jamk.fi

Juniper cSRX konttipalomuuri kont-tiklusterissa

Santtu Turunen

Opinnäytetyö
Toukokuu 2019
Tekniikan ja liikenteen ala
Insinööri (AMK), Tieto- ja viestintätekniikan koulutusohjelma
Tietoverkkotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Turunen Santtu	Opinnäytetyö, AMK	Toukokuu 2019
	Sivumäärä 75	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: Kyllä

Työn nimi

Juniper cSRX konttipalomuuri konttiklusterissa

Tutkinto-ohjelma

Tieto- ja viestintätekniikan tutkinto-ohjelma, tietoverkkotekniikka (AMK)

Työn ohjaaja(t)

Marko Rintamäki, Juha Saarisilta

Toimeksiantaja(t)

Kela (Kansaneläkelaitos)

Tiivistelmä

Opinnäytetyön toimeksiantajan toimi Kansaneläkelaitoksen ICT-palvelujen tulosyksikkö. Opinnäytetyön tavoitteena oli tutkia ja testata Juniper cSRX konttipalomuurin toimintaa konttiklusterin sisällä. Alkuperäisen suunnitelman mukaan testiympäristönä toimi Openshift Container Platform, mutta ympäristössä törmätyt ongelmat pakottivat siirtymään toiseen ympäristöön. Toimeksiantajan päätöksellä siirryttiin toiseen konttiklusteriympäristöön Docker Swarmiin.

Opinnäytetyönteoria vaiheessa pyrittiin käyttämään paljon virallisia lähteitä, kuten IEEE:n (Institute of Electrical and Electronics Engineers) sähköistä kirjastoa ja keskityttiin tarkemmin virtualisointiin, Dockerin ominaisuuksiin ja sen haavoittuvuuksiin, sekä Juniperin konttipalomuuriratkaisun tuomiin ominaisuuksiin ja niiden käyttämiseen.

Opinnäytetyön toteutusvaiheessa todennettiin ensiksi valmistajan antaman levykuvan toiminta, jonka jälkeen konttipalomuuria yritettiin saada toimintaan Openshift Container Platform ympäristössä. Useiden ongelmien vuoksi siirryttiin Docker Swarm ympäristöön, jossa törmättiin myös useisiin ongelmiin. Konttipalomuuri saatiin toimimaan Docker Swarmin kanssa, mutta ei klusterin sisällä, kuten toimeksiantaja oli tavoitteeksi määritellyt.

Lopputulokseksi ei saatu klusterin sisällä toimivaa konttipalomuuriratkaisua, mutta valmistaja on työstä saatujen tietojen perusteella alkanut muokkaamaan tuotettaan, jotta tulevaisuudessa konttipalomuuri toimisi klusterin sisällä.

Avainsanat (asiasanat)

Docker, konttiteknologiat, konttipalomuuraus, cSRX, swarm

Muut tiedot (salassa pidettävät liitteet)

-

jamk.fi

Description

Author(s)	Type of publication	Date
Turunen Santtu	Bachelor's thesis	May 2019
		Language of publication: Finnish
	Number of pages 75	Permission for web publication: Yes

Title of publication
Juniper cSRX container firewall inside container cluster

Degree programme
Information and Communications Technology

Supervisor(s)
Marko Rintamäki, Juha Saarisilta

Assigned by
Kela (The Social Insurance Institution)

Abstract

The bachelor's thesis was assigned by the ICT department of Social Insurance Institution of Finland. The purpose of the thesis was to study and test features and operation of the Juniper cSRX container firewall product inside container cluster. According to the original plan, the Openshift Container Platform was meant to act as sole test environment, but the problems encountered in the platform forced to use alternate container cluster environment. Decision of the employer was to move to Docker Swarm container cluster.

The aim of the thesis's theory phase was to use a lot of official sources, such as the IEEE (Institute of Electrical and Electronics Engineers) electronic library. Theory phase was focused more on virtualization, Docker's features and vulnerabilities, as well as the features and use of the Juniper container firewall solution.

The implementation phase started with verifying the operation of Docker image provided by manufacturer, after which main focus of the implementation phase was to get container firewall to work inside Openshift Container Platform. Due to several problems with Openshift Container Platform, it was decided to move to Docker Swarm environment, where several problems were also encountered.

Result of thesis was a not working container firewall solution inside container cluster, but based on the information from the thesis, the manufacturer has started to modify the product so that in future the product will work inside the cluster.

Keywords/tags (subjects)
Docker, container technology, container firewall, cSRX

Miscellaneous (Confidential information)
-

Sisältö

Kuviot

Taulukot

Lyhenteet

AET	Advanced Evasion Technique
ARP	Address Resolution Protocol
CLI	Command Line Interface
CVE	Common Vulnerabilities and Exposures
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDP	Intrusion Detection and Prevention
IEEE	Institute of Electrical and Electronics Engineers
IKE	Internet Key Exchange
IP	Internet Protocol
IPAM	IP Address Management
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
Kela	Kansaneläkelaitos
MAC	Media Access Control
MTU	Maximum Transmission Unit
NDP	Neighbor Discovery Protocol
NSD	Network Security Daemon
PaaS	Platform as a Service
REST API	Representational State Transfer Application Programmable Interface
RHEL	Red Hat Enterprise Linux
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UCP	Universal Control Plane
UDP	User Datagram Protocol
UnionFS	Union File System
UserFW	User Firewall
URL	Uniform Resource Identifier

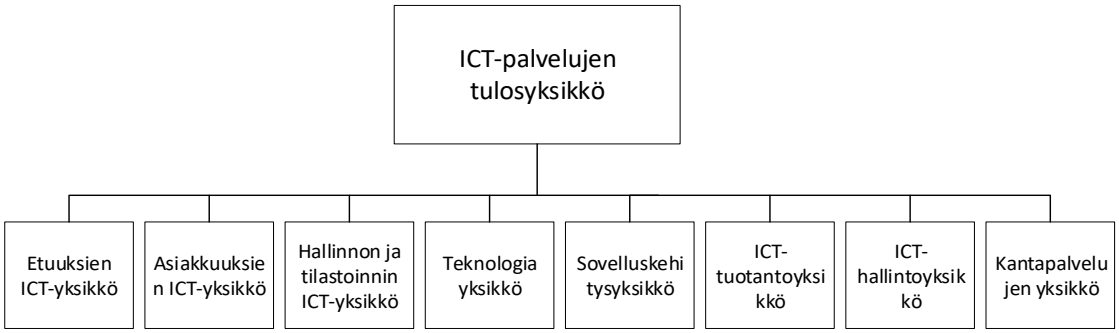
VNF	Virtualized Network Function
VXLAN	Virtual Extensible LAN
YAML	YAML Ain’t Markup Language

1 Lähtökohdat

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja toimi Kela (Kansaneläkelaitos). Kela on itsenäinen sosiaaliturvalaitos, joka toimii eduskunnan valvonnan alla. Kela on linjaorganisaatio, joka on organisoitu neljään tulosyksikköön: etuuspalvelut, asiakkuuspalvelut, ICT-ja kehittämispalvelut ja yhteiset palvelut. Kelan henkilöstön määrä 31.12.2017 oli yhteensä 7226 henkilöä. Suurin osa Kelan henkilöstöstä kuuluu etuuspalveluihin. Etuuspalvelut työllistää reilut 4700 henkilöä. Palvelupisteitä Kelassa on ympäri Suomea yhteensä 161 kappaletta, joista 139 on asiointipisteitä. (Organisaatio. 2018)

ICT-ja kehittämispalveluiden tulosyksikkö vastaa ICT-palvelujen tuottamisesta ja toimittamisesta Kelan sidosryhmien, henkilöstön ja asiakkaiden käyttöön. ICT-palvelujen tulosyksikössä työskentelee lähes 600 toimihenkilöä ympäri Suomea. ICT-tulosyksikkö koostuu 8 yksiköstä, jonka jokaisen alla on useampia ryhmiä. Kuviossa 1 on havainnollistettu ICT-palvelujen tulosyksikön organisaatiokaavio. (Organisaatio n.d.)



Kuvio 1. ICT-palvelujen tulosyksikkö

1.2 Tavoitteet

Kansaneläkelaitoksen ICT-palvelujen tulosyksikössä on testikäytössä Openshift Container Platform- ja Docker Swarm konttiympäristöt. Toimeksiantaja halusi tietää, minkä lisäarvon konttipalomuuraus antaa jo olemassa olevien suojausjärjestelmien

lisäksi. Päätavoitteena oli saada konttipalomuuri testiympäristöissä toimivaksi klusterin sisällä. Tavoitteena oli myös tutkia Dockerin verkkoliikennettä ja saada siitä mahdollisimman paljon lokitietoja konttipalomuurauksen avulla. Konttipalomuuriin tarvittava kontin levykuva saatiin testikäyttöön Juniperilta. Konttipalomuurin käyttöönotto edellytti kuitenkin kattavaa perehtymistä Docker-konttiteknologiaan, sekä Openshift-konttiajoympäristöön, jossa alkuperäisten suunnitelmien mukaan konttipalomuuria haluttiin testattavan. Opinnäytetyön toteutusvaiheessa jouduttiin kuitenkin vaihtamaan testausympäristöä, koska konttipalomuurin käyttöönotossa törmättiin yllättäviin ongelmiin, eikä tuote myöskään valmistajan mukaan tue Openshift-ympäristöä. Ongelmien takia toimeksiantajan päätöksellä siirryttiin Docker Swarm-ympäristöön testaamaan konttipalomuurituotetta, jolloin tavoitteeksi päätettiin saada konttipalomuuri toimivaksi Docker Swarm klusterin sisällä.

Opinnäytetyön teoriavaiheessa pyrittiin käyttämään paljon virallisia lähteitä kuten IEEE:n (Institute of Electrical and Electronics Engineers) sähköistä kirjastoa ja keskityttiin tarkemmin virtualisointiin, Dockerin ominaisuuksiin ja sen haavoittuvuuksiin sekä Juniperin konttipalomuuriratkaisun tuomiin ominaisuuksiin ja niiden käyttämiseen. Opinnäytetyössä pyrittiin käsittelemään konttipalomuurauksen valmiutta tuotantoon ja toteutusvaiheen pääkohtana oli Openshift konttiajoympäristössä sekä Docker Swarmilla ajettava Juniper cSRX-konttipalomuuri. Opinnäytetyön lopussa pohditaan konttipalomuurin toimivuutta ja soveltuvuutta tuotantokäyttöön.

1.3 Tutkimusmenetelmät

Opinnäytetyö on rakenteeltaan soveltava tutkimus, jonka tavoitteena on käytännön sovellutukset. Soveltavassa tutkimuksessa pyritään hyödyntämään tunnettuja faktoja aiheesta ja tekemään uusia ratkaisuja, joista on toimeksiantajalle teknistä hyötyä. Tutkimuksessa sovelletaan teoriaosuudessa saatua tietoa käytännön osuuteen. Tutkimuksen lähdekirjallisuutena on pyritty käyttämään virallisia lähteitä, joita on käytetty useiden muidenkin tutkimusten lähteinä, kuten esimerkiksi IEEE:n sähköistä kirjastoa. Tutkimuksen tavoitteena on pyrkiä tuomaan teknistä hyötyä toimeksiantajalle ja tutkia konttipalomuurauksen tuomia hyötyjä sekä toiminnallisuutta määrätyissä ympäristöissä.

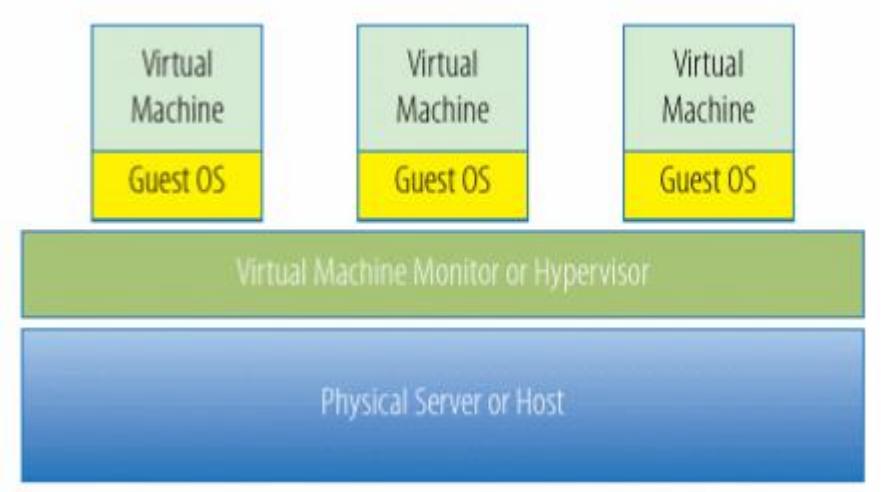
2 Virtualisointi

2.1 Yleistä

Virtualisoinnin historia kulkeutuu aina 1960-luvulle asti, jolloin IBM (International Business Machines) alkoi hyödyntämään virtualisointia osioimalla suuria keskustietokoneita. Idea virtualisointiin lähti alun perin tietokoneiden välimuistin rajallisuuden takia. Tietokone saatiin käyttämään erillistä virtuaalilevyä välimuistin kaltaisiin toimituksiin, jolla pystyttiin huijaaman tietokonetta luulemaan, että sillä on käytössä enemmän muistia kuin todellisuudessa. Virtualisointia kehitettiin kovasti vuosien varrella, mutta varsinaisen läpimurron se teki vasta vuosituhannen vaihteessa, kun VMware julkaisi tietokonearkkitehtuuria virtualisoivan ohjelmiston (Portnoy 2018, 1-2.)

Virtualisointi tarkoittaa virtuaalisten koneiden luontia käyttäen ainoastaan yhtä fyysistä konetta. Virtuaalikoneet käyttäytyvät ja toimivat fyysiseen koneen lailla, mutta ne eivät käytä omia resurssejaan vaan fyysisen niin sanotun isäntäkoneen resursseja, kuten muistia ja tallennustilaa. Tyypillisesti fyysisissä palvelimissa käyttöasteet ovat pieniä, minkä vuoksi laitekustannukset kasvavat. Virtualisoinnin avulla palvelimista saadaan kaikki hyöty irti ja niitä voidaan käyttää täydellä kapasiteetilla, mikä tuo säästöjä laitteistokustannuksissa. Virtualisoinnin avulla pystytään myös ajamaan samassa fyysisessä koneessa useita eri tyyppisiä käyttöjärjestelmiä ja ohjelmistoja. (Virtualization n.d.)

Oletuksena virtuaalikone voi virtualisoida kaikkia laitteistoja, kuten suorittimia, muistia, levytilaa ja verkkoyhteyksiä. Virtuaalikoneita monitoroi hypervisor ohjelmisto, joka tarjoaa ympäristön, jossa virtuaalikoneet voivat operoida. Kuviossa 2 on kuvailtu yleinen virtualisointimalli. (Portnoy 2016, 1-2.)



