin 2 virtualisointialustat toimivat isäntäkäyttöjärjestelmän päällä ja käyttöjärjestelmä kohtelee virtualisointialustaa kuin mitä tahansa sovellusta. Tyypin 2 virtualisointialustat toimivat kehämallin alueella 3. Tyypin 2 virtualisointialustojen eri komennot täytyy läpäistä useita eri kehämallin kerroksia, ennen kuin toiminto suoritetaan virtuaalikoneella. Virtuaalikoneet toimivat kauempana komentoja suorittavasta laitteistosta kuin tyypin 1 virtualisointialustat ja tästä syystä tyypin 2 virtualisointialustat kuormittavat enemmän laitteistoa. (Dittner & Rule. 2019, alaluku The Virtual Machine Monitor and Ring-0 Presentation)



Kuva 9 Esimerkkirakenne tyypin 2 virtualisointialusta ratkaisusta

Tyypin 2 virutalisointialustat ovat usein ilmaisia ja ovat helppokäyttöisiä. Tyypin 2 virutalisointialustat ovat usein tarkoitettu lähinnä testaus käyttöön, kuten uusien ohjelmien sekä projektien testaamiseen. (Phoenixnap 2019)

3.3 Täysvirtualisointi

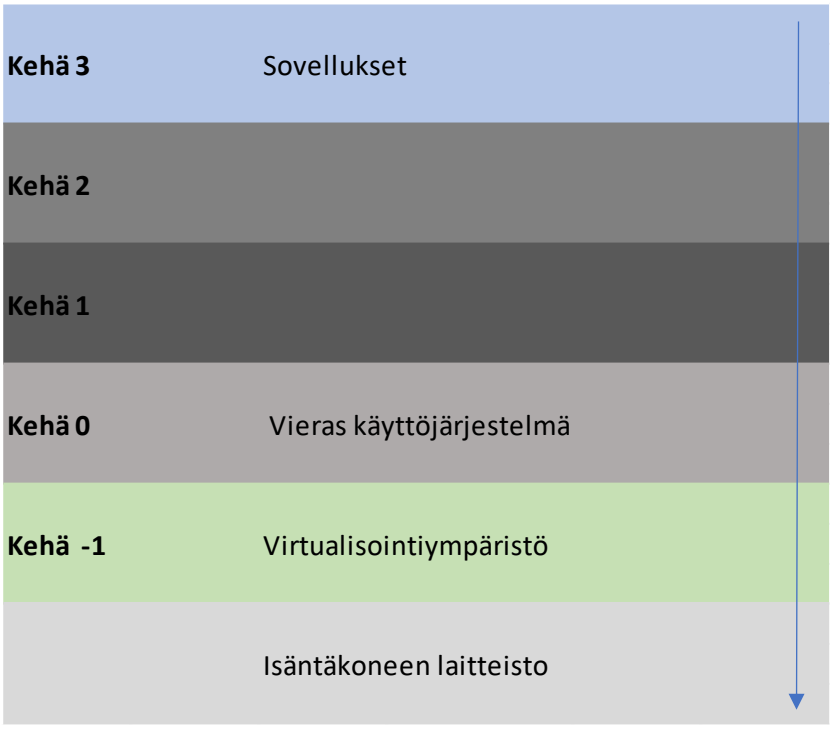
Virtualisointialustat voidaan jakaa kahteen tyyppiin, mutta erityyppisiä virtuaalikoneita on useampi. Täysvirtualisointi on tekniikka, jossa virtualisointialusta emuloi saman laitteiston vieraskoneelle millä virtualisointialusta toimii.

Täysvirtualisointi voi toimia kahdella eri tavalla. Täysvirtualisointi voi toimia ohjelmistoavusteisesti, jolloin virtualisointialusta käyttää binäärikäännöstä (eng. Binary Translation). Binäärikäännös on tekniikka, joka tarkkailee vieraskoneen ajamaa koodia. Binäärikäännöksessä vieraskoneen käskyt korvataan uusilla käskyillä, jos vieraskoneen käskyt ovat sellaisia, että voivat aiheuttaa virtualisointialustalle tai sen muille vieraskoneille ongelmia. Binäärikäännös tapahtuu reaaliaikaisesti, joten ohjelmistoavusteinen täysvirtualisointi kuormittaa laitteistoresursseja enemmän kuin ei virtualisoitu käyttöjärjestelmä. (VMWare 2007 s.4)

Lisäksi täysvirtualisointi voi toimia laitteistoavusteisesti. Laitteistoavusteisessa virtualisoinnissa

tarvitaan x86 prosessoriarkkitehtuuria käyttävän suorittimen olla sellainen, joka tukee Intel VT-x tai AMD-V teknologioita. Teknologiat ovat kahden suurimman prosessorivalmistajan teknologioita ja mahdollistavat usean vieraskoneen ajamisen samalla laitteistolla ilman ohjelmallista erottelua. Teknologiat mahdollistavat virtualisointialustan toimimisen -1 kehällä, jolloin vieraskäyttöjärjestelmät voivat toimia kehällä 0. Laitteistoavusteisesti virtualisoidulla vieraskäyttöjärjestelmällä on suora pääsy laitteistoresursseihin prosessoriteknologian ansioista. (Chirammal, Mukhedkar, & Vettathu 2016, alaluku Hardware assisted virtualization)

Kuvasta 10 voidaan havainnoida, että laitteistoavusteinen virtualisointi ei noudata täysin perinteistä kehämallia, vaan VT-x ja AMD-V prosessoriteknologioiden ansiosta vieraskäyttöjärjestelmä voi toimia perinteisen kehämallin mukaan 0 kehällä. Verrattuna kehämalliin….



Kuva 10 laitteistoavusteinen virtualisointi

3.4 Paravirtualisointi

Paravirtualisoinnissa virtualisointiympäristössä ajetaan sopivaksi muokattuja vieraskäyttöjärjestelmiä. Paravirtualisoinnissa virtualisoimattomat järjestelmäkutsut ovat jo valmiiksi muokattu käyttöjärjestelmään. Järjestelmäkutsut ovat korvattu hyperkutsuilla (eng. hypercall). Virtualisointiympäristö tarjoaa hyperkutsurajanpintoja kernelin toiminnoille, kuten muistinhallinnalle ja prosessorin toiminnoille.

