Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TUOMAS SUONIEMI

Sovelluskonttien hallinta

TIETOJENKÄSITTELYN KOULUTUSOHJELMA
2022

Tekijä(t)
Suoniemi, Tuomas

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
Heinäkuu 2022

Sivumäärä
66

Julkaisun kieli
Suomi

Julkaisun nimi
Sovelluskonttien hallinta

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Tiivistelmä
Opinnäytetyössä käsiteltiin sovelluskonttien hallinnoimista. Toimintaympäristöjen muutos -kappaleessa käsiteltiin ohjelmisto-, ja järjestelmäpuolen haasteita ja niitä sovelluskonttien ominaisuuksia, joiden odotetaan tuovan haasteisiin helpotusta. Sovelluskontit-kappaleessa tutustuttiin ensin sovelluskonttien historiaan ja niissä käytettävien tekniikoiden kehitykseen. Historian jälkeen tutustuttiin sovelluskonttien ominaisuuksiin sekä niiden tarjoamiin mahdollisuuksiin.

Sovelluskonttien hallinta -osiossa tutustuttiin sovelluskonttien hallintaan Kubernetesin kautta. Kubernetes valikoitui käsittelyn kohteeksi sen hallitsevan markkina-aseman vuoksi, sillä yli 70 prosenttia sovelluskonttien hallinnasta tapahtuu Kubernetesiin perustuvien sovelluskonttihallintajärjestelmien kautta. Kubernetes itsessään on alun perin Googlen julkaisema avoimen lähdekoodin projekti, jota hallinnoi nykyisin Cloud Native Computing Foundation. Osaltaan tämän vuoksi Kubernetesilla on valtava kehittäjäyhteisö, joka vie projektia jatkuvasti eteenpäin. Ilmaisuudestaan huolimatta avoimen lähdekoodin Kubernetesin käyttö sellaisenaan on vähäistä ja suurin osa Kubernetesin käyttäjistä käyttää sitä jonkinlaisena palveluna. Tästä johtuen sovelluskonttiympäristöt-osiossa tutustuttiin neljään mahdollisimman erityyppiseen ympäristöön, joissa sovelluskontteja voidaan hallinnoida.

VMware Tanzu, OpenShift ja Azure Container Apps perustuvat Kubernetesiin vaikka ovat muuten hyvin erityyppisiä keskenään. Apache Mesos on alun perin kehitetty klusterien hallintaan, mutta siihen on tuotu myöhemmin tuki sovelluskonttien suorittamiseen. Mesos ei oletuksena käytä Kubernetesta sovelluskonttien hallintaan, mutta siihenkin on saatavissa Kubernetes-laajennus. VMware Tanzu on tutussa vSphere ympäristössä toimiva sovelluskonttien hallintajärjestelmä. Red Hatin OpenShift -tuote on monien suurten yhtiöiden käyttämä ja erityisesti tietoturvaan keskittyvä hallintajärjestelmä. Azure Container Apps on Microsoftin uusin palvelu sovelluskonttien helppoon käyttöönottoon. Johtopäätöksissä todetaan sovelluskonttien tulleen pysyäkseen ja niiden suosion kasvavan yhä. Sovelluskonttihallintajärjestelmien perustana tulee pysymään pitkään Kubernetes, mutta siihen perustuvia erilaisia uusia tuotteita ja palveluja tullaan julkaisemaan tiheään.

Avainsanat
Sovelluskonttien hallinta**, s**ovelluskontti, kontitus, mikropalvelu, Kubernetes, OpenShift, Apache Mesos, Docker, VMware Tanzu.

Author(s) Last name, First name Suoniemi, Tuomas	Type of Publication Bachelor's thesis	Date July 2022
	Number of pages 66	Language of publication: Finnish

Title of publication
Container orchestration

Degree programme
Degree Programme in Information Technology

Abstract
The thesis dealt with the management of application containers. Change of operating environments was discussed in the section on software and system-side challenges and the features of application containers that are expected to bring relief to the challenges. In the application containers chapter, we first got acquainted with the history of application containers and the development of the technologies used in them. After the history, we got to know the properties of application containers and the possibilities they offer.

In the application container management section, we got acquainted with the management of application containers through Kubernetes. Kubernetes was selected as the target for processing due to its dominant market position, as more than 70 percent of application container management takes place through application container management systems based on Kubernetes. Kubernetes itself is an open-source project originally published by Google and currently managed by the Cloud Native Computing Foundation. Partly because of this, Kubernetes has a huge developer community that constantly moves the project forward. Despite its free nature, the use of open-source Kubernetes as such is low and the majority of Kubernetes users use it as some kind of service. Because of this, in the application container environments section, we got to know four different types of environments where application containers can be managed.

VMware Tanzu, OpenShift and Azure Container Apps are based on Kubernetes, although they are otherwise very different types. Apache Mesos was originally developed to manage clusters, there was later added support for running application containers. By default, Mesos does not use Kubernetes to manage application containers, but a Kubernetes plugin is available for that as well. VMware Tanzu is an application container management system that works in the familiar vSphere environment. Red Hat's OpenShift product is a management system used by many large companies and especially focused on information security. Azure Container Apps is Microsoft's latest service for easy deployment of application containers. The conclusions state that application containers are here to stay, and their popularity is still increasing. Kubernetes will remain the basis of application container management systems for a long time, but various new products and services based on it will be published frequently.

Keywords
Container orchestration**,** application containers, container, containerization, microservice, Kubernetes, OpenShift, Apache Mesos, Docker, VMware Tanzu.

SISÄLLYS

1 JOHDANTO 5
2 TOIMINTAYMPÄRISTÖJEN MUUTOS 7
3 SOVELLUSKONTIT 12
3.1 Sovelluskonttien historia 12
3.2 Sovelluskontit 13
3.2.1 Container image 16
3.2.2 Container image layers 17
3.2.3 Dockerfile 18
3.2.4 Container Repository 19
3.2.5 Container registry (sovelluskonttirekisteri) 21
3.2.6 Container 22
3.3 The Open Container Initiative (OCI) 25
4 SOVELLUSKONTTIEN HALLINTA 26
4.1 Kubernetes 26
4.2 Kubernetesin osat 28
4.2.1 Nodes 28
4.2.2 Node Components (solmun komponentit) 29
4.2.3 Control Plane Components 29
4.3 Namespace 31
5 SOVELLUSKONTTIYMPÄRISTÖT 36
5.1 VMware Tanzu 36
5.2 Apache Mesos 43
5.3 Red Hat OpenShift 46
5.4 Microsoft Azure 50
6 JOHTOPÄÄTÖKSET 54
LÄHTEET

1 JOHDANTO

Tietokoneiden ylläpito ja järjestelmäresurssien riittävyyden varmistaminen ovat olleet tietojenkäsittelyn haasteina jo ensimmäisistä tietokoneista lähtien. Ylläpidossa haasteina ovat ohjelmistojen ja laitteiden päivitykset, rikkoutuneiden laitteiden ja uusien laitteiden asetusten ja ohjelmien määrittely käyttäjille sekä tarjottavien palveluiden toimittaminen käyttäjille. Järjestelmäresurssien riittävyyden suurin haaste on kustannustehokkuus, resurssit eivät saa loppua hallitsemattomasti, toisaalta fyysisten resurssien varaaminen kulutushuipun mukaan johtaa yleensä valtavaan ylikapasiteettiin muulloin.

Ylläpidon ja järjestelmäresurssien ongelmia on yritetty hallita jo pitkään erilaisilla virtualisoinnin ratkaisuilla. Varsinkin palvelimien ylläpito ja resurssien hallinta perustuvat nykyisin erittäin pitkälti virtualisointiin ja virtuaalikoneisiin. Virtuaalikoneet eivät kuitenkaan skaalaudu tehokkaasti todella suurille asiakasmäärille tarjottaviin palveluihin, sillä niissä virtualisoidaan myös koko käyttöjärjestelmä. Käyttöjärjestelmän virtualisointi vie resursseja ja vaatii lisää ylläpitoa kuten päivityksiä.

Varsinkin mobiilipäätelaitteiden määrän räjähdysmäinen kasvu ajoi ohjelmistoalan etsimään uusia ratkaisuja. Sovellusten testaaminen, muokkaaminen ja jakelu kaikille mobiililaitteille alkoi muuttua mahdottomaksi hallita. Ensimmäisenä reaktiona sovelluksia alettiin muuttamaan selainpohjaisiksi, mutta samalla entistä suurempi osa sovelluksen toimintaa siirtyi asiakkaan päätelaitteesta palveluntarjoajan palvelimille. Selaimissa toimivat sovellukset asettivat kuitenkin rajoituksia toiminnoille. Useat selainvaihtoehdot, selaimien jatkuvat päivitykset sekä muutokset tietoturvaan jättivät paljon haasteita jäljelle.

Suosituimmaksi ratkaisuksi tähän ovat nousseet sovelluskontit. Sovelluskonteissa käyttöjärjestelmää ei virtualisoida eikä tietoja tallenneta pysyvästi sovelluskonttiin. Tämä vähentää sovelluksen tarvitsemia resursseja ja mahdollistaa todella nopean ja

tehokkaan skaalautuvuuden. Sovelluskontit ovat helppoja ylläpitää ja lisäävät tavallaan vikasietoisuutta, sillä sovelluskontteja ei päivitetä tai varmuuskopioida vaan ne korvataan tarvittaessa uusilla. Kaatunut sovelluskontti lopetetaan ja käyttäjälle osoitetaan uusi kontti. Tällöin yksittäisten konttien kaatumiset vaikuttavat vain kyseisen kontin käyttäjään eivätkä koko palveluun. Esimerkiksi Google Gmail ja NetFlix ovat toteutettu sovelluskonteilla.

Opinnäytetyön tavoite on tutustua sovelluskontteihin ja erityisesti niiden hallintaan. Ensisijaisesti työssä tutustutaan Kubernetes -pohjaiseen Docker -sovelluskonttien hallintaan.

2 TOIMINTAYMPÄRISTÖJEN MUUTOS

Tarve sovelluskonteille ja muille vastaaville ratkaisuille on syntynyt toimintaympäristöjen vaatimusten muuttuessa. Yleisesti ottaen voidaan todeta, että sovelluskontteihin kohdistuu korkeita odotuksia ainakin ylläpidon, tehokkuuden sekä ohjelmistokehityksen suunnalta.

Ohjelmistokehitykselle sovelluskontit tarjoavat standardoidun (OCI) ohjelmistopaketoinnin sekä standardoidun (OCI) suoritusympäristön. Välillisesti sovelluskontit tarjoavat GitHub -integraatiota, julkisia sovelluskonttirekistereitä, helppoa testaamista sekä kaupallisia erittäin pitkälle automatisoituja ympäristöjä sovelluskehitykselle. On vaikea määritellä, mitkä sovelluskonttien tarjoamista eduista sovelluskehitykselle johtuvat suoraan sovelluskonteista itsestään ja mitkä räjähdysmäisesti kasvaneesta ekosysteemistä, mutta julkisuudessa ja markkinoinnissa sovelluskehitykseen liittyvät ehkä suurimmat odotukset.

Fyysisen suorituskyvyn haasteet voidaan jakaa kahteen osaan, eli tavoite on tietenkin tuottaa palvelu käyttämällä mahdollisimman vähän resursseja kuten muistia, laskentatehoa ja tallennustilaa sen toteuttamiseen. Toisaalta taustalla on aina palvelinympäristö, jossa on ennalta määrätty määrä kyseisiä resursseja, eikä palvelun tuottaminen vähemmillä muisti-, CPU- tai tallennusresursseilla ole hyödyllistä, jos taustalla olevia palvelimia ei voida vähentää tai tuottaa vastaavasti enemmän palveluita. Huomattavalla vajaakäytöllä toimivan palvelimen resurssien käytön vähentäminen vielä pienemmäksi ei lähtökohtaisesti tuota säästöjä.

Palvelinlaitteiden suorituskyvyssä on paljon erilaisia mittareita ja ongelmia, mutta opinnäytetyön aiheen kannalta kiinnostavia ovat perusresurssit, eli tallennustila, laskentakyky, muisti (RAM) sekä erityisesti niiden käytön optimointi. Palvelimien työkuorma voidaan jakaa karkeasti kahteen ryhmään. Eräajo (batch) -tyyppisiin tehtäviin, jotka suoritetaan yksi kerrallaan loppuun ja aloitetaan sitten uusi tehtävä. Tällaiseksi voidaan lukea esimerkiksi videon uudelleen koodaaminen tai vaikkapa erilaiset laskennalliset mallinnukset. Eräajo -tyyppisten tehtävien suorittamiseen tietokoneet ovat soveltuneet aina hyvin ja niissä on mahdollista päästä erittäin korkeisiin palvelimen

käyttöasteisiin (> 90 %), koska kaikki resurssit voidaan ohjata suoritettavalle prosessille. Toisena tehtävätyyppinä ovat reaaliaikaiset prosessit. Ne ovat vuorovaikutuksessa käyttäjän tai sensorien kanssa ja ovat huomattavasti vaikeampia toteuttaa tehokkaasti. Tällaisia ovat esimerkiksi verkkokaupan taustaprosessit, navigointi tai vaikkapa videoiden suoratoisto. Kyseisten prosessien määrä ja CPU:n käyttö vaihtelevat erittäin paljon ajankohdan mukaan. Asiakasta ei voida jättää odottamaan verkkokaupan ostoskorin käsittelyä, vaan sen on toteuduttava muutamien sekuntien sisällä, tämän toteuttaminen edellyttää kuitenkin aivan eri kokoisia resursseja juhannusyönä, kuin joulua edeltävällä viikolla. Eräs monille tuttu esimerkki on tapahtumien lipunmyynti verkossa, jonka tekninen toiminta sai muutamia vuosia sitten erittäin kovaa kritiikkiä palvelun kaaduttua toistuvasti heti lipunmyynnin alettua. Ongelma johtuu toisaalta resurssien käytön valtavasta kasvusta ja toisaalta tarpeesta lähes reaaliaikaiseen vuorovaikutukseen. Teollisuuden IOT ja sensorit tarvitsevat vielä viiveettömämpää vuorovaikutusta palvelimen kanssa, mutta teollisuuden automaatiossa se on monesti helpommin ennakoitavissa, kuin kuluttajien kanssa. Eräs voimakkaasti resurssien kulutushuippuun vaikuttava asia on paradoksaalinen, sillä esimerkiksi lipunmyynnin sivuston jumiutuessa käyttäjät yrittävät epätoivoisesti päivittää sivua toistuvasti tai lisätä tuotetta uudelleen ostoskoriin, jolloin jo vähissä olevien resurssien kuormitus alkaa nopeasti moninkertaistumaan. Palveluiden skaalaaminen kuormituksen mukaan onkin eräs tärkeimmistä ongelmista, johon sovelluskonteista on haettu apua ja niiltä osin voidaan sanoa sovelluskonttien lunastaneen odotukset. Sovelluskontit tarjoavat erittäin resurssitehokkaan tavan toteuttaa palveluita ja sovelluskonttien hallintajärjestelmät tarjoavat mahdollisuuden skaalata näitä palveluita käytännössä reaaliaikaisesti.

Microsoft Windowsin yleistyessä ja PC-tietokoneiden tehojen kasvaessa 90-luvulla otettiin käyttöön yhä enemmän palveluja, sillä Windows oli tuttu it-osastoille ja kynnys uusien palveluiden käyttöönottoon laski merkittävästi verrattuna aiempiin räätälöityihin järjestelmiin. Alun perin Windows oli suunniteltu yhdelle käyttäjälle ja yritys ajaa useampaa sovellusta samanaikaisesti samalla palvelimella johti helposti ongelmiin. Monet tahot ottivat tämän vuoksi käyttöön periaatteen "yksi palvelin, yksi sovellus". Pc-tietokoneiden tulo oli "laskenut" it-kustannuksia aluksi, mutta nyt kustannukset alkoivat kohoamaan hälyttävästi. Kustannusten lisäksi alkoi palvelinten hallinnointi muuttua painajaiseksi ja syntyi esimerkiksi termi "a lost server" (tomatoe) tarkoittaen palvelinta, jonka hallinnoijaa ja tarkoitusta ei tunnettu, mutta jolle ei

uskallettu myöskään tehdä mitään, ettei vaarannettaisi organisaation kriittisten palveluiden toimintaa. Palvelin määrien jatkuva kasvu alkoi aiheuttamaan fyysisiä ongelmia palvelintilojen kanssa. Palvelintiloista alkoi loppumaan tila palvelimilta, rakennuksien sähkönsyöttö saavutti rajansa ja palvelinten vaatima jäähdytys alkoi muuttua mahdottomaksi. (Portnoy, 2016, s. 4-6.)

Vuonna 2001 julkaistiin kaksi hyvin erilaista ratkaisumallia ongelmaan. VMware julkaisi ensimmäisen x86 -pohjaisen palvelinvirtualisoinnin tuotteen ja RLX Technologies julkaisi ensimmäisen kaupallisen Blade -palvelimen. Blade -palvelimien tarkoitus oli kasvattaa palvelintiheyttä ja helpottaa ahtaiksi muuttuneiden palvelinhuoneiden tilannetta. Vanhojen palvelinhuoneiden tullessa käyttöikänsä päähän ja siirryttäessä uusiin datasaleihin ja verkkotekniikoihin tilojen kustannukset muuttuivat helpommin mitattavaksi ja osoittautuivat melko pieniksi.

Alkoi tulla ilmeiseksi, että palvelimien tarve oli enemmänkin oire palvelimien käyttöasteesta kuin laskentatehon tarpeesta. Puhuttiin 10 prosentin käyttöasteesta palvelimissa, mutta monet arvioivat senkin olleen optimistinen arvio. Virtualisoinnista alettiin etsimään ratkaisua, sillä se mahdollisti useamman loogisen palvelimen ajamisen yhdellä fyysisellä palvelimella, jolloin palvelimen käyttöastetta saatiin nostettua ja palvelintiheyskin nousi samalla. "Verkkojen" nopeutuminen mahdollisti ulkoiset tallennusjärjestelmät ja yhdessä virtualisoinnin kanssa kuormantasaus ja käyttöasteen optimointi helpottui, sillä virtualisoitujen palvelinkuormien siirtäminen oli selvästi joustavampaa kuin perinteinen kuormantasaus.

Uptime Instituten arvio julkaisussaan 2014 Data Center Industry Survey on, että jopa 20 prosentia palvelinkeskusten palvelimista on käynnissä, mutta eivät suorita mitään eli ovat koomassa (comatose). Kuitenkin suurin osa operaattoreista uskoi, että alle viisi prosenttia heidän koneistaan on comatose -tilassa. (Heslin, 2014)

Palvelimen keskimääräisen käyttöasteen määrittäminen suorittimen käyttöasteen perusteella saattaa olla harhaanjohtavaa, sillä monesti järjestelmän käytettävissä oleva muisti (RAM) loppuu ennen kuin suorittimen suorituskyky muuttuu pullonkaulaksi. Tästä johtuen osa palvelimista, jotka vaikuttavat toimivan huomattavan vajaalla käyttöasteella saattavat olla muistin osalta jo äärirajoilla. (Benik, 2013)

"Mooren lain" periaatteen mukaisesti neljän vuoden välein vaihdetun palvelimen suorituskyky on nelinkertainen edelliseen palvelimeen verrattuna. Vaikka sovellusten tarvitsema laskentateho on täysin sovelluskohtainen, voidaan melko turvallisesti todeta, että keskimääräisesti sovelluksen käyttämä laskentateho ei nelinkertaistu neljässä vuodessa. Noudatettaessa ajatusta "yksi palvelin, yksi sovellus" palvelinlaitteistojen uusimisen yhteydessä palvelimen käyttöaste laskee. Laskennallisesti vuonna 2006 käyttöönotetun ja 48 kuukauden välein vaihdetun sähköpostipalvelimen suorituskyky on vuonna 2018 64 kertainen verrattuna alkuperäiseen palvelimeen.
(Portnoy, 2016, s. 6-9.)