Kuvio 2. Yleinen virtualisointimalli (Portnoy 2016, 2)

2.2 Hypervisor

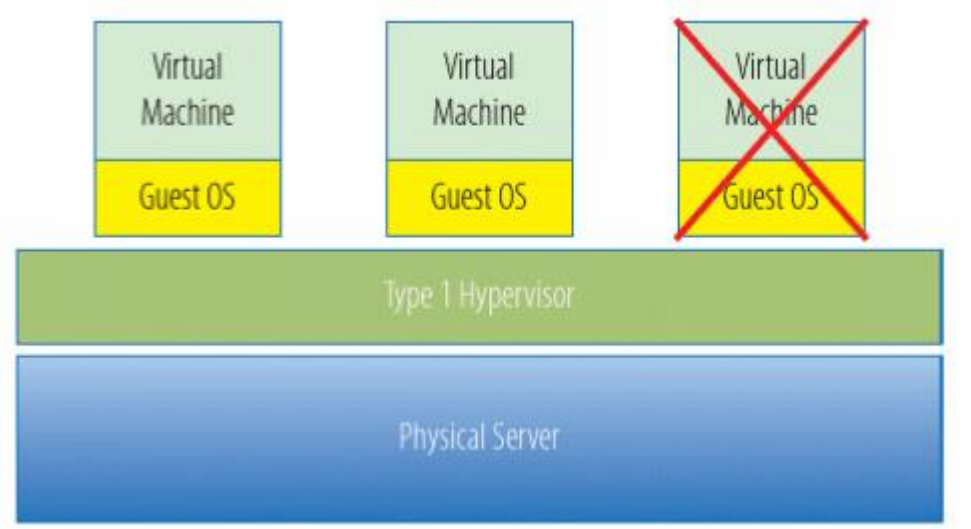
2.2.1 Yleistä

Hypervisor on sovelluskerros laitteiston ja yhden tai useamman virtuaalikoneen välillä. Hypervisor pyrkii tarjoamaan ympäristön, joka on identtinen fyysiseen ympäristöön verrattuna, sekä tarjoamaan ympäristön mahdollisimman pienillä suorituskykykustannuksilla ja säilyttämään täydellisen hallinnan järjestelmän resursseista. Hypervisor kehitettiin alun perin monitoroimaan virtuaalikoneita, jotta useat virtuaalikoneet pystyvät toimimaan samalla alustalla. Monitoroinnin lisäksi Hypervisor kykenee kohdistamaan fyysisen isäntäkoneen resursseja käyttäjän tarpeiden mukaisesti ja se myös hallitsee isäntäkoneen ja virtuaalikoneiden vuorovaikutusta. Hypervisorista on olemassa kaksi eri luokkaa: Tyyppi 1 ja Tyyppi 2. (Portnoy 2016, 26-27.)

2.2.2 Tyyppi 1

Tyypin 1 hypervisoria käytetään suoraan palvelimen fyysisellä laitteella siten, että sen alla ei ole käyttöjärjestelmää. Kyseistä Tyyppi 1:stä kutsutaan myös bare-metal toteutukseksi eli yksittäishaltijan fyysiseksi palvelimeksi, koska hypervisorin ja fyysisen laitteiston välissä ei ole muita ohjelmistoja. Tyypin 1 hypervisor kykenee kommunikoimaan suoraan laitteiston resurssien kanssa ilman välikäsiä, mikä tekeekin siitä Tyypin

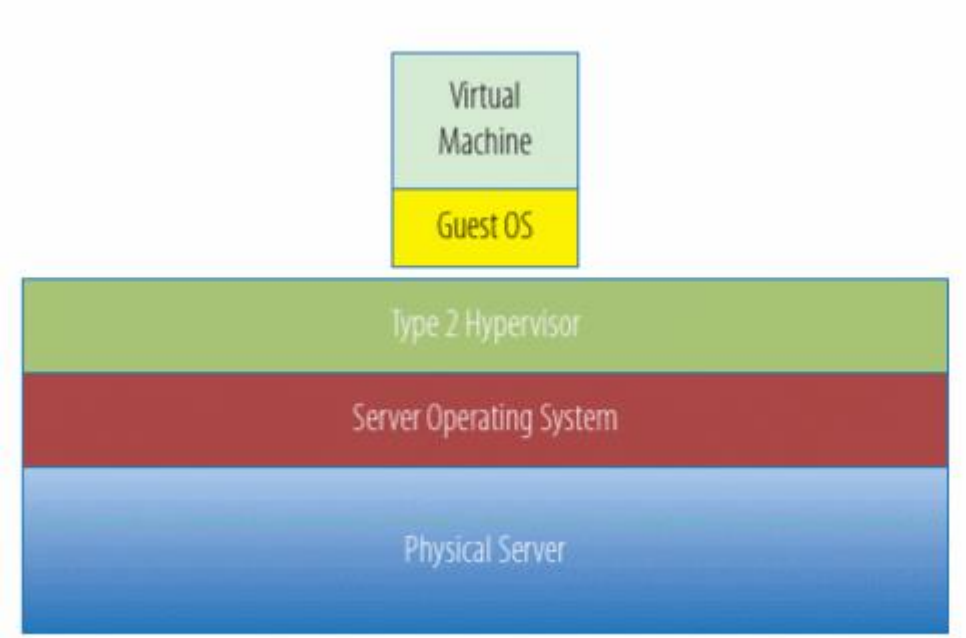
2 hypervisoria tehokkaamman ja takaa paremman hyötysuhteen, jonka ansiosta Tyypin 1 hypervisorilla voidaan ajaa useampia virtuaalikoneita yhdellä isäntäkäyttöjärjestelmällä. Paremman hyötysuhteen lisäksi Tyyppi 1 takaa paremman tietoturvan kuin Tyypin 2 hypervisor, koska Tyypin 1 hypervisor voi vahingoittaa vain itseään, jolloin ainoastaan yksi virtuaalikone kaatuu, mutta mikään viallinen tapahtuma ei voi karata virtuaalikoneen rajalta. Muut virtuaalikoneet jatkavat normaalia prosessointia ja myös itse hypervisor säilyy koskemattomana. Kuviossa 3 on havainnollistettu Tyypin 1 hypervisor ja yhden virtuaalikoneen virhetilanne. (Portnoy 2016, 20-24.)



Kuvio 3. Tyypin 1 hypervisor ja virhetilanne

2.2.3 Tyyppi 2

Tyypin 2 hypervisor on itsessään sovellus, jota ajetaan perinteisen käyttöjärjestelmän päällä. Ensimmäiset x86-ratkaisut oli toteutettu Tyypillä 2, koska se oli helpoin ja nopein keino. Varsinainen käyttöjärjestelmä pystyi jo valmiiksi käsittelemään laitteiston resursseja, ja hypervisorin avulla pyrittiin nostamaan sen kapasiteettia. Tyypin 2 suurin etu on, että se tukee laitteistoa erittäin laajasti, koska perii kaiken käyttämältään käyttöjärjestelmältä. Tyypin 2 hypervisor on myös erittäin helppo asentaa ja ottaa käyttöön, koska suurin osa konfiguroinneista, kuten verkot ja tiedontallennusvälineet on jo määritetty käyttöjärjestelmässä. Kuviossa 4 on esitelty Tyypin 2 hypervisor. (Portnoy 2016, 25-26.)



Kuvio 4. Tyypin 2 hypervisor (Portnoy 2018, 25.)

Tyypin 2 hypervisor ei ole yhtä kustannustehokas kuin Tyypin 1, koska Tyypissä 2 on ylimääräinen kerros hypervisorin ja laitteiston välillä. Joka kerta, kun virtuaalikone toteuttaa verkko -tai kovalevytoiminnon, täytyy Tyypin 2 hypervisorin ohjata pyyntö käyttöjärjestelmälle, joka käsittelee I/O -pyynnön. Tämän jälkeen käyttöjärjestelmä välittää tiedon takaisin hypervisorille, joka välittää sen virtuaalikoneelle. Tämän kaltainen toiminto lisää kaksi ylimääräistä askelta prosessiin ja vähentää kustannustehokkuutta. Tyypin 2 hypervisor on myös vähemmän luotettava ja alttiimpi vikatilanteille. Kaikki toiminnot, jotka vaikuttavat alla olevan käyttöjärjestelmän käytettävyyteen, voivat myös vaikuttaa hypervisoriin ja sen tukemiin virtuaalikoneisiin. Esimerkiksi alla olevan käyttöjärjestelmän uudelleen käynnistyksen vaativa päivitys tulee sammuttamaan myös kaikki virtuaalikoneet. Yleisesti Tyypin 2 hypervisoria käytetään työpöytäympäristöissä, joissa yksi kehittäjä työskentelee yhdellä tai useammalla virtuaalikoneella rakentaen sovellusta. Työpöytäympäristöissä Tyypin 2 hypervisor on enemmänkin työpöytäsovellus kehittäjälle toisin kuin Tyypin 1 hypervisor, jonka päätavoite on suorittaa virtuaalikoneita, eikä mitään muita sovelluksia laitteistolla. (Portnoy 2016, 25-25.)

2.3 Palvelinvirtualisointi

Eniten virtualisointia käytetään palvelimissa. Palvelinvirtualisoinnilla saadaan laskettua huimasti laitekustannuksia, koska tyypillisessä konesalissa saattaa olla useita palvelimia, joiden kapasiteetista käytetään vain murto-osa, mikä aiheuttaa jatkuvia lisäkustannuksia jäähdytys- ja ylläpitokustannuksissa. Palvelinvirtualisoinnin päätavoitteena on nostaa resurssien hyötykäyttöä jakamalla fyysinen palvelimen useaksi virtuaalipalvelimeksi, jossa jokainen virtuaalipalvelin käyttää omaa käyttöjärjestelmää ja sovelluksia. Palvelinvirtualisointi saa jokaisen virtuaalipalvelimen näyttämään ja käyttäytymään kuten normaali fyysinen palvelin moninkertaistamalla fyysisen palvelimen käytössä olevaa kapasiteettia. (Server virtualization n.d.)

2.4 Verkon virtualisointi

Verkon virtualisoinnilla tarkoitetaan koko yrityksen verkon hallinnointia ja valvontaa yhtenä kokonaisuutena yhdestä ohjelmistopohjaisesta hallintakonsolista. Verkon virtualisointi mahdollistaa verkon paremman optimoinnin datansiirron määrissä, joustavuudessa ja skaalautuvuudessa sekä tietoturvassa. Sillä pystytään automatisoimaan monia verkon hallinnoinnin tehtäviä, jolla voidaan ns. naamioida verkon monimutkaisuus. Verkon kaikki palvelimet ja palvelut luokitellaan yhdeksi resurssivarannoksi, jota pystytään käyttämään ottamatta huomioon fyysisiä laitteita. Verkon virtualisoinnin avulla kuormantasaus helpottuu sekä verkon ohjelmointi ja valmistelu ovat helpompaa, siten että verkon infrastruktuuria ei tarvitse muokata. (Network Virtualization n.d.)

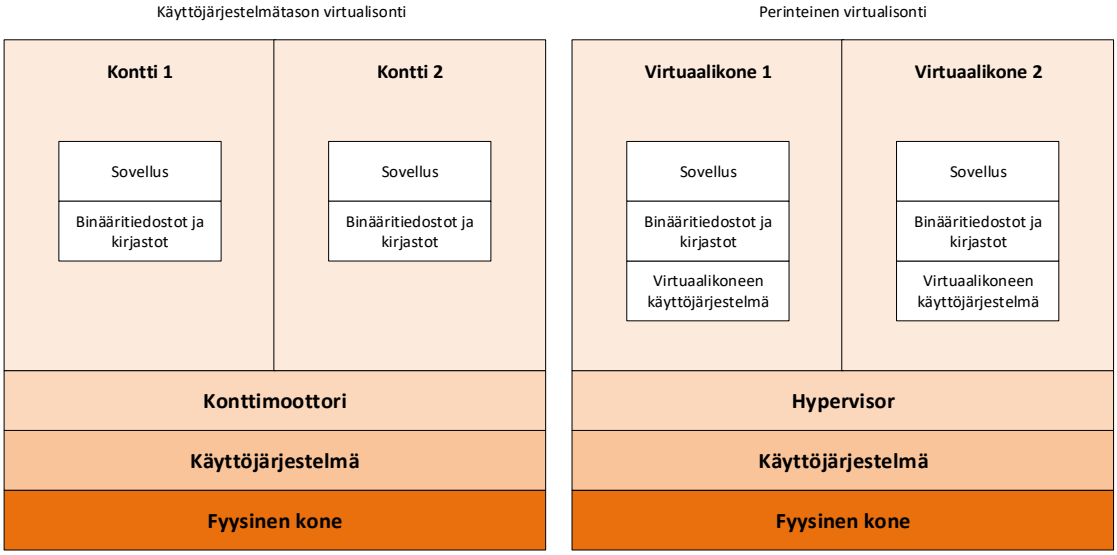
2.5 Työpöydän virtualisointi

Työpöydän virtualisoinnilla erotetaan käyttäjän työpöytäympäristö ja fyysinen työasema toisistaan. Tällä mahdollistetaan käyttäjien pääsy omaan työympäristöönsä, milloin ja mistä tahansa. Työympäristöllä tarkoitetaan omaa työpöytää ja työhön tarvittavia sovelluksia. Työpöydän virtualisoinnilla pystytään vähentämään yrityksen kustannuksia ylläpidossa ja hallinnoimisessa. Virtuaaliset työpöydät sijaitsevat palvelimella, josta käsin voidaan käyttäjälle antaa käyttöön simuloituja työpöytiä ja sovel-

luksia ilman, että niitä tarvitsee asentaa käyttäjän fyysiselle työasemalle. Käyttäjätasolla muutos ei juurikaan näy, koska virtuaalisia työpöytiä ja sovelluksia käytetään aivan kuten fyysisellä laitteellakin. Työpöydän virtualisoinnilla voidaan myös ajaa helpommin tarvittavat päivitykset käyttäjätasolle ja hallinnoimisesta sekä valvonnasta tulee yksinkertaisempaa ja tehokkaampaa. (HyungJik & JeunWoo. 2010.)

2.6 Käyttöjärjestelmätason virtualisointi

Käyttöjärjestelmätason virtualisointi on tekniikka, jossa ydin (engl. kernel) sallii useiden toisistaan eristäytyneen käyttäjäinstanssin (engl. user space) käytön. Käyttäjätasolle nämä instanssit vaikuttavat oikeilta palvelimilta. Käyttöjärjestelmätason virtualisointi on kevyt ja resurssiystävällinen virtualisointitekniikka, koska se ei tarvitse laitteiston tukea toimiakseen kustannustehokkaasti. Käyttöjärjestelmätason virtualisoinnissa virtuaalikoneita kutsutaan yleisesti konteiksi ja ne eroavat perinteisistä virtuaalikoneista siten, että ne eivät tarvitse omaa virtuaalista käyttöjärjestelmää, vaan pärjäävät pelkillä binääritiedostoilla ja kirjastoilla. Kuviossa 5 on perinteisen virtualisoinnin erot käyttöjärjestelmätason virtualisointiin verrattuna. (Jimenez, Maltzahn, Lofstead, Moody, Mohror, Arpaci-Dusseau & Arpaci-Dusseau 2016.)



Kuvio 5. Virtualisontitekniikoiden vertailu

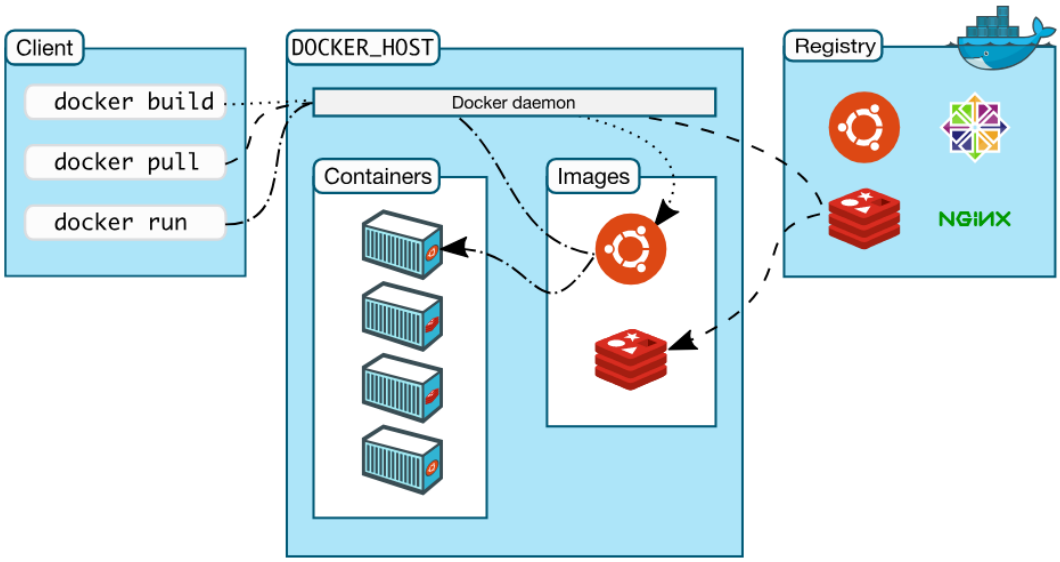
3 Docker

3.1 Yleistä

Docker on yksi suosituimmista konttiteknologioista. Konttiteknologialla tarkoitetaan sovelluksen ja sen tarvitseman sovellusalustan paketointia kontteihin. Kontit ovat täysin eristettyjä ympäristöjä ympäröivästä käyttöjärjestelmästä ja muista konteista. Kontitus tapahtuu käyttöjärjestelmätason virtualisoinnilla eli kaikki kontit jakavat saman käyttöjärjestelmän. Konteilla ei ole siis omaa käyttöjärjestelmää, eivätkä ne tarvitse erillistä hypervisoria. Tämän takia konttien koko on ainoastaan useita megatavuja, jonka ansiosta kontit käynnistyvät sekunneissa verrattuna virtuaalikoneiden gigatavujen kokoihin ja jopa minuuttien käynnistysaikoihin. (Jatin 2017.)