Paravirtualisoinnin hyötyjä on parempi suorituskyky, koska paravirtualisoidut käyttöjärjestelmät eivät tarvitse kuormittavaa binäärikäännös tekniikkaa, koska järjestelmäkutsut ovat jo valmiiksi käännetty. Paravirtualisoinnnin haittoja on taas hankalampi tuki ja ylläpito, koska muokattuja käyttöjärjestelmiä on rajallinen määrä. Lisäksi paravirtualisoituja ympäristöjä on vaikeaa siirtää eri alustojen välillä. (VMWare 2007 s.5) Tunnettuja tuotteita, jotka tukevat paravirtualisoituja ratkaisuja ovat esimerkiki Xenserver ja Oracle Virtualbox (ServerWatch 2020)

4 Tutkimuksen toteutus

Tutkimusosuudessa suunniteltiin esimerkkijärjestelmän arkkitehtuuri ensiksi perinteisellä virtualisointitekniikalla toteutettuna ja sitten konttiorkestroinnilla toteutettuna. Kukin järjestelmä kuvattaan kappaleessa 5.1.1 ja 5.1.3 käyttäen arkkitehtuurikuvaa esimerkkitoteutuksesta. Esimerkkijärjestelmiä käytetään havainnollistamaan sitä, millaisen määrän resursseja kukin toteutustapa tarvitsee. Esimerkkijärjestelmässä käytetään tuotteiden resurssien suositusarvoja, jotka ovat tuotteiden valmistajien ja eri palveluntuottajien määritelmiä suositusarvoja.

Järjestelmät piirretään korkean saatavuuden periaatetta käyttäen, koska silloin saadaan enemmän otantaa tutkimukseen. Korkean saatavuuden järjestelmässä kaikki on vähintään kahdennettu. Tämä tarkoittaa sitä, että jos jokin järjestelmän palvelin hajoaa tai kaatuu, niin pystyy järjestelmä palvelemaan käyttäjiä normaalisti muilla samaa palvelua ajavan palvelimen turvin. (Cisco 2022)

Sovellusten ja tietokantojen resurssitarvetta on vaikea arvioida etukäteen, ellei kyseessä ole valmis tuote, jonka resurssitarpeet ovat testattu erilaisilla kuormilla. Tutkimusasetelma onki teoreettinen, sillä oikeaa järjestelmää ei rakenneta käytännössä. Siitä huolimatta tulokset antavat hyvin suuntaa oikeaan tilanteeseen.

Sovellusten ja tietokantojen resurssitarpeet selviävät yleensä vasta, kun virtualisoitua järjestelmää ajetaan erilaisilla käyttäjämäärillä ja asetuksilla. Tietokantojen resurssien käyttöön vaikuttaa tietokannan koko, tietokannan käsittelemä tieto, tapahtumien määrä, sekä tietokantatuotteen toimivuus. Sovellukseen vaikuttaa se, että mitä sovellus tekee ja kuinka paljon sitä käytetään. Tästä syystä opinnäytetyössä käytetään ennalta sovittuja tarvemääriä tietokantojen ja sovellusten osalta

Tuotteiden ja sovellusten osalta testissä käytetään niiden oikeita resurssisuosituksia. Tuotteilla tarkoitetaan esimerkiksi käyttöjärjestelmää, virtualisointialustaa, kuormanjakotuotetta, tietokantatuotetta ja sovelluspalvelintuotetta. Esimerkiksi verkkokaupalla on eri toimintoja palvelussaan, jotka ovat käyttöliittymä, kirjautumistoiminnot, tuotekatalogi, maksutoiminnot ja postitustoiminnot. Joskus nämä toiminnot ovat omia sovelluksia kuten mikropalveluarkkitehtuuria käytettäessä, mutta koska virtualisointialusta taipuu todella huonosti mikropalveluarkkitehtuurilla toteutettuun ratkaisuun, niin testissä käytetään palvelukeskeistä arkkitehtuuria. Lisäksi osalla näistä sovelluksista saattaa olla tietokanta takanaan.

Tutkimuksessa on käytetty lähteinä vapaasti saatavilla olevaa kirjallisuutta, joka on valittu sillä perusteella, että lähteet ovat eri ammattilaisten kirjoituksia, suurien tunnettujen yritysten julkaisua ja valmistajien dokumentaatiota tuotteista.

4.1 Laitteistoresurssien vertailu

Laitteistoresurssien käyttöä mitattiin kahdella mittarilla. Ensimmäinen mittari on se, että kuinka paljon virtuaalipalvelimelle tai konttialustalle täytyy varata resursseja. Resurssien varaaminen tarkoittaa sitä, että palvelimelle asetetaan ylärajat keskusmuistille, kovalevytilalle ja prosessoriteholle. Varaaminen ei tarkoita sitä, että resurssit olisivat käytössä, mutta se tarkoittaa sitä, että palvelimien ylläpitäjien on täytynyt varautua siihen, että ne olisivat käytössä.

Resurssien loppuminen saattaa aiheuttaa ongelmia palvelimella kuten hidastumista ja jopa palvelun tai palvelimen kaatumisen, siksi palvelimelle olisi määriteltävä sellainen määrä resursseja, että resurssit eivät lopu edes silloin kun palvelimella toimivaa palvelua käytetään paljon.

Toinen mittari on se, kuinka paljon se todellisuudessa kuluttaa, eli kuinka monta prosenttia prosessori käyttää laskentatehosta, kuinka paljon todellisuudessa käyttää kovalevyä ja kuinka paljon sovellus käyttää keskusmuistia.

4.2 Laitteistoresurssien testaaminen

Laitteistoresurssien mittaamista varten piirretään kuvitteellinen järjestelmä, joka vastaisi kuvitteellisen yrityksen järjestelmää. Järjestelmä koostuu samoista sovelluksista ja tietokannoista mistä oikeatkin järjestelmät koostuvat.

Ensimmäiseksi järjestelmä sovitetaan arkkitehtuuriin, joka käyttää järjestelmän alustana virtuaalikoneita ja tämän jälkeen arkkitehtuuriin, joka käyttää järjestelmän alustana konttien orkestrointijärjestelmää.

Järjestelmät piirretään korkean saatavuuden periaatetta käyttäen. Korkean saatavuuden järjestelmässä kaikki on vähintään kahdennettu. Tämä tarkoittaa sitä, että jos jokin järjestelmän palvelin hajoaa tai kaatuu, niin pystyy järjestelmä palvelemaan käyttäjiä normaalisti muilla samaa palvelua ajavan palvelimen turvin. (Cisco 2022)

5 Konttiteknologian hyödyt verrattuna virtuaalikoneisiin

Tässä kappaleessa tarkastellaan konttiteknologian hyötyjä kolmesta näkökulmasta jotka ovat: laitteistoresurssien käyttö, sovellusten käyttöönotto ja kehitys, sekä järjestelmän siirtämisen helppous eri laitteistojen välillä. Ensiksi esitellään esimerkkiarkkitehtuurit. Lopuksi esitellään vertailun tulokset.