Useita ytimiä sisältävät prosessorit ja niitä tukevat käyttöjärjestelmät toivat käyttöön moniajon, jossa useita sovelluksia voidaan suorittaa samaan aikaan. Tämän tehokkaassa hyödyntämisessä oli kuitenkin pitkään ongelmia, sillä virheellisesti tai muuten ennakoimattoman suuren muisti- tai prosessorikuorman aiheuttava sovellus saattoi käytännössä lamaannuttaa koko järjestelmän. Tähän ongelmaan Google esitteli 2006 Cgroup -järjestelmän resurssien käytön hallinnointiin Linuxissa, mutta erityisesti siitä kehittyi perusta sovelluskonteille.

Sovelluskontit näyttävät tarjoavan ratkaisua useisiin suorituskyvyn haasteisiin tarjoten vähäisempää resurssien kulutusta toteutettavaa palvelua kohden sillä:

-Sovelluskonttien muistijälki on lähtökohtaisesti huomattavasti pienempi verrattuna virtuaalikoneisiin ja fyysisiin palvelimiin, joten voidaan sanoa sovelluskonttien lähtökohtaisesti tarjoavan mahdollisuuden selvästi korkeampaan käyttöasteeseen kuin mihin fyysiset- tai virtualisoidut palvelimet pystyvät.

-Sovelluskonttien vaatima levytila on todella pieni verrattuna virtuaalikoneisiin, koska käyttöjärjestelmää ei sisällytetä sovelluskonttikuvaan. Vaikka tallennustila on nykyisissä järjestelmissä edullista, niin epäsuorat hyödyt vähäisestä tallennustilan käytöstä ovat merkittäviä, sillä sovelluskonttien käynnistyminen ja erityisesti sovelluskonttikuvien siirto verkossa on sen vuoksi äärettömän nopeaa verrattuna virtuaalikoneisiin.

-Suorittimen käyttöaste on sovelluskonttia kohden pienempi kuin virtuaalikoneessa, koska sovelluskontissa ei ole käyttöjärjestelmän omia taustaprosesseja. Tämän hyöty on luultavasti melko vähäinen, sillä taustaprosessit käyttävät keskimäärin melko vähän CPU-aikaa, lisäksi virtuaalikoneissa on käytetty pitkään räätälöityjä Linux-jakeluita, jotka on optimoitu virtualisointiin riisumalla turhia palveluita pois.

3 SOVELLUSKONTIT

3.1 Sovelluskonttien historia

Sovelluskonttien historia alkaa vuodesta 1979 UNIX V7 -käyttöjärjestelmän yhteydessä esitellyn chroot -komennon kautta, joka mahdollisti prosessikohtaisen juurihakemiston. Ominaisuus otettiin mukaan BSD-käyttöjärjestelmään vuonna 1982. Seuraavaa edistysaskelta jouduttiin kuitenkin odottamaan vuoteen 2000, jolloin FreeBDS Jails julkaistiin. Jails sai alkunsa pienen jaetun ympäristön isännöintipalveluntarjoajan (R&D Associates, Inc.) tarpeesta eriyttää omat ja asiakkaiden palvelut. Poul-Henning Kampin ratkaisuna oli jakaa järjestelmän resurssit ja tiedostot pienemmiksi itsenäisiksi järjestelmiksi "jails", näin luotuihin jails-osioihin pääsyä pystyttiin rajoittamaan käyttäjäkohtaisesti ja tarjoamaan luoduille osioille oma ip-osoite. (Strotmann, 2016) Vaikka sovelluskonttien tekniikka otti ensiaskeleensa 1979, niin esimerkiksi Red Hat katsoo FreeBDS 4.0:n mukana julkaistun Jailsin esitelleen ensi kerran nykyisin käytetyn sovelluskonttitekniikan idean. (Red Hat, 2022F) Linux Vserver otti 2001 Jails -tekniikan käyttöön Linuxissa lisäten mahdollisuuden resurssien osiointiin. (Basyildiz, 2019)
2004 julkaistu Solaris Containerin beetaversio esitteli alueisiin (zone) perustuvan sovellustason virtualisoinnin, joka pystyi hyödyntämään ZFS-tiedostojärjestelmän ominaisuuksia, kuten kloonausta ja tilannevedoksia. (Osnat, 2020) Googlen 2006 kehittämä process containers -tekniikan tarkoitus on rajoittaa, valvoa ja eristää prosessikokoelman resurssien käyttöä. Tekniikka uudelleen nimettiin sekaannusten välttämiseksi muotoon control groups 2007 ja myöhemmin samana vuonna tekniikka liitettiin Cgroups -nimellä osaksi Linux-kerneliä sen versiosta 2.6.24 lähtien. (Strotmann, 2016)
LinuX Containers eli LXC julkaistiin 2008. Se perustui Linuxin cgroups -resursinhallintaan sekä Linuxin nimitilan eristämiseen (namespace isolation) LXC:ä pidetään aikansa kattavimpana ja vakaimpana versiona konttitekniikasta ja siksi monet myöhäisemmät sovelluskontituksen ratkaisut perustuvat siihen. Näistä mainiten 2011 julkaistu Cloud Foundry Warden sekä erityisesti 2013 julkaistu Docker, joka ladattiin ensimmäisen vuoden aikana 100 miljoonaa kertaa. (Basyildiz, 2019) Dockerin saama suosio teki sovelluskonteista yhden it-alan kuumimmista trendeistä, kiihdyttäen samalla niiden kehityksen ennennäkemättömään vauhtiin. Käsittelen kuitenkin tässä

työssä sovelluskonttien historiaa vain vuoteen 2013 asti, koska työssäni sovelluskontteja käsitellään pitkälti Dockerin kautta ja Docker julkaistiin tuolloin.
Työn laajuuden vuoksi oli mahdotonta esitellä sovelluskonttien historiaa kattavasti, mutta yritin esitellä sen tärkeimmät tapahtumat, jotka ovat vaikuttaneet sovelluskonttien tekniseen kehitykseen. Pois rajatuista kiinnostavin oli ehkä Virtuozzo -yhtiö (Swsoft, Parallers), joka kehitti jo vuonna 2001 ensimmäisen sovelluskontitukseen perustuvan kaupallisen tuotteen. Tuote julkaistiin avoimen lähdekoodin projektina vuonna 2005 nimellä OpenVZ. Virtuozzo on yhä erittäin tärkeä vaikuttaja avoimen lähdekoodin yhteisössä tukemalla ja toimimalla mukana useissa projekteissa, kuten OpenVZ, CRIU, KVM, Docker, OpenStack ja Linux -kernel. (Strotmann, 2016)

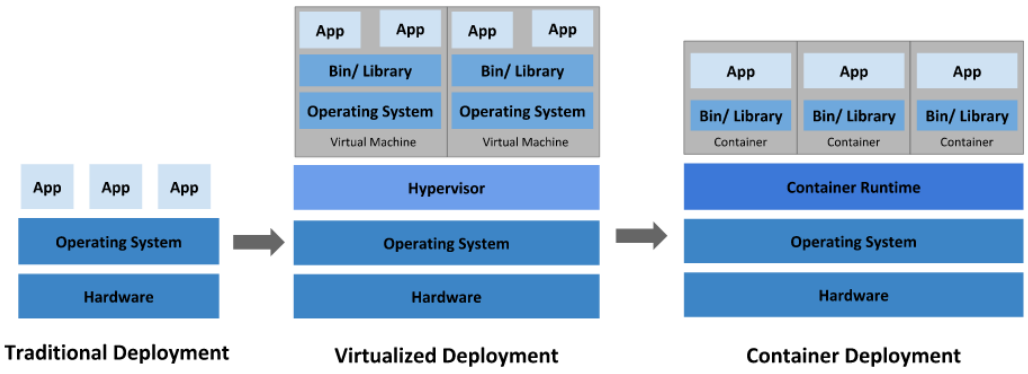
3.2 Sovelluskontit

Jatkossa sovelluskonteilla tarkoitetaan OCI image format -standardia noudattavia sovelluskontteja, jotka ovat periaatteessa yhteensopivia Docker-sovelluskonttien kanssa. Dockerin Container images noudattaa OCI image format -standardia, joten opinnäytetyössä käsiteltyjen asioiden pitäisi olla sovelluskonttien osalta yhteensopiva muidenkin OCI image format -standardia noudattavien sovelluskonttien kanssa.

Sovelluskontin määritelmä on muuttunut sovelluskonttien kehittyessä. Yleisluonteinen määritelmä, jolla sovelluskonttien kehitystä voidaan seurata myös viime vuosituhannelle ja joka sopisi sovelluskonttien historiaosassa käsiteltyihin tapahtumiin olisi seuraava: Sovelluskontit ovat käyttöjärjestelmätason eristys- eli virtualisointimenetelmä useiden sovellusten suorittamiseen yhdellä palvelimella (McCabe & Friis, 2017, s. 14.).
IBM:n moderni määritelmä sovelluskonteista on, että kontit ovat suoritettavia yksiköitä ohjelmia, joissa sovelluskoodi kirjastoineen ja riippuvuuksineen pakataan määrätyllä tavalla yhteen, jolloin se voidaan suorittaa missä tahansa, olipa kyseessä työpöytä, perinteinen it -ympäristö tai pilvi. (IBM, 2021) Tätä määritelmää eivät yleensä täytä kokonaan ennen vuotta 2013 tehdyt sovelluskonttiratkaisut, varsinkaan siirrettävyyden osalta.

Kontitus (containerization) tarkoittaa ohjelmistokoodin pakkaamista vain käyttöjärjestelmän kirjastoihin ja koodin suorittamiseen tarvittavin riippuvuuksin, josta luodaan yksi kevyt suoritettava tiedosto nimeltään kontti ja tämä toimii missä tahansa infrastruktuurissa. (IBM, 2021A)
Kontitus (containerization) on siis tekniikka, joka tuottaa kontti -kuvan (container image), josta tulee kontti (container), kun sitä suoritetaan kontti -ajoympäristössä (container runtime).

Sovelluskonteista puhuttaessa saattaa törmätä tilanteeseen, jossa sovelluskontteja ja virtualisointia verrataan toisiinsa ja annetaan mahdollisesti jopa kuva, etteivät sovelluskontit olisi virtualisointia. Sovelluskontit ovat ehdottomasti virtualisointia, mutta niiden virtualisointi tapahtuu käyttöjärjestelmän tasolla toisin kuin perinteisen virtualisoinnin, joka tapahtuu laitteistotasolla. Käyttöjärjestelmätason virtualisoinnissa konteille tarjotaan virtuaalisesti pilkottua käyttöjärjestelmää, kun perinteisessä virtuaalikoneisiin (VM) perustuvassa virtualisoinnissa annetaan VM:lle virtuaalisesti pilkottuja laitteistoja, joita VM:n käyttöjärjestelmä hallinnoi. Alla olevasta kuvasta näkee perinteisen virtualisointiin ja sovelluskontitukseen perustuvan ohjelman suorittamisen eron.



Kuvio 1. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Perinteisen aikakauden malli on rakenteeltaan yksinkertaisin ja on periaatteessa optimaalisin ratkaisu sovelluksen suorittamiseen. Käytännössä näin harvemmin on, sillä fyysisellä palvelimella ei voida määritellä sovelluksille resurssirajoja, johtaen ongelmiin resurssien allokoinnissa. Yksittäinen sovellus saattaa viedä jonkin resurssin lähes kokonaan, jolloin muiden sovellusten toiminta häiriintyy. Ratkaisuna olisi ajaa jokainen sovellus omalla fyysisellä palvelimellaan, mutta se johti perinteisissä konesaleissa tunnetusti tilanteeseen, jossa palvelinten käyttöaste pyöri pahimmillaan 5 prosentin paikkeilla. Tämä johti kohtuuttomiin laite- ja ylläpitokustannuksiin.

Virtualisoinnin aikakautena ratkaisuna käytettiin virtualisointia, jonka avulla voitiin käyttää useita virtuaalikoneita (VM) yhden fyysisen palvelimen resursseilla. Virtualisointi tarjosi myös merkittävästi suojaa, sillä eri virtuaalikoneissa suoritettavat sovellukset on eristetty toisistaan. Virtualisointi mahdollistaa fyysisen palvelimen resurssien optimoinnin ja skaalautuvuuden tehokkaammin, vähentäen laitteisto- ja ylläpitokustannuksia, koska sovellusten lisääminen ja päivittäminen on helpompaa. Virtualisoinnilla voidaan esittää joukko fyysisiä resursseja virtuaalikoneiden klusterina.
Jokainen virtuaalikone vastaa toiminnaltaan fyysistä tietokonetta käyttäen kaikkia komponentteja mukaan lukien oma käyttöjärjestelmänsä, virtualisoidun laitteiston päällä.

Sovelluskonttien aikakaudella kontit ovat samankaltaisia virtuaalikoneiden kanssa, mutta niiden eristystä on kevennetty, jotta ne pystyvät jakamaan käyttöjärjestelmän keskenään. Jaetun käyttöjärjestelmän vuoksi useamman yhdessä ajettavan kontin vaatimat resurssit ovat yleensä huomattavasti kevyemmät verrattuna virtuaalikoneisiin. Kuten virtuaalikoneella on kontillakin oma tiedostojärjestelmä sekä osuus prosessorista, muistista ja prosessitilasta. Koska kontit ovat irrallaan taustalla olevasta infrastruktuurista, ne ovat siirrettävissä pilven, konesalin ja työpöydän välillä kuten myös eri käyttöjärjestelmäjakeluiden välillä.

Sovelluskontit mahdollistavat käyttöjärjestelmätason tietojen ja mittareiden lisäksi myös sovelluksen kunto- ja muiden signaalien seuraamisen. Sovelluskontti toimii Ubuntussa, RHEL:ssä, CoreOS:ssä, paikan päällä, suurissa julkisissa pilvissä ja missä tahansa muualla tarjoten ympäristöllisen yhdenmukaisuuden kehitystyön, testauksen ja tuotannon välillä. Resurssien eristäminen mahdollistaa ennustettavan suorituskyvyn

sovellukselle. Löyhästi kytketyt, hajautetut ja joustavat mikropalvelut soveltuvat erityisen hyvin sovelluskontteihin, sillä sovellukset voidaan jakaa pienempiin, itsenäisiin osiin, ja niitä voidaan ottaa käyttöön ja hallita dynaamisesti.
(Kubernetes, 2022C)

3.2.1 Container image

Container image on tiedosto, jota käytetään koodin/ohjelman suorittamiseen sovelluskontissa. Container image toimii ohjeena ja mallina sovelluskontin muodostamiseen. Container image toimii aloituspisteenä kontteja käytettäessä. Se on verrattavissa tilannekuvaan (snapshot) virtuaalikone (VM) -ympäristössä. Container image sisältää sovelluskoodin, riippuvuudet, työkaluja, kirjastoja, ympäristömuuttujat sekä muut tiedostot, joita tarvitaan kontin suorittamiseen. Käyttäjän suorittaessa container imagen siitä luodaan yksi tai useampi sovelluskontti. Container imaget sisältävät yleensä useita tasoja (layers), jotka perustuvat periaatteessa edelliseen tasoon, mutta sisältävät/esittävät vain muutokset edelliseen tasoon. Tasojen käyttö nopeuttaa container imagen luomista ja vähentää uudelleen käytettävyydellään levytilan käyttöä. Tasot ovat container imagen tapaan vain lukutiedostoja. (Gillis, 2021) Container image on vain lukumuotoinen malli konttien luomiseen, ja se tarjoaa tiedostojärjestelmän, joka perustuu useiden tiedosto- ja hakemistokerrosten järjestetylle yhdistelmälle ja se voidaan jakaa muiden kuvien ja konttien kanssa. (Brown, 2016)

3.2.2 Container image layers

Jokainen container imagen muodostava tiedosto tunnetaan tasona. Nämä kerrokset muodostavat sarjan välikuvia, jotka on rakennettu vaiheittain päällekkäin. Jokainen taso tallentaa muutokset verrattuna kuvaan, johon se itse perustuu. Jokainen taso on itsessään kuva, mutta toisin kuin lopullisella container imagella, sillä ei ole käyttäjän antamaa nimeä vaan automaattisesti luotu ainutkertainen tunniste (ID). Container image voi koostua vain yhdestä tasosta, jos se on muodostettu Docker squash -komennolla, joka yhdistelee tasoja. Tasojen järjestyksen suunnittelu on tärkeä osa container imagen elinkaaren suunnittelua, sillä aina tehtäessä muutoksia kuvan tasoon, rakennetaan kyseisen tason lisäksi kaikki siitä periytyvät eli myöhemmin lisätyt tasot uudelleen. Tästä johtuen pinon ylimpien tasojen vaihtaminen on laskennallisesti huomattavasti kevyempää ja nopeampaa kuin alempien tasojen vaihtaminen. Useimmiten vaihtuvat tasot pitäisi sijoittaa mahdollisimman korkealle pinossa, jolloin muutoksien vaatima laskenta saadaan mahdollisimman vähäiseksi. (Kisler, 2021) Kaikki container imagen luomiseen käytetyt tasot näkee, esimerkiksi Dockerissa komennolla: docker history <imagename>. Docker tukee useita tallennusohjaimia (storage drivers) kuten, aufs, overlay2, btrfs, devicemapper, zfs. Ne kaikki tarjoavat Dockerille tavan toteuttaa tasoja (layers) ja kopioida kirjoittaessa. Näin sovelluskontti voi nähdä muiden tasojen tiedot, mutta kun tietoja muokataan, ne kirjoitetaan eri paikkaan alkuperäisen sisällön korvaamisen sijaan. Tätä tekniikkaa kutsutaan nimellä "Copy on write", sillä tiedot vain kopioidaan uuteen paikkaan, kun kirjoitettaan muutokset talteen. Tähän perustuu sovelluskonttikuvan ja sovelluskonttikuvan tasojen muuttumattomuus, kumpaakaan ei ikinä tehdä muutoksia, vaan muutoksista kirjoitetaan pelkästään uusi taso, joka sisältää tehdyt muutokset. (Crippa, 2021) Tarvittaessa tasot voidaan yhdistää yhdeksi esimerkiksi luomalla ”valmiista” kuvasta uusi kuva Dockerin FROM scratch -komennolla. (Docker Inc. n.d. B)

Useimmissa tapauksissa container imagen ensimmäinen taso tunnetaan parent imagena. Se toimii perustana, jolle kaikki muut tasot rakennetaan ja sisältää konttiympäristön vaatimat peruselementit. (Kisler, 2021) Valmiita parent imageja löytyy esimerkiksi julkisista sovelluskonttirekistereistä (public container registry), kuten Docker

Hubista, joka on suurin sovelluskonttirekisteri. (Docker Inc. 2022) Tyypillinen parent image on riisuttu Linux -jakelu tai esiasennettu palvelu, kuten www- ja proxy-palvelin Nginnx tai esimerkiksi muistinvarainen tietokantapalvelin Redis.
Container image voi myös perustua peruskuvaan (base image). Se on yksinkertaisesti tyhjä ensimmäinen taso, johon voi tehdä konttikuvan alusta asti omin määrityksin. Peruskuvan luontiin voidaan käyttää jo aiemmin mainittua FROM scratch -komentoa. (Kisler, 2021)

OCI image tai OCI Image Specification on Open Container Initiativen (OCI) standardoima sovelluskonttikuvan tekninen määritelmä, joka perustuu Docker-imagen rakenteeseen. OCI image specification toteuttaa Docker Image Manifest Version 2, Schema 2 muotoa. Standardilla on erittäin laaja tuki ja lähes kaikki sovelluskonttiympäristöt tukevat sitä. Esimerkiksi avoimen lähdekoodin rakennustyökalut BuildKit, Podman ja Buildah tukevat OCI -standardin mukaisten sovelluskonttikuvien luomista.
Mikä tahansa OCI Runtime Specificationin täyttävä container runtime voi purkaa OCI-kuvan ja suorittaa sen sisällön. (Busser, 2021)

3.2.3 Dockerfile

Dockerfile on komentosarja, joka luo automaattisesti Docker -imagen. Dockerfile on yksinkertainen tekstitiedosto, joka sisältää täydelliset ohjeet Docker -imagen luomiseen. (Suse, n.d.) Docker -imagen voi tehdä Docker clientin tarjoaman komentoriviliittymän kautta interaktiivisesti suorittamalla komentoja yksi kerrallaan. Se ei kuitenkaan ole yleensä tarkoituksenmukaista, sillä vasta Dockerfilen käyttö mahdollistaa monet sovelluskonttien eduista. Dockerfilen käyttö on joustavaa, sillä tekstitiedostona sen sisältö on helposti luettavissa antaen samalla selkeän kuvan tulevan kuvan sisällöstä. Dockerfilen etuna on, ettei se sisällä valmista binäärikuvaa vaan kuva muodostetaan ”viittauksista” ja komennoista, jolloin automaattisten koontiversioiden käytöllä voidaan varmistaa, että käytössä on aina uusimmat versiot. (Butler, 2015)