Docker konttien tavoitteena on ratkaista pitkäikäinen ongelma sovelluskehityksessä eli siirrettävyys. Konttien avulla käyttäjät pystyvät helposti luomaan erilaisia ympäristöjä, jotka on muokattu täydellisesti tarpeisiin soveltuviksi. Docker mahdollistaa myös täydellisten kopioiden tekemisen omasta ympäristöstä millä tahansa infrastruktuurilla nopeammin ja tehokkaammin kuin virtuaalikoneita käyttäessä. (Abdelbaky, Diaz-Montes, Parashar, Unuvar & Steinder 2015.)

Docker-arkkitehtuuri muodostuu asiakas-palvelinympäristö-mallista (engl. client-server). Kyseisessä arkkitehtuurissa Dockerin asiakasohjelma (engl. client) keskustelee Dockerin palveluprosessille (engl. daemon), joka tekee monimutkaisemmat prosessit, kuten konttien rakentamisen ja ajamisen. Asiakasohjelmaa ja palveluprosessia voidaan ajaa erillään tai samassa järjestelmässä. Kuviossa 6 on kuvailtu Dockerin perusarkkitehtuuri. (Docker overview 2017.)



Kuvio 6. Dockerin perusarkkitehtuuri

3.2 Docker-verkot

3.2.1 Yleistä

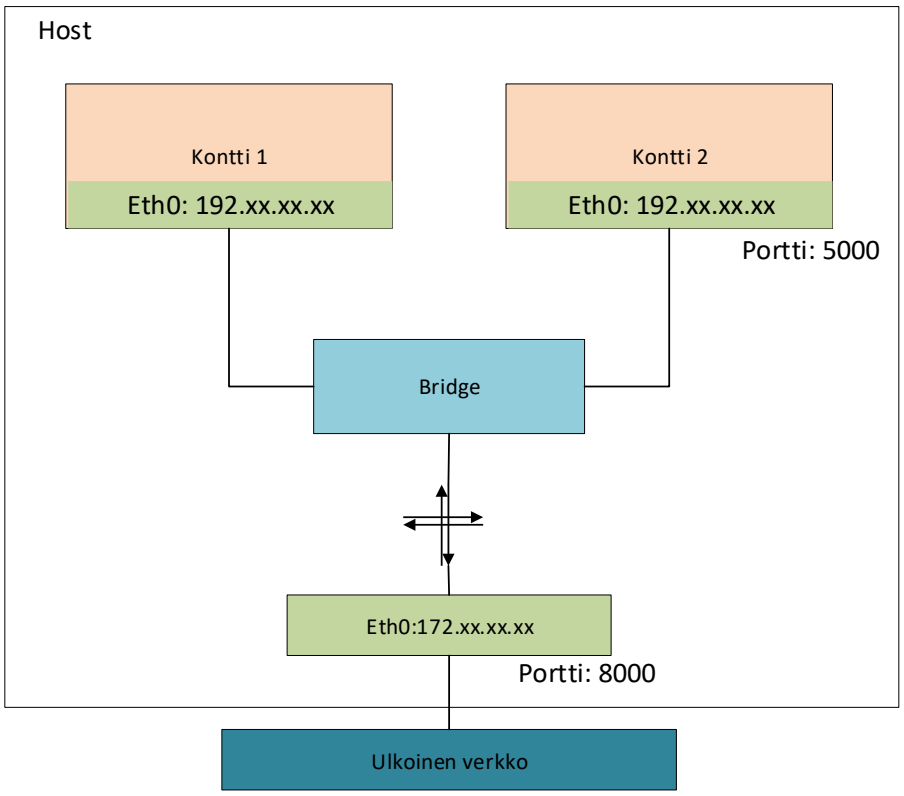
Docker luo asennuksen yhteydessä automaattisesti kolme verkkoa: bridge, none ja host. Nämä kolme verkkoa on rakennettu Dockerin sisään. Kontin käynnistyksen yhteydessä voi erillisellä parametrilla määrittää, mihin verkkoon kontin halutaan yhdistävän. Bridge-verkko tarkoittaa docker0-verkkorajapintaa, joka on ethernet-siltauslaite. Mikäli kontille ei määritetä käynnistyksen yhteydessä verkkoa, yhdistää verkko aina oletuksena bridge-verkkoon, jolloin kaikki tuleva ja menevä liikenne kontilta kulkee bridgen yli Dockerin palveluprosessille, joka käsittele reitityksen kontin puolesta. Docker konfiguroi docker0-verkkorajapintaan oletuksena IP-osoitteen, aliverkon peitteen ja IP-osoitteiden jakamisalueen. Näitä asetuksia pystyy muuttamaan dockerd -komennolla tai muokkaamalla daemon.json tiedostoa /etc/docker -polusta. Turvallisuussyistä Docker konfiguroi iptables -sääntöjä siten, että se estää kontteja edelleen lähettämästä liikennettä ulos isäntäkoneelta. Docker määrittää edelleen lähetysketjun oletuskäytänteen pudottamaan paketit. Tämän asetuksen voi yliajaa tarvittaessa manuaalisesti. (Understand container communication n.d.)

Kontin kommunikointi ulkomaailmaan riippuu yleisesti kahdesta tekijästä: edelleenlähettääkö isäntäkone sen IP-paketteja ja sallivatko isäntäkoneen iptables-säännöt kyseisen yhteyden. IP-pakettien edelleen lähetystä hallitaan ip_forward -parametrillla, jossa paketit voivat kulkea konttien välillä ainoastaan, jos parametrin arvo on määritetty oikein. Kahden kontin välinen liikenne riippuu myös kahdesta tekijästä: toimiiko verkkotopologia oikein, ja yhdistyykö kontin verkkorajapinnat oikein ja salliiko iptables-säännöt kyseisen yhteyden. (Understand container communication n.d.)

Dockerissa on muutamia erilaisia verkkorajapinta-ajureita, jotka soveltuvat erilaisiin käyttötarkoituksiin. Jokainen ajuri tarjoaa erilaiset hyödyt käyttökohteesta riippuen. Eniten käytettyjä Dockeriin sisäänrakennettuja verkkorajapinta-ajureita ovat bridge, overlay ja Macvlan. Yhdessä nämä kolme verkkoajuria kattavat suurimman osan yleisistä käyttökohteista. (Church 2016.)

3.2.2 Bridge-verkkoajuri

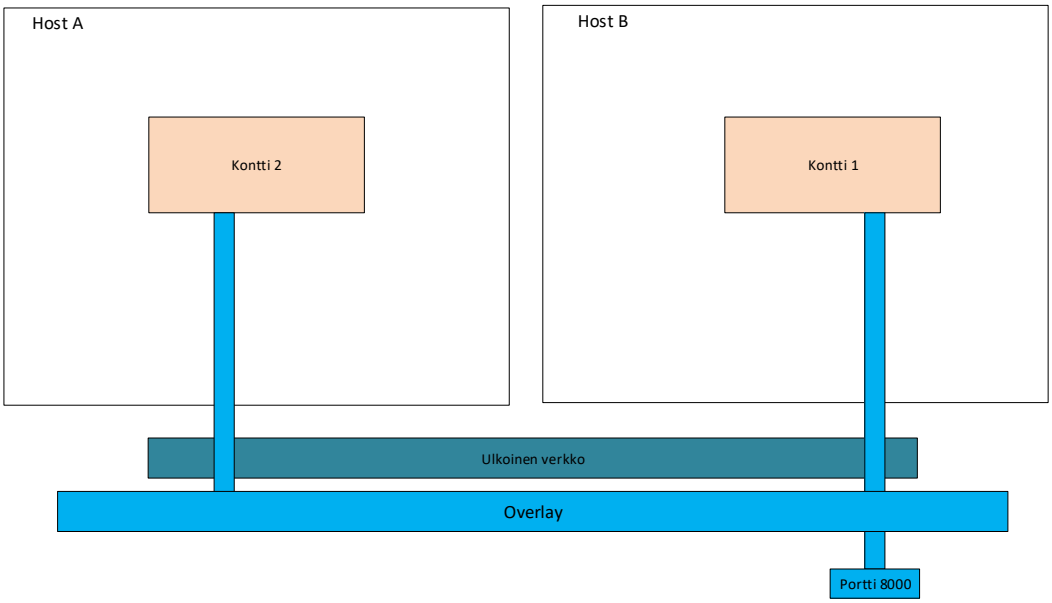
Bridge-verkkoajuri on yksinkertaisin ymmärtää, käyttää ja sen vianselvitys on helppoa, minkä takia se on hyvä valinta ohjelmistokehittäjille ja uusille Docker käyttäjille. Bridge verkkoajuri tekee yksityisen sisäisen verkon isäntäkoneelle, jonka kautta kontit voivat kommunikoida. Ulkoinen yhteys mahdollistetaan paljastamalla porttien näkyvyys konteille. Docker turvaa verkon hallinnoimalla sääntöjä, jotka estävät yhteyden erilaisten Docker verkkojen välillä. Docker tekee automaattisesti tarvittavat Linux siltaukset, sisäiset rajapinnat, iptables-säännöt ja isännän reititykset, jotta yhteydet ovat mahdollisia käyttää. Kuviossa 7 on kuvailtu bridge verkkoajurin toiminta. (Church 2016.)



Kuvio 7. Bridge-ajurin toiminta

3.2.3 Overlay-verkkoajuri

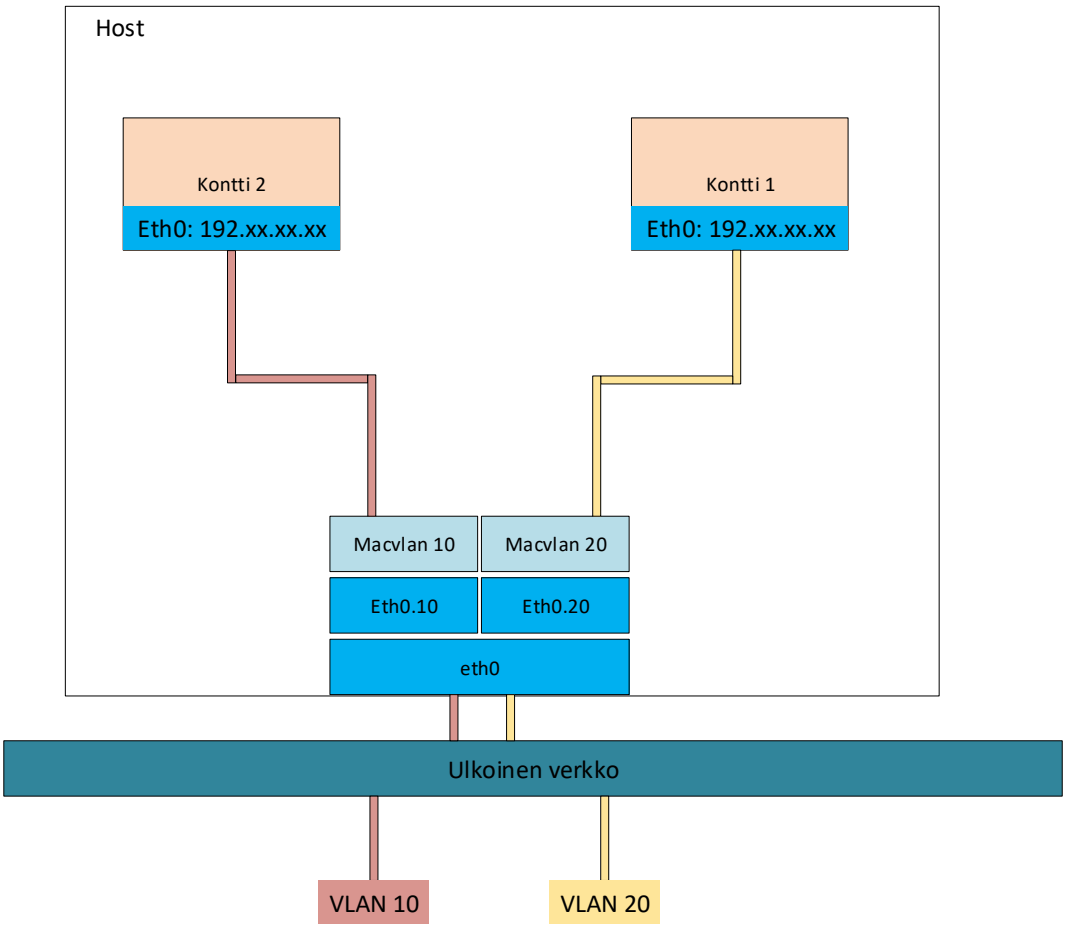
Overlay verkkoajuri yksinkertaistaa monimutkaisen multi-host-verkon. Overlay on Swarm näkyvyysalueen (engl. scope) ajuri, mikä tarkoittaa, että se operoi koko Swarmin tai klusterin alueella eikä ainoastaan yksittäisillä isännillä. Overlay ajuria käytettäessä multi-host-verkot ovat ensisijaisia Dockerin sisällä ilman ulkoista provisiointia tai muita komponteja. IPAM (IP address management), multi-host yhteydet, salaus ja kuormanhallinta on rakennettu suoraan ajurin sisään. Overlay-ajuri hyödyntää VXLANin (Virtual Extensible LAN) datapinta (engl. data plane) standardia, joka erottaa kontin verkon sen alla olevasta fyysisestä verkosta. Tämän toiminnon avulla saavutetaan korkean tason siirrettävyys erilaisissa pilviverkoissa. Verkko- ja turvallisuuspolitiikkaa sekä näkyvyyttä hallitaan keskitetysti Dockerin UCP:n (Universal Control Plane) kautta. Kuviossa 8 on kuvailtu overlay ajurin toimintaa siten, että kontit ovat eri isännillä. (Church 2016.)



Kuvio 8. Overlay-ajurin toiminta

3.2.4 Macvlan-verkkoajuri

Macvlan mahdollistaa usean MAC (Media access control)- ja IP-osoitteen määrittämisen yhdelle fyysiselle rajapinnalle. Macvlan-ajurin on erittäin kevyt ajuri, ja se yhdistää kontin rajapinnan suoraan isännän rajapintaan. Kontit saavat reitittävät IP-osoitteet, jotka ovat ulkoisen verkon aliverkossa. Reitittävän IP-osoitteen avulla kontit voivat kommunikoida suoraan Swarm klusterin ulkopuolella olevien resurssien kanssa ilman, että mukaan tarvitaan NAT (Network Address Translation) tai porttikartoitusta. Nämä toimenpiteet auttavat verkon näkyvyyden kanssa ja vianselvityksessä sekä suora reitti kontin ja isännän rajapinnan välillä vähentää viivettä. Kuviossa 9 on havainnollistettu Macvlan-verkkoajurin toimintaa luomalla kaksi macvlan verkkoa, jotka liittyvät kahteen eri alirajapintaan. Tämän kaltaista konfiguraatiota voi soveltaa, jos haluaa jatkaa useaa Layer 2-vlania isäntä rajapinnan läpi suoraan konteille. (Church 2016.)