5.1 Esimerkkiarkkitehtuuri virtualisointia käyttäen

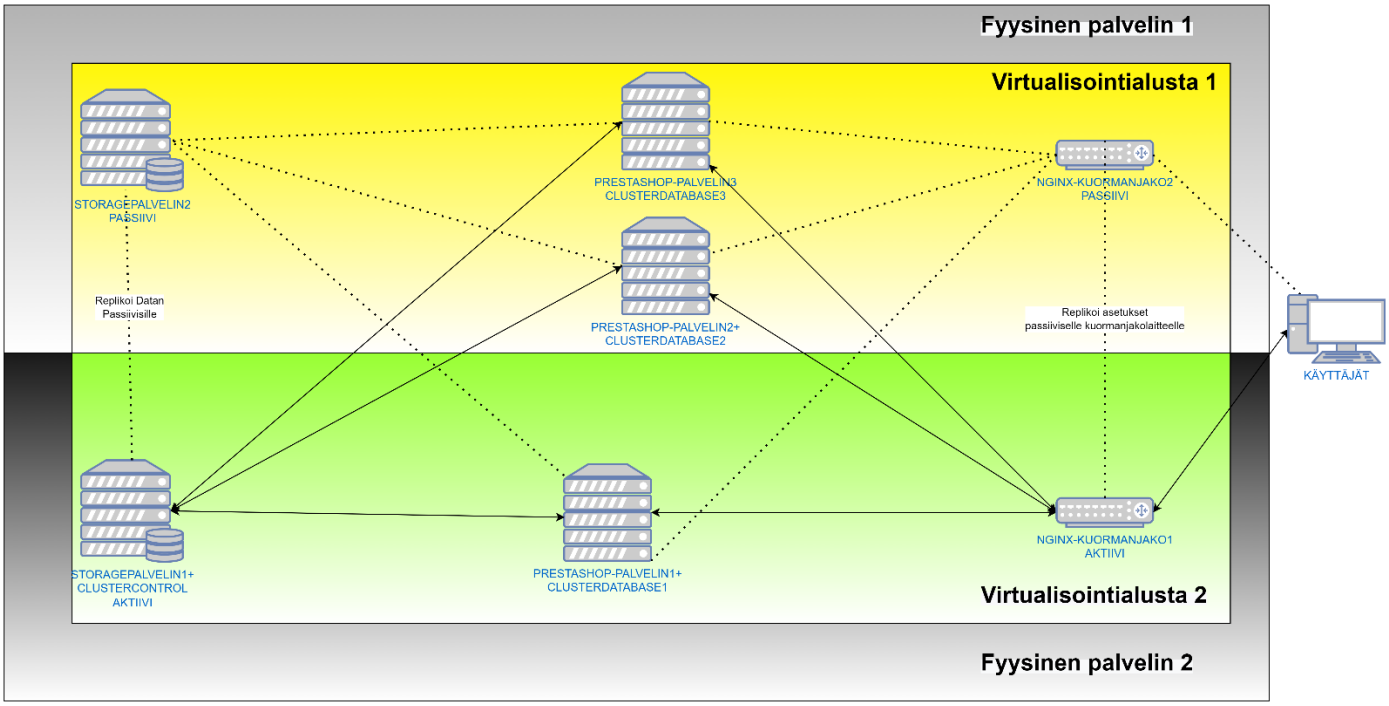
Sovellusten ja tietokantojen resurssitarvetta on vaikea arvioida etukäteen, ellei kyseessä ole valmis tuote, jonka resurssitarpeet ovat testattu erilaisilla kuormilla. Sovellusten ja tietokantojen resurssitarpeet selviävät yleensä, kun järjestelmää ajetaan erilaisilla käyttäjämäärillä ja asetuksilla. Tietokantojen resurssien käyttöön vaikuttaa tietokannan koko, tietokannan käsittelemä tieto, tapahtumien määrä, sekä tietokantatuotteen toimivuus.

Koska tietokannan ja sovelluksen käyttämiä resursseja on vaikea määrittää, arvioinnissa on käytetty Appedology- yrityksen tarjoamien verkkokauppa palvelimien suositusarvoja. Tällaiselle sovellukselle, ja sen vaatimalle tietokannalle suositellaan yhteensä 200 gigatavua kovalevyä, 8-ydinprosessoria, sekä 16 gigatavua keskusmuistia. (Appeldology 2021)

Testissä käytetään Red Hat Enterprise Linux käyttöjärjestelmää virtuaalipalvelimilla, koska Red Hatin tuotteet ovat yleisesti käytettyjä ammattimaisissa ratkaisuissa, sekä Red Hat:in tuotteista on paljon tietoa saatavilla.

Testiin valitaan PrestaShop-verkkokauppa-alusta. PrestaShop on valmis verkkokauppaohjelmisto. PrestaShop tarvitsee minimissään yhden palvelimen. Korkean saatavuudella suunniteltu järjestelmä tarvitsee kumminkin kuormanjaon, useamman sovelluspalvelimen, sekä tietokantapalvelimen. PrestaShop on hyvä valinta esimerkkijärjestelmäksi, koska se voidaan asentaa sekä konttiorkestroinnilla toimivaksi ja virtualisoinnilla toimivaksi

Kuvassa 11 voidaan nähdä ratkaisupiirros korkealla saatavuudella toteutetusta verkkokaupparatkaisusta. Kuvasta löytyy yhteensä 7 virtualisoitua palvelinta ja kaksi fyysistä palvelinta, joihin on asennettu virtualisointialusta.