Alla olevassa kuvassa 2 on esimerkki Dockerfilen sisällöstä.

```
#
# Each instruction in this file generates a new layer that gets pushed to your local image cache
#


#
# Lines preceeded by # are regarded as comments and ignored
#


#
# The line below states we will base our new image on the Latest Official Ubuntu
FROM ubuntu:latest

#
# Identify the maintainer of an image
LABEL maintainer="myname@somecompany.com"

#
# Update the image to the latest packages
RUN apt-get update && apt-get upgrade -y

#
# Install NGINX to test.
RUN apt-get install nginx -y

#
# Expose port 80
EXPOSE 80

#
# Last is the actual command to start up NGINX within our Container
CMD ["nginx", "-g", "daemon off;"]
```

Kuvio 2. https://codefresh.io/docker-tutorial/build-docker-image-dockerfiles/

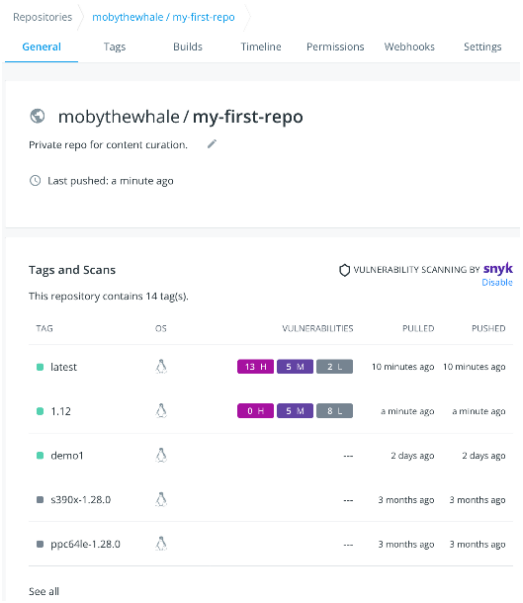
Dockerfile on tekstitiedosto, jonka Docker -runtime suorittaa rivi kerrallaan. # rivin alussa tekee rivistä kommentin, jonka Docker -runtime poistaa suorituksen yhteydessä. Isolla kirjoitetut ovat komentoja (INSTRUCTION), joita seuraa pienellä kirjoitettuja argumentteja. FROM määrittelee ensimmäisen tason eli BASE-imagen, jonka päälle muodostetaan kuva. Tässä tapauksessa se on Ubuntun ”uusin versio” sillä hetkellä, kun dockerfile suoritetaan. LABEL maintainer on etiketti, josta selviää kuvan ”ylläpitäjä”. RUN lataa ja suorittaa tässä esimerkissä päivityksiä, jotka on putkitettu, ettei luoda ”turhaan” kahta tasoa. Seuraavaksi RUN lataa ja asentaa NGINX www-palvelimen. EXPOSE 80 määrittelee kuvasta käynnistettävälle sovelluskontille käyttöön portin 80. CMD määrittelee komennot, joita suoritetaan kontin **käynnistyttyä**. Esimerkkitapauksessamme CMD suoritetaan kontissa, jolloin NGINX www-palvelin käynnistyy automaattisesti kontin käynnistyessä. (Codefresh, 2019)

3.2.4 Container Repository

Repository on kokoelma sovelluskontin kuvia, joilla on sama nimi, mutta eri tagit. Tagi viittaa Artifactiin, joka tarkoittaa yleensä container imagea. Tagi on kuitenkin teknisesti pelkkä viittaus siihen. Tagin taustalla on container image, mutta koska samalla container imagella voi olla useampi tagi, eikä muuttumattomia container

imageja kopioida, niin tagi voi olla myös kovakoodattu linkki, joka näyttäytyy järjestelmässä levykuvana. Pelkistetysti kuvattuna repository sisältää jaettavaksi tarkoitetun palvelun eri versiot. Versiot voivat olla uudempia ja vanhempia versioita, mutta ne voivat olla myös versioita eri tekniseen tai ohjelmalliseen ympäristöön. (Microsoft, 2022D)

Kuvassa xi näkyy Docker Hubissa käyttäjän mobythewhale yksityinen (Private repo) repository ”my-first-repo”, joka sisältää 14 tagia eli sovelluskontin kuvaa. Kuvassa kannattaa huomioida myös kuinka tag:1.12 on lisätty myöhemmin kuin tag:latest. my-first-repo:latest on loogisesti uusin, mutta ”latest” on vain tagi, joten ei ole mitään taetta, että se on viimeisin versio.



Kuvio 3. https://docs.docker.com/docker-hub/repos/

Kuvassa 3 näkyy tagin latest tiedot. Ensimmäisenä on koko nimi joka koostuu käyttäjästä (DOCKER_HUB_ID)/repositoryn nimestä/tagista eli kuvan nimestä. Seuraavana on Docker v2 registry formatin mukainen DIGEST. Digest on funktiolla sha256 laskettu tiivistelmä sovelluskonttikuvan ”latest” luonnista. Tiivistelmä lasketaan kuvasta sen luonnin yhteydessä. Muuttumattomasta Dockerfilesta ja muuttumattomista koontikomponenteista (peruskuva, tiedostot ym.) luotu uusi image tuottaa aina saman digestin. Digestin muuttuminen tarkoittaa, että sovelluskonttikuvassa on tapahtunut muutos. (Newswanger, 2020) Lisäksi on nähtävissä kuvan koko, arkkitehtuuri, käyttöjärjestelmän tyyppi sekä kuvan julkaisusta kulunut aika.

Kuvio 4. https://docs.docker.com/docker-hub/repos/

3.2.5 Container registry (sovelluskonttirekisteri)

Sovelluskonttirekisteri on repository -kokoelma ja sitä käytetään sovelluskonttikuvien säilyttämiseen ja jakeluun. Sovelluskonttirekisteriä voi kuvata sovelluskonttikuvien arkistoksi, johon ladataan (push) sovelluskonttikuvia ja josta ladataan sovelluskonttikuvia (pull), lisäksi se tarjoaa mahdollisuuden hallinnoida sovelluskonttikuvia ja niistä tehtyjä kokoelmia (repository). (Zhang, 2017)

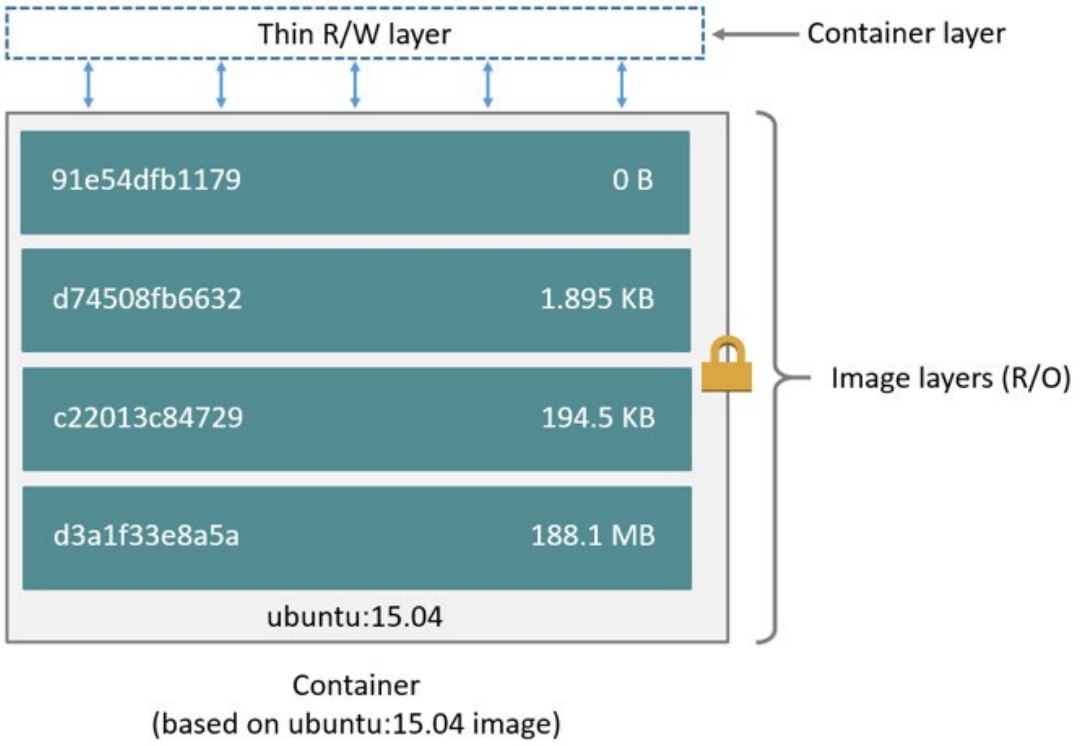
Sovelluskonttirekisteri voi olla julkinen (public), yksityinen(private) tai näiden yhdistelmä. Julkisessa sovelluskonttirekisterissä sinne ladatut sovelluskonttikuvat ovat yleensä kaikkien ladattavissa. Ne ovat helppokäyttöisiä ja soveltuvat hyvin yksityishenkilöille ja pienille ryhmille, mutta suuremmille tiimeille julkisten sovelluskonttirekisterien rajoittuneet hallintamahdollisuudet muuttuvat helposti ongelmaksi. Yksityisissä sovelluskonttirekistereissä organisaatio pystyy hallitsemaan roolipohjaisilla käyttöoikeuksilla pääsyä sovelluskonttikuviin. (Wong, 2019)

Tunnetuin ja laajin sovelluskonttirekisteri Docker Hub on julkinen sovelluskonttirekisteri, joka myy lisäksi yksityisiä arkistoja (repositories). Yksityinen repository toimii kuin yksityinen konttirekisteri, mutta sisältää vain yhden arkiston, eli käytännössä yhden jaeltavan sovelluksen tarvittavine versioineen. (Docker Inc. n.d. C)
Sovelluskonttirekisteri on tärkeä osa sovelluskontti ympäristöä, sillä juuri se tarjoaa monet sovelluskonttien eduista. Esimerkiksi versionhallinta on käytännössä sisäänrakennettu ominaisuus sovelluskonttikuvissa, mutta sitä ei saa hyödynnettyä ilman sovelluskonttirekisteriä. (Wong, 2019)

Vaikka tässä puhutaan sovelluskonttirekistereistä tarkoittaen palvelinympäristössä hallinnoituja palveluita, niin sovelluskonttien versionhallinta perustuu oikeastaan eräänlaiseen paikalliseen rekisteriin, jollaisen esimerkiksi Docker perustaa aina asennusvaiheessa. Tämän paikallisen rekisterin tehtävä on käsitellä sovelluskonttikuvien tasoja ja niiden periytymisiä. Tätä ei pidä kuitenkaan sekoittaa paikallisesti suoritettuun Dockerin sovelluskonttirekisteri palveluun, joka toimii sovelluskonttina ja toteuttaa sovelluskonttirekisterin lataus mahdollisuuksineen. (Docker Inc. n.d. D)

3.2.6 Container

Container eli sovelluskontti on sovelluskonttikuvasta suoritettava sovellus ympäristömuuttujineen (asetuksineen ja muutoksineen). Sovelluskonttikuvasta suoritettava sovelluskontti muodostuu sovelluskonttikuvan kuvakerroksista, jotka ovat vain lukutilassa sekä sovelluskontti kerroksesta, joka on kaikista muista kerroksista poiketen luku- ja kirjoitustilassa. Sovelluskonttikerrokseen kirjoitetaan kaikki muutokset, joita kontin suorituksessa syntyy verrattuna sovelluskonttikuvaan. (Docker Inc. n.d. E)

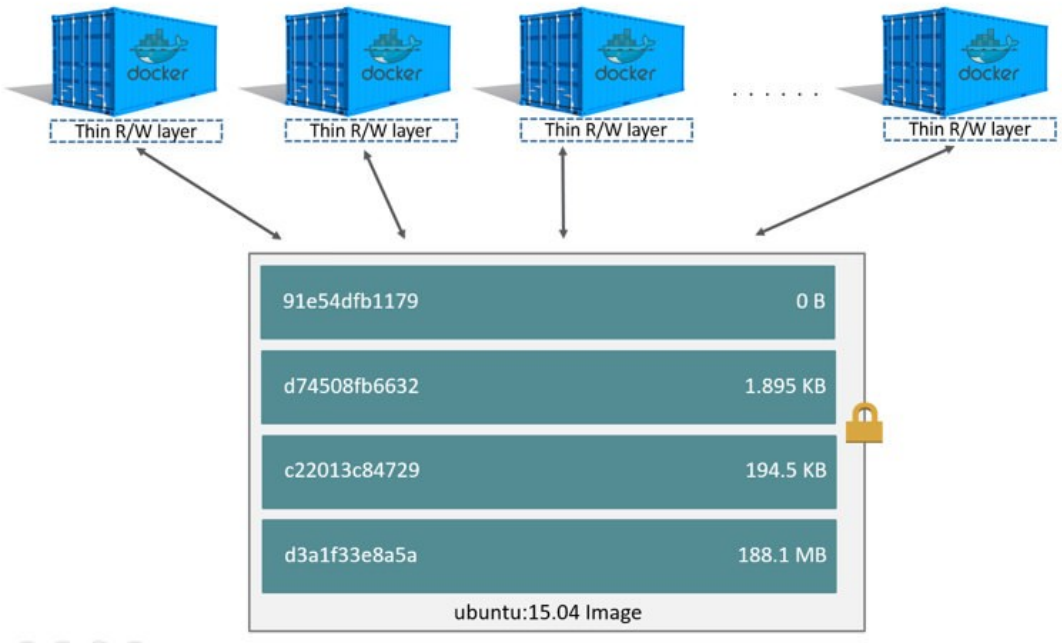


Kuvio 5. https://docs.docker.com/storage/storagedriver/

Alla olevassa kuvassa näkyy sovelluskonttikuva, joka on muodostettu Ubuntusta ja siitä on ”käynnistetty” neljä erillistä sovelluskonttia. Kaikki käynnistetyt sovelluskontit käytävät samaa sovelluskonttikuvaa, joka koostuu neljästä vain lukutilassa olevasta kerroksesta. Lisäksi jokaisella sovelluskontilla on oma luku-/kirjoitustilainen konttitaso, johon tallentuvat kaikki muutokset verrattuna käynnistyskuvaan. Toisin kuin sovelluskonttikuva konttitaso säilyy vain siihen asti, kunnes sovelluskontti lopetetaan. Periaatteessa konttitaso mahdollistaa, vaikka koko tietokantapalvelun toteuttamisen kontissa, mutta sovelluskonttien toiminta-ajatukseen sellainen istuu huonosti. Itse asiassa sovelluskontteja käytetään paljon tietokantapalvelujen toteuttamiseen, mutta tällöin tietokanta yleensä tallennetaan sovelluskontin ulkopuolelle pysyvään tallennustilaan, joka ei tyhjene sovelluskontin lopettamisen tai kaatumisen yhteydessä, sillä vaikka sovelluskonttien varmuuskopiointi on mahdollista, on se vastoin koko sovelluskonttien filosofiaa. Se perustuu ajatukseen nopeasti käynnistyvistä sovelluspaketeista, joita voidaan siirtää, kloonata ja korvata uudella versiolla nopeasti ja mahdollisimman kevyellä hallinnalla.

(Docker Inc. n.d. E)

.



Kuvio 6. https://docs.docker.com/storage/storagedriver/

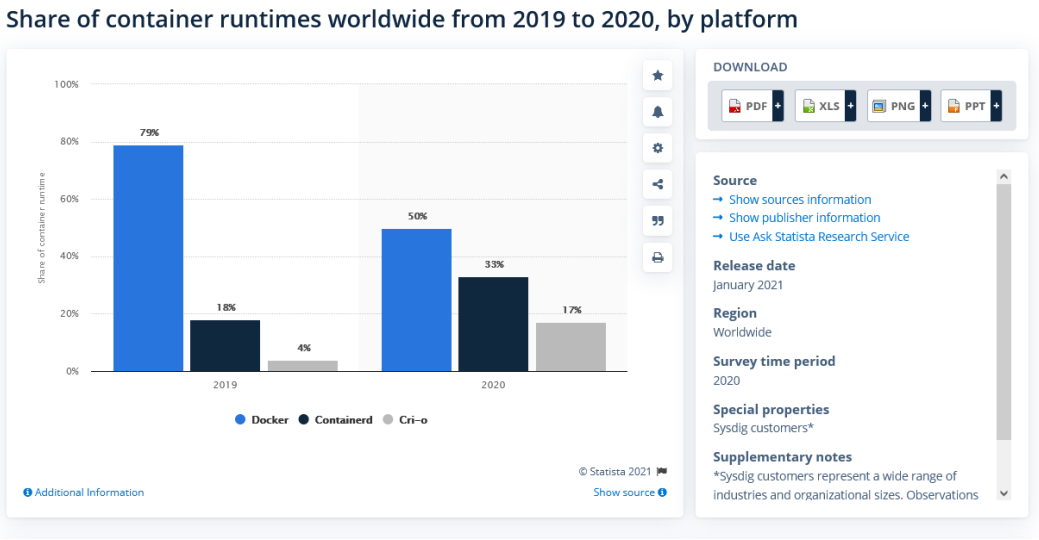
Container runtime

Container runtime on ohjelmisto, joka suorittaa sovelluskontin ja hallinnoi siihen tarvittavia komponentteja. Container runtime voi olla ylemmän tason (higher-level container runtimes) tai alemman tason (lower-level container runtimes). Tässä työssä container runtime tarkoittaa alemman tason container runtime ja container engines tarkoittaa ylemmän tason higher-level container runtimea.
Jako ei ole erityisen selkeä, kuten low- ja high -nimistä voi arvata. Container runtime ja container engine -termejä käytetään monesti sekaisin, koska monessa yhteydessä sillä ei ole merkitystä. Molemmat suorittavat kontteja ja hallinoivat suorituksessa tarvittavia sovelluskomponentteja. (Velayudhan, 2021)

Container runtimen toiminta perustuu ensisijaisesti kolmeen Linuxin -ytimen peruskomponenttiin. Namespaces (nimiavaruudet) määrittelee, keiden kanssa palvelu voi keskustella, sillä vain samaan nimiavaruuteen kuuluvat palvelut näkevät toisensa. Cgoups mahdollistaa resurssien jakamisen sovelluskohtaisesti, antaen mahdollisuuden hallita ja rajoittaa tehokkaasti sovelluksen käytössä olevien resurssien määrää. LSM (Linux security modules) ja erityisesti sen SELinux -moduuli mahdollistaa käyttöoikeuksien määrittelyn sovelluskohtaisesti ja käyttöoikeuksien eheyden valvonnan estäen esimerkiksi puskuriylivuodon kautta tapahtuvat käyttöoikeuksien korotukset. (Red Hat, 2019) Yhdessä nämä palvelut luovat edellytykset sovellusten suorittamiseen eristetyssä ja suojatussa ympäristössä. Container runtimen perustehtävä on automatisoida ja yhdistää näiden kolmen palvelun toiminta. (Baker, 2020)

Ensimmäinen container runtime LXC (2008) julkaistiin pian sen jälkeen, kun cgoups lisättiin Linuxin ytimeen. LXC (Linux containers) ei kuitenkaan tavoittanut suuria massoja, mutta se toimi perustana Canonnial JuJu- ja Docker -projekteille. Docker (silloiselta nimeltään dotCloud) aloitti omien käyttäjä- kehittäjäystävällisten työkalujensa luomisen LXC:n ympärille. Marraskuussa 2013 julkaistu Docker käytti LXC:tä suoritusympäristönään, mutta jo vuonna 2014 julkaistussa 0.9 versiossa Docker oli korvannut LXC:n omalla libcontainer -ympäristöllään. (Docker Inc., n.d. F)

Docker-sovelluskontteilla on yhä noin 50% markkinaosuus



Kuvio 7. https://www.statista.com/statistics/1224618/container-platforms-deployed-runtime/

3.3 The Open Container Initiative (OCI)

OCI on Linux Foundationin alaisuudessa perustettu kevythallintoinen projekti, jonka tarkoituksena on luoda avoimia standardeja sovelluskonttiformaateille sekä sovelluskonttien ajoympäristölle. OCI sisältää kaksi määrittelyä. Image Specification (image-spec) määrittelee sovelluskonttikuvan rakenteen. Runtime Specification (runtime-spec) määrittelee, kuinka sovelluskonttikuva ladataan, avataan ja suoritetaan. (Open-containers, 2020)

Yleisemmin tunnetuista tuotteista (Docker, OpenShift,Kubernetes, Azure yms.) lähes kaikki tukevat OCI-määrityksiä. Käytännössä saattaa olla pieniä eroja standardin toteutuksen suhteen, kuten esimerkiksi OpenShift -osassa käsitelty Docker konttien yhteensopivuus OpenShift -ympäristön kanssa.