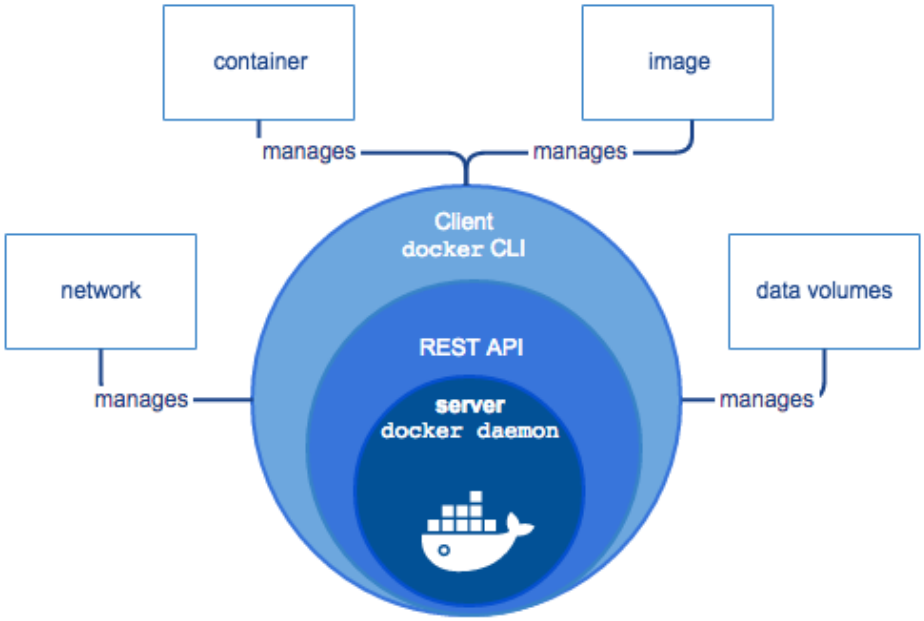


Kuvio 9. Macvlan-ajurin toiminta

3.3 Docker engine

Docker engine on Dockerin perustana oleva asiakas-palvelinympäristö-tekniikka, joka rakentaa ja ajaa kontteja käyttäen Dockerin komponentteja ja palveluita. Useasti ihmisten puhuessa Dockerista he tarkoittavat oikeasti Docker engineä. Docker engine rakentuu Dockerin CLI:stä (Command line interface), REST API:stä (Representational state transfer application programmable interface) ja Docker daemonista eli palveluprosessista. Docker daemon on koko järjestelmän ydin. Se on prosessi, joka muodostaa ja käsittelee Dockerin objekteja. Daemonin hallitsemiin objekteihin kuuluvat verkko, kontit, levykuvat ja data. REST API toimii kommunikointikanavana daemonin ja Docker CLI:n välillä. Docker CLI lähettää komentoja tai skriptejä REST API:lle, jonka jälkeen se välittää ne daemonille, joka suorittaa ne. Docker CLI vaatii REST API:n,

jotta se voi olla vuorovaikutuksessa Daemoniin. Docker CLI:llä käyttäjä pystyy syöttämään komentoja, jolla varsinainen Dockerin ajaminen tapahtuu. Kuviossa 10 havainnollistetaan Docker enginen toiminta. (Docker overview, 2017.)



Kuvio 10. Docker engine (Docker overview, 2017).

3.4 Docker-levykuva

Docker-levykuvien avulla luodaan Docker kontteja. Levykuvat ovat Dockerin rakennuskomponentteja ja ne ovat kirjoitussuojattuja. Levykuvat voivat pitää sisällään esimerkiksi pelkän käyttöjärjestelmän pohjan tai jonkin sovelluksen. Levykuvia voi rakentaa Dockerissa itse tai vaihtoehtoisesti ladata muiden käyttäjien rakentamia levykuva julkisesta dockerin rekisteristä. Useimmat Docker-levykuvat alkavat jostain valmiista niin kutsutusta pohjalevykuvasta, mutta käyttäjä voi tarvittaessa myös rakentaa alusta alkaen oman levykuvansa. (Docker overview, 2017.)

Levykuvat koostuvat useista kerroksista, jotka yhdistetään yhdeksi levykuvaksi käyttäen Dockerin UnionFS-tiedostojärjestelmää (Union File System). UnionFS:n avulla pystytään muodostamaan yhtenäisiä tiedostojärjestelmiä, sekä pinoamaan tiedostoja ja tiedostojärjestelmien haaroja päällekkäin. Aina, kun Dockeriin levykuvaan tehdään muutos, muodostetaan myös uusi kerros levykuvaan. Esimerkiksi sovellusversiota päivittäessä voidaan ainoastaan muodostaa uusi kerros tai päivittää nykyistä, jolloin

koko levykuvaa ei tarvitse korvata ja levykuvasta voidaan jakaa vain päivitettyä versiota tarvittaessa. (Docker overview, 2017.)

3.5 Dockerfile

Dockerfilen avulla voidaan automatisoida levykuvien rakentaminen. Dockerfile on komennoista ja muuttujista rakentuva skripti. Oikeilla komennoilla ja muuttujilla voidaan automaattisesti suorittaa toimintoja pohjalevykuvaan luomalla kokonaan uusi levykuva. Dockerfile sisältää kaikki tarvittavat komennot levykuvan muodostamiseen, kuten porttien avaamisen ja ympäristömuuttujien luomisen, sekä ohjelmien lataamisen ja asentamisen. Dockerfilejä tarkoituksena on yksinkertaistaa koko käyttöönottoprosessia alusta loppuun. Dockerfileen voidaan myös lisätä muita konfiguraatiotiedostoja isäntokoneelta tai halutusta URL (Uniform Resource Identifier) -osoitteesta. Ennen levykuvan rakentamista dockerfilellä Docker tarkastaa, että Dockerfilen komennot on määritetty oikein, mikäli samaa levykuvaa on jo rakennettu, on mahdollista hakea aikaisemmin käytetyn komennon kerrosta välimuistista. Välimuistin ja kerrosten käyttö nopeuttaa levykuvien rakentamista, jos tavoitteena on korjata ainoastaan jokin aikaisemmin levykuvan rakennuksessa käynyt vika. (Dockerfile reference. N.d.)

3.6 Rekisteri

Dockerin rekisteri on tallennus- ja jakelujärjestelmä Dockerin levykuville. Rekistereitä on olemassa julkisia ja yksityisiä. Julkisia rekistereitä ovat muun muassa Docker Hub ja Docker Cloud, joista kuka tahansa voi ladata levykuvia. Docker on myös konfiguroitu oletuksena hakemaan levykuvia Docker Hubista. Yksityiset rekisterit ovat usein käytössä vain yrityksen sisäisesti palomuurien takana. Docker rekisteri on organisoitu säilytyspaikoiksi (engl. repositories), joka sisältää tietyn levykuvan kaikki versiot. Dockerilla voidaan työntää levykuvia haluttuun rekisteriin tai vaihtoehtoisesti hakea levykuvia halutusta rekisteristä. (Docker Registry. N.d.)

3.7 Docker Compose

Compose on työkalu, jolla voidaan määrittää ja luoda useita kontteja sisältäviä palveluita. Composella luotu kokonaisuus on myös yksittäisten konttien tapaan eristyksissä muusta isäntäkoneesta. Composessa käytetään YAML (YAML Ain't Markup Language)-kielellä kirjoitettua konfiguraatiotiedostoa, jossa määritetään sovelluksen palvelut ja halutut määritykset. Kuviossa 11 on esimerkki YAML-tiedostosta, jota käytettiin cSRX konttipalomuurin kanssa. (Overview of Docker Compose. N.d.)

```
version: '3.7'

services:
  csrx:
    image: [redacted]/csrx:18.2r1.9
    privileged: true
    networks:
    - mgt_bridge
    - left_bridge
    - right_bridge
    deploy:
      mode: replicated
      replicas: 1
      resources:
        limits:
          memory: 512M
      restart_policy:
        condition: on-failure
        delay: 1s
        max_attempts: 3
      update_config:
        delay: 10s
    environment:
    - TZ=Europe/Helsinki
    - CSRX_FORWARD_MODE=wire
    labels:
      app: csrx
    ports:
    - 22:22/tcp
    - 830:830/tcp
    volumes:
    - /config
    - /tmp
    - /var
    - /home/p[redacted]esktop/lokit:/var/log
networks:
  mgt_bridge:
    driver: bridge
  left_bridge:
    driver: bridge
  right_bridge:
    driver: bridge
```

Kuvio 11. cSRX:n docker-compose.yml tiedosto.

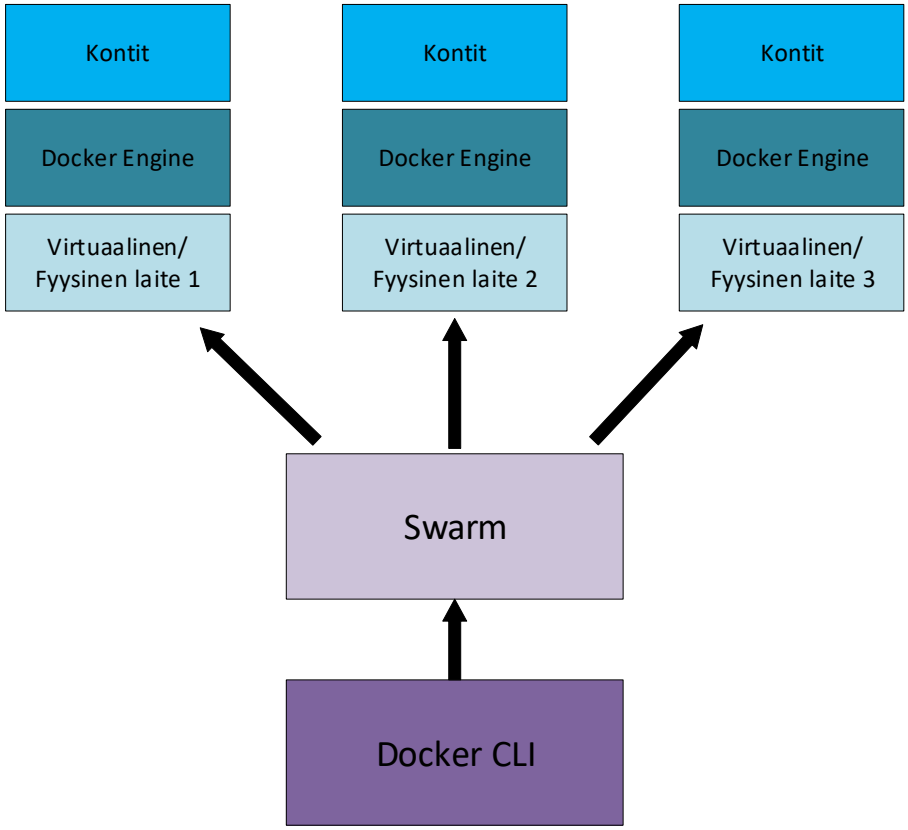
Compose varastoi konfiguraation, jota on käytetty kontin tekemiseen. Jos käynnistetään uudestaan palvelu, joka ei ole muuttunut, käyttää compose uudestaan olemassa olevaa konttia. Kontin uudelleen käyttämisen avulla pystytään tekemään tarpeen tullen nopeita muutoksia ympäristöön. Compose säilyttää kaikki levyt, joita palvelu on käyttänyt. Mikäli 'docker-compose up' löytää yhdenkään konttin edellisistä ajoista, kopioi se levyt vanhasta kontista uuteen konttiin, jolla varmistetaan, että levyillä oleva tieto ei katoa. (Overview of Docker Compose. N.d.)

3.8 Docker Swarm

Docker swarm on avoimen lähdekoodin konttien orkesterointialusta, joka tarjoaa klusterointiominaisuuden Dockeriin. Jokainen Dockerilla toimiva sovellus tai palvelu toimii myös Swarmissa ja Swarm käyttää samoja komentorivikomentoja kuin Docker. Swarm on Dockeria ajavia laitteita, jotka on laitettu samaan klusteriin. Dockerin ajamista voi jatkaa kuin normaalisti Dockerissa, mutta tällä kertaa komennot suorittaa swarm manager klusterissa. Laitteet swarmissa voivat olla fyysisiä tai virtuaalisia ja swarmiin liittymiseen jälkeen liittyneitä laitteita kutsutaan nodeiksi. Kuviossa 12 on havainnollistettu Swarmin perustopologiaa. Swarmin tärkeimmät ominaisuudet ovat seuraavat:

- Pystytään tasavertaistamaan kontteja ja kohdistamaan tehtäviä konttiryhmille tarpeen mukaan.
- Voidaan tarkastella sekä ylläpitää konttien kuntoa ja hallinnoida yksittäisten konttien elinkaarta sekä tarjota parempaa redundanssia.
- Helppo skaalata konttien määrää kuorman perusteella.
- Sovelluspäivitysten helppo ajaminen useille konteille.

(Get Started, Part 4: Swarms n.d.)



Kuvio 12. Swarmin perustopologia

Swarm manager voi käyttää useita erilaisia toimintasuunnitelmia konttien ajamiseen. Esimerkiksi emptiest node -suunnitelman avulla täytetään vähiten käytetty laite konteilla tai global-suunnitelmalla, joka varmistaa, että jokaisella laitteella on tasan yksi instanssi samasta kontista ajossa samanaikaisesti. Palvelut saa helposti pystyy Docker Compose- tiedostolla, jolla voidaan käynnistää haluttu kontti tietyillä parametreilla. Swarmin käyttöönotto on nopeaa, koska se tarvitsee vain sallia Dockerilla. Ottamalla Swarm käyttöön tulee ajavasta laitteesta Swarm manager, minkä jälkeen Docker komentoja ajetaan Swarmissa eikä pelkästään yhdellä laitteella. Swarmin voi ottaa käyttöön docker swarm init -komennolla, jolloin laitteesta tulee Swarm manager. Muilla laitteilla pystyy liittymään Swarmiin docker swarm join -kommenolla, jolloin muut Swarmiin liittyneet laitteet saavat worker-roolin Swarmissa. (Get Started, Part 4: Swarms n.d.)

Manager node käsittelee klusterin hallinnointitehtävät, joihin kuuluvat klusterin tilan ylläpito, palveluiden ajoitus ja swarm moden HTTP (Hypertext Transfer Protocol) ohjelmointirajapinnan päätepisteiden tarjoaminen. Worker nodejen päätarkoitus on ainoastaan ajaa kontteja, ja ne tarvitsevat aina rinnalleen myös manager noden. Oletuksena kaikki manager nodet ovat myös worker nodeja, mutta manager noden voi erillisellä komennolla estää suorittamasta kontteja. Swarm myös mahdollistaa worker noden ylentämisen manageriksi, jos on esimerkiksi tarve tehdä manager nodelle päivityksiä. (Get Started, Part 4: Swarms n.d.)