Kuva 11 Korkean saatavuuden prestashop-verkkokauapparatkaisu. Yhtenäiset viivat kuvaavat käytössä olevaa yhteyttä ja katkoviivat kuvaavat varayhteyksiä. Ratkaisu perustuu Serveralnines yrityksen Prestashop-ratkaisuun pienillä muutoksilla (Serveralnines 2014)

Virtualisointialusta tarvitsee muistia, prosessoritehoa, sekä kovalevyä toimiakseen. Testissä käytetään KVM-virtualisointiratkaisua, joka perustuu laitteistoavusteiseen täysvirtualisointiin ja on täten tyypin 1 virtualisointialusta. (Redhat 2022)

KVM-virtualisointialustan suositusresurssimäärä on 4-ydinprosessori, 16gb keskusmuistia, sekä 50gb kovalevyä. (Redhat 2022) Todellisuudessa kuvan virtualisointialustat tarvitsevat paljon enemmän fyysisiä resursseja, koska verkkokaupan järjestelmät virtualisoidaan virtualisointialustalle. Tämä tarkoittaa sitä, että laitteistoresurssit on löydyttävä virtualisointialustalla löytyvältä koneelta, jotta alusta voi jakaa niitä virtualisoiduille palvelimille.

Virtualisoiduille palvelimille ei tarvitse kumminkaan löytyä täsmälleen samaa määrää fyysisiä laitteistoresursseja palvelimelta, vaan virtualisoiduille palvelimille voidaan antaa enemmän resursseja käyttöön mitä fyysiseltä palvelimelta oikeasti löytyy. Tämä perustuu siihen, että esimerkiksi kuvassa 11 löytyy passiivisia laitteita, jotka eivät syö läheskään niin paljon resursseja kuin niille on osoitettu. Passiivinen laite on laite, joka on olemassa varalta, jos aktiivinen menee vikatilaan.

Lisäksi kuvasta 11 löytyy Nginx kuormanjakopari. Nginx kuormanjako on virtualisoitu palvelin, josta löytyy Nginx-ohjelmisto. Nginx ohjelmiston tehtävänä on tasapainottaa kuormaa verkkokaupan käyttäjien välillä ohjaamalla heidän web-selainten pyynnöt eri sovelluspalvelimille kuorman mukaan. Kuormanjaon tehtävänä on lisäksi välittää järjestelmän sisäistä liikennettä kuten tietokantaklusterin liikennettä, sekä lähettää erilaisia tilatarkistuksia (eng. healthcheck) järjestelmän muille palvelimille. Tilatarkistusten tehtävän on selvittää viallisesti toimivat laitteet ja ohjata tarvittaessa liikenne toimiville palvelimille. Nginx kuormanjakopalvelimia on kaksi, toinen passiivinen ja toinen aktiivinen. Aktiivisen palvelimen vioittuessa passiivinen ryhtyy jakamaan kuormaa.

Nginx kuormanjakopalvelimelle suositellaan laitteistoresursseiksi 8-ydinprosessori, 8 gigatavua keskusmuistia, sekä 255 gigatavua kovalevytilaa. (Nginx 2022)

Kuvasta voidaan löytää lisäksi storage-palvelimet. Storage-palvelimen tehtävänä on luoda jaettu levy PrestaShop-palvelimelle. Jaetulla levyllä säilytetään PrestaShop-verkkokaupan sovellusta. Jos sovellukseen tehdään konfigurointimuutoksia, niin muutokset kopioituvat kaikille palvelimille jaetun levyn välityksellä.

Storage-palvelin sisältää myös MySQL Galera klusterin Cluster Control-ohjelman. Ohjelmalla voidaan konfiguroida Galera klusterille asetuksia. Galera Clusteri on klusteroitu tietokanta, jonka toimii PrestaShop-sovellukselle tietokantana. Varsinaiset tietokantasolmut toimivat samoilla palvelimilla, kuin PrestaShop-sovellus ja synkronoivat tietokannan tietoja kuormanjaon välityksellä. (Severalnines 2014)

Storage-palvelin ei ole kovalla kuormituksella, niin sille allokoidaan 4-ydinprosessori, 8gigatavua keskusmuistia ja 250gigatavua kovalevyä. Resurssien määrä perustuu Red Hat-yrityksen linux-käyttöjärjestelmän suosituksiin. (Redhat 2021)

5.2 Yhteenveto virtuaalipalvelimien resurssitarpeista

Yhteenvetona voidaan laskea, että kuvan 11 järjestelmä tarvitsee laitteistoresursseja seuraavasti:

Palvelimet	Prosessoriytimien määrä	Keskusmuisti(Gt)	Kovalevytila (Gt)
Nginx kuormanjako passiivinen	8	8	255
PrestaShop palvelin 2	8	16	200
Prestashop palvelin 3	8	16	200
Stroage-palvelin 2 passiivinen	4	8	200
KVM-virtualisointialustaohjelma	4	16	50
Fyysinenpalvelin 1 yhteensä	**32**	**64**	**905**

Fyysisen palvelin 1 käyttämät laitteistoresurssit

Palvelimet	Prosessoriytimien määrä	Keskusmuisti(Gt)	Kovalevytila (Gt)
Nginx kuormanjako aktiivinen	8	8	255
Prestashop palvelin 1	8	16	200
Stroage-palvelin 1 aktiivinen	4	8	200
KVM-virtualisointialustaohjelma	4	16	50
Fyysinenpalvelin 2 yhteensä	**24**	**48**	**705**