4 SOVELLUSKONTTIEN HALLINTA

4.1 Kubernetes

Avoimen lähdekoodin Kubernetes tunnetaan myös nimellä K8s ja se on tarkoitettu sovelluskonttien käyttöönottoon, skaalaukseen ja hallinnointiin. (Kubernetes, n.d. A) Kubernetes on Googlen kehittämä avoimen lähdekoodin projekti, jonka Google julkaisi 2014. Kubernetesin sanotaan monessa yhteydessä olevan Googlen omassa käytössä olevan Borgin avoimen lähdekoodin versio ja Kubernetesin historian katsotaan yleisesti alkavan Borgin mukana. (Hámori, 2022)

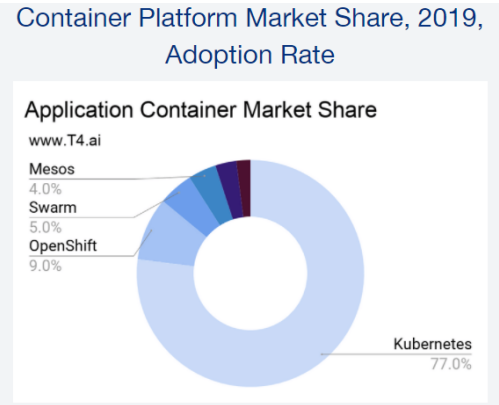
Googlen ensimmäinen yhtenäinen sovelluskonttien hallintajärjestelmä oli vuonna 2003 julkaistu Borg, mutta sen edeltäjinä toimivat Babysitter (long-running services) ja eräajoista vastannut Global Work Queue. Varsinkin Global Work Queue -arkkitehtuurin kerrotaan vaikuttaneen vahvasti Borgin kehitykseen, vaikka sen toiminta keskittyi pelkästään eräajoihin. (Burns ym., 2016, s. 70-90)
Borgin tehtävä on jakaa koneita molempien sovellustyyppien kesken tehostaen resurssien käyttöä. Tämän mahdollisti Linuxin -ytimeen lisätty tuki sovelluskonteille ja sen lähdekoodista Google oli lahjoittanut huomattavan osan. Borgissa ajettavien sovellusten lisääntyessä jatkuvasti kehittivät Googlen sovellus- ja järjestelmätiimit kokonaisen ekosysteemin työkaluja ja palveluita Borgin ympärille. Kehitys perustui Googlen eri tiimien tarpeisiin ja sen tuloksena Borgin ekosysteemi muodostui hyvin tehtäväkohtaisista ratkaisuista ja koko ympäristöstä tuli erittäin heterogeeninen. Tästä johtuen Borgin ohjaaminen vaati useita hallinnointikieliä ja -prosesseja.

Borgin jatkuvasti laajetessa sen heterogeeninen rakenne sai Googlen aloittamaan uuden sovelluskonttien hallintajärjestelmän kehittämisen. Projektin nimeksi tuli Omega ja sen tavoitteena oli parantaa Borgin ohjelmistosuunnittelua. Vuonna 2013 esiteltyyn Omegaan otettiin mukaan monia Borgin hyväksi havaittuja malleja, mutta se suunniteltiin alusta alkaen noudattamaan yhtenäisempää ja säännönmukaisempaa arkkitehtuuria. Se tallensi klusterin tilan keskitettyyn Paxos -pohjaiseen tietovarastoon, johon klusterin ohjaustason toimijat, kuten vuorottajat (scheduler) pääsivät käsiksi käyttäen optimistisen samanaikaisuuden hallintaa (Optimistic concurrency control) ristiriitojen

ratkaisemiseksi. Muutos mahdollisti Borgmasterin toiminnallisuuden pilkkomisen itsenäisiin osiin, jotka kykenivät toteuttamaan muutoksia ilman jatkuvaa keskitettyä hallintaa. Monia Omegan ratkaisuja on otettu käyttöön myöhemmin Borgissa. (Burns ym., 2016, s. 70-90)

Vuonna 2013 Dockerin aloittama räjähdysmäinen sovelluskonttien suosion kasvu sekä Googlen päätös alkaa myydä julkista pilvi-infrastruktuuria sai Googlen aloittamaan kolmannen sovelluskonttien hallintajärjestelmänsä kehityksen. Projektin vetäjien tavoitteena oli luoda avoimen lähdekoodin sovelluskonttien hallintajärjestelmä, joka yhdistäisi Googlen Borg- ja Omega-sovelluskonttihallintajärjestelmien parhaat ominaisuudet Docker -sovelluskonttien kanssa. (Smith, 2021) Brendan Burns kertoi 2020 Kubecon Europen haastattelussa, että heiltä kului huomattava määrä työajastaan, kun he yrittivät saada Googlen johdon vakuuttumaan, että Project Seven of Nine kannattaa toteuttaa avoimen lähdekoodin projektina. Alkukesästä 2014 Googlen avoimen lähdekoodin projekti esiteltiin nimellä Kubernetes. Kubernetes v1.0 julkaistiin 21.7.2015 ja samalla Google liittoutui Linux Foundationin kanssa Cloud Native Computing Foundationin (CNCF) muodostamiseksi sekä tarjosi Kubernetesin sen perusteknologiaksi. (Hámori, 2022)

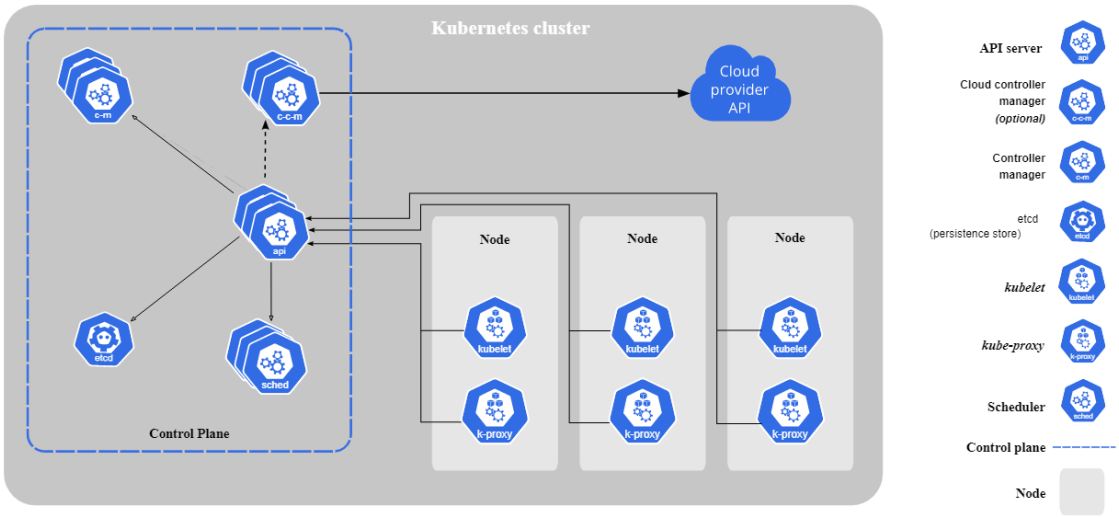
Kubernetesin kehitys on ollut nopeaa ja siitä on tullut neljässä vuodessa sovelluskonttien hallintajärjestelmien kiistaton markkinajohtaja ja alan defacto, kuten kuvasta 8 selviää.



Kuvio 8. https://www.t4.ai/industries/container-platform-market-share

4.2 Kubernetesin osat

Kubernetes klusterin osat alla olevassa kuvassa hallinnollisessa muodossa.



Kuvio 9. https://kubernetes.io/docs/concepts/overview/components/

4.2.1 Nodes

Kubernetesin Nodes eli solmut ovat klusterista riippuen tietokoneita tai virtuaalikoneita, jotka suorittavat sovelluskontteja sisältäviä Podeja. Normaalissa Kubernetes-ympäristössä on useita solmuja, mutta esimerkiksi testiympäristössä voi olla vain yksi solmu. Solmun komponentteina on aina kubelet, kube-proxy sekä ajoympäristö. Solmun tila (node status) sisältää neljä kenttää.

-Osoitetietoihin kuuluu hostname eli isäntänimi, jolla laite tunnetaan verkossa, ulkoinen ip-osoite (external ip), joka on tavoitettavissa klusterin ulkopuolelta, sekä sisäinen ip (internal ip), joka on reititettävissä vain klusterin sisällä.

-Conditions -kenttä kertoo solmun tilan. Niitä ovat esimerkiksi valmis, muisti vähissä (MemoryPressure), levytila vähissä (DiskPressure) ja verkko tavoittamattomissa (NetworkUnavailable).

-Capacity and allocatable kertoo solmun resursseista kuten muistista ja prosessorista sekä suurimman mahdollisen määrän podeja. Kapasiteetti kertoo resurssien kokonaismäärän ja allokoitavissa -kohta tavallisille podeille saatavissa olevat resurssit.

-Info-kenttä sisältää yleistä tietoa solmusta, kuten kernelin version, Kubernetesin version, ajoympäristön tiedot sekä solmun käyttämän käyttöjärjestelmän. Solmut lähettävät säännöllisesti heartbeats -viestejä, joista klusteri tietää solmun olevan tavoitettavissa. (Kubernetes, 2022A)

4.2.2 Node Components (solmun komponentit)

Jokaisessa solmussa toimivat solmukomponentit ylläpitävät käynnissä olevia Podeja sekä tarjoavat Kubernetesin ajonaikaisen ympäristön.

-Kubelet toimii jokaisen solmun ensisijaisena agenttina ja rekisteröi solmut API-serveriin. Kubelet toimii PodSpecin perusteella ja valvoo, että PodSpeciin merkityt sovelluskontit ovat käynnissä ja terveitä. (Kubernetes, 2022F)

-Kube-proxy on Kubernetesin välityspalvelin, joka toteuttaa jokaisessa solmussa Kubernetesin palvelumallia. Sen tehtävänä on toteuttaa solmun verkkosääntöjä, mahdollistaen tarvittavat yhteydet Podiin. (Kubernetes, 2022G)

4.2.3 Control Plane Components

API Server on Kubernetesin ohjaustason API-rajapinnan käyttöliittymä, jonka pääosa on kube-apiserver. Kube-apiserverin tehtävä on määritellä sekä vahvistaa tiedot api-objekteille, kuten esimerkiksi podeille, palveluille ja replikointiohjaimille. (Kubernetes, 2022E)

Etcd on hajautettu avainarvotietokantavarasto (key-value database store), jota Kubernetes käyttää Kubernetes Clustereiden konfigurointitietojen säilyttämiseen. Lisäksi se tallentaa Kubernetesin toimintaideologian kannalta ehkä kaksi tärkeintä asiaa, eli objektin tämänhetkisen tilan (status) sekä halutun tilan (specification), joita vertaamalla kyseisestä objektityypistä vastaava kontrolleri tekee tarvittavat toimenpiteet saavuttaakseen ja ylläpitääkseen halutun tilan. Etcd sisältää kaikki *kubectl get 'namespace'* tulosteen tiedot. (Palmer, n.d.)

Kubernetesissä schedulerin tehtävä on vastata, että PODit on sidottu NODEen, jotta kubelet voi suorittaa PODIN. Kube-scheduler seuraa äskettäin luotuja Podeja joille ei ole vielä osoitettu omaa NODEa ja sitoo (binding) PODin optimaalisimpaan vaatimukset täyttävään NODEen. Kube-scheduler toteuttaa PODille sopivan solmun valinnan kahdessa vaiheessa. Ensimmäisessä vaiheessa suodatus luo listan niistä solmuista, jotka täyttävät PODin vaatimukset. Mikäli lista jää tyhjäksi, ei PODia voida vielä sijoittaa, vaan joudutaan odottamaan resurssin vapautumista. Mikäli taas listalle jää useampi vaihtoehtoinen NODE, niin scheduler aloittaa solmujen pisteytyksen, jonka säännöt voivat perustua *Scheduling Policies* tai *Scheduling Profiles* -määrityksiin. Lopuksi Scheduler valitsee PODille korkeimmat pisteet saaneen NODEn. (Kubernetes, 2022D)

Kube-controller-manager hallinnoi eri ohjaus prosesseja (controller processes), joiden tehtävänä on valvoa itselleen kuuluvien Kubernetes -objektien tilaa (*spec field)* ja tehdä tarvittavat toimenpiteet, jotta objektien tila saavuttaisi halutun tilan (*desired state*). Kubernetesin ohjausprosessit toteuttavat objektien ohjauksen joko Kubernetesin API-serverin kautta viestimällä tai vaikuttamalla suoraan kohteisiin. Ohjausprosessin tehdessä muutoksia klusterin ulkopuolisiin kohteisiin se kommunikoi suoraan kohteen kanssa, mutta ensin se selvittää API-serverin kautta halutun tilan ja päätyessään tekemään muutoksia ohjain ilmoittaa lopuksi API-serverille tekemänsä muutokset. Kubernetesin mukana on eräitä ohjaimia valmiiksi, vaikka Kubernetes mahdollistaa myös muiden ohjaimien käytön. Mukana tulevia ohjaimia ovat esimerkiksi Node controller, joka vastaa solmujen tilan seuraamisesta ja toimenpiteistä, mikäli solmu kaatuu. Job controller seuraa kertaluonteisia tehtäviä ja luo niille tarvittavat PODit. Endpoint Controller yhdistää PODit ja niihin kuuluvat palvelut. Service Account & Token controllers luo API-käyttövaltuudet ja tilit uusille nimiavaruuksille. (Kubernetes, 2021)

Cloud-controller-manager on pilvipalveluiden ohjain, joka keskustelee pilvipalveluntarjoajan API-rajapinnan kanssa mahdollistaen klustereiden hallinnan pilvipalveluissa. Cloud-controller-managerin käyttämä ohjain riippuu käytettävästä pilvipalvelusta. Vain Node Controllerin, Route controllerin sekä Service controllerin toiminnalla on

riippuvuuksia palveluntarjoajan ympäristön kanssa. Node -kontrolleri seuraa vastaamattomien solmujen tilaa pilvipalvelussa. Reititys -kontrolleri vastaa reitityksen toteuttamisesta pilvipalvelussa. Palvelu -ohjain huolehtii palveluntarjoajan kuormantasaajista. (Kubernetes, 2022B)

4.3 Namespace

Linuxin kerneliin vuodesta 2002 (2.4.19 mount) asti kuuluneet nimiavaruudet ovat sovelluskonttien perustavanlaatuinen osa, sillä ne mahdollistavat sovelluskonttien edustaman käyttöjärjestelmätason virtualisoinnin. (Rathnayaka,2018)
Nimiavaruudet eristävät ajettavat sovelluskontit isäntäjärjestelmästä sekä toisistaan. Nimiavaruus määrittelee resurssit, joita sovelluskontti näkee, vain samaan nimiavaruuteen kuuluvat resurssit näkyvät sovelluskontille. Kernelin versiosta 5.6 lähtien erilaisia nimiavaruuksia on yhteensä 8 kappaletta.

- Mount (mnt) eristää nimiavaruuden tiedostot toisten nimiavaruuksien tiedostoista. Tuloksena on prosessikohtainen ja nimiavaruuskohtainen juuritiedostojärjestelmä. Toimii samoin kuin Linuxin ensimmäinen käyttöjärjestelmätason virtualisointiratkaisu, eli chroot-komento.
- Process ID (pid) eristää nimiavaruudessa ajettavat prosessit muista nimiavaruuksissa ajettavista prosesseista. Uudessa nimiavaruudessa ensimmäinen prosessi saa pid -numerokseen 1 ja seuraavat aliprosessit perivät siltä juoksevan numeroinnin. Tavallinen prosessi voi saada vain nimiavaruudessa itselleen pid -numerokseen 1, sillä Linux ympäristössä pid 1 on varattu järjestelmän käynnistysprosessille (init).
- Network (net) mahdollistaa jokaiselle nimiavaruudelle oman IP-osoitteen kaikkine portteineen. IP-osoitteiden luominen nimiavaruuksille on välttämätöntä, sillä vaikka prosessit on eroteltu toisistaan, niin esimerkiksi sähköpostipalvelimen sisältävien sovelluskonttien skaalaaminen edellyttää, että jokaiselle samalle palvelimelle käynnistetylle sähköpostisovelluskontille pystytään osoittamaan oma varaamaton portti 25. (Red Hat, 2021B)

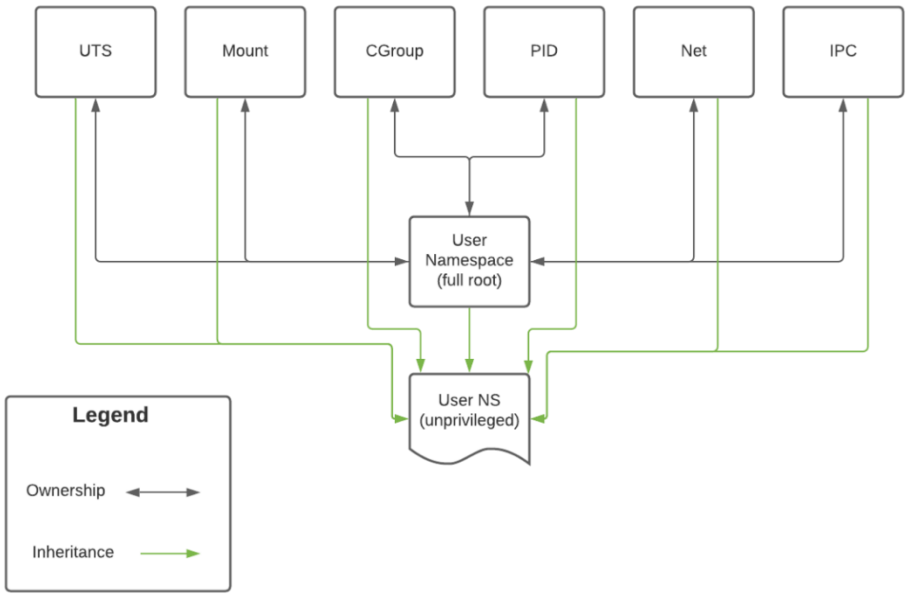
- Interprocess Communication (ipc) IPC-nimiavaruuden kautta on mahdollista eristää prosessien viestiketjut toisistaan. Normaalisti prosessi perii kaikki

avatut IPC-resurssit, kuten semaforat, putket, signaalit, viestijonot ja jaetut muistisegmentit, mutta IPC-nimitilassa ne eivät näe ylemmän nimitilan resursseja. (Kirov, 2021)

- UTS Unix Timesharing System toteuttaa nimestään huolimatta hyvin toisenlaisen toiminnon. UTS mahdollistaa samassa järjestelmässä suoritettaville prosesseille erilliset host names (isäntänimet) ja domain names (domain-nimet). Muutokset näkyvät kaikille samassa UTS-nimiavaruudessa oleville prosesseille, mutta eivät muissa nimiavaruuksissa oleville prosesseille. (Ovens, 2022)

- Time Namespace on uusin nimiavaruus ja se julkaistiin maaliskuussa 2020 Linux kernelin 5.6 versiossa. Time Namespace on tarkoitettu ensisijaisesti juuri sovelluskontteihin tarjoten nimiavaruuskohtaisen aikamäärityksen. Ensisijainen käyttötarkoitus on mahdollistaa siirtymän (offset) CLOCK_MONOTONIC ja CLOCK_BOOTTIME kelloille. Kyseisillä kelloilla mitataan aikaviipaleita ja asetetaan ajastimia. Kyseiset tiedot ovat tärkeitä prosesseja palautettaessa sekä siirrettäessä, sillä aika-arvot eivät saa milloinkaan pienentyä. (Larabel, 2020)

- User ID (user) käyttäjänimiavaruus on Linuxin ensisijainen nimiavaruus ja se hallitsee kaikkia muita nimiavaruuksia. Nimiavaruuden oikeudet määräytyvät aina kyseisen nimiavaruuden pääkäyttäjän nimiavaruuden (user namespace) oikeuksien mukaan. Kuvasta 10 näkyy, kuinka pääkäyttäjän nimiavaruuden omistussuhteet voivat olla kaksisuuntaisia. Tällöin verkkonimiavaruudessa pääkäyttäjänä suoritettu prosessi pystyy vaikuttamaan kaikkiin pääkäyttäjän nimiavaruuden omistamiin prosesseihin.