3.9 Haavoittuvuudet

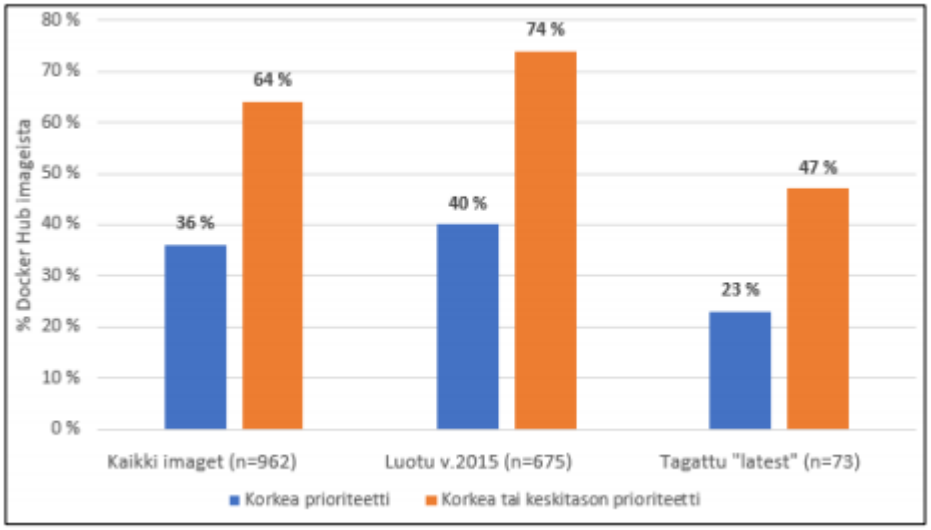
3.9.1 Yleistä

Tällä hetkellä Dockeriin on CVE (Common Vulnerabilities and Exposures) -tietokannan mukaan 18 julkisesti tunnettua suoraa haavoittuvuutta, joista vain yksi on saanut täyden kymmenen pisteen luokituksen, eli se on määritetty erittäin suureksi haavoittuvuudeksi, joka havaittiin vuonna 2014 Dockerin 1.3.2 versiossa. Kyseinen haavoittuvuus korjattiin jo kuukausi sen havaitsemista seuraavassa Dockerin versiossa 1.3.3. Useimmat vakavimmat Dockerin haavoittuvuudet ilmenivät sen ensimmäisiä versioissa, ja ne on saatu jo korjattua uudemmissa päivityksissä. Tällä hetkellä Dockerin uusimmassa versiossa ei ole havaittu suoranaisia haavoittuvuuksia, mutta Dockeriin

pätevät samat hyökkäysmahdollisuudet kuin muihinkin konttitekniikkaa käyttäviin järjestelmiin. (Docker Security Vulnerabilities n.d.)

3.9.2 Myrkyttyneet levykuvat

Vuonna 2015 tehdyn BanyanOpsin tutkimuksen mukaan Docker hubissa noin 30 % levykuvista olivat alttiita erilaisille hyökkäyksille, kuten Heartbleedille ja Shellshockille. Tutkimuksen teon aikana Docker hubissa oli virallisia repositoryjä 75 kappaletta. Kuviossa 13 on havainnollistettu diagrammien avulla haavoittuvuuksia virallisista levykuvista, mistä voidaan huomata, että vuonna 2015 korkean prioriteetien haavoittuvuuksia löytyy 40 % levykuvista. (Desikan, Gummaraju & Turner 2015.)



Kuvio 13. Virallisten levykuvien haavoittuvuusmäärät (Kuoppala 2017.)

Vuonna 2018 havaittiin 17 kappaletta levykuvia Docker hubissa, johon oli avattu takaovi (engl. backdoored) ja joita käytettiin reverse shell -hyökkäyksiin ja kryptovaluutan louhinnan aloittamiseen. Kyseiset pahantahtoiset levykuvat säilyivät Docker hubissa vuoden, ennen kuin ne havaittiin ja saatiin poistettua rekisteristä. Jotkut levykuvista ehdittiin ladata miljoonia kertoja ennen niiden poistamista, jolloin todella iso osa peruskäyttäjistä ehti altistua kyseisille hyökkäyksille. Nykyään Docker tarjoaakin liitännäisiä, joiden avulla pystyy skannaamaan levykuvia. Levykuvia suositellaankin skannattavaksi käyttöjärjestelmä -ja sovellustasolla ennen niiden käyttöönottoa tuotantoympäristöihin. (Cimpanu 2018.)

3.9.3 Arp spoofing ja Mac flooding -hyökkäykset

Dockerin luodessa uuden kontin, se luo myös uuden virtuaalisen Ethernet rajapinnan uniikilla nimellä, jonka jälkeen se yhdistää kyseisen rajapinnan siltaan (engl. bridge). Rajapinta on yhteydessä myös kontin eth0 rajanpintaan, joka mahdollistaa, että kontti kykenee lähettämään paketteja siltaan. Tämä normaali Dockerin yhdistämismalli tekee sen haavoittuvaiseksi ARP (Address Resolution Protocol) spoofingille ja Mac flooding hyökkäyksille, koska silta edelleen lähettää kaikki sille tulevat paketit ilman suodatusta. (Chelladhurai, Chelliah & Kumar. 2016.)

3.9.4 Docker Daemon hyökkäykset

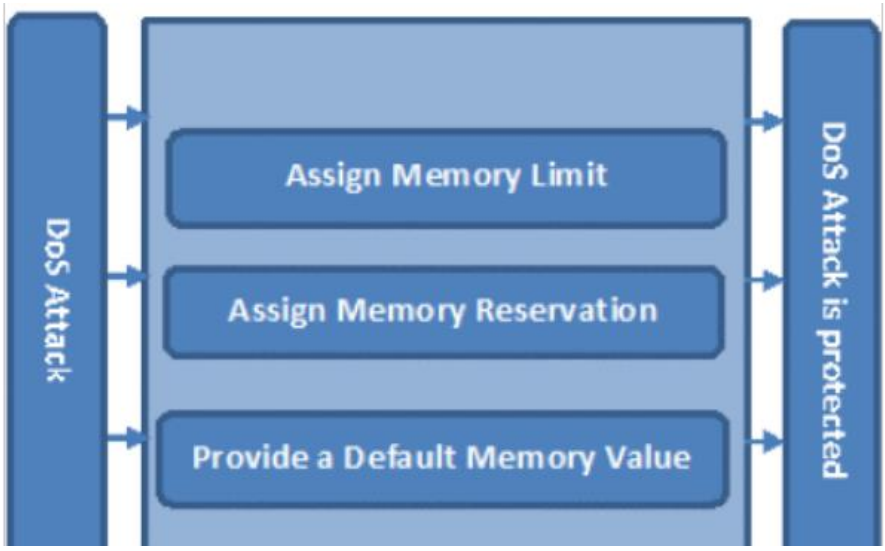
Docker Daemon eli Dockerin palveluprosessi sijaitsee Docker asiakasohjelman (engl. client) ja konttien välillä. Käyttäjä ei suoranaisesti ole vuorovaikutuksessa palveluprosessin kanssa, vaan käyttää Dockerin asiakasohjelmaa. Dockerin palveluprosessi vaatii root-oikeudet, joten ainoastaan valtuutetut käyttäjät pystyvät hallitsemaan palveluprosessia. Docker sallii hakemistojen jakamisen Docker isännän (engl. host) ja vieraskontin välillä rajoittamatta pääsyoikeuksia kontteihin. Ongelmaksi muodostuu se, että kuka vain voi käynnistää kontin siellä, missä isäntä hakemisto on tai missä hakemisto on Docker isännällä, jolloin kontti kykenee muuttamaan isännän tiedostojärjestelmää ilman rajoitteita. tämä aiheuttaa suuremman luokan tietoturva-aukon. Jos esimerkiksi Docker isännällä sijaitsee web-palvelin, on hyökkääjällä mahdollista syöttää luodut parametrit luodakseen omia kontteja web-palvelimen isännälle. (Chelladhurai ym. 2016.)

3.9.5 Palvelunestohyökkäykset

Kaikki kontit jakavat saman ytimen resursseja, joten jos jokin kontti pystyy haukkamaan liikaa ytimen resursseja, voi se kaataa muut kontit isäntäkoneella, jonka tuloksena on palvelunestohyökkäys. Kontit pystyvät kommunikoimaan suoraan isäntäkoneen ytimen kanssa, joka mahdollistaa hyökkääjälle säästämään runsaasti aikaa tun-

keutuessaan isäntäjärjestelmään. Tämän takia kontit ovat hyviä kohteita palvelunesto hyökkäyksille. Virtuaalikoneet onkin yleisesti kerrottu olevan suojatumpia kuin kontit, koska virtuaalikoneet lisäävät yhden kerroksen eristystä sovellusten ja isännän välille. Virtuaalikoneen sisällä ajettava sovellus pystyy kommunikoimaan ainoastaan virtuaalikoneen ytimen, eikä isäntäkoneen ytimen kanssa. Sen seurauksena sovelluksen täytyy ulos päästäkseen ohittaa ensin virtuaalikoneen ydin ja hallintaalusta, ennen kuin se voi hyökätä isännän ytimeen. (Chelladhurai ym. 2016.)

Palvelunestohyökkäyksiä vastaan pystyy suojautumaan Dockerilla rajaamalla muistin ylärajaa jokaiselle kontille niitä luodessaan tai rajaamalla muistin varausta. Mikäli kumpaakaan ylläolevista ei ole määritetty, täytyy pienentää muistin kokonaiskäyttöä jakamalla kokonaismuistinkäyttö, jotta ainoastaan minimimäärä kontteja on mahdollista ajaa samanaikaisesti. Näillä muutoksilla pyritään siihen, että palvelunestohyökkäykset eivät kaada koko järjestelmää, vaan ainoastaan pahimmassa tapauksessa yksittäisiä kontteja. Kuviossa 14 on esitetty yleisimmät keinot palvelunestohyökkäyksiltä suojautumiseen. (Chelladhurai ym. 2016.)



Kuvio 14. Palvelunestohyökkäykseltä suojautuminen Dockerissa (Chelladhurai ym. 2016).

4 Openshift

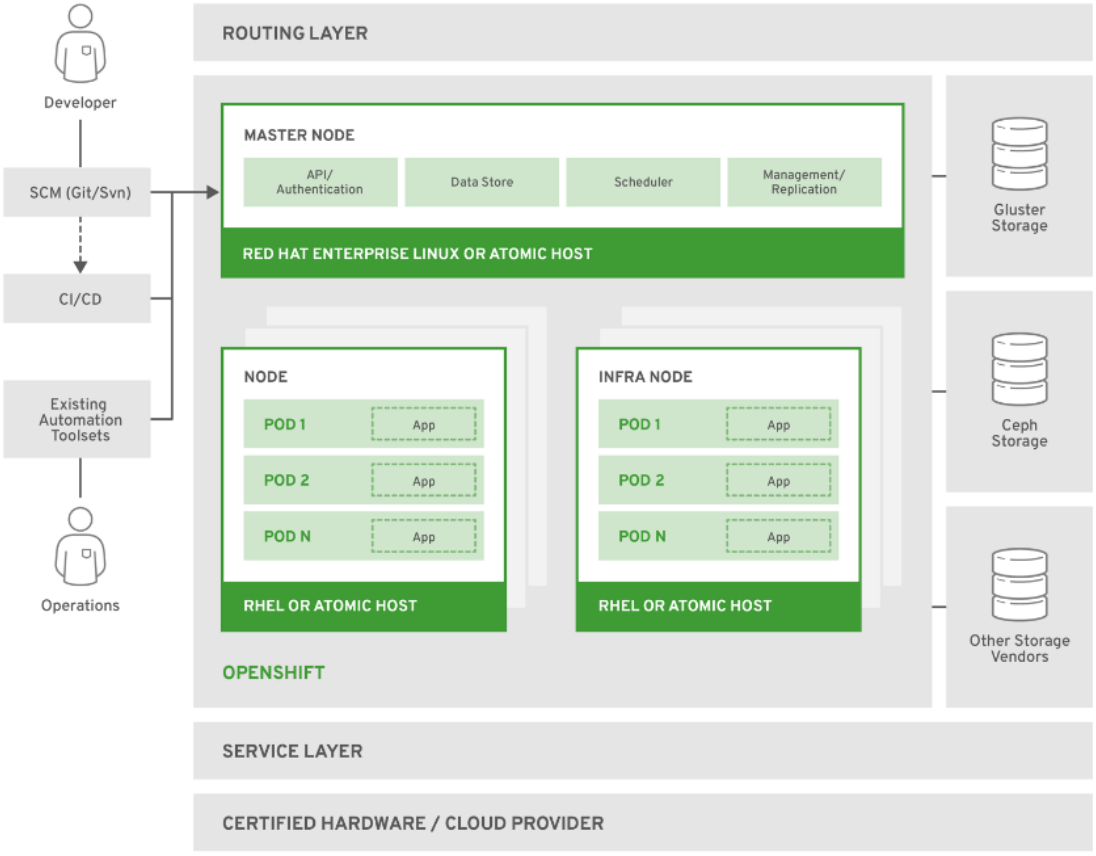
4.1 Yleistä

Openshift on PaaS (Platform as a Service- alusta, jonka Red Hat julkaisi vuonna 2011. PaaS-alustalla tarkoitetaan sovellusalustaa, jolla käyttäjät voivat ajaa ja asentaa ohjelmia, mutta käyttäjiltä on evätty pääsy ohjelmia ajavaan käyttöjärjestelmään. Openshiftin avulla ohjelmistokehittäjät pystyvät helposti ottamaan käyttöön sovelluksia ja mahdollistaa suuren konttimäärän helpomman hallinnan sekä automatisoinnin. Ohjelmistokehittäjien ei tarvitse enää manuaalisesti hallinnoida kontteja Openshiftissä. Openshift on yhdistelmä Docker kontteja ja Kuberneteksen orkesterointia, jossa molemmat on rakennettu RHEL (Red Hat Enterprise Linux) käyttöjärjestelmän päälle. Kun Openshiftiin pusketaan ohjelmistokehittäjän puolelta komentoja, aloittaa Openshift orkesteroimaan, että kuinka ja milloin sovellus käynnistyy. Red Hat tarjoaa Openshiftistä kolmea erilaista versiota; Openshift Online, Openshift Dedicated ja Openshift Container Platform. (Overview, n.d.)

4.2 Openshift Container Platform

OCP (Openshift Container Platform) on konttipohjainen ylläpitoympäristö, joka tuo yhteen Dockerin ja Kuberneteksen, ja tarjoaa ohjelmointiorajapinnan näiden palveluiden hallintaan. OCP mahdollistaa yksinkertaisen ja helpon konttien luomisen sekä hallinnoimisen. OCP: arkkitehtuuri on mikropalveluihin perustuva, jossa pienet ja irti kytketyt yksiköt toimivat yhdessä. OCP toimii Kubernetes klusterin päällä, jossa on tieto etcd:n sisään tallennetuista objekteista. Kyseiset palvelut jaetaan toimintojensa mukaan kahteen osaan: REST API:n, jotka paljastavat jokaisen ydinobjektin sekä ohjaimiin, jotka lukevat ohjelmointirajapintoja ja lisäävät muutoksia muihin objekteihin sekä raportoivat tilojen muutoksista. Käyttäjät voivat ottaa yhteyden REST API:n järjestelmän tilan muuttamiseksi ja ohjaimet käyttävät REST API:a lukemaan käyttäjän haluaman tilan ja pyrkivät tuomaan sen järjestelmään muiden osien synkronoitavaksi. Esimerkiksi, kun käyttäjä pyytää rakentamista (engl. build) tekemällä ’build’ ob-

jektin. Rakentamisohjain huomaa, että uusi rakennelma on luotu, jolloin ohjain suorittaa prosessin klusterilla ja viimeistelee rakennelman. Rakennelman valmistuttua ohjain päivittää objektin REST API:n kautta, jolloin käyttäjä saa tiedon, että rakennelma on valmis. Ohjaimet vastaavat järjestelmän logiikasta ottamalla vastaan käyttäjän toimintoja ja muuttamalla ne todellisiksi. Ohjaimia mukauttamalla voidaan toteuttaa erilaisia käyttäytymismalleja omien tarpeiden mukaisesti. Järjestelmänhallinnan näkökulmasta voidaan ohjelmointirajapintaa käyttää komentamaan hallinnollisia toimintoja automaattisesti toistuvalla aikataululla. OCP:n arkkitehtuuri on kuvattu kuviossa 15. (Overview, n.d.)