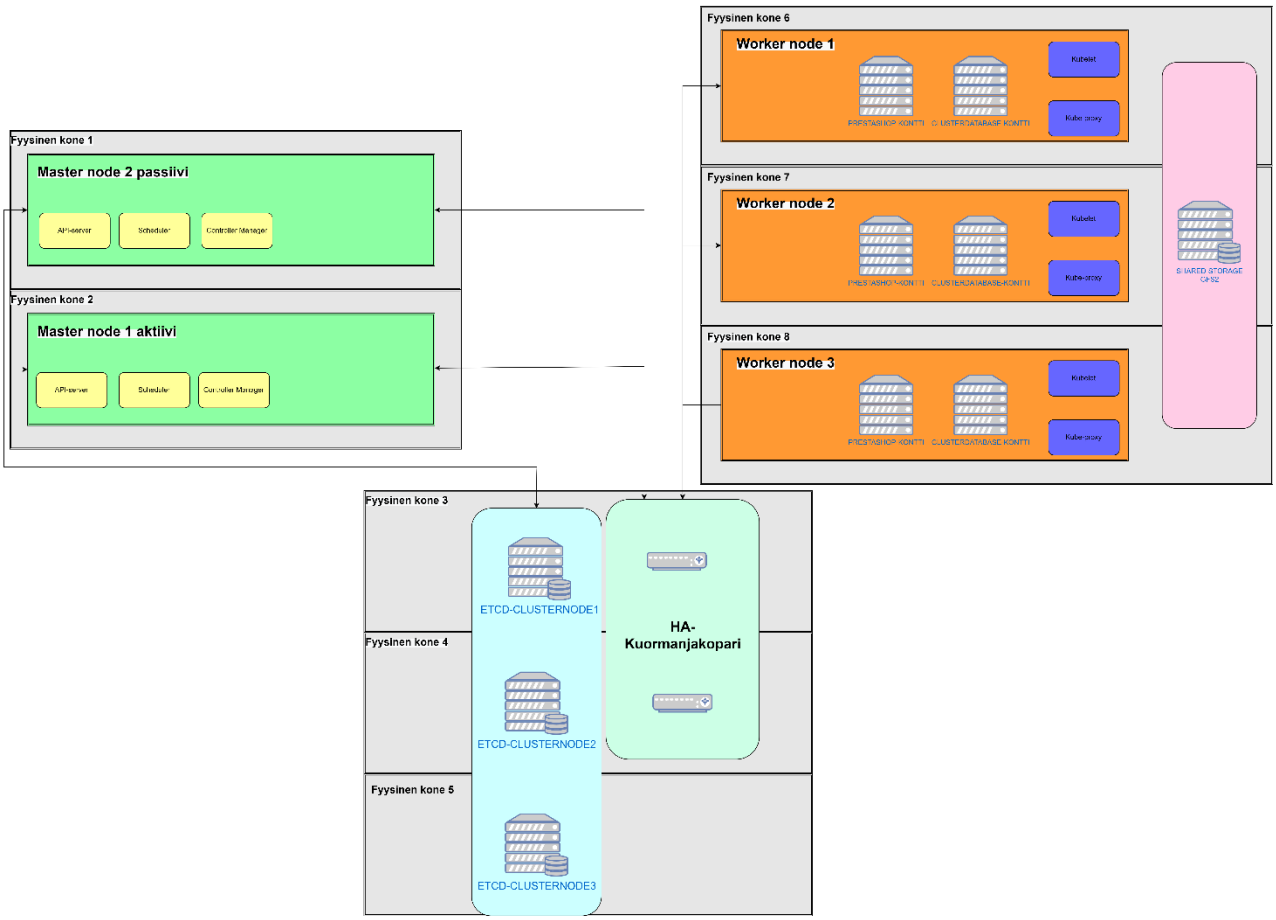
Fyysinen palvelin 2 käyttämät laitteistoresurssit

Koska fyysinen palvelin 1:llä on kaksi passiivista laitetta, voidaan molemmille fyysisille palvelimille asettaa tarpeeksi 24 prosessoriydintä, 48 gigatavua keskusmuistia, sekä 1 teratavu kovalevytilaa.

Virtualisoinnilla voidaan antaa järjestelmän palvelimille enemmän resursseja, mitä fyysiseltä palvelimelta löytyy, jos tiedetään etteivät fyysiset laitteistoresurssit ole käytössä 100 prosenttisesti. (Redhat 2022b)

5.3 Konttiorkestroinnin resurssitarpeet ja tuotteet

Konttiratkaisu piirretään käyttämällä pohjana suosituinta orkestrointiohjelmaa, Kubernetesiä.



Kuva 12 HA-ratkaisu Kubernetes-tuotteella. Ratkaisu perustuu Faun Publicationin blogi julkaisussa olevaan kubernetes alustaratkaisuun (FAUN 2019) Lisäksi ratkaisuun on lisätty PrestaShop-tuotteen pakolliset komponentit.

Ratkaisusta löytyy 2 master nodea, aktiivinen ja passiivinen. Master noden tarkoituksena on ohjata Worker nodeja. Toinen on aktiivisena koko ajan ja jos aktiiviselle käy jotain, niin passiivinen muuttuu aktiiviseksi.

Master nodet ei tarvitse paljon resursseja tämän kokoisessa järjestelmässä, joten niille voidaan osoittaa seuraavat resurssien suositusarvot 2 prosessoriydintä, 8 gigatavua keskusmuistia ja 100 gigatavua kovalevyä. (DellTechnologies 2022)

Kuvasta 11 löytyvä etcd-klusteri toimii avainarvokantana Kubernetes-tuotteelle. Avainarvokannassa säilytetään erilaisia tietoja, joiden avulla Kubernetes pystyy orkestroimaan konttien toimintaa. (Redhat 2019) Etcd-kanta tai vastaava tuote on pakollinen Kubernetes-tuotteelle.

Etcd-klusteri tarvitsee toimiakseen vähintään 3 nodea. Etcd-tuotteen noden laitteistosuositukset pienissä ympäristöissä on 2 prosessoriydintä, 8 gigatavua keskusmuistia. Etcd-tuote ei juuri tarvitse kovalevytilaa, joten 10gigatavua riittäneen.

HA-klusteripari tarvitsee saman verran resursseja, kuin virtualisoinnilla, joten arvot pysyvät samoina näiden osalta.

Worker-nodejen pidetään samoina kuin virtualisoinnissa olevien palvelimet, eli 8 prosessoriydintä, 16 gigatavua keskusmuistia, sekä 200gigatavua kovalevyä. Lisäksi Samoille palvelimille on asennettu GFS2-Shared storage. GFS2 on kirjainlyhenne sanoista Global File Storage. Global File Storage on ohjelmisto, jolla voidaan replikoida samat tiedostot palvelimille käytettäväksi.

Kontit eivät oletusarvoisesti tallenna mitään itseensä, niin on muuttuvat tiedot tallennettava erilliseen varastoon, esimerkiksi tietokantakontin tietokanta on tallennettava sijaintiin, joka on pysyvä, sekä on saatavilla jokaisella Worker-nodella.

Saatavuus on oltava jokaisella worker-nodella, koska konttijärjestelmä saattaa käynnistää kaatuneen kontin toisella Worker-nodella esimerkiksi tasapainottaakseen kuormaa.

5.4 Yhteenveto konttiorkestroinnin resurssitarpeista

Fyysisetpalvelimet ja tuotteet	Prosessoriytimien määrä	Keskus-muisti(Gt)	Kovalevy-tila (Gt)
Fyysinen kone 1 + Masternode 2	2	8	100
Fyysinen kone 2 + Masternode 1	2	8	100
Fyysinen kone 3 + Etcd node 1 + Kuormanjako 1	10	10	265
Fyysinen kone 4 + Etcd node 2 + Kuormanjako 2	10	10	265
Fyysinen kone 5 + Etcd node 3	2	8	60
Fyysinen kone 6 + worker node 1 + prestashop + database	8	16	200
Fyysinen kone 7 + worker node 2 + prestashop + database	8	16	200
Fyysinen kone 8 + worker node 3 + prestashop + database	8	16	200
Järjestelmä yhteensä	**50**	**92**	**1390**

Konttiorkestroinnilla toteutetun järjestelmän komponenttien yhteenlasketut resurssivaatimukset

Orkestrointityökalulla toimiva järjestelmä vie hieman vähemmän resursseja suositusresurssiarvoilla mitattuna, kuin täysvirtualisoinnilla toimiva järjestelmä tai ainakin PrestaShop verkkokauppaohjelman osalta.