Kuvio 10. https://www.redhat.com/sysadmin/building-container-namespaces

Luotaessa uusi etuoikeudeton käyttäjä hänen käyttäjänimiavaruutensa perii pääsyn muihin luotaessa käytettyihin nimiavaruuksiin. Periytymisessä ei kuitenkaan siirry omistajuus, joka on sidottu nimiavaruuden pääkäyttäjän käyttäjänimiavaruuteen. Tämän vuoksi uusi käyttäjä pystyy käyttämään (lukemaan) verkkonimiavaruuden ip-osoitetta ping-komennolla, mutta ei määrittelemään uutta ip-osoitetta verkkoliitännälle, jonka verkkonimiavaruuden omistaa verkkonimiavaruuden pääkäyttäjäksi merkitty käyttäjänimiavaruus.
(Ovens, 2021)

- Control group (cgroup) Namespace Cgroup on Linux-ytimen järjestelmä resurssien hallinnointiin. Googlen insinöörien vuonna 2006 esittelemä ominaisuus sisällytettiin Linux-ytimeen tammikuussa 2008 julkaistussa 2.6.24

versiossa. Cgroupilla on neljä toisiinsa läheisesti liittyvää ominaisuutta, joiden tarjoamat toiminnallisuudet ovat erityisen tärkeitä sovelluskonttiympäristössä. Cgroupista on Linux-ytimeen kaksi versiota, eli versiot v1 ja v2. Cgroup v2 julkaistiin jo 2016, mutta sen käyttö on yhä vähäistä. Versio 2 on esimerkiksi sisällytetty Red Hat Enterprise Linux 8 (RHEL8) jakeluun, mutta se on oletuksena kytketty pois päältä. Suurin syy versiossa 1 pysymiseen saattaa löytyä sovelluskonteista. Sovelluskonteissa on käytännössä vain versio 1 käytössä, sillä Kubernetes, Openshift ja Docker käyttävät yhä Cgroupin versiota 1 ja yhdessä näiden kolmen toimijan markkinaosuudet kattavat lähes koko sovelluskonttien markkinat. Cgroup koostuu nimensä mukaisesti useista alijärjestelmistä. Esimerkiksi RHEL6 jakelussa Cgroup sisältää 10 alisysteemiä, joista lisää tietoa on esimerkiksi alla olevassa lähteessä. (Red Hat, n.d. A)
Cgroupin ensisijainen ja käytetyin ominaisuus on resurssien käytön rajoittaminen. Tällöin Cgroup valvoo, että prosessien käyttämät resurssit kuten RAM-muisti, prosessorin käyttö tai I/O-laitteen käyttö pysyy ennalta määritetyissä rajoissa. (Riel, 2016)

Toinen Cgroupin tehtävä on toteuttaa prosessien priorisointi. Teknisesti kyseessä on resurssien rajoittaminen, mutta priorisoinnissa rajoitukset eivät perustu suoraan ennalta määritettyihin rajoihin, vaan eri prosessien prioriteettiin. Yksinkertaisimmillaan se tarkoittaa, että prosessi A saa enemmän aikaa kuin prosessi B. Eräs sovelluskonttien suosion syistä on tavoite saada mahdollisimman korkea käyttöaste järjestelmän resursseille. Sovelluskonttien suorittamisen yhteydessä priorisoinnin ja rajoittamisen ero tulee ehkä selvimmin esille. Priorisoinnissa Cgroup puuttuu resurssien käyttöön vain tarvittaessa, esimerkiksi microserviceistä koostuvassa palvelussa, muita sovelluskontteja ohjaavalla palvelulla olisi määritelty korkeampi prioriteetti sekä cpu:ssa että verkossa. Tällöin sillä on etuoikeus esimerkiksi cpu:n ja verkkon käyttöön, mahdollistaen ohjausprosessin välittömän suorittamisen sekä ohjauskomentojen nopean ja mahdollisimman yhtäaikaisen välittämisen ohjattaville sovelluskonteille. Tämä mahdollistaa huomattavasti tehokkaamman järjestelmän resurssien käytön kuin mitä ennalta määrätyt rajoitukset mahdollistaisivat. (Ovens, 2020; Riel, 2016)

Kolmas Cgroupin ominaisuus on Accounting. Ominaisuus on useimmissa kaupallisissa Linux -versioissa oletuksena pois päältä, mutta tarjoaa mahdollisuuden seurata tarkasti resurssien käyttöä prosessi- ja Cgroup -kohtaisesti. (Ovens, 2020)

Neljäs Cgroupin tarjoama ominaisuus on prosessien hallinta, joka perustuu Cgroupin Freezer -alijärjestelmään. Freezer mahdollistaa käynnissä olevien tehtävien pysäytämisen (suspend) säilyttäen niiden tilan. Ominaisuus on käytännöllinen esimerkiksi analysoitaessa prosessin toimintaa tai prosessin kuluttaessa liikaa resursseja, jolloin freezer mahdollistaa sen keskeyttämisen, kunnes resursseja on jälleen tarpeeksi käytettävissä. Sovelluskonttien kannalta kiinnostavinta on kuitenkin keskeytyksen yhteydessä luotava tilannevedos, joka mahdollistaa suorituksessa olevan prosessin keskeyttämisen ja siirtämisen toiselle palvelimelle. Siirron jälkeen prosessi palautetaan tilannevedoksesta ja se jatkaa suoritusta keskeytystä edeltäneessä tilassa. (Containerlabs, n.d.) Cgroup yhdessä muiden Linuxin nimiavaruuksien kanssa tarjoaa pitkälti tekniikan, johon sovelluskonttien toteutus perustuu. (Docker Inc. n.d. A)

5 SOVELLUSKONTTIYMPÄRISTÖT

5.1 VMware Tanzu

Tanzu on virtualisointiratkaisujen pioneerin VMwaren ratkaisu sovelluskonttien hallinnointiin. Tuotteen itsensä lisäksi kiinnostavuuteen vaikuttaa myös tuotteen tarjoaja VMware. Vuonna 1998 perustetun yhtiön nimi on parhaimmillaan ollut lähestulkoon synonyymi virtualisoinnille. Pilvipalveluiden ja sovelluskonttien suosio on kasvattanut virtualisoinnin markkinoita voimakkaasti ja VMware on menettänyt huomattavasti suhteellista osuuttaan kokonaismarkkinoista. Se hallitsee kuitenkin SDI (Software-Defined Infrastructure) ja HCI (Hyperconverged Infrastructure) -markkinoita, eli laiteympäristöjen virtualisointia abstraktiksi ympäristöksi, jossa laitteiden sijasta hallitaan ja käytetään virtuaalisia resursseja. (VMware, 2021A)
(Wong, 2020)

VMware Cloud Provider -listalta löytyy hieman yli 500 palveluntarjoajaa ja strateginen yhteistyö kaikkien tärkeimpien pilvipalveluiden tuottajien kanssa, kuten AWS, Azure, Google Cloud, IBM Cloud, Oracle Cloud. (VMware, n.d.), (VMware, n.d.)

VMwaren asemaa markkinoilla sekä siihen kohdistuvista odotuksista kertoo Broadcomin päätös ostaa VMware 61 miljardilla dollarilla. Perspektiiviä kauppaan voi hakea vanhentuneesta, mutta suomalaisille tutusta vertailukohdasta, eli Nokia-kaupasta 2013, jossa Nokia-puhelimet myytiin Microsoftille 3,79 miljardilla eurolla. (VMware, 2022) (Microsoft, 2013)

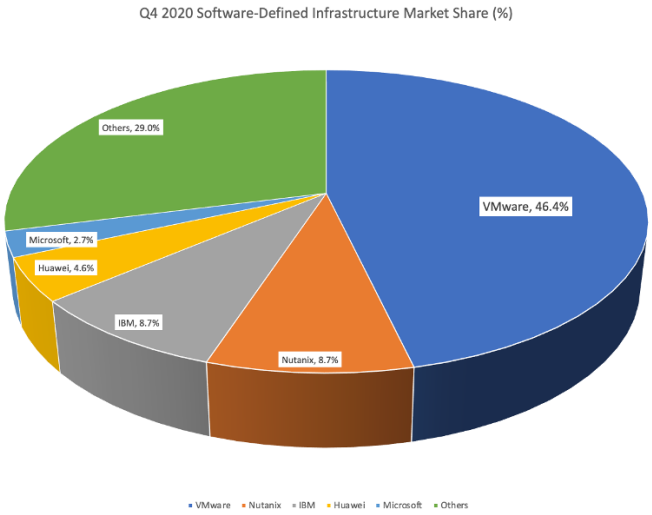
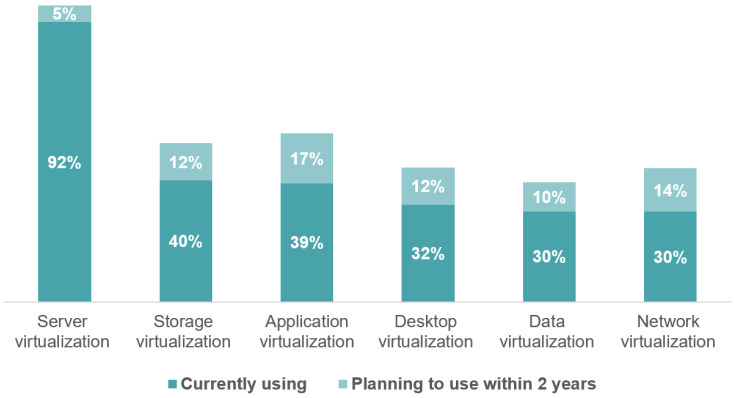
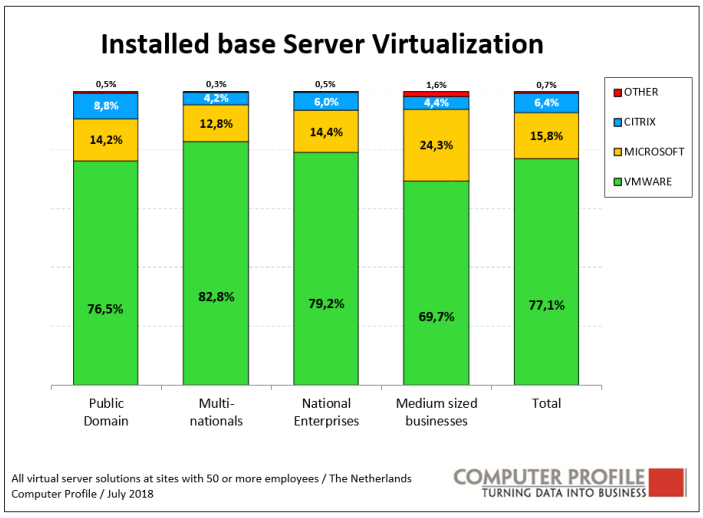


Figure 1: Software-Defined Infrastructure Market Share by Revenue (Source: IDC Semi-annual Software-Defined Infrastructure Tracker 2020 H2 Dated June 3, 2021)

Kuvio 11. https://blogs.VMware.com/cloud-foundation/2021/08/10/VMware-leads-software-defined-infrastructure-software-market/



Kuvio 12. https://www.spiceworks.com/marketing/reports/state-of-virtualization/



Kuvio 13. https://www.smartprofile.io/analytics-papers/VMware-further-expands-market-share-server-virtualization/

Elokuussa 2019 julkaistun Tanzun perusta on VMwaren ostaman Pivotal Labs -yhtiön sovelluskehitystekniikka sekä sovelluskonttien hallinnoinnista vastaava Kubernetes. (VMware, 2019B) Lisäksi Bitnamin hankinnalla VMware sai haltuunsa johtavan sovelluspakkausratkaisujen valmistajan, joka oli erikoistunut suurien pilvi- ja Kubernetes -ympäristöjen sovellusten ja kehityspintojen asennusten yksinkertaistamiseen paketoinnin avulla. (VMware, 2019A), (VMware, 2019C)

VMware Tanzu mahdollistaa virtuaalikoneiden, sovelluspohjaisen infrastruktuurin ja sovelluskonttien hallinnan saman ylläpitäjille ennestään tutun käyttöliittymän ja logiikan kautta.

Taulukossa 1 esitellään lyhyesti useimmat VMware Tanzu tuoteperheen tuotteet ja laajennukset.

Cloud native app development and data

Tanzu Application Platform	App aware platform that runs on any Kubernetes and any cloud
Tanzu Application Service	A modern runtime for microservices
Tanzu Build Service	Build containers from source code for Kubernetes
Tanzu Data Services	Cloud native data and messaging including GemFire, RabbitMQ, SQL, and Greenplum
VMware Application Catalog	Curated image catalog

Enterprise grade Kubernetes

Tanzu for Kubernetes Operations	Multi-cloud container infrastructure
Tanzu Kubernetes Grid	Enterprise-ready Kubernetes runtime
Tanzu Mission Control	Multi-cloud Kubernetes management
Tanzu Observability	Enterprise observability for multi-cloud environments
Tanzu Service Mesh	Connectivity and security for modern applications
Tanzu Standard Edition	Operate a multi-cloud Kubernetes platform

Taulukko 1. https://tanzu.VMware.com/products

Lisäksi on helmikuussa 2022 julkaistu avoimen lähdekoodin projekti Tanzu Community Edition, jonka sivut löytyvät osoitteesta: https://tanzucommunityedition.io/

Seuraavassa kuvassa xxi vertaillaan Tanzu -tuoteperheen eri versioihin sisältyviä ominaisuuksia. Basic- ja Standard -versioiden erot painottuvat pitkälti hallinnointiin ja erityisesti sovelluskonttien tilan seurantaan sekä mahdollisuuteen käyttää useita eri pilvipalveluita. Standard- ja Advanced -versioiden ero on pitkälti Advanced versiossa huomattavasti laajennettu tuki sovelluskehitykselle ja sen automatisoinnille.

	Tanzu Basic Edition LEARN MORE	Tanzu Standard Edition LEARN MORE	Tanzu Advanced Edition LEARN MORE
Support type	Commercial	Commercial	Commercial
Developer Frameworks			✓
Data Services			✓
Curated App Catalog			✓
Container Build			✓
Container Registry	✓	✓	✓
Service Mesh			✓
Observability/Monitoring		✓	✓+
Policy Management		✓	✓+
Conformance/Diagnostics		✓	✓+
Container Networking	✓	✓	✓+
Load Balancing	✓	✓	✓+
Ingress		✓	✓+
Kubernetes Runtime	✓	✓+	✓+
Identity and Access Management	✓	✓+	✓+
Cluster Lifecycle Management	✓	✓+	✓+
SaaS multi-cluster management		✓+	✓+
Operating System	✓	✓+	✓+
Data Protection		✓	✓
Logging	✓	✓	✓
Multi-Cloud Support		✓	✓
vSphere Support**	✓	✓	✓

*** vSphere 7 provides the following capabilities for all editions: Pod Service (requires NSX), Storage Service, Network Service and Registry Service (requires NSX)*

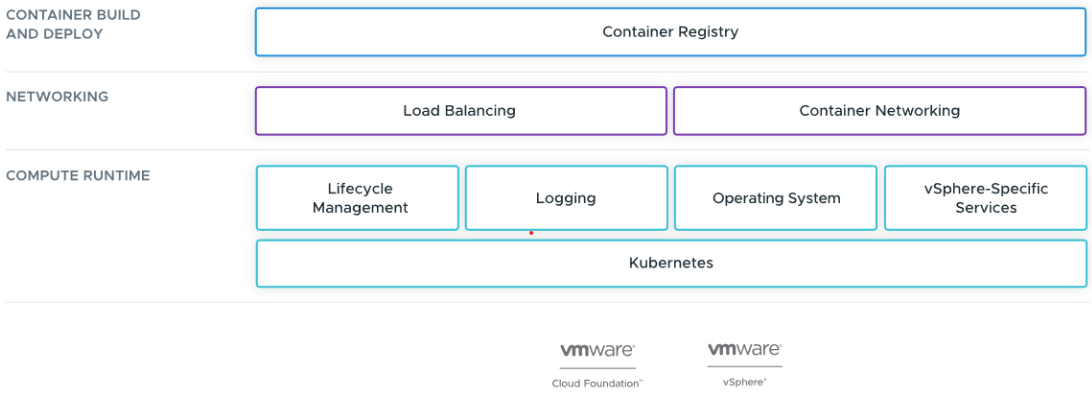
✓+ Includes expanded capabilities

Kuvio 14. https://tanzu.VMware.com/tanzu/compare

Työssä käsitellään Tanzu Basic -tuotetta, joka on VMwaren edullisin ja helppokäyttöisin kaupallinen versio Tanzusta. Tanzu voidaan integroida vSphere7 control planeen tai asentaa erillinen hallintaklusteri vSphere6.7u3 tai vSphere7- ympäristöön.

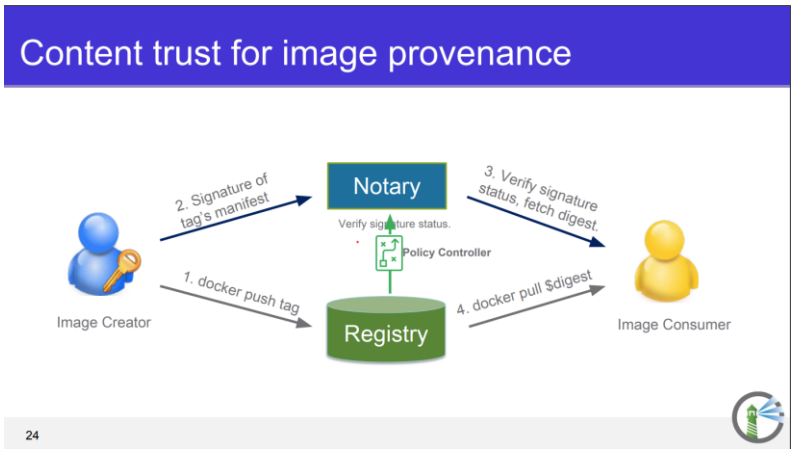
Integrointi vSphere7 -ympäristöön tekee mahdolliseksi ylläpitäjien hallita virtuaalikoneita ja sovelluskontteja rinnakkain ja yksinkertaistaa näiden hallintaa. Kehittäjille ratkaisu tarjoaa mahdollisuuden ottaa käyttöön itsepalveluresursseja Kubernetesin API:n kautta ja nopeuttaa siten sovelluskehitystä. Tanzu Basicin käyttämä Kubernetes -jakelu on avoimen lähdekoodin mukainen, mutta se on paketoitu asennusohjelmaan käyttöönoton helpottamiseksi. Tanzu Basic hyödyntää avoimen lähdekoodin projekteja toteuttaakseen Kubernetes -pohjaisen sovelluskonttien hallinnan. (VMware, 2021B)

Kuvassa 15 näkyy Tanzu Basic Editionin rakennekaavio.