Kuvio 15. Openshift Container Platform -arkkitehtuuri. (Overview, n.d.)

Openshift Container Platform ja Kuberneteksen ohjelmointirajapinta todentavat käyttäjät, jotka esittävät tarpeelliset tunnistetiedot, jonka jälkeen käyttäjät valtuutetaan heidän rooliensa perusteella. Ohjelmistokehittäjät ja ylläpitäjät voidaan todentaa useilla erikeinoilla, mutta pääasiassa OAuth-tunnisteilla ja X.509 asiakassertifikaateilla. OAuth-tunnisteet allekirjoitetaan JSON Web-algoritmi RS256:lla, joka on RSA:n allekirjoitusalgoritmi SHA-256 salauksella. Ohjelmistokehittäjät tekevät tyypillisesti REST API-pyynnöt asiakasohjelmistosta tai verkkoselaimen kautta, jolloin he käyttävät myös OAuth kantotunnistetta (engl. bearer token) kommunikointiin. (Overview, n.d.)

Lupia käsitellään OCP:n käytännemoottorilla (engl. policy engine), joka määrittää toimintoja, kuten podin luomisen. Moottori ryhmittelee luvat ryhmittelemällä niitä rooleihin käytänne asiakirjoissa. Kun käyttäjä yrittää tehdä toimintoa, tarkistaa moottori käyttäjän roolit, ennen kuin sen annetaan jatkaa toimintojaan. (Overview, n.d.)

5 Palomuuri

5.1 Yleistä

Palomuuri on verkon turvallisuuslaite, joka hallinnoi sisään- ja ulostulevaa verkkoliikennettä. Palomuuri päättää sille ennalta määritellyin säännöin, että salliiko se liikenteen vai estääkö se liikenteen. Määritettyjä sääntöjä voivat olla esimerkiksi paketin osoite, protokolla tai portit. Palomuuri vertaa paketin saapuessa sen tietoja ja vertaa niitä olemassa oleviin sääntöihin, joiden perusteella se päättää paketin kohtalosta. Erikaltaiset palomuurit voivat toimia eri OSI-mallin kerroksissa, mutta kaikki palomuurit kykenevät suodattamaan liikennettä minimissään IP-osoitteiden perusteella. Palomuurit jaetaan karkeasti kahteen päätyyppiin; tilattomaan (engl. staless) ja tilalliseen (engl. stateful). Kyseiset kaksi eri tilaa eroavat keskenään tavoissa tutkia paketteja ja liikennettä. (What is firewall?. N.d.)

5.2 Tilaton palomuuri

Tilaton palomuuri vahtii verkon liikennettä ja joko sallii tai estää paketit sen lähteen, päämäärän tai jonkin muun muuttumattoman arvon perusteella. Tilaton palomuuri käyttää yksinkertaisia sääntöjoukkoja, jotka eivät ota huomioon sitä mahdollisuutta, että tuleva paketti esittää olevansa jotain mitä palomuuri sallii. Kyseisen palomuuri-tyypin suodatus ei dynaamisesti tarkasta liikennettä, vaan se arvioi paketin sisällön staattisesti ja ei pidä verkkoliikenteen tilasta minkäänlaista tietoa. Tilattoman palomuurin suodatuksen päätoiminta on parantaa turvallisuutta pakettien suodatuksella. Pakettien suodatuksen avulla voidaan tarkastella sisään- ja ulostulevia paketteja ja niiden perusteella tehdä tarvittavat toiminnot paketeille. Suodatuksen tyypillisin käyttökohde on suojella reititys moottorin (engl. Routing Engine) prosesseja ja resursseja mahdollisilta hyökkäyksiltä tai tuntemattomilta paketeilta (Mojidra, N. 2016).

5.3 Tilallinen palomuuri

Tilallinen palomuuri vahtii verkon liikennettä tilattoman palomuurin tavoin, mutta se myös kykenee esimerkiksi muistamaan, että missä vaiheessa TCP (Transmission Control Protocol) -yhteyden muodostus on tai onko MTU (Maximum Transmission Unit) muuttunut sekä havaitsemaan rikkoutuneen paketin. Tilallisuus nopeuttaa palomuurin toimintoja ja vähentää sen tarvitsemia resursseja, koska palomuurin ei enää tarvitse käsitellä aivan kaikkia sen läpi kulkevia paketteja. Palomuuri kykenee esimerkiksi UDP (User Datagram Protocol) -protokollaa käyttäessä tekemään omaan yhteystauluunsa merkinnän, jotta se tietää minkä kahden laitteen välillä UPD-liikennettä on tunnistettu ja se voi jatkossa tarvittaessa sallia kyseisen liikenteen automaattisesti. (Mojidra, N. 2016.)

Tilallinen palomuuri ei tarvitse pitkiä konfigurointeja, koska se kykenee muistamaan jokaisen sen läpi tulevan liikenteen tilan. Toiminnaltaan se ensiksi vertaa liikennettä olemassa oleviin sääntöihin ja mikäli paketti sallitaan, alkaa palomuurin dynaaminen prosessi. Esimerkiksi TCP-yhteydessä tilallinen palomuuri kykenee muistamaan TCP-yhteyden sen hetkisen tilan, koska myös TCP-yhteys on toiminnaltaan tilallinen. TCP-yhteyden kolmisuuntakättelyn (engl. three way handshake) alkaessa alkuperäinen

pyynnön lähettäjä lähettää SYN-paketin, joka tallentuu automaattisesti palomuurin tilataulukkoon. Mikäli kohde vastaa SYN-ACK-paketin yhteyden aloittamisesta, palomuurin tilataulukko heijastaa sen itselleen. Viimeiseksi, kun alkuperäinen yhteyden muodostaja vastaa viimeisellä ACK-paketilla, muuttuu palomuurin tilataulukkoon tilaksi vahvistunut (engl. established). Kyseinen tila säilyy taulukossa niin pitkään, kunnes yhteys katkaistaan tavalla tai toisella. (Wilkins, S. 2013.)

5.4 Web-sovellustason palomuuri

Web-sovellustason palomuuri (engl. Web application firewall) toimii nimensä mukaisesti OSI-mallin ylimmällä kerroksella ja sen päätavoitteena on suojata web-sovelluksia. Web-sovellustason palomuuri. suorittaa syvällistä pakettien tarkastelua verkkoliikenteessä palvelimen ja asiakkaan (engl. client) välillä. Palvelimen ja asiakkaan välistä liikennettä tutkimalla palomuuri kykenee tunnistamaan mahdolliset hyökkäykset, vaikka sovelluksesta itsestään puuttuisikin mahdollisuus havaitsemiseen. Web-sovellustason palomuureja käytetään lähinnä HTTP-protokollan kanssa tapahtuvassa liikenteessä ja niillä ei ole tarkoitus korvata muita perinteisiä palomuuriratkaisuita (Clincy & Shahriar. 2018).

Web-sovellustason palomuuri voi olla kahta eri mallista käytänteisiin (engl. policies) perustuvaa turvallisuusmallia: positiivinen ja negatiivinen. Positiivinen turvallisuusmalli sallii ainoastaan käytänteisiin sopivan liikenteen läpi pääsyn ja kaikki muut liikenne on estetty. Negatiivinen malli sallii kaiken liikenteen läpi pääsyn ja yrittää estää ainoastaan liikenteen, joka on merkitty säännöillä vihamieliseksi. Useat palomuurit käyttävät joko positiivisia tai negatiivisia sääntöjä ja todella harvoin molempia (Clincy & Shahriar. 2018).

5.5 Seuraavan sukupolven palomuuri

Seuraavan sukupolven palomuurit (engl. next-generation firewall) ovat osa kolmatta palomuuriteknologian sukupolvea, joka kehitettiin havaitsemaan ja estämään nykypäivän monimutkaisempia hyökkäyksiä, kuten AET (Advanced Evasion Technique) -hyökkäykset, joissa pyritään hyökkäämään useilla kerroksilla samanaikaisesti peittä-

mällä haitalliset kuormat, jakamalla ne pienempiin kehyksiin harvoin käytettyjen protokollien avulla. Seuraavan sukupolven palomuureilla pyritään estämään hyökkäykset ennen niiden pääsyä sisäverkkoon. (Neupane, J., Haddad, R. & Chen, L, 2018.)

Tärkein ominaisuus seuraavan sukupolven palomuureissa on pakettien syvätarkastelu (engl. deep packet inspection) -tekniikka, jota voidaan käyttää SSL (Secure Sockets Layer) -liikenteen purkamiseen ja tarkasteluun sisään -ja ulosmenevässä liikenteessä, jolloin palomuuri pystyy havaitsemaan ja estämään SSL:n kautta tulevat hyökkäykset. Perinteinen palomuuri ei kykene purkamaan ja tarkastelemaan SSL-liikennettä, jolloin hyökkääjät voivat tehdä SSL-tunneleita sisältä ulospäin. Seuravan sukupolven palomuurien edistyneitä ominaisuuksia ovat:

- Sovellus tietoisuus (engl. application awareness).
- Integroitu IPS (Intrusion Prevention System)
- Identiteetti tietoisuus eli käyttäjien ja ryhmien hallinta
- Siltaus -ja reititystilat.
- Pakettien syvätarkastelu

SoutheastCon konferenssissa vuonna 2018 pidetty esitys seuraavan sukupolven ja perinteisen palomuurin eroista toi hyvin esiin tärkeimmät erot ja niiden vaikutukset nykypäivän tarpeisiin. Kuviossa 16 on konferenssissa esitetyt erot perinteisen palomuurin ja seuraavan sukupolven palomuurin välillä. (Chen, L & ym. 2018)

Goals	Traditional Firewall	Next Generation Firewall
Prevent Advanced Persistent Attacks	• Only part of network security supplemented with IPS, URL filtering, gateway antimalware-malware products • Separately managing security tools is expensive	• Offer complete set of security technologies in one package • Combine all features of traditional firewall • Integrated package is easy to install, configure, deploy and manage as a unit which reduces administrative cost
Inspect SSL Traffic	• Cannot decrypt and inspect SSL traffic • Attacker can create SSL tunnels inside out to exchange command and control message	• Use Deep Packet Inspection technology to decrypt and inspect SSL traffic in both inbound and outbound direction • Detect and block botnet command and control message • Prevent advanced persistent threats using SSL
Control Web Applications	• Not application aware • Application control is a serious deficiency	• Offer application intelligence and control • Recognize specific application • Provide chart to visualize and control traffic by application
Manage Users & Use Policy	• No correlation of network traffic with users	• Allow application control at user group and individual level • Impose acceptable policies at high level of granularity • Allow to identify traffic by user and user group who pose security threats or involuntarily affect productivity through traffic visualization
Trade off Security vs Performance	• Administrator turn off monitor on specific ports, disable firewall rule and limit deep packet inspection which affect performance	• Parallel processing hardware architecture • Apply efficient approaches

Kuvio 16. Perinteisen ja seuraavan sukupolven palomuurin erot. (Chen, L & ym. 2018.)

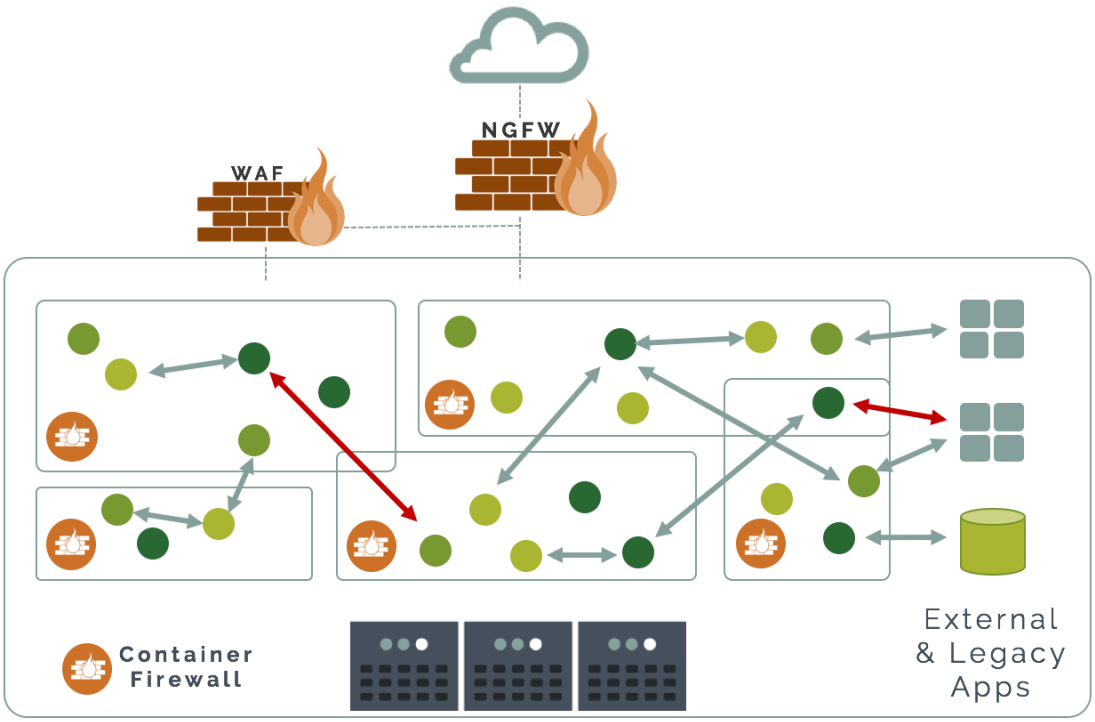
6 Konttipalomuuri

6.1 Yleistä

Konttipalomuuri on kontitettu versio palomuurista. Konttipalomuurien tarkoituksena ei ole korvata olemassa olevia suojausmenetelmiä, kuten web-sovellustason palomuureja tai seuraavan sukupolven palomuureja, koska konttipalomuurien päätehtävä on suojata konttien liikennettä. Konttipalomuurit tarjoavat samankaltaista suojaa kuin seuraavan sukupolven palomuurit, mutta konttiympäristön sisällä. Konttipalomuureilla voidaan hallita useita eri tyyppisiä liikenteitä, kuten north-south -ja east-west liikennettä, sekä kontilta kontille ja kontilta muualle menevää liikennettä. North-south tarkoittaa liikennettä palvelimen ja asiakasohjelman (engl. client) välillä. East-west liikenne taas tarkoittaa kahden palvelimen välistä liikennettä. (Duan, G. & Huang, F., n.d.)

Konttipalomuuri tarjoaa konttiympäristöille paljon enemmän, kuin web-sovellustason palomuurit tai seuraavan sukupolven palomuurit, koska konttipalomuurien avulla voidaan monitoroida isännän (engl. host) ja kontin aktiivisuutta konttien verkon käyttäytymisen ohella. Tämän ansiosta mahdollisia hyökkäyksiä voidaan paremmin ennaltaehkäistä ja havaita. (Duan, G. & Huang, F., n.d.)

Konttipalomuuraus mahdollistaa nopean palomuurin nopean käyttöönottoon ja turvallisuuspalveluiden skaalautumisen dynaamisiin tarpeisiin. Konttipalomuurit ovat myös huomattavasti kevyempiä ja nopeampia käynnistää konttitekniikan vuoksi, joka tuo huomattavasti pienemmän jalanjäljen ja helpottaa mahdollista palomuurin tarpeellista siirtämistä. Useimmat konttipalomuurit ovat parhaimmillaan yksityisessä, -julkisessa -tai sekamuotoisessa pilviympäristössä. Konttipalomuurien käyttö olemassa olevien suojausmenetelmien lisäksi luo erittäin hyvän suojauksen hyökkäyksiä vastaan. Kuviossa 17 on seuraavan sukupolven palomuuri tarjoamassa moniprotokolla tarkastelua ja suojausta epäluotettavilta verkoilta, web-sovellustason palomuuri suojaamassa web front-end sovelluksia ulkoisilta hyökkäyksiltä ja konttipalomuuri suojaamassa sisäistä konttiliikennettä. (Duan, G. & Huang, F., n.d.)