6 Konttiorkestroinnin hyödyt järjestelmien kehitystyössä ja käyttöönotossa

Organisaatioissa on monesti käytössä erilaisia kehitys-, testi-, laadunvarmistus- ja tuotantoympäristöjä järjestelmille. Erilaisten ympäristöjen tarkoitus on varmistaa, että kokonaisuus eli järjestelmä ja sen komponentit toimivat odotetusti ennen kuin järjestelmä menee tuotantokäyttöön. Järjestelmiä testataan erilaisilla testeillä ja osallistujakokoonpanoilla eri ympäristöissä, kunnes voidaan todeta, että järjestelmä tai sovellus on valmis tuotantokäyttöön. (2022 Altexsoft)

Erilaisia ympäristöjä voi olla sovellustoimittajalla, sekä asiakasorganisaatiolla ja sovelluksia voidaan testata sekä asiakkaan ympäristöissä ja sovellustoimittajan ympäristöissä, jonka vuoksi testeistä voi tulla eri tuloksia johtuen ympäristöjen erilaisuudesta.

6.1 Siirrettävyys

Sovelluskonttia voidaan siirtää ja kopioida sen jälkeen helposti, kun kontti on luotu. Sovelluskontteja voidaan luoda, ajaa ja testata työasemalla, palvelimella ja pilvessä. Sovelluskontti ei tarvitse kuin konttialustaohjelmiston. Työasemalla ajettavuus tuo hyötyjä esimerkiksi testauksessa.

Sovelluskehittäjä voi ohjelmoida sovelluksen toimimaan kontissa ja tehdä testit ennekuin sovellus siirretään esimerkiksi orkestrointialustalle toimimaan muiden sovelluskonttien kanssa. Työasemalla toimiva sovelluskontti toimii suurella todennäköisyydellä samanlailla myös muilla alustoilla. (IBM 2019)

Virtualisointialustalla olevia palvelimia voidaan myös kopioida, mutta palvelimen yksilöiviä tiedostoja täytyy muuttaa manuaalisesti. Esimerkiksi, jos palvelimet kommunikoivat toisillensa nimillä ja IP-osoitteilla, niin täytyy nämä käydä konfiguroimassa vanhoille palvelimille ja tarvittaessa muuttaa uuden palvelimen asetuksia siten, ettei palvelinympäristössä ole kahta samanimistä palvelinta.

Konttiorkestrointiohjelmistoon voidaan rakentaa taas älyä, että se tunnistaa uuden kontin automaattisesti uudeksi instanssiksi ajaa sovellusta. Konttialustan sisällä kontit kommunikoivat toisillensa ja muille komponenteille virtuaaliverkon avustuksella, joka kuuluu orkestrointialustan ominaisuuksiin. Orkestrointialustalle määritykset tarvitsevat tehdä kerran ja tämän jälkeen konttialusta voi kopioida konttia toimimaan mulla worker nodeilla, jolloin palvelu saa enemmän resursseja vastata sovellusten ja asiakkaiden tekemiin pyyntöihin.

6.2 Riippuvuudet

Tilanteessa, jossa sovellus siirtyy sovelluskehittäjältä asiakkaan käyttöön testattavaksi tai käytettäväksi voi tulla erilaisia ongelmia, kun asiakas ajaa sovellusta omilla laitteillaan, koska sovellukset saattavat olla riippuvaisia erilaisista lisäohjelmista ja koodikirjastoista. Konttiin nämä riippuvuudet

voidaan pakata jo sovellustoimittajan toimesta, silloin organisaation tietojärjestelmien ylläpidosta vastaavan tiimin tai henkilön ei tarvitse erikseen asentaa tai pohtia riippuvuuksia.

Virtuaalipalvelimille, sekä normaaleille rautapalvelimille nämä riippuvuudet tulee asentaa erikseen, joka vaatii manuaalista työtä. Virtuaalipalvelimilla saattaa olla lisäksi eri käyttöjärjestelmiä ja eri versiota niistä tai eri versioita riippuvuuksista, joka saattaa edelleen hankaloittaa asennuksia. Kontit eivät juuri välitä missä ne toimivat, koska ajettavasta kontista löytyy kaikki samat riippuvuudet, mitä siellä on ollut, kun se on lähtenyt sovellustoimittajalta asiakkaalle.

6.3 Poikkeukset

Kontteihin voi vaikuttaa isäntäkoneen laitteistoresurssit. Erilaiset laitteistoresurssit kuten prosessori, keskusmuisti, grafiikkapiiri voivat tuoda epävakautta kontin suorittamiseen, mutta ei sen enempää, kun virtualisoinnilla testattaviin sovelluksiin. (Limepoint 2019)

6.4 Kehitystyö

Kehitystyö saattaisi olla helpompaa mikropalveluarkkitehtuuria ja kontteja käytettäessä, silloin kun uusia järjestelmiä ja sovelluksia kehitetään, koska kehittäjät voisivat työskennellä yhden kontin ympärillä kerrallaan. Lisäksi kehittäjät voisivat käyttää eri ohjelmointikieliä eri sovelluskonteissa kehittäjien parhaaksi katsomalla tavalla. Jos kehittäjiä olisi useampi taho, niin vastuut voitaisiin jakaa helposti kontteihin.

Lisäksi kehittäjien ei tarvitsisi ymmärtää muiden konttien koodia perinpohjaisesti, vaan vain sen kuinka ne kommunikoivat löyhien liitosten avulla muille konteille. Sovelluskontteja voitaisiin myös päivittää mikropalvelu kerrallaan, joka nopeuttaisi päivitysten tekemistä. (IBM 2020)

7 Pohdinta ja oma oppiminen

Tutkimuksen tavoitteena on selvittää miksi konttiteknologialla toimivat palvelut voisi olla parempi valinta organisaatiolle kuin perinteiset virutaalikoneratkaisut. Opinnäytetyössäni tarkastelen hyötyjä kolmesta näkökulmasta: laitteistoresurssien käyttö, sovellusten käyttöönotto ja kehitys, sekä järjestelmän siirtämisen helppous eri laitteistojen välillä.