Kuvio 15. https://tanzu.VMware.com/tanzu/basic

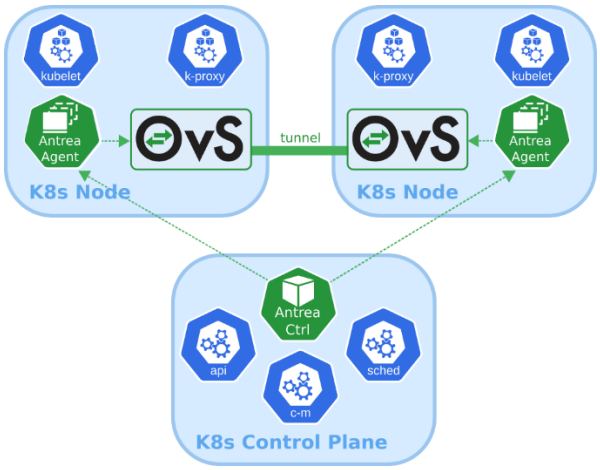
Container Registry on avoimen lähdekoodin CNCF Graduated -projekti Harbor, joka täyttää OCI-määritykset. (Harbor, 2020A) Harbour tukee rooliperusteista kulunvalvontaa antaen mahdollisuuden määritellä käyttäjäoikeudet sovelluskonttikuvaan projektikohtaisesti. Esimerkiksi sovelluskehityksessä ohjelmistokehittäjällä voi olla erittäin laajat käyttöoikeudet konttikuvaan, mutta saman kuvan käyttöoikeudet tuotantoympäristössä ovat rajoitetumpia. (Rosland, 2020) Harbour sisältää Trivy scannerin haavoittuvuuksien havaitsemiseen konttikuvista. (Harbor, 2020B) Harbour tukee Notaryn kautta konttikuvien aitouden tarkastamista kuvan XXXI mukaisesti.



Kuvio 16. https://www.cncf.io/wp-content/uploads/2020/08/harbor-cncf-webinar-1.pdf

Harbour integroituu LDAP/AD-palveluihin käyttäjien todentamista ja hallintaa varten, tämä mahdollistaa ylläpitäjille LDAP/AD-ryhmien tuomisen projekteihin jo aiemmin mainittujen projektikohtaisten käyttäjäoikeuksien määrittämiseksi. (VMware, 2022A)

Container Network on toteutettu avoimen lähdekoodin Antrea- tai Calico -ratkaisuilla. (VMware, 2022B) Oletuksena Tanzussa on käytössä Antrea, joka toimii OSI-mallin tasoilla 3-4 tarjoten verkko- ja tietoturvapalveluita Kubernetes-klusterille avoimen lähdekoodin Open vSwitch (OvS) -virtuaalikytkimien kautta. Kuten kuvassa XXXII näkyy. (Dong & Xin, 2022)



Kuvio 17. https://github.com/antrea-io/antrea#overview

Antrea lisää Kubernetesin verkkoon klusteritason verkkokäytännöt, erilliset tasot käytännöille sekä sääntöjen priorisoinnin. Lisäksi Antrea tarjoaa mahdollisuuden salata solmun podien välinen verkkoliikenne IPsec tai WireGuard tunneloinnilla. (Dong & Xin, 2022)

Load Balancing voidaan toteuttaa VMwaren NSX Advanced Load Balancerilla tai avoimen lähdekoodin HAProxyllä, mutta vSphere Pod servicen ja Registry servicen käyttö edellyttää VMwaren kaupalllisen lisäosan NSX:n käyttämistä. (VMware, 2021B)

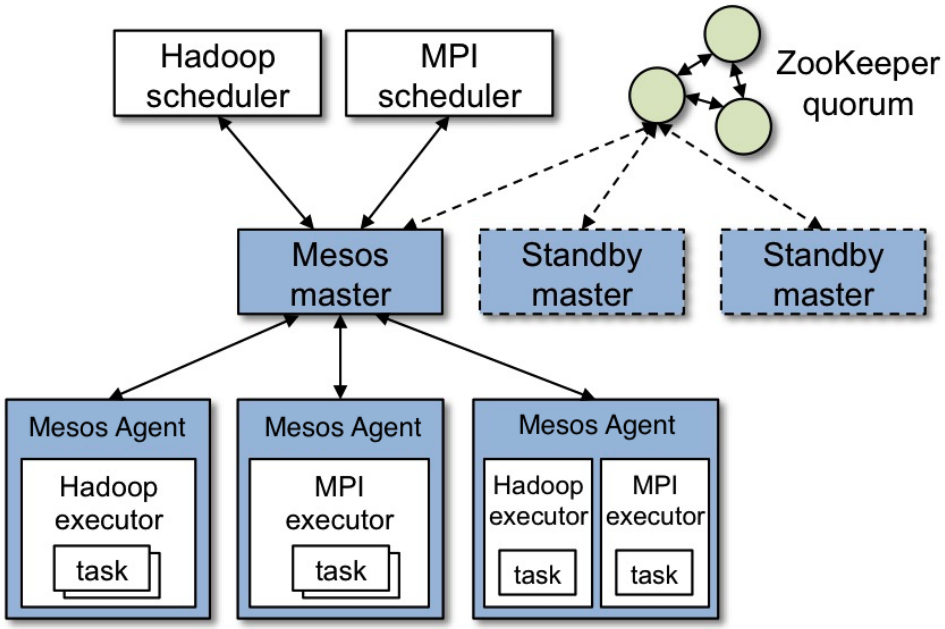
Tanzu -tuotteista on huomioitava, etteivät ne ole käyttövalmiita tuoteratkaisuja vaan tarvitsevat toimiakseen vSphere- tai pilviympäristön sekä NSX-lisenssin. Testi- ja harjoitusympäristössä käytetään myös HAproxya, mutta tuotantoympäristössä se ei luultavasti ole tarkoituksenmukaista edellä mainittujen rajoitteiden vuoksi. Tanzu Basic on saatavilla ainakin vuoden ja kolmen vuoden lisenssillä sekä VMware vSphere with Tanzu tuotepakettina. (VMware, 2022C), (VMware, 2021B)

5.2 Apache Mesos

Apache Mesos on alunperin kolmen Berkleyn yliopiston opiskelijan sekä professorin opintoprojekti, jota esiteltiin vuonna 2009 nimeltä Nexus, mutta seuraavalla kerralla vuonna 2011 se esiteltiin nimellä Mesos. (Bhadwal, 2022) Apache Mesos sai nykyisen nimensä 2016, kun Apache lisäsi version 1 ohjelmistoihinsa nykyisellä nimellään. (Eh-neß, 2021) Mesos on kehitetty klusterin resurssien hallintaan, eikä se itse käsittele tai hallinnoi sovelluskontteja suoraan. (Kasthuri & Nayyar, 2021)

Mesos koostuu kolmesta peruskomponentista:

- Mesos master toimii välittäjänä Mesos agentin ja Mesos frameworkin kesken ja vastaa resurssien ohjaamisesta frameworkeille.
- Mesos agent vastaa fyysisten solmujen resurssien hallinnasta sekä suorittaa frameworkin executoria.
- Mesos framework koostuu kahdesta osasta, jotka ovat scheduler (ajoittaja) sekä executor (suorittaja). Scheduler toimii hallinnoijana ja executor suorittaa schedulerilta saamansa prosessit.



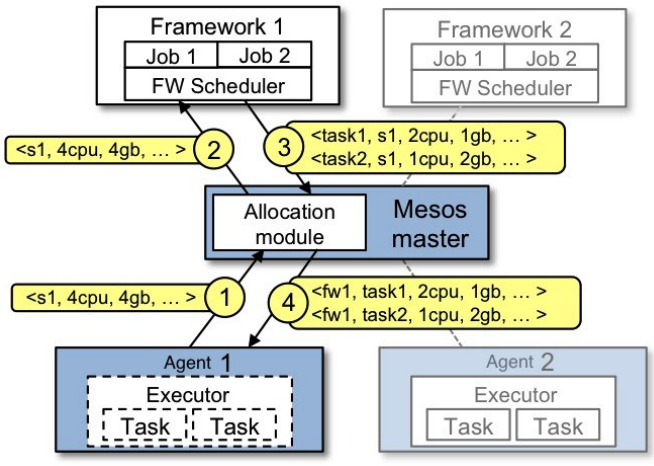
Kuvio 18. https://mesos.apache.org/documentation/latest/architecture/

Eräitä Mesosin frameworkkeja ovat:

- Marathon on sovelluskonttienhallintajärjestelmä, joka tukee Mesosin kontteja sekä Docker- ja AppC -pohjaisia kontteja. (Mesospehere, 2019), (Apache Mesos, n.d.)
- Aurora on pitkäaikaisten palvelujen scheduler.
- Hadoop-ohjelmistokirjasto tarjoaa mahdollisuuden laskennan hajauttamiseen useiden tietokoneklustereiden kesken. Kirjasto tarjoaa korkeaa saavutettavuutta valvomalla ja käsittelemällä virhetilanteita ohjelmistotasolla. (Apache Hadoop, 2022)
- Jenkins on sovelluskehitykseen tarkoitettu avoimen lähdekoodin integrointi- ja käyttöönottotyökalu. (Manning, 2022)

Kuvan 17 esimerkki Mesos masterin resurssin tarjoamisesta:

1. Agentti 1 ilmoittaa Mesos masterille ”noden” käytettävissä olevat resurssit. 4 Cpu, 4gb muistia.
2. Mesos masterin allocation moduuli välittää resurssitiedot framework 1:lle
3. Framework 1:n scheduler vastaa ja tarjoaa kahta tarjottuihin resursseihin sopivaa tehtävää. (task1 2 cpu ja 2 gb -muistia, task2 1 cpu ja 1gb -muistia)
4. Mesos masters lähettää tehtävät Agentti 1:lle ja Agentti 1:n executor käynnistää saadut tehtävät.
5. Mesos Master tarjoaa Agentti 1:n käyttämättä olevia resursseja Framework 2:lle. (1 cpu ja 1 gb muistia)



Kuvio 19. https://mesos.apache.org/documentation/latest/architecture/

Mesos on monien lähteiden ja ominaisuuksiensa puolesta tarkoitettu lähinnä suuriin järjestelmiin. (Hindman ym., 2011, s. 2) Kubernetes pystyy hallitsemaan enintään 5000 noden klustereita, kun Mesos pystyy hallitsemaan yli 10000 noden klustereita. (Bhadwal, 2022). Mesosin kehittäjät ovat testanneet sitä 50000:n simuloidun noden kuormalla onnistuneesti, eikä suurempien kuormien simulointi ollut tuolloin mahdollista Amazonin EC2 -ympäristössä. (Hindman ym., 2011, s. 12)
Mesosia ovat käyttäneet tai käyttävät monet suuret yritykset kuten, Twitter (2010-2019), PayPal ja Uber. (Kasthuri & Nayyar, 2021) Applen avustaja Siri siirrettiin 2015 Mesos-pohjaiselle alustalle. (Sverdlik, 2015)
Netflix käyttää Titus-nimistä sovelluskonttialustaa AWS EC2-pilvessä suoratoisto-, suositus- ja sisältöjärjestelmissään. Titus on Netflixin kehittämä framework Apache Mesosin päälle. (Netflix, n.d.)

Mesosin katsotaan soveltuvan erityisesti niille käyttäjille, joilla on sovelluskonttien lisäksi kontittamattomia työkuormia, sillä Mesos tukee frameworkiensa kautta työkuormien suoritusta kontitettuna sekä kontittamattomana jopa samassa nodessa. (Nolle, 2019)

Kubernetesin valtava suosio on syönyt sovelluskehittäjien kiinnostusta Mesosiin ja sen tulevaisuus näytti keväällä 2021 erittäin epävarmalta, kun Mesos ehdittiin jo käytännössä äänestää siirrettäväksi Apache Attic:in. Apache Attic on paikka projekteille, jotka ovat saavuttaneet virallisesti elinkaarensa lopun (End of life). Parin päivän kuluttua äänestys kuitenkin peruutettiin ja projektia päätettiin jatkaa uuden puheenjohtajan johdolla. (Wallen, 2021)

Tässä osiossa Mesosia käsiteltiin sen perustoiminnan kannalta ja esiteltiin eräänlaisena Kubernetesiin perustumattomana vaihtoehtoisena alustana sovelluskonttien hallintaan. Todellisuudessa asia ei ole niin yksiselitteinen, sillä Mesosille on myös kehitetty kubernetes-mesos framework, jolla sovelluskonttien hallinta voidaan toteuttaa Kubernetesin avulla Mesos-ympäristössä. (Doyle, 2014)
Tämä alun perin Mesospheren ja Google kehittämä ja myöhemmin IBM:n ylläpitämä projekti on lopetettu, mutta esimerkiksi 27.8.2021 on julkaistu vastaava M3S - Apache Mesos Kubernetes Framework. (M3S, n.d.)

5.3 Red Hat OpenShift

OpenShift on Red Hatin kaupallinen tuoteperhe sovelluskonttien hallintaan. Sovelluskonttien hallinta on toteutettu OpenShiftissä käyttäen Kubernetesia, mutta OpenShift tarjoaa lisäksi paljon muita ominaisuuksia, keskitetympää hallintaa sekä maksullisen tuen.

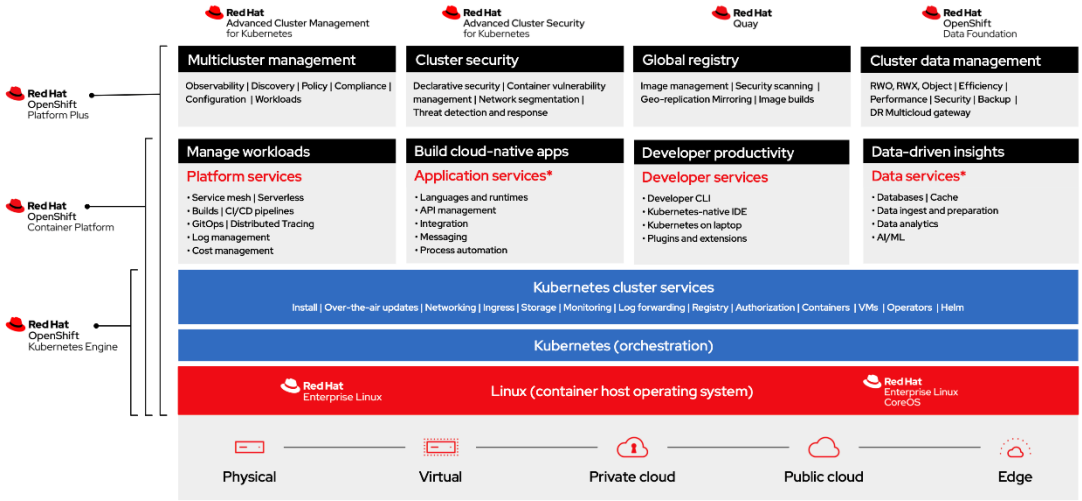
OpenShift tuli alun perin Red Hatin haltuun sen ostaessa Linux -kontteja PaaS -palveluna tarjonneen Makara:n marraskuussa 2010. (Rosenberg, 2010) 8.6.2022
OpenShift-nimellä tuote julkaistiin 2011 suljetun lähdekoodin tuotteena, mutta 2012 toukokuussa OpenShift Origin -projekti julkaistiin Apache v2-lisenssin alla. (Red Hat, 2012) OpenShiftin versiot 1 ja 2 perustuivat vielä Red Hatin omaan container runtimeen sekä container orchestration engineen. OpenShiftin version 3 yhteydessä Red Hat siirtyi Docker-kontteihin sekä käyttämään konttien hallinnassa Kubernetesia. (Fernandes, 2016) OpenShiftin version 4.1 mukana tuli useita rakenteellisia muutoksia, kuten container enginen vaihtaminen Dockerista CRI-O:on, Docker Daemonin korvaaminen Podmanilla sekä luopuminen sovelluskonttikuvien luonnissa Dockerin käytöstä ja siirtyminen Buildahin käyttöön. (Willliam, 2019) Näillä muutoksilla OpenShift vapautui riippuvuudesta Dockeriin ja erityisesti Docker Daemonista, jolla on hieman kyseenalainen maine tietoturvan kannalta. (Chen, 2020)

OpenShiftin tuoteperheessä on kaksi tuoteryhmää Cloud services (pilvipalvelut) sekä Self-managed, jota voisi monesti kutsua On-Premise-ratkaisuksi, mutta nimensä mukaisesti kyseessä on itsehallittava ympäristö, joka ei kuitenkaan rajoitu yrityksen omaan laiteympäristöön. Cloud service -tuoteperheeseen kuuluu Red Hat OpenShift Service on AWS, Microsoft Azure Red Hat OpenShift, Red Hat OpenShift Dedicated sekä Red Hat OpenShift on IBM Cloud. Kyseisten pilvipalveluiden sisältö on pitkälti yhteneväinen, erona osaan kuuluu Red Hat OpenShift API Management ja osaan taas ei kuulu. Lisäksi palvelujen Service level agreement (SLA) on 99,95 prosenttia, poikkeuksena IBM joka ainoana lupaa 99,99 prosenttin palvelutason. Self-managed Red Hat OpenShift -editionin alta löytyy kolme tuotetta: Red Hat OpenShift Kubernetes Engine (OKE), Red Hat OpenShift Container Platform (OCP) sekä Red Hat OpenShift Platform Plus. (Red Hat, n.d. D)

Työssä käsitellään Red Hat OpenShift Kubernetes Engine-tuotetta esimerkkinä itsehallittavasta Kubernetes-ympäristöstä, joka on kuitenkin suunniteltu yhtenäiseksi kokonaisuudeksi valmiiksi määriteltyine asetuksineen. OpenShift Kubernetes Engine ja OpenShift Container Platform ovat täysin sama ohjelmistopaketti, jonka eroavuudet johtuvat käytössä olevasta lisenssistä. Tästä johtuen OpenShift Kubernetes Engineen on lähtökohtaisesti tilattavissa samat ominaisuudet kuin OpenShift Container Platformiin. (Red Hat, 2022A)

Red Hat OpenShift Kubernetes Engine voidaan asentaa AWS-ympäristöön, IBM Cloudiin, Google Cloudiin, Azureen, vSphereen tai suoraan sopivaan laiteympäristöön. (Red Hat, 2022D) Red Hat OpenShift Kubernetes Engine koostuu neljästä tilauksesta, jotka ovat Red Hat OpenShift Kubernetes Engine, Red Hat Enterprise Linux and Red Hat Enterprise Linux CoreOS, Red Hat OpenShift Virtualization ja Red Hat Application Streams. Red Hat OpenShift Kubernetes Engine sisältää Kubernetes -jakelun ja -klusterin komponentit kuten OpenShift asennusohjelman, monitoroinnin, lokien edelleen lähetyksen, SDN-verkkokomponentin, rekisterin sekä sisään tulevan liikenteen reitityksen (ingress router). Red Hat Enterprise Linux and Red Hat Enterprise Linux CoreOS sisältää niemensä mukaiset Linux jakelut, joita tarvitaan Red Hat OpenShiftin käytössä. (Red Hat, 2022C) Red Hat OpenShift Virtualization mahdollistaa sovelluskonttien ja virtuaalikoneiden hallinnan OpenShiftin kautta. Ominaisuus perustuu alun perin Red Hatin kehittämään, mutta nykyisin CNCF:n (Cloud Native Computing Foundation) ylläpitämään KubeVirt-projektiin. (Red Hat, 2022E) Red Hat Application Streams on Red Hatin ylläpitämä ohjelmistokirjasto Red Hat Enteprise Linuxille (Sagat, 2020). Kirjaston ohjelmistoilla on tarkoitus tarjota ennakoitavuutta ja siksi sen ohjelmistoilla on Red Hatin ennalta määrittelemät tukiajat (Red Hat, n.d. B). Red Hat OpenShift Kubernetes Engine ei sisällä OpenShift Container Platformin edistyneitä ominaisuuksia tai kehittäjätyökaluja. (Red Hat, 2022C)

Kuvassa 20 näkyy OpenShift Self-managed-tuoteperheen hierarkia sekä tärkeimmät erot tuotteiden ominaisuuksissa.