Kuvio 17. Palomuurien yhteiskäyttö. (Duan, G., n.d.)

6.2 Konttipalomuurin ja seuraavan sukupolven palomuurin erot

Seuraavan sukupolven palomuurien oli tarkoitus käsitellä tuoreimmat uhkat, mutta uudet pilvipalveluiden mikropalveluympäristöt saattavat olla niille liikaa. Konttipohjaiset mikropalvelut ovat tuoneet räjähdysmäisesti lisää East-West- ja sisäistä isännältä isännälle -liikennettä. Kontteja käynnistetään ja sammutetaan jatkuvasti, sekä päivitysten käyttöönotto sovelluksissa ja jatkuva integrointi tuovat mukanaan suuria haasteita monitorointiin ja konttiliikenteen suojaamiseen verkkotasolla. Perinteisten palomuurien on likimain mahdotonta pitää silmällä konttien muutoksia, koska kontit käynnistyvät ja sammuvat jatkuvasti. Konttitekniikan kanssa manuaalinen palomuurisääntöjen muokkaaminen on todella vaikeaa ja tuskastuttavaa, jonka konttipalomuurien on luvattu korjaavan. (Duan, G. N.d.).

Konttien ajon aikainen monitorointi on tärkeää, koska konttien rakentamisessa käytetään säännöllisesti avoimen lähdekoodien ohjelmistoja ja ohjelmistokehittäjät eivät välttämättä aina saata ymmärtää sovelluksen haavoittuvuuksia avoimen lähdekoodin paketissa tai käytetyissä kirjastoissa. Tuotannossa pystyy myös helposti unohtumaan, että mitkä konteista ovat haavoittuvaisia uusille löydetyille haavoittuvuuksille, koska

kaikki kontteja ei jatkuvasti päivitetä. Konttipalomuurin ehdoton etu onkin konttitason suojaus, jolla voidaan pudottaa epäilyttävät yhteydet tai laittaa koko kontti karanteeniin tarvittaessa sekä palomuurin integroiminen konttien orkesterointiin. (Duan, G. N.d.)

6.3 Konttipalomuurin ja web-sovellustason palomuurin erot

Sovelluskontit tarjoavat tehokkaan tavan ottaa käyttöön ja hallita sovelluksia, mukaan lukien web-pohjaisia, mutta kontittamisen yhteydessä sovellusten turvaaminen on tullut entistä haastavammaksi. Konttipalomuurilla ei pyritä korvaamaan web-sovellustason palomuureja. Web sovellustason palomuurit on suunniteltu suojaamaan ulkoista pääsyä web-pohjaisiin sovelluksiin, kun taas konttipalomuurit pystyvät suojaamaan kaikkia sisäisiä east-west -liikenteitä konttien välillä ja konttipalomuurit sisältävät myös joitain web-sovellustason palomuurien ominaisuuksia. Yhtenäisistä sovelluksista mikropalveluihin siirtyminen luo erilaisia verkkotopologioita ja uusia turvallisuusuhkia. Web-sovellustason palomuurien on vaikeampaa havaita sisäistä east-west -liikennettä isäntäkoneessa ja isäntien välillä, varsinkin konttiympäristöissä, joissa muutokset ovat jatkuvia konttien käynnistyessä ja katoamisessa. (Duan, G., n.d.)

Kuten aikaisemminkin mainittiin, konttipalomuuri kykenee tarkastelemaan ja suojaamaan konteille tulevaa -ja menevää liikennettä sekä se kykenee myös suojaamaan sovellusten pinoja (engl. stacks) ja palveluja. Konttipalomuuri pyrkii myös suojaamaan sisäistä -ja ulkoista liikennettä ulkoisista verkoista sekä olemassa olevista sovelluksista kontteihin. Toisinkuin web-sovellustason palomuuri suojelee web-pohjaista asiakkaiden pääsyä front-end sovellukseen. (Duan, G., n.d.)

7 Juniper cSRX konttipalomuuri

7.1 Yleistä

Juniper cSRX on kaupallinen Juniperin kehittämä konttipalomuuri tuote. cSRX on kontitettu versio Juniperin SRX sarjan seuraavan sukupolven palomuuri tuotteesta ja se on rakennettu Juniperin tuotteille tutun Junos (Juniper Network Operating System)

käyttöjärjestelmän päälle. Junos on FreeBSD-pohjainen käyttöjärjestelmä, jota käytetään Juniper Networksin valmistamissa verkkolaitteissa. CSRX konttipalomuuri tarjoaa suurimman osan samoista ominaisuuksista kuin Juniperin seuraavan polven palomuurit. Tarkemmat cSRX:stä löytyvät seuraavan sukupolven palomuurin ominaisudeet on listattu taulukossa 1. (Understanding cSRX with a Bare-Metal Linux Server, 2018).

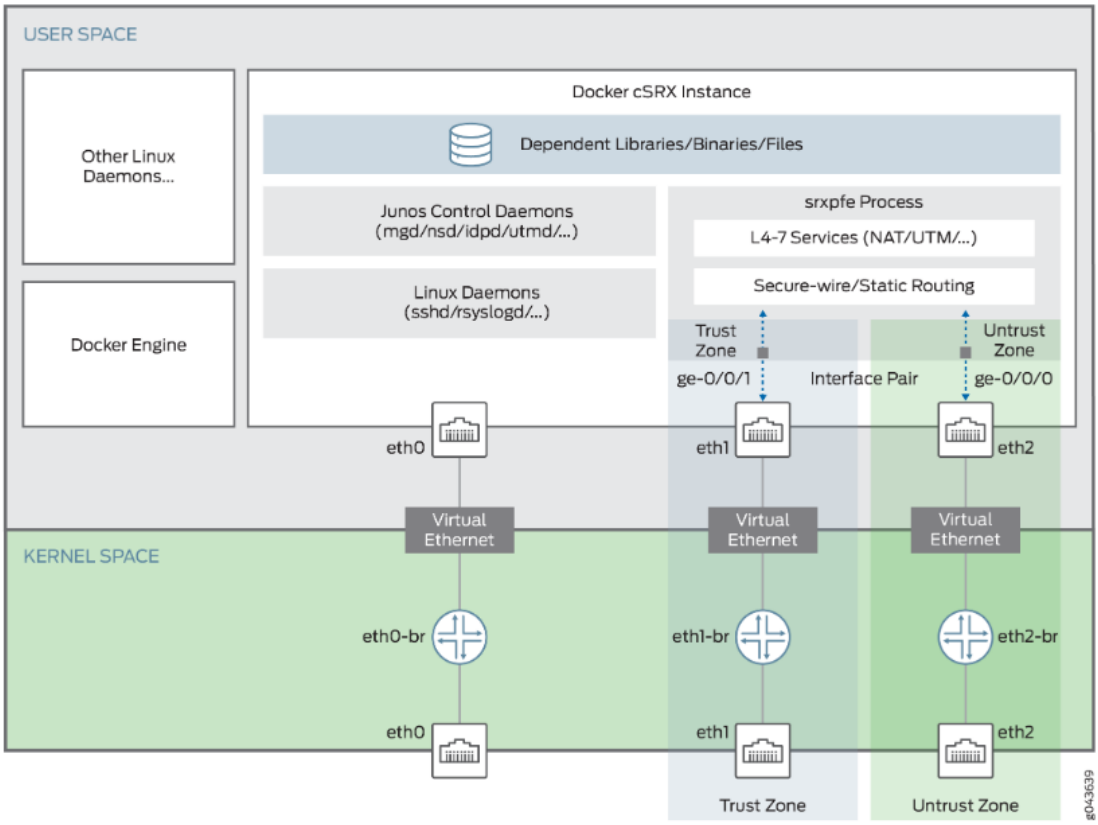
Taulukko 1. cSRX-konttipalomuurin ominaisuudet (Understanding cSRX with a Bare-Metal Linux Server, 2018).

SRX-sarjan tuetut ominaisuudet cSRX:ssä
Application Firewall
Application Identification
Application Tracking
Basic firewall policy
Brute force attack migitation
Central management
DDoS protection
DoS protection
Interfaces (network & revenue)
Intrusion Detection and Prevention (IPS)
IPv4 & IPv6
Jumbo frames
Malformed packet protection
Network Address Translation (NAT)
Routing
SYN cookie protection
User firewall
Unified Threat Management (UTM)
Zones & zone-based IP-spoofing

Juniperin cSRX konttipalomuuri pyörii yhdessä kontissa fyysisellä Linux palvelimella ja se käyttää fyysistä palvelinta isäntä alustana Dockerin konttiympäristölle. CSRX kontti pakkaa kaikki sille riippuvaiset prosessit ja kirjastot tukeakseen erilaisia Linux isäntäkoneen jakeluita, kuten Ubuntua, Red Hatin kaupallisia jakeluita ja CentOs:a. Itse konttia hallitaan normaaleilla Docker komennoilla, mutta konttipalomuurin konfigurointi tapahtuu Juniperin Junos käyttöjärjestelmän komennoilla komentoriviltä. (Understanding cSRX with a Bare-Metal Linux Server, 2018)

Kun konttipalomuuri käynnistetään ja se tulee aktiiviseksi, käynnistyy Docker kontin sisällä useita palveluprosesseja automaattisesti. Jotkin näistä palveluprosesseista tukevat Linuxin toimintoja, tarjoten saman palvelun kuin Linux isännällä ajettaessa. Tällaisia palveluprosesseja ovat mm. sshd, rsyslogd ja monit. Muut palveluprosessit ovat käännetty ja siirretty Junos OS:sta, jotta ne voivat suorittaa konfiguraatiot ja hallinnointi tehtävät turvallisuuspalveluilla. Tämänkaltaisia palveluprosesseja ovat mm. MGD, NSD, UTM, IDP ja AppID. CSRX käyttää srxpfe prosessia edelleenlähetyksen (engl. forwarding) funktioihin Layer 2:lta Layer 3:lle, kuten secure-wire edelleenlähetystä ja staattisen reitityksen edelleenlähetystä. Srxpfe prosessia käytetään cSRX:ssä myös verkon turvallisuuspalveluissa Layer 4:lta Layer 7:lle (Understanding cSRX with a Bare-Metal Linux Server, 2018).

Konttipalomuuri mahdollistaa edistyneen turvauksen monihaltijan (engl. multitenant) virtualisoidun ympäristön verkon reunalla (engl. edge) tarjoamalla Layer 4:lta Layer 7:lle ulottuvia edistyneitä turvallisuus ominaisuuksia, kuten IPS (Intrusion Prevention System) -ja AppSecure ominaisuudet. CSRX konttipalomuuri tarjoaa kaksi tuloporttia liitettyinä trusted- ja untrusted -alueisiin ja myös yhden ylimääräisen rajapinnan konttipalomuurin hallinnointiin. Kun cSRX toimii Layer 2:lla secure-wire tilassa, tulevat Layer 2 kehykset yhdeltä rajapinnalta menevät läpi Layer 4:lta läpi Layer 7:n prosessien riippuen konfiguroiduista palveluista. Tämän jälkeen cSRX lähettää kehykset ulospäin toisesta rajapinnasta ja joko päästää kehykset läpi tai pudottaa ne, riippuen turvallisuuspolitiikasta. Kuviossa 18 on esimerkki näkymä cSRX kontin instanssista, jossa on nähtävissä, kuinka kontti on sillattu ulkopuoliseen verkon kanssa ja eth1 sekä eth2 on sillattu isännän fyysisiin verkkorajapintakontrollereihin. Docker konfiguraatioissa täytyy siis erikseen määrittää, että cSRX kontti yhdistetään 3 virtuaaliseen verkkoon; yksi virtuaalinen verkko hallintasessioille ja kaksi muuta vastaanottamaan ja lähettämään in-band liikennettä. (Understanding cSRX with a Bare-Metal Linux Server, 2018.)



Kuvio 18. cSRX-kontin yleiskuva (Understanding cSRX with a Bare-Metal Linux Server, 2018).

7.2 Juniper cSRX:n hyödyt

Juniperin konttipalomuuri mahdollistaa nopean käyttöönoton uusille palomuuripalveluille ja se pystyy skaalaamaan turvallisuuspalvelut dynaamisten tarpeiden mukaisesti. Konttipalomuuri eroaa virtuaalikoneista siten, että se ei tarvitse vieras käyttöjärjestelmää, se käyttää huomattavasti vähemmän muistia, sillä on huomattavan pieni jalanjälki (engl. footprint) ja se on helpompi siirtää tai vaihtoehtoisesti ladata. Ja tietysti sen uudelleenkäynnistyksen aika on vain muutamia sekunteja, verrattaen virtuaalikoneen useisiin minuutteihin. Ideaalisimmat käyttökohteet cSRX:lle ovat yksityiset- ja julkiset pilvet, sekä hybrid pilviympäristöt. (Understanding cSRX with a Bare-Metal Linux Server, 2018.)

CSRX konttipalomuurin tärkeimpiä ominaisuuksia ovat; tilallinen palomuurisuojaus haltijan (engl. tenant) reunalla, konttipalomuurin nopea käyttöönotto, skaalautuu helposti pysyen mukana asiakkaiden tarpeissa, sovelluksen turvallisuusominaisuudet

(IPS ja AppSecure), UTM (Unified Threat Management) ominaisuudet, kuten antispam, Sophos Antivirus, web- ja sisältösuodatus (Understanding cSRX with a Bare-Metal Linux Server, 2018).

7.3 Juniper cSRX:n ympäristömuuttujat

Docker mahdollistaa ympäristömuuttujien tallentamisen konfiguraatioasetuksiin. Ajon aikana ympäristömuuttujat paljastetaan sovellukselle kontin sisällä. Kontin käynnistyksen yhteydessä voidaan määrittää cSRX:lle useita eri muuttujia -e parametrin avulla. Taulukossa 2 on lueteltu kaikki ympäristömuuttujat, joita cSRX konttipalomuurille voi määrittää. (cSRX Environment Variables Overview, 2018.)

Taulukko 2. cSRX-kontin ympäristömuuttujat

Muuttuja	Arvo	Oletusarvo
CSRX_SIZE	"small" / "middle" / "large"	"large"
CSRX_FORWARD_MODE	"routing" / "wire"	"routing"
CSRZ_PACKET_DRIVER	"poll" / "interrupt"	"poll"
CSRX_ROOT_PASSWORD	oma arvo	ei salasanaa
CSRX_CTRL_CPU	hexa-arvo	ei arvoa
CSRX_DATA_CPU	hexa-arvo	ei arvoa
CSRX_ARP_TIMEOUT	desimaalinen arvo	sama kuin Linux isännällä
CSRX_NDP_TIMEOUT	desimaalinen arvo	sama kuin Linux isännällä

7.3.1 CSRX_SIZE

Konttipalomuurin kokoa pystyy muuttamaan omien tarpeiden mukaisesti kolmeen eri kokoon; pieneen, keskikokoiseen tai suureen. Jokaisella cSRX:n koolla on tietyntyyppiset ominaisarvot ja ne ovat sovellettavissa erilaisissa käyttötarpeissa. Oletuksena cSRX kontti käynnistyy käyttäen suurta kokoa. Kuviossa 19 on vertailtu muuttujan avulla määritettäviä kontin kokoja. (Changing the Size of a cSRX Container, 2018.)