7.1 Laitteistoresurssit

Laitteistoresurssien käytössä suositusresurssiarvoilla mitattuna konttiorkestroinnilla toimiva järjestelmä tarvitsi vähemmän resursseja. Toisaalta konttiorkestroinnilla toimiva ratkaisu tarvitsi 8 fyysistä palvelinta, joiden ylläpito voisi olla hieman raskasta. Perinteinen HA-virtualisointialustaratkaisu tarvitsi vain 2. Esimerkiksi laitteistoresurssien säätäminen on paljon vaikeampaa fyysisessä palvelimessa, kuin pilvessä tai virtualisointialustalla. Laitteistoresurssien säätäminen tarvitsisi koneen sammutuksen, sekä koneen pitäisi vaihtaa tehokkaampi prosessori tai lisää muistia käsin tai vaihtoehtoisesti vähentää resursseja. Virtualisointialustalla voisi olla tietenkin sama ongelma edessä, jos fyysiseltä koneelta loppuisi resurssit, mutta virtualisointialustalla toimivien virtuaalipalvelimien välillä resurssien lisääminen ja vähentäminen tapahtuisi ohjelmallisesti, joka on tietenkin paljon helpompaa ja nopeampaa.

Kontteja käytettäessä suoraan fyysisen koneen päällä tapahtuisi vähemmän virtualisointialustasta johtuvaa viivettä ja resurssihukkaa, koska konteilla on suora pääsy laitteistoresursseihin, kun taas virtualisointialustalla toimivien virtuaalikoneiden on läpäistävä ylimääräinen kerros kuvan 7 kehämallissa. Toisaalta laitteistoavusteinen virtualisointi on melko kehittynyttä ja resurssihukka on nykyään häviävän pieni vanhempiin virtualisointitekniikoihin verrattuna, kuten ohjelmistoavusteiseen virtualisointiin tai käyttöjärjestelmän päällä toimivaan virtualisointiin.

Testissä käytettiin palvelukeskeisellä toimivaa arkkitehtuuria, eli tietokanta, sovellus ja käyttöliittymä oli omissa lokeroissaan, kuten kuvassa 4. Mikropalveluarkkitehtuurilla toimivassa järjestelmässä sovelluskehittäjien tarvitsisi pilkkoa sovellus pieniin suoritettaviin yksiköihin, joita ajettaisiin omissa konteissa.

Mikropalveluarkkitehtuurilla voitaisiin saada enemmän hyötyjä resurssien suhteen, koska konttiorkestroinnilla toimivaan järjestelmään voitaisiin rakentaa toiminnallisuuksia, jotka lisäisivät aina kovalla kuormalla olevien sovelluskonttien määrää kuten kuvassa 5 ja vähän käytettyjä kontteja voitaisiin taas vähentää, jolloin resurssien tarjonta kohtaisi kysynnän optimaalisemmin.

7.2 Kehitys, käyttöönotto ja siirrettävyys

Kehitys, käyttöönotto ja siirrettävyys vaikuttaisi olevan helpompaa konttien kanssa, koska ensimmäiset testit voitaisiin tehdä helposti jo sovelluskehittäjän työasemalla ennen kuin kontti siirtyisi ajettavaksi orkestrointialustalle, koska kontti käyttäytyy suurella todennäköisyydellä samanlailla alustasta riippumatta.

Lisäksi riippuvuuksien pakkaaminen konttiin virtaviivaistaisi sovelluksen käyttöönottoa tilanteessa, jossa esimerkiksi sovellus asennettaisiin asiakkaan toimesta sen omalle konttialustalle, koska riippuvuuksia ei tarvitsisi erikseen asentaa ja tietää kuten virtuaalikoneelle asennettaessa, vaan kaikki tarvittava tulisi aina kontin mukana.

Toisaalta konttiorkestrointialustan hallinnointi tarvitsisi syvää asiantuntemusta orkestroinnista ja varsinkin jos kyseessä olisi mikropalveluarkkitehtuurilla toimiva järjestelmä, koska silloin lukuiset toisillensa kommunikoivat kontit täytyisi asentaa orkestrointialustalle ja varmistaa, että ne todella kommunikoivat ja toimivat kuten pitää.

7.3 Oma oppiminen

Opinnäytetyön tekeminen kokonaisuudessaan opetti minulle pitkien tekstien kirjoittamista, sekä opinnäytetyössä jouduin perehtymään käsiteltäviin aiheisiin teoreettisella tasolla. Lisäksi perehdyin tuotteisiin, myös käytännön tasolla, vaikka se ei opinnäytetyöstä käykään ilmi. Perehtyminen käsiteltäviin aiheisiin helpottaa varmasti minua myöhemmin työtehtävissä. Uskon, että erityisesti konttiteknologiasta oppiminen helpottaa työmahdollisuuksia tulevaisuudessa. Myös virtualisointiteknologiaan perehtyminen opetti minulle paljon tietokoneiden toiminnasta ja tietoteknisten palveluiden tarjoamisesta, sekä en uskon, että vaikka konttiteknologiassa on paljon hyötyjä, niin palvelinvirtualisointi ei ole katoamassa mihinkään.

Lisäksi teknisten kuvien piirtotaidot kehittyivät opinnäytetyötä tehdessä. Aluksi piirsin Paint- ja Excel tuotteilla kuvia. Tämän jälkeen löysin Draw.io tuotteen, jolla sai piirrettyä parempia ja ammattimaisemman näköisiä kuvia. Draw.io tuotteesta löytyi tarkoituksenmukaisempia työkaluja teknisten kuvien piirtämiseen.

Olen suorittanut koko ammattikorkeakoulun ja opinnäytetyön siten, että olen ollut täyspäiväisesti töissä. Opinnäytetyön kirjoittaminen katkesi aina lähes viideksi päiväksi viikossa, koska olin varannut arki-illat muille kursseille ja harrastamiselle. Viikonloput olivat varattu opinnäytetyölle. Opinnäytetyöhön joutui perehtymään uudelleen ja selvittää, että mihin sitä viimeksi jäi.

Jos joudun tai saan kirjoittaa opinnäytetyön jostain toisesta aiheesta, niin anoisin töistä hieman pidemmän loman kirjoittamiseen, niin opinnäytetyön voisi kirjoittaa keskeytymättömästi ja ei tarvitsisi perehtyä aina uudelleen.