Kuvio 20. https://www.redhat.com/en/resources/openshift-kubernetes-engine-datasheet

Kuten edellisestä kuvasta ilmenee, niin useimmat OpenShiftin tunnetuimmista ominaisuuksista eivät sisälly OpenShift Kubernetes Engine -lisenssiin. Seuraavaksi käsittelen OKE:n mielenkiintoisimmiksi arvioimiani ominaisuuksia. Ominaisuuksien valinta on väistämättä subjektiivista kirjoittajan, käyttötarpeen, vertailukohdan ja valintahetken vaikutuksesta. OKE on CNCF:n sertifioima Kubernetes-jakelu ja taakaa sen yhteensopivuuden Kubernetes-ekosysteemiin. (Red Hat, 2021A) OKE sisältää erityisesti tietoturvaan liittyviä parannuksia. Esimerkiksi sovelluskonttien suorittaminen root -käyttäjänä on oletuksena estetty ja sovelluskontit suoritetaan projektille satunnaisesti luodun käyttäjätunnusalueen ensimmäisellä vapaalla ID:llä. Tämä suojaa muiden käyttäjien hallinnoimia prosesseja tilanteessa, jossa sovellus pääsee ajonaikaisen ympäristön haavoittuvuuden vuoksi vuorovaikutukseen hostin kanssa. Tästä johtuen Docker -kuvat eivät ole aina suoraan yhteensopivia OpenShiftin kanssa. Suoranaisesti kyseessä ei ole yhteensopimattomuus vaan best security practices -käytännöistä. Docker ei pakota noudattamaan näitä käytäntöjä toisin kuin OpenShift ja tämä mahdollistaa sellaisten kuvien luomisen Dockerilla, jotka eivät toimi suoraan OpenShiftissä. (Redhat, 2020)

Toinen OpenShiftin oletuksena noudattama best security practices -tietoturvakäytäntö on porttimääritykset. OpenShift estää oletuksena sovelluskonttien prosessien pääsyn etuoikeutettuihin portteihin (Priviliged ports), joita ovat porttinumerot alle 1024:n. Tilanne on sama kuin käyttäjä ID:n kanssa, eli Docker sallii luoda kontin, joka toimii tällä etuoikeutetulla alueella, mutta OpenShift ei päästä sovellusta käsiksi porttiin. Tämä tuottaa esimerkiksi "Permission denied: AH00072: make_sock: could not bind to address [::]:80" ilmoituksen. (Datadog, n.d.) (Number One,2019)

OpenShiftin skaalautuvuutta pidetään nopeampana ja parempana erityisesti klustereiden ja nodien osalta. (Marinelli, 2021)
Automaattinen solmujen skaalaus seuraa Podien solmuvarauksia ja, mikäli solmuja on vähemmän käytössä kuin niiden ylärajaksi on määritelty, skaalaus käynnistää uuden solmun. Skaalain myös valvoo automaattisesti solmujen käyttöä ja poistaa käyttämättömät solmut automaattisesti. (Marinelli, 2021)
OpenShift tukee jopa hybridi-infrastruktuurissa automaattista asennusta, päivittämistä ja elinkaarihallintaa koko OpenShift -ympäristölle. Tätä voidaan pitää OpenShiftin yhtenä tärkeimmistä ominaisuuksista, mutta samalla se luo eräitä tiukkoja rajoituksia OpenShift -ympäristölle. OpenShiftin tarjoama mahdollisuus päivittää master-nodet ja working-nodet muutamalla painalluksella perustuu Red Hatin päätökseen sallia vain Red Hat Enterprise Linux (RHEL) ja Red Hat Enterprise Linux CoreOS (RHCOS) -jakeluiden käytön OpenShift-ympäristössä. (Red Hat, 2021A) RHCOS perustuu Red Hat Enterprise Linuxiin, mutta on huomattavasti kevyempi ja optimoitu sovelluskonttiympäristöön, lisäksi se on tarkoitettu hallittavaksi vain OpenShift-ympäristön kautta. Useimmat RHCOSin -asetukset eivät ole muokattavissa, vaan ne on määritelty valmiiksi. (Red Hat, n.d. C) Uusien klusterien asennus tapahtuu OpenShiftin -asennusohjelmalla, joka mahdollistaa uusien klusterien käyttöönoton parhaimmillaan minuuteissa. (Marinelli, 2021)

5.4 Microsoft Azure

Azure itsessään on Microsoftin pilvipalvelu eivätkä sen tarjoamat palvelut rajoitu sovelluskontteihin, mutta koska Microsoftilla ei ole yhtenäistä nimeä sovelluskonttien käsittelyyn tarjoamilleen palveluille, niin käytämme Azurea ympäristön yläkäsitteenä.

Seuraava taulukko on Microsoftin tuotekartta Azuren kaupallisista palveluista sovelluskonteille.

Find the Azure service for your container needs	
IF YOU WANT TO	**USE THIS**
Deploy and scale containers on managed Kubernetes	Azure Kubernetes Service (AKS)
Deploy and scale containers on managed Red Hat OpenShift	Azure Red Hat OpenShift
Build and deploy modern apps and microservices using serverless containers	Azure Container Apps
Execute event-driven, serverless code with an end-to-end development experience	Azure Functions
Run containerized web apps on Windows and Linux	Web App for Containers
Launch containers with hypervisor isolation	Azure Container Instances
Deploy and operate always-on, scalable, distributed apps	Azure Service Fabric
Build, store, secure, and replicate container images and artifacts	Azure Container Registry

Taulukko 2. https://azure.microsoft.com/en-us/product-categories/containers/

Azure Container Apps (ACA) on Microsoftin uusin PaaS -tuote Azuressa ja sitä käsitellään työssä esimerkkinä helposti lähestyttävästä sovelluskonttien hallinta-alustasta, mutta esimerkiksi Red Hat, Google ja Amazon tarjoavat hyvin samankaltaisia tuotteita. Kontitettujen kuormien aloittaminen palvelussa on parhaimmillaan erittäin yksinkertaista, mutta käyttäjän tarpeet määrittelevät tietysti lopullisen toteutuksen monimutkaisuuden.

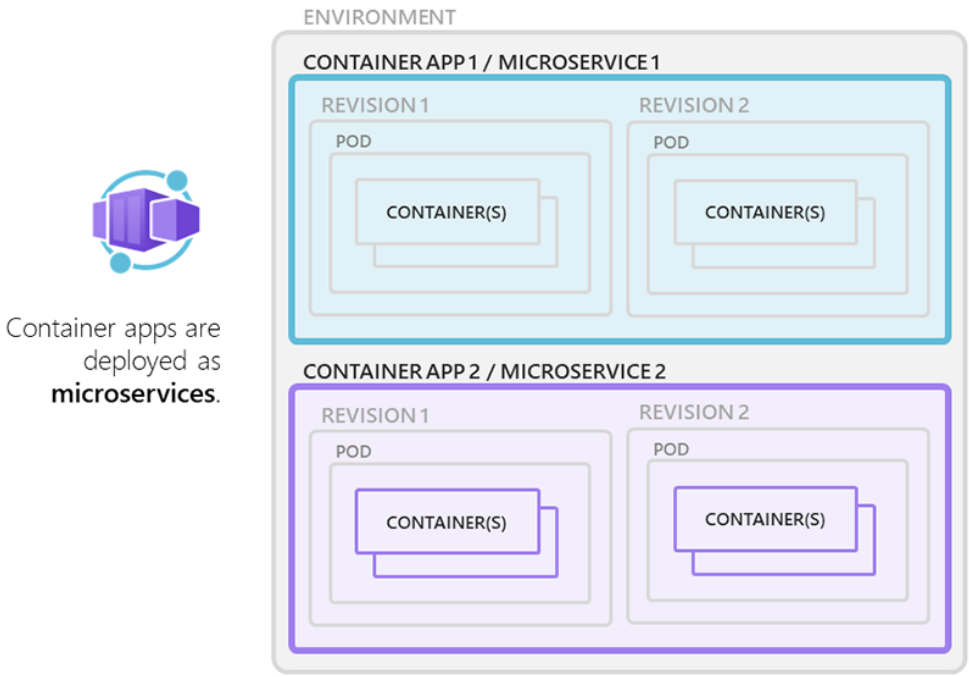
Azure Container Apps on tarkoitettu sovelluskonttien ja mikropalveluiden (microservices) suorittamiseen palvelittomasti (serverless). (Microsoft, 2022B)

Azure Container Apps on tilauspohjainen PaaS -palvelu, jonka hinnoittelu perustuu resurssien käyttöön sekunnissa. Kuukausittaisessa laskutuksessa mitattavia arvoja ovat vCPU-s- ja GiB-s -resurssien käyttö sekä kuukausittaisten pyyntöjen määrä. Kaikissa edellä mainituissa resursseissa on käytössä pieni kuukausittainen ilmaismäärä, joka mahdollistaa maksuttoman tutustumisen palveluun. (Microsoft, 2022)

Azure Container Apps on Kubernetes -pohjainen PaaS -palvelu, jossa asiakkaan ei tarvitse huolehtia laitteistosta tai Kubernetes-ympäristön hallinnoinnista. Azure Container Apps ei toimintafilosofiansa mukaisesti tarjoa pääsyä Kubernetes-sovellusliittymiin tai ohjaustasoon. Kubernetesin klusteritason hallintaa tarvitseville Microsoft tarjoaa vaihtoehdoksi Azure Kubernetes Service -ympäristöä. (Microsoft, 2022F)

Azure Container Apps -palvelua hallitaan Azure CLI -laajennuksella, joka on saatavilla Windowsiin käytettäväksi Command Promptin tai PowerShellin kautta sekä Linux-ympäristöihin BASH-laajennuksena. (Microsoft, 2022E)

Azure Container Apps -palvelun rakenne muodostuu kuvan ix mukaisesti ympäristöstä (Environment), sovelluksesta (Container App), Versiosta (Revision) ja suoritettavista sovelluskonteista.

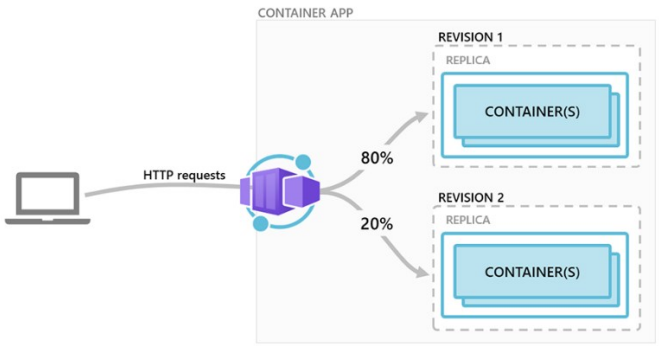


Kuvio 21. https://samcogan.com/wth-are-azure-container-apps/

Enviroment kuvastaa Kubernetesin Namespace -aluetta, jonka sisällä olevat sovellukset pystyvät kommunikoimaan keskenään. Sovellus koostuu versioista ja siitä käynnistetyistä sovelluskonteista. Versio perustuu Kubernetesin PODiin ja sen sisällä suoritettavat sovelluskontit omaavat yhtenäisen elinkaaren, jakavat saman levyn sekä pystyvät kommunikoimaan suoraan toisilleen. Mikropalveluista koostuvassa

sovelluksessa erilaisen elinkaaren omaavat mikropalvelut otetaan käyttöön omina sovelluksinaan (Container App). (Cogan, 2021)

Azure Container Apps tukee sovelluskonttien automaattista versiointia kuten useimmat kilpailijatkin, mutta tarjoaa siihen lisäksi muutamia helppokäyttöisiä sääntöjä. Revision management -sivulla voidaan määritellä uuden version vaikutus sovellukseen eli siihen, poistetaanko vanhat versiot automaattisesti käytöstä ja korvataan uusimmalla vai voiko sovelluksesta olla käytössä eri versioita. Mikäli käyttöön otetaan Multiple revision mode, niin käytettävissä olevat revisionit määritellään Azure -CLI:n tai Azure Portalin kautta. Multiple revision mode on erityisesti sovelluskehityksen tarpeisiin suunniteltu ominaisuus, joka tukee http-pyyntöjen suhteellista jakamista eri versioiden välillä, jolloin voidaan esimerkiksi testata sovelluksen uutta versiota rajoitetun kokoisella, satunnaisesti valitulla käyttäjäjoukolla, ilman että palvelun toiminta vaarantuu mahdollisten ohjelmistovirheiden vuoksi.



Kuvio 22. https://docs.microsoft.com/en-us/azure/container-apps/revisions#revision-scope-changes

Suoranaiseen testaamiseen Multiple revision mode tarjoaa Revision Labels -ominaisuuden, jolla eri versiot voidaan merkitä osoitelapulla (labels). Osoitelappu luo url -osoitteen, jonka kautta halutut käyttäjät voidaan ohjata käyttämään eri versiota. Sama osoitelappu voi olla käytössä vain yhdessä versiossa kerrallaan, mutta on siirrettävissä toiseen versioon. Tällöin testiversion osoite voidaan pitää samana version vaihtuessa. (Microsoft, 2022C)

Azure Container Apps -palvelu tukee vaakasuuntaista skaalausta eli sovelluskonttien monistamista (replikointia). Skaalaus voi perustua http-pyyntöjen määrään, erilaisiin tapahtumiin kuten mihin tahansa KEDA-tapahtumaan tai resurssien käyttöön joko suorittimen tai muistin mukaan. Azure Container Apps tukee skaalausta nollaan replikaan,

jolloin laskutus kyseisen sovelluskontin/revisionin kohdalta pysähtyy. Resursseihin eli cpu:n tai muistin käyttöön perustuva replikointi ei tue skaalausta nollaan. (Microsoft, 2022G)

Azure Container Apps seuraa sovelluskonttien tilaa automaattisesti ja käynnistää kaatuneet kontit uudelleen. Lokitietojen kerääminen on automaattisesti käytössä ja se toteutetaan Microsoftin Log Analyticsin kautta. Erityisesti mikropalveluiden kohdalla automaattisesti yhteen kerätty lokitieto tarjoaa tehokkaan tavan käsitellä tietoja. (Belmansour, 2022)

Azure Container Apps tukee tällä hetkellä vain Linux-pohjaisia sovelluskontteja. Sovellusten käyttöönotto toteutetaan Azure CLI:n tai ARM Templatesin kautta, eikä Kubernetesin kanssa käytettyjä YAML- tai Helm -kaavioita tueta. (Cogan, 2021)

Sovelluksen sovelluskonttien varaamien muisti- ja cpu -resurssien pitää vastata jotakin Microsoftin valmiiksi määrittelemistä malleista. Lyhyesti sanottuna mallissa kerrotaan vCPU:n käyttö kahdella ja saadaan käytettävissä olevan muistin määrä. (Microsoft, 2022A)

6 JOHTOPÄÄTÖKSET

Mahdollisuudet

Sovelluskehityksen tulevaisuutta on mahdoton ennustaa, mutta sovelluskontit ovat selvästi suunta, johon kaikessa ohjelmistokehityksessä ollaan menossa, eikä paluuta vanhaan ole. Uusia haastajia sovelluskonteille voi tulla, mutta todennäköisemmin sovelluskontit ovat tulleet pysyäkseen hyvin pitkään ja tulevat ohjaamaan sovelluskehitystä vähintään seuraavan vuosikymmenen. Sovelluskontit ja niihin erityisen hyvin sopivat mikropalvelut vaikuttavat tarjoavan ratkaisun lähes kaikkiin suuriin haasteisiin, joita ohjelmistokehitys sekä järjestelmäpuoli ovat vuosikymmenien aikana kohdanneet. Ohjelmistokehitykselle sovelluskontit tarjoavat ympäristöllisen yhdenmukaisuuden kehitystyön, testauksen ja tuotannon välillä. Tämä on erityisen tärkeää ketterän kehittämisen menetelmiin luottavassa modernissa sovelluskehityksessä, jossa muutokset ohjelmakoodiin halutaan toteuttaa nopeasti. Yhdenmukainen sovelluskonttiympäristö antaa ohjelmoijan testata palvelinympäristöön tarkoitetun ohjelman toimintaa kannettavalla tietokoneellaan mahdollistaen entistä nopeamman ja ketterämmän ohjelmistokehityksen.

Järjestelmäpuolelle sovelluskontit tarjoavat erityisesti palvelujen siirrettävyyttä ympäristöjen välillä, mikä itsessään mahdollistaa ennennäkemättömät mahdollisuudet palvelujen skaalautuvuudelle. Skaalautuvuus on itsessään sovelluskonttien ominaisuus, mutta, juuri siirrettävyys mahdollistaa työasemassa pyörivän palvelun skaalaamisen yhtä helposti paikalliselle palvelimelle, kuin kansainväliseen pilvipalveluun. Sovelluskontit kuluttavat itsessään erittäin vähän resursseja ja ovat siksi optimaalinen ratkaisu palvelujen skaalaamiseen. Erityisesti sovelluskontit näyttävät kuitenkin tarjoavan ratkaisun ikiaikaiseen ongelmaan resurssien tehokkaasta käytöstä, sillä sovelluskonttien siirrettävyys, skaalautuvuus ja mahdollisuus hallita/rajoittaa sen käyttämiä resursseja mahdollistaa erittäin korkean käyttöasteen palvelimille. Palvelimien korkea käyttöaste laskee investointi- ja ylläpitokuluja vähentämällä tarvittavien palvelimien määrää. Palvelimien määrällä on myös suora vaikutus energian kulutukseen, sillä kolme palvelinta 30 prosentin kuormalla kuluttaa huomattavasti enemmän energiaa kuin yksi 90 prosentin kuormalla toimiva palvelin. (Barraso ym., 2018, s. 107, 112)

Sovelluskontit ja niiden hallintajärjestelmät tarjoavat järjestelmäpuolelle huomattavasti helpompaa sovellusten ylläpitoa ja päivittämistä. Vaikka konttisovellukset vaativat päivityksiä aivan samoin kuin perinteiset sovellukset, niin toteutus on ylläpidon kannalta aivan erilainen. Sovelluskontteja itsessään ei päivitetä, sillä vain sovelluskonttikuva päivitetään ja sen jälkeen sovelluskonttihallintajärjestelmä huolehtii vanhaan kuvaan perustuvien konttien sammutuksesta ja käynnistää ne sitten uudesta päivitetystä kuvasta.

Sovelluskontit muuttavat monessa tapauksessa varmuuskopioinnin täysin, sillä sovelluskonteista ei tarvita varmuuskopioita. Parhaimmillaan varmuuskopioinniksi riittää tietokantojen, sovelluskonttikuvien ja sovelluskonttihallintajärjestelmän ohjauspalvelimen varmuuskopiointi. Varmuuskopion palautuksessakaan ei tarvitse olla vastaavaa ympäristöä, sillä sovelluskontit voidaan käynnistää lähes missä vain.

Haasteet

Edellä mainitut johtopäätökset kertovat sovelluskonttitekniikan ja erityisesti sovelluskontti-filosofian tarjoamista mahdollisuuksista. Sovelluskonttiteknologia tarjoaa nuo mahdollisuudet melko helposti käyttöön uusille organisaatioille ja yrityksille, joilla ei ole ennestään omaa laiteympäristöä tai sovelluksia. Pitkään toimineet yritykset, joilla on omat laiteympäristöt ja sovellukset, joutuvat suunnittelemaan sovelluskontteihin siirtymisen melko huolellisesti, sillä perinteiset sovellukset ja ympäristöt ovat vaikeasti siirrettävissä. Sovellusten täytyy olla kehitetty sovelluskontti käyttöön, jotta niitä voidaan ajaa sovelluskonteissa, mutta lisäksi on pohdittava oman ympäristön soveltuvuutta sovelluskonttien käyttöön. Perinteiseen sisä-, ulkoverkko -ajatteluun perustuva toimintaympäristö ei luultavasti taivu suoraan sovelluskonttien käyttöön. Monesti neuvona tuntuukin olevan, että sovelluskontti-ratkaisut kannattaisi aloittaa alusta, jotta saataisiin käyttöön kaikki sovelluskonttien mahdollisuudet, ilman menneisyyden taakkaa. Sovelluskontituksen tekniikka perustuu pitkälti Linuxiin, minkä takia tuki Windows sovellusten suorittamiseen konteissa vaihtelee, eikä kaikki sovelluskonttien ominaisuudet ole silloin välttämättä käytettävissä. OCI-standardit tarjoavat mahdollisuuden toteuttaa ympäristöstä riippumattomat helposti siirrettävissä olevat sovelluskontit. Kuitenkin esimerkiksi Docker sallii luoda sovelluskonttikuvia, joiden määritykset eivät ole porttien ja käyttöoikeuksien osalta standardin mukaisia. Vaikka sovelluskontit itsessään ovat siirrettävissä ympäristöstä toiseen helposti, eivät palvelut

kokonaisuudessa siirry aina kivuttomasti yhden pilvitoimijan ympäristöstä toisen toimijan pilviympäristöön. Tähän on kuitenkin saatavilla ratkaisuja kuten työssä käsitelty OpenShift, joka tukee useimpia suuria pilvipalveluita ja mahdollistaa siirtymisen niiden välillä.