Opinnäytetyöprosessi kesti 01.01.2022 – 15.05.2022 ja jouduin aloittaa opinnäytetyön alusta kahteen otteeseen, koska en saanut aiemmista aiheista tarpeeksi kirjoitettavaa. Kokonaisuutena opinnäytetyön tekeminen oli yksi opettavaisimmista kokemuksista ammattikorkeakoulussa ja antaa uskoa siitä, että pitkätkin projektit valmistuvat joskus.

Lähteet

Appeldology 2021. Hardware And Software Requirements for E-Commerce Websites. Luettavissa: https://appedology.com/hardware-and-software-requirements-for-e-commerce-websites. Luettu 27.3.2022

Altexsoft 2022. Quality Assurance, Quality Control and Testing — the Basics of Software Quality Management. Luettavissa: https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/ Luettu 11.4.2022

Avinetworks 2021. Container Orcestration. Luettavissa: https://avinetworks.com/glossary/container-orchestration/. Luettu 5.2.2022

Capitalone 2020. How to Avoid Coupling in Microservices Design. Luettavissa: https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/ Luettu 20.3.2022

Chirammal ,H. Mukhedkar, P. Vettathu., D 2016. Mastering KVM Virtualization. Packt Publishing. E-kirja. Luettu 13.3.2022

Cisco 2022. What Is High Availability?. Luettavissa: https://www.cisco.com/c/en/us/solutions/hybrid-work/what-is-high-availability.html#~infrastructure-elements Luettu 13.3.2022

Cloud Native Computing Foundation 2020. CNCF SURVEY 2020. Luettavissa: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf. Luettu 4.2.2022.

Dell 2022. Introduction to hardware design. Luettavissa: https://infohub.delltechnologies.com/l/design-guide-red-hat-openshift-container-platform-4-2/introduction-to-hardware-design-5. Luettu 3.4.2022

Docker 2021. Mange swarm service networks. Luettavissa: https://docs.docker.com/engine/swarm/networking/#configure-service-discovery. Luettu 5.2.2022

Digital Ocean 2015. The Docker Ecosystem: Scheduling and Orchestration Luettavissa https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-scheduling-and-orchestration. Luettu 5.2.2022

Dittner, R. & Rule, D. 2019 The Best Damn Server Virtualization Book Perioid. Syngress. E-kirja. Luettu 6.3.2022

Eurostat. Cloud computing – statistics on the use by enterprises 2014. Eurostat. Luettavissa: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises. Luettu 4.2.2022

Eurostat. Cloud computing – statistics on the use by enterprises 2022. Eurostat. Luettavissa: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises. Luettu 4.2.2022

Gartner 2020. Gartner Forecasts Strong Revenue Growth for Global Container Management Software and Services Through 2024. Luettavissa: https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co. Luettu 4.2.2022.

IBM 2019 The true benefits of moving to containers. Luettavissa: https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/. Luettu: 13.4.2022

IBM 2020. why should we use microservices and containers Luettavissa: https://developer.ibm.com/articles/why-should-we-use-microservices-and-containers/. Luettu 5.2.2022

IBM Cloud Education 2021. Containers. Luettavissa: https://www.ibm.com/cloud/learn/containers. Luettu 4.2.2022.

Iximiuz 2021. Why and How to Use containerd From Command Line. Luettavissa: https://iximiuz.com/en/posts/containerd-command-line-clients/ Luettu 20.3.2022

Iximiuz 2022. Implementing Container Runtime Shim: runc. Luettavissa: https://iximiuz.com/en/posts/implementing-container-runtime-shim/ Luettu 20.3.2022

Kubernetes 2021. Operating etcd clusters for Kubernetes. Luettavissa: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/. Luettu: 5.2.2022

Limepoint 2019. How are containers influencing world of software testing? Luettavissa: https://www.limepoint.com/blog/how-are-containers-influencing-the-world-of-software-testing. Luettu: 10.4.2022

Manpages 2021. chroot – Linux manual page. Luettavissa: https://man7.org/linux/man-pages/man2/chroot.2.html. Luettu 5.2.2022

Nginx 2022. NGINX Controller Tech Specs. Luettavissa: https://docs.nginx.com/nginx-controller/admin-guides/install/nginx-controller-tech-specs/ Luettu 30.3.2022

Opensource.com 2021. 4 Linux technologies fundamental to containers. Luettavissa: https://opensource.com/article/21/8/container-linux-technology. Luettu 20.3.2022

Phoenixnap 2019. What is a Hypervisor? Types of Hypervisors 1 & 2. Luettavissa: https://phoenixnap.com/kb/what-is-hypervisor-type-1-2 Luettu: 28.2.2022

Redhat 2020. What is Hypervisor? Luettavissa: https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor Luettu 30.4.2022

Redhat 2019. What is etcd?. Luettavissa: https://www.redhat.com/en/topics/containers/what-is-etcd Luettu: 03.04.2022

Redhat 2021. Red Hat Enterprise Linux technology capabilities and limits. Luettavissa: https://access.redhat.com/articles/rhel-limits#minimum-required-memory-3 Luettu 22.3.2022

Redhat 2022. Red Hat Virtualization Manager Requirements. Luettavissa: https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.0/html/installation_guide/chap-system_requirements Luettu 27.3.2022

Redhat 2022b. Luettavissa: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-overcommitting_with_kvm-overcommitting_virtualized_cpus. Luettu 29.3.2022

ServerWatch 2020. Paravirtualization: Improved Server Performance. Luettavissa: https://www.serverwatch.com/guides/paravirtualization-improved-server-performance/ Luettu: 28.2.2022

Serveralnines 2014. PrestaShop on Multiple Servers with MariaDB Galera Cluster. Luettavissa: https://severalnines.com/database-blog/how-setup-high-availability-prestashop-multiple-servers-mariadb-galera-cluster

Termipankki 2017. virtualisointiympäristö; virtualisointialusta Luettavissa: https://termipankki.fi/tepa/en/search/virtualisointialusta Luettu 25.2.2022

VMWare 2007. Understanding Full Virutalization, Paravitualization, and Hardware Assist. Luettavissa: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf Luettu: Luettu 20.3.2022

Wallenius Consulting 2019. Konttiteknologia – mitä kontit ovat ja mitä hyötyä niistä on?. Luettavissa: https://niklaswallenius.fi/konttiteknologia-mita-hyotya/ Luettu: 4.2.2022