Sovelluskonttienhallintajärjestelmän valintaa on vaikea tehdä. Lähes kaikki hallintajärjestelmät Mesosta lukuun ottamatta perustuvat avoimenlähdekoodin Kubernetekseen. Kuberneteksen osuus hallintajärjestelmistä oli vuonna 2019 77 prosenttia ja on todennäköisesti siitä kasvanut, kun esimerkiksi OpenShift siirtyi käyttämään Kubernetesta. Ilmainen avoimenlähdekoodin Kubernetes ei kuitenkaan ole sellaisenaan erityisen käytetty vaan 90 prosenttia Kubernetesta käyttävistä organisaatioista käyttää sitä palveluna (Datadog, 2021). Palveluna tarjottavien Kubernetes-ratkaisujen kesken on huomattavia eroja hinnoittelun, ominaisuuksien ja hallinnan osalta. Eräät tuotteet perustuvat helppokäyttöisyyteen eivätkä edes tarjoa pääsyä Kuberneteksen omiin rajapintoihin. Hallintajärjestelmää tai sen tarjoajaa valitessa yrityksen täytyisikin tietää tarkasti nykyiset ja tulevat tarpeensa. Kaikilla pilvipalvelujen tarjoajilla on ratkaisuja sovelluskonttien hallintaan ja valinta niiden välillä riippuu omista tarpeista. Niiden lisäksi erityisen kiinnostavia tuotteita ovat työssä käsitellyt OpenShift ja Tanzu. Ne eivät välttämättä tarjoa aivan samaa helppokäyttöisyyttä kuin pilvipalvelut, mutta ne sisältävät huomattavasti ominaisuuksia ja tukevat on-premisen lisäksi useita pilvipalveluita mahdollistaen erilaiset hybridi ratkaisut on-premisen ja eri pilvipalveluiden välillä.

Sovelluskonttien hallinnassa Kubernetes on ylivoimainen markkinajohtaja ja kiinnostus siihen on vielä vahvempaa kuin sen tämänhetkinen markkina-asema. Kubernetesille ei ole näkyvissä haastajia ja se tulee säilyttämään luultavasti pitkään asemansa hallintajärjestelmien standardina. Markkinoilla ei esiinny Apache Mesosiin sellaista kiinnostusta tai odotuksia kuin esimerkiksi Dockeriin tai Kubernetesiin. Tämä kiinnostuksen puute vaikuttaa voimakkaasti Mesosin asemaan ja näkyvyyteen markkinoilla, vaikka Mesosilla on yhä erittäin suuria käyttäjiä, joiden järjestelmät ovat pohjautuneet siihen parhaimmillaan jo yli vuosikymmenen.

LÄHTEET

Apache Hadoop. (2022). Apache Hadoop. Haettu 20.05.2022 osoitteesta https://hadoop.apache.org/

Apache Mesos. (n.d.). Apache Mesos. Haettu 22.05.2022 osoitteesta https://mesos.apache.org/

Baker, E. (10.06.2020). A Comprehensive Container Runtime Comparison. https://www.capitalone.com/tech/cloud/container-runtime/

Barroso, L. A., Hölzle, U., & Ranganathan, P. (2018). The datacenter as a computer: Designing warehouse-scale machines. Synthesis Lectures on Computer Architecture.

Basyildiz, B. (19.08.2019). A Brief History of Container Technology. https://www.section.io/engineering-education/history-of-container-technology/

Belmansour, T. (08.02.2022). Azure Container Apps – an overview. https://www.serverless360.com/blog/azure-container-apps-an-overview

Benik, A. (30.11.2013). The sorry state of server utilization and the impending post-hypervisor era. https://gigaom.com/2013/11/30/the-sorry-state-of-server-utilization-and-the-impending-post-hypervisor-era/

Bhadwal, P. (10.06.2022). Mesos vs Kubernetes - Which One to Choose. https://www.techgeekbuzz.com/blog/mesos-vs-kubernetes/

Brown, N. (08.06.2018). Explaining Docker Image Ids. https://windsock.io/explaining-docker-image-ids/

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade.

Busser, A. (12.02.2021). From Docker to OCI. https://www.padok.fr/en/blog/container-docker-oci

Butler, T. (28.09.2015). What Is a Dockerfile. https://www.cloudbees.com/blog/what-is-a-dockerfile

Chen, J. (29.01.2020). Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed. https://unit42.paloaltonetworks.com/attackers-tactics-and-techniques-in-unsecured-docker-daemons-revealed/

Codefresh. (30.06.2019). Building Docker Images with Dockerfiles. https://codefresh.io/blog/build-docker-image-dockerfiles/

Cogan, S. (06.11.2021). WTH are Azure Container Apps. https://samcogan.com/wth-are-azure-container-apps/

Containerlabs. (n.d.). The cgroup freezer subsystem. Haettu 18.05.2022 osoitteesta https://www.containerlabs.kubedaily.com/LXC/Linux%20Containers/The-cgroup-freezer-subsystem.html

Crippa, F.S. (16.02.22021). Understanding Container Images, Part 3: Working with Overlays. https://blogs.cisco.com/developer/373-containerimages-03

Datadog (n.d.). Privileged ports are not mapped within containers. Haettu 28.06.2022 osoitteesta https://docs.datadoghq.com/security_platform/default_rules/cis-docker-1.2.0-5.7/

Datadog. (10.2021). Container report. https://www.datadoghq.com/container-report/

Docker Inc. (2022). Build and Ship any Application Anywhere. Haettu 12.07.2022 osoitteesta https://hub.docker.com/

Docker Inc. (n.d. A). Runtime options with Memory, CPUs, and GPUs. Haettu 15.05.2022 osoitteesta https://docs.docker.com/config/containers/resource_constraints/

Docker Inc. (n.d. B). Best practices for writing Dockerfiles. Haettu 16.05.2022 osoitteesta https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Docker Inc. (n.d. C). Docker Hub Quickstart. Haettu 14.05.2022 osoitteesta https://docs.docker.com/docker-hub/

Docker Inc. (n.d. D). Deploy a registry server. Haettu 14.05.2022 osoitteesta https://docs.docker.com/registry/deploying/

Docker Inc. (n.d. E). About storage drivers. Haettu 16.05.2022 osoitteesta https://docs.docker.com/storage/storagedriver/

Docker Inc. (n.d. F). Introducing execution drivers and libcontainer. Haettu 11.05.2022 osoitteesta https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer/

Dong, W. & Xin, G (22.06.2022). antrea-io/antrea. https://github.com/antrea-io/antrea#overview

Doyle, C. (12.12.2014). Kubernetes on Mesos. https://d2iq.com/blog/kubernetes-on-mesos

Ehneß, J. (07.10.2021). Was ist Apache Mesos. https://www.storage-insider.de/was-ist-apache-mesos-a-1048815/

Fernandes, J. (07.11.2016). Why Red Hat Chose Kubernetes for OpenShift. https://cloud.redhat.com/blog/red-hat-chose-kubernetes-openshift

Gillis, A.S. (05.2021). Docker image. https://www.techtarget.com/searchitoperations/definition/Docker-image

Hámori, F. (31.05.2022). The History of Kubernetes on a Timeline. https://blog.risingstack.com/the-history-of-kubernetes/

Harbor. (13.05.2020A). Harbor the first OCI-compliant open source registry. https://goharbor.io/blog/harbor-2.0/

Harbor. (13.05.2020B). Vulnerability Scanning. https://goharbor.io/docs/2.0.0/administration/vulnerability-scanning/

Heslin, K. (21.11.2014). 2014 Data Center Industry Survey. https://journal.uptimeinstitute.com/2014-data-center-industry-survey/

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., ... & Stoica, I. (2011). Mesos: A Platform for {Fine-Grained} Resource Sharing in the Data Center. In 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11).

IBM. (23.06.2021). Containers. https://www.ibm.com/cloud/learn/containers

IBM. (23.06.2021). Containerization. https://www.ibm.com/cloud/learn/containerization

Kasthuri, M. & Nayyar, A. (03.07.2021). Container Orchestration Using Kubernetes and Apache Mesos. https://www.opensourceforu.com/2021/06/container-orchestration-using-kubernetes-and-apache-mesos/

Kirov, M. (18.11.2021). Digging into Linux namespaces - part 2. https://blog.quarkslab.com/digging-into-linux-namespaces-part-2.html

Kisler, E. (04.05.2021). A Beginner’s Guide to Understanding and Building Docker Images. https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

Kubernetes. (04.05.2022E). kube-apiserver. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/

Kubernetes. (08.06.2022G). kube-proxy. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/

Kubernetes. (10.05.2022D). Kubernetes Scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

Kubernetes. (14.06.2021). Controllers. https://kubernetes.io/docs/concepts/architecture/controller/
Kubernetes. (14.07.2022F). kubelet. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/
Kubernetes. (18.06.2022A). Nodes. https://kubernetes.io/docs/concepts/architecture/nodes/
Kubernetes. (30.04.2022B). Kubernetes Components. https://kubernetes.io/docs/concepts/overview/components/
Kubernetes. (30.04.2022C). What is Kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
Kubernetes. (n.d. A). Production-Grade Container Orchestration. Haettu 04.05.2022 osoitteesta https://kubernetes.io/
Larabel, M. (14.01.2020). The Time Namespace Appears To Finally Be On-Deck For The Mainline Linux Kernel. https://www.phoronix.com/news/Linux-Time-Namespace-Coming
M3S. (n.d.). M3S - Apache Mesos Kubernetes Framework. Haettu 20.06.2022 osoitteesta https://aventer-ug.github.io/mesos-m3s/index.html
Manning. (2022.). List of Mesos frameworks and tools. https://livebook.manning.com/book/mesos-in-action/appendix-b/1
Marinelli, C. (29.11.2021). What is OpenShift. https://www.dynatrace.com/news/blog/what-is-openshift-2/
Marinelli, C. (29.11.2021). What is OpenShift? And how to make OpenShift monitoring easy. https://www.dynatrace.com/news/blog/what-is-openshift-2/
Mesosphere Inc. (11.09.2019). Marathon. http://mesosphere.github.io/marathon/
Microsoft. (03.07.2022A). Containers in Azure Container Apps. https://docs.microsoft.com/en-us/azure/container-apps/containers
Microsoft. (03.09.2013). Microsoft to acquire Nokia’s devices & services business. https://news.microsoft.com/2013/09/03/microsoft-to-acquire-nokias-devices-services-business-license-nokias-patents-and-mapping-services/
Microsoft. (15.06.2022G). Set scaling rules in Azure Container Apps. https://docs.microsoft.com/fi-fi/azure/container-apps/scale-app
Microsoft. (15.07.2022F). Comparing Container Apps with other Azure container options. https://docs.microsoft.com/fi-fi/azure/container-apps/compare-options

Microsoft. (2022). Azure Container Apps pricing. https://azure.microsoft.com/en-gb/pricing/details/container-apps/?ocid=AID3042118
Microsoft. (23.07.2022D). About registries, repositories, and artifacts. https://docs.microsoft.com/en-us/azure/container-registry/container-registry-concepts
Microsoft. (24.05.2022C). Revisions in Azure Container Apps. https://docs.microsoft.com/en-us/azure/container-apps/revisions#revision-scope-changes
Microsoft. (28.06.2022B). Azure Container Apps overview. https://docs.microsoft.com/en-us/azure/container-apps/overview
Microsoft. (30.05.2022E). Install Azure CLI on Windows. https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli
Netflix. (n.d.). Titus. Haettu 11.05.2022 osoitteesta https://netflix.github.io/titus/overview/
Newswanger, M. (10.06.2020). Digests in Docker. https://www.mikenewswanger.com/posts/2020/docker-image-digests/
Nolle, T. (07.02.2019). Evaluate the Mesos architecture for massive container deployments. https://www.techtarget.com/searchitoperations/tip/Evaluate-the-Mesos-architecture-for-massive-container-deployments
Number One. (03.10.2019). Openshift will not run your container as a root user. https://number1.co.za/openshift-will-not-run-your-container-as-a-root-user/
Opencontainers. (2020). About the Open Container Initiative. Haettu 26.05.2022 osoitteesta https://opencontainers.org/about/overview/
Osnat, R. (10.01.2020). A Brief History of Containers: From the 1970s Till Now. https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016
Ovens, S. (05.01.2022). Building a container by hand using namespaces: The UTS namespace. https://www.redhat.com/sysadmin/uts-namespace
Ovens, S. (08.03.2021). Building a Linux container by hand using namespaces. https://www.redhat.com/sysadmin/building-container-namespaces
Ovens, S. (29.09.2020). A Linux sysadmin's introduction to cgroups. https://www.redhat.com/sysadmin/cgroups-part-one
Palmer, M. (n.d.). How Does Kubernetes Use etcd?. Haettu 20.06.2022 osoitteesta https://matthewpalmer.net/kubernetes-app-developer/articles/how-does-kubernetes-use-etcd.html
Portnoy, M. (2016). Virtualization essentials (Second edition). Sybex.

Rathnayaka, K. (19.02.2018). Docker Namespace and Cgroups. https://medium.com/@kasunmaduraeng/docker-namespace-and-cgroups-dece27c209c7
Red Hat. (10.05.2022E). Virtualization with Red Hat OpenShift. https://www.redhat.com/en/technologies/cloud-computing/openshift/virtualization
Red Hat. (11.01.2021). The 7 most used Linux namespaces. https://www.redhat.com/sysadmin/7-linux-namespaces
Red Hat. (11.05.2022F). What's a Linux container. https://www.redhat.com/en/topics/containers/whats-a-linux-container
Red Hat. (15.06.2022). About OpenShift Kubernetes Engine. https://github.com/openshift/openshift-docs/commits/enterprise-4.7/welcome/oke_about.adoc
Red Hat. (15.06.2022D). About OpenShift Kubernetes Engine. https://docs.openshift.com/container-platform/4.7/welcome/oke_about.html
Red Hat. (23.04.2021). Red Hat OpenShift Kubernetes Engine. https://www.redhat.com/en/resources/openshift-kubernetes-engine-datasheet
Red Hat. (23.06.2022C). Self-managed Red Hat OpenShift sizing and subscription guide. https://www.redhat.com/en/resources/self-managed-openshift-sizing-subscription-guide
Red Hat. (30.04.2012). Announcing OpenShift Origin. https://www.redhat.com/en/blog/Announcing-OpenShift-Origin-Open-Source-Code-For-Platform-as-a-Service?source=author&term=2661
Red Hat. (30.08.2019). What is SELinux. https://www.redhat.com/en/topics/linux/what-is-selinux
Red Hat. (n.d. A). Chapter 1. Introduction to Control Groups (Cgroups). Haettu 21.05.2022 osoitteesta https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01
Red Hat. (n.d. B). Red Hat Enterprise Linux Application Streams Life Cycle. Haettu 22.05.2022 osoitteesta https://access.redhat.com/support/policy/updates/rhel-app-streams-life-cycle
Red Hat. (n.d. C). Red Hat Enterprise Linux CoreOS. Haettu 20.05.2022 osoitteesta https://docs.openshift.com/container-platform/4.10/architecture/architecture-rhcos.html
Red Hat. (n.d. D). Red Hat OpenShift editions and pricing. Haettu 27.06.2022 osoitteesta https://www.redhat.com/en/technologies/cloud-computing/openshift/pricing

Redhat. (16.04.2020). Why do my applications run as a random user ID. https://cookbook.openshift.org/users-and-role-based-access-control/why-do-my-applications-run-as-a-random-user-id.html
Riel, J. (08.11.2016). linux_memory_control_to_avoid_swap_thrashing. https://gist.github.com/JPvRiel/bcc5b20aac0c9cce6eefa6b88c125e03
Rosenberg, D. (30.11.2010). Red Hat acquires Makara for cloud platform. https://www.cnet.com/tech/services-and-software/red-hat-acquires-makara-for-cloud-platform/
Rosland, J. (13.04.2020). Managing Users. https://goharbor.io/docs/2.0.0/administration/managing-users/
Sagat, P. (17.06.2020). Introduction to Application Streams in Red Hat Enterprise Linux. https://www.redhat.com/en/blog/introduction-appstreams-and-modules-red-hat-enterprise-linux
Smith, T. (19.11.2021). Where It Started: The Rich History of Kubernetes. https://www.appvia.io/blog/history-of-kubernetes
Strotmann, J. (13.12.2016). A Brief History of Containerization. https://www.plesk.com/blog/business-industry/infographic-brief-history-linux-containerization/
Suse. (n.d.). Dockerfile. Haettu 28.06.2022 osoitteesta https://www.suse.com/suse-defines/definition/dockerfile/
Sverdlik, Y. (24.04.2015). Mesos Powers Data Center Backend for Apple's Siri. https://www.datacenterknowledge.com/archives/2015/04/24/cluster-management-software-mesos-powers-apples-siri
Velayudhan, N. (01.090.2021). What are container runtimes. https://opensource.com/article/21/9/container-runtimes
VMware, (15.05.2019A). VMware to Acquire Bitnami. https://blogs.VMware.com/cloud/2019/05/15/VMware-to-acquire-bitnami/
VMware, (24.06.2021A). VMware vSAN Powered HCI Systems Leads the Market in Q1, 2021, According to IDC. https://blogs.VMware.com/virtualblocks/2021/06/24/vsan-hci-leads-q1-2021/
VMware, (n.d.). Providers. Haettu 22.5.2022 osoitteesta https://cloud.VMware.com/providers/search-result

VMware. (02.05.2022B). Tanzu Kubernetes Cluster Networking. https://docs.VMware.com/en/VMware-vSphere/7.0/VMware-vsphere-with-tanzu/GUID-A7756D67-0B95-447D-A645-E2A384BF8135.html
VMware. (2021B). VMware Tanzu Basic. Haettu 12.07.2022 osoitteesta https://docs.VMware.com/en/VMware-vSphere/7.0/VMware-vsphere-with-tanzu/GUID-A7756D67-0B95-447D-A645-E2A384BF8135.html
VMware. (2022C). VMware Store. Haettu 18.057.2022 osoitteesta https://store-us.VMware.com/VMware-tanzu-basic-per-cpu-5471172000.html
VMware. (21.06.2022). VMware Harbor Registry. https://docs.VMware.com/en/VMware-Harbor-Registry/services/VMware-harbor-registry/GUID-index.html
VMware. (26.05.2022). Broadcom to Acquire VMware. https://www.VMware.com/content/dam/digitalmarketing/VMware/en/pdf/company/VMware-broadcom.pdf
VMware. (26.08.2019). VMware Announces VMware Tanzu Portfolio. https://www.globenewswire.com/news-release/2019/08/26/1906432/0/en/VMware-Announces-VMware-Tanzu-Portfolio-to-Transform-the-Way-Enterprises-Build-Run-and-Manage-Software-on-Kubernetes.html
VMware. (30.12.2019). VMware Completes Acquisition of Pivotal. https://news.VMware.com/releases/VMware-completes-acquisition-of-pivotal
VMware. (n.d.). Transform Your Apps and Cloud Faster with VMware Cloud. Haettu 28.06.2022 osoitteesta https://www.VMware.com/cloud-solutions.html
Wallen, J. (13.04.2021). Apache Mesos Narrowly Avoids a Move to the Attic (for Now). https://thenewstack.io/apache-mesos-narrowly-avoids-a-move-to-the-attic-for-now/
William, H. (21.02.2019). Podman and Buildah for Docker users. https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users#how_does_docker_work_
Wong, S. (27.06.2019). Private vs. Public Container Registries. https://thenewstack.io/how-a-container-registry-can-both-save-and-harm/
Wong, W. (11.08.2020). Nutanix’s Hyperconverged Infrastructure Comes to AWS. https://www.datacenterknowledge.com/hyper-convergence/nutanix-s-hyperconverged-infrastructure-comes-aws

Zhang, H. (21.06.2017). What is a Container Registry.
https://blogs.VMware.com/cloudnative/2017/06/21/what-is-a-container-registry/