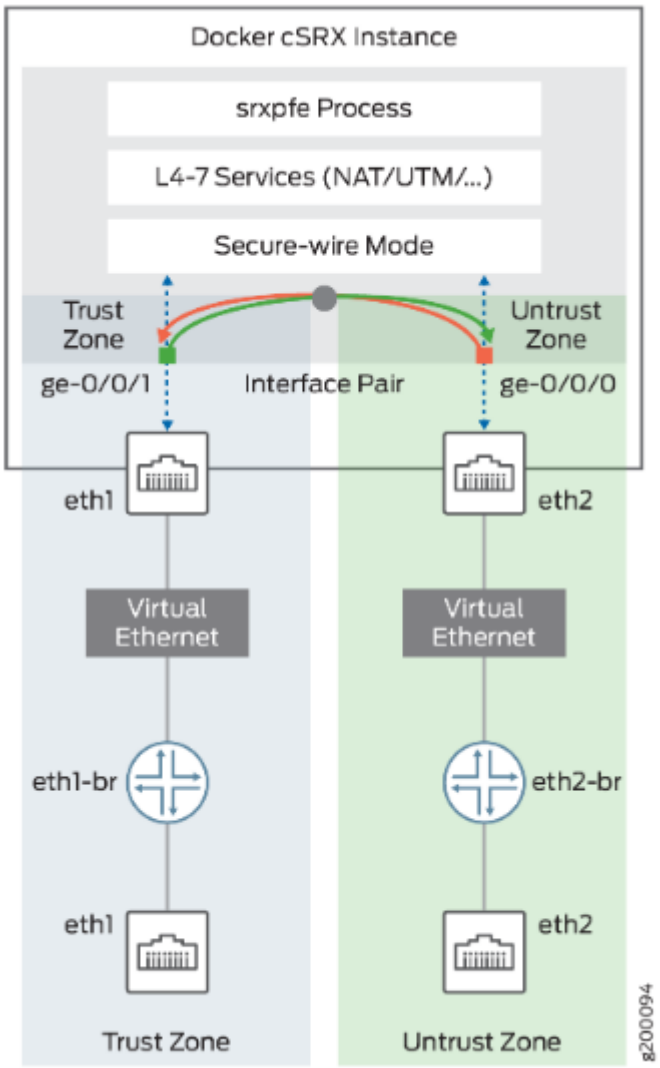
Specification	cSRX: Small Size	cSRX: Middle Size	cSRX: Large Size (Default)
Physical Memory Overhead	256M	1G	4G
Number of Flow Sessions	8K	64K	512K

Kuvio 19. cSRX-kontin kokojen vertailu. (Changing the Size of a cSRX Container, 2018.)

7.3.2 CSRX_FORWARD_MODE

CSRX konttipalomuurin liikenteen edelleen lähetystilan pystyy määrittelemään kahdella eri tavalla; staattisesti reitittäväksi tai secure-wire tilalla. Tästä on hyötyä esimerkiksi silloin, jos halutaan välttää virtuaalisen verkkotopologian muutoksia tai kontille halutaan määrätyn lainen next-hop osoite ulosmenevään liikenteeseen. Staattisesti reitittävänä cSRX käyttää staattisia reittejä liikenteen edelleen lähetykseen reiteille, jotka on tarkoitettu rajapinnoilla ge-0/0/0 ja ge-0/0/1, jolloin täytyy erikseen konfiguroida staattinen reitti ja määritellä next-hop osoite. (Configuring Traffic Forwarding on a cSRX Container, 2018.)

Secure-wire tilassa kaikki liikenne, mikä saapuu tiettyyn rajapintaan eli ge-0/0/0 tai ge-0/0/1 edelleen lähetetään muuttumattomana rajapinnan läpi. Secure-wire mahdollistaa cSRX:n käyttöönoton ilman muutoksia reititystauluihin tai naapurilaitteiden uudelleen konfigurointia. Rajapintojen ge-0/0/0 ja ge-0/0/1 välille asetetaan ns- ristikytkentä, jolloin voidaan ohjata liikenne portista toiseen riippuen IIF (Interworking and Interoperability Function) syöttötietojen avaimesta. CSRX muodostaa automaattisesti oletus secure-wiren nimeltään csrx_sw srxpfe prosessiin, johon myös ge-rajapintapari lisätään. Tämän jälkeen voi itse määrittää turvallisuuspolitiikat, jotka on pakotettu trust- ja untrust -alueiden välille. Kuviossa 20 on kuvailtu secure-wire tilan toiminta. (Configuring Traffic Forwarding on a cSRX Container, 2018.)



Kuvio 20. cSRX secure-wire-tila (Configuring Traffic Forwarding on a cSRX Container, 2018).

7.3.3 CSRX_PACKET_DRIVER

CSRX kontti vaihtaa (engl. exchange) paketteja käyttäen Linux-isännän VETH (Virtual Ethernet)-rajapintaa. I/O pakettiajurin asetukset vaikuttavat edelleen lähetyksen tehokkuuteen ja skaalautuvuuteen. CSRX kontin käynnistyksessä voi valita kahdesta eri ajurista; poll mode -ajurista, joka on oletuksena tai interrupt mode -ajurista. Nämä määrittävät, kuinka paketteja vaihdetaan. Poll mode -ajuria suositaan käytettäväksi silloin, kun cSRX halutaan tukevan VNF:ää (Virtualized network function) ja interrupt

mode -ajuria silloin, kun tarvitaan cSRX:n tuki suurelle määrälle samanaikaisia turvallisuuspalveluita. Taulukossa 3 on vertailultu ajureiden tehokkuutta ja skaalautuvuutta. (Specifying the packet I/O Driver for a cSRX container, 2018.)

Taulukko 3. I/O-ajureiden vertailu

Määrittely	**Poll mode -ajuri**	**Interrupt mode -ajuri**
Tehokkuus	Korkeampi uudelleen lähetys tehokkuus	Matalampi uudelleen lähetys tehokkuus
Skaalautuvuus	Vähentää skaalautuvuutta. Tukee yhtä cSRX konttia per vCPU	Paranneltu skaalatauvuus. Tukee useita cSRX kontteja per vCPU

Poll mode -ajuri käyttää PCAP:n (Packet Capture) pohjautuvaa DPDK (Data Plane Development Kit) -ajuria pakettien kyselyyn Linuxin VETH ajurilta. Paketteja vaihdetaan käyttäjän kernel tilan välillä käyttäen BPF:ää (Berkeley Packet Filte). Poll mode -ajurin avulla saavutetaan paras suorituskyky yhdelle cSRX kontille. Interrupt mode -ajuri ottaa vastaan ja lähettää paketteja käyttäen pakettisuoritinkantaa (engl. packet socket). Linux käyttöjärjestelmän epoll -mekanismin avulla interrupt mode -ajuri kykenee auttamaan srxpfe prosessia odottamaan siihen asti, kunnes paketti saapuu VETH-rajapinnalle. Mikäli tuloporteissa ei ole pakettikuormaa, pysyy srxpfe prosessi uni-tilassa säästääkseen suorittimen resursseja. Epoll-mekanismin avulla Linux-palvelin pystyy ylläpitämään suurta määrää cSRX instansseja, varsinkin tilanteissa, joissa on monta cSRX instanssia käytössä yhdellä suorittimella. (Specifying the packet I/O Driver for a cSRX container, 2018.)

7.3.4 CSRX_CTRL_CPU & CSRX_DATA_CPU

CSRX instanssi vaatii kaksi suorittimen ydintä Linux palvelimella. CSRX kontin hallinnointi -ja dataprosesseja pystyy muuttujien avulla määrittämään tietylle suorittimelle, joka auttaa ajoittamaan Linux palvelimen tehtäviä ja säätämään CSRX kontin suorituskykyä. Ympäristömuuttujat varmistavat, että cSRX:n hallinnointi -ja datatason palveluprosessit on sidottu määritettyyn fyysiseen suorittimeen, joka parantaa cSRX kontin suorituskykyä, koska suorittimen välimuistia (engl. cache) käytetään tehok-

kaasti. Oletuksena mitään arvoja ei määritetä ja palveluprosessit pyörivät suorittimella, jonka on määritetty Linux ytimen aikataulutuksessa (engl. scheduling). (Configuring CPU Affinity for a cSRX Container. 2018.)

7.3.5 CSRX_ARP_TIMEOUT & CSRX_NDP_TIMEOUT

Oletuksena ARP -ja NDP (Neighbor Discovery Protocol) -merkintöjen aikakatkaisu on määritetty 1200 sekuntiin. Aikakatkaisun arvoa pystyy muuttamaan ympäristömuuttujilla cSRX kontin käynnistymisen yhteydessä. ARP merkintöjen maksimi arvoa hallinnoi Linux isännän ydin. Mikäli tiedossa on useita naapureita, on ARP -ja NDP merkintöjen rajoitusta tarpeellista yleensä nostaa myös Linux isännällä. (Configuring Traffic Forwarding on a cSRX Container, 2018.)

7.3.6 Pysyvä lokitiedosto

CSRX konttipalomuurissa lokitiedostot tallennetaan oletuksena /var/log -hakemistoon. Oletuksena lokitiedostoja pidetään yllä ainoastaan, mikäli ulkoisia massamuistin sisältöä ei ole otettu käyttöjärjestelmän käyttöön /var/log -hakemistolle, joten cSRX kontin tuhoutuessa myös lokitiedostot hävitetään. Pysyvän lokitiedoston pystyy ottamaan käyttöön kontin käynnistymisen yhteydessä komennolla ”v <polku>:/var/log”. (Enabling Persistent Log File Storage to a Linux Host Directory, 2018.)

8 Toteutus

8.1 Toteutuksen vaiheet

Toteutus jakautuu kolmeen eri vaiheeseen. Ensimmäisessä vaiheessa viedään Juniperilta saatu konttipalomuurin levykuva yrityksen yksityiseen rekisteriin, josta se voidaan helposti ottaa käyttöön ja testataan levykuvan toimintaa virtuaalikoneella Dockerilla ennen kuin sitä testataan muissa ympäristöissä. Toteutuksen toisessa vaiheessa konttipalomuuria yritetään saada toimivaksi Openshift Container Platformilla,

jonka oli tarkoituksena olla ainut testiympäristö. Toteutuksessa ilmaantuneiden ongelmien vuoksi toimeksiantajan päätöksellä siirryttiin Docker Swarm -ympäristöön, jossa konttipalomuuri pyrittiin saamaan käyttöön Docker Composen avulla. Toteutuksen kolmannessa tutustaan konttipalomuurin lokitiedostoihin, jonka tavoitteena oli saada lokitiedostot pysyväisiksi ja testata lokitiedostojen reaaliaikasuutta. Kolmannen vaiheen oli alun perin tarkoitus olla testausvaihe, mutta tuotteen yhteensopivuusongelmien vuoksi testausta ei voitu halutussa ympäristössä toteuttaa.

8.2 CSRX -konttipalomuurin käynnistäminen Dockerissa

Toteutus alkoi levykuvan viemisellä rekisteriin, jotta se saadaan helposti käyttöön kaikissa halutuissa ympäristöissä ja sijainneissa. Juniperin omaa rekisteriä ei voitu käyttöoikeuksien vuoksi käyttää, joten yrityksessä käytössä oleva yksityinen rekisteri valikoitui parhaaksi vaihtoehdoksi, koska sieltä voidaan hakea levykuvat helposti myös Openshift Container Platformiin. Ensimmäiseksi kokeiltiin levykuvan viemistä yksityiseen rekisteriin Dockerin import-toiminnolla, mutta siinä ilmeni ongelmaksi, että Dockerin import-komento poisti levykuvasta entrypointin, minkä vuoksi konttia ei saatu käynnistymään edes perus-Dockerilla virtuaalikoneella, jonka vuoksi import-toiminnolla viety levykuva päätettiin poistaa. Toisena keinona oli hakea levykuva docker load -komennolla, jotta entrypoint pysyisi entisellään, jonka jälkeen se olisi mahdollista puskea rekisteriin. Tällä keinolla levykuva toimikin moitteetta.

Ennen cSRX kontin käynnistämistä Dockeriin täytyi luoda Linux siltaverkkorajapinnat (engl. bridge networks). Ensimmäisenä toimenpiteenä tehtiin kolmisilta verkko cSRX kontille, johon kuuluu mgt_bridge (eth0), left_bridge (eth1) ja right_bridge (eth2). Konttipalomuuri käyttää mgt_bridgeä ”out-of-band” hallintaan, jonka avulla voidaan hyväksyä hallintaistuntoja ja liikennettä. Left_bridgeä ja right_bridgeä käytetään tulo portteina, joilla prosessoidaan in-band liikennettä. CSRX:n vaatimat trusted ja untrusted rajapinnat liittyvät Linux siltaan eth1 ja eth2 rajapinnoista. Docker liittää eth0 hallintarajapinnan automaattisesti Linux siltaan ja määrittää sille IP-osoitteen. Kuviossa 21 on luotu cSRX -konttipalomuurille tarpeelliset verkkorajapinnat ja tarkastettu, että ne on luotu oikein.

```
root@                :/# docker network create --driver bridge mgt_bridge
1801002c7aefc913ddadc8e85383ae48ba30f23d4e3dc8e307f45cc703b2fba4
root@                :/# docker network create --driver bridge left_bridge
3223a291a64baf292ad6f88313e54d4cacb4b6c4657f6400ba66399fb851ba71
root@                /# docker network create --driver bridge right_bridge
698ad49c05131f401b1eb7aac2cb317dd790997ae8a206088f4d4b3ee42a1b24
root@                /# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
bb001f8cf338        bridge              bridge              local
5b8750aacc32        host                host                local
3223a291a64b        left_bridge         bridge              local
1801002c7aef        mgt_bridge          bridge              local
cde0cc06c4f4        none                null                local
698ad49c0513        right_bridge        bridge              local
```

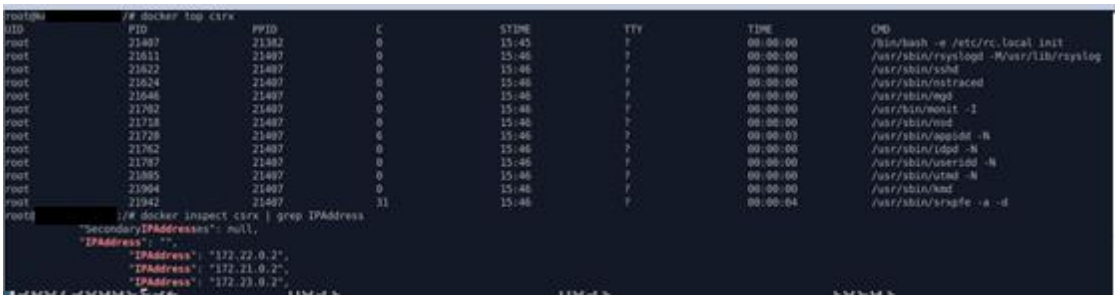
Kuvio 21. Verkkorajapintojen luonti Dockeriin

Verkkorajapintojen luomisen jälkeen voitiin käynnistää cSRX kontti. Kontin käynnistämisessä tuli ottaa huomioon, että mitä muuttujia halutaan käyttää ja yhdistää kontti aikaisemmin luotuun hallintaverkkorajapintaan sekä kontti täytyy muistaa käynnistää –privileged parametrilla, jotta kontti pyörii etuoikeutetussa tilassa. Käynnistämisen yhteydessä voitiin myös määrittää, että halutaanko kontti käynnistää reititystilassa (engl. routing mode) vai secure-wire-tilassa. Konttipalomuurin käynnistämisen jälkeen, täytyy kontti yhdistää myös muihin aikaisemmin luotuihin verkkorajapintoihin. Kuviossa 22 on käynnistetty konttipalomuuri reititystilassa ja yhdistetty se luotuihin verkkorajapintoihin.

```
root@u          /# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="routing" --name=csrx                /csrx:18.2r1.9
0dbbbd7c3ec93751c334ad7f4cc230c56a68fc15f9239188e852baedc88abe47
root@u          /# docker network connect left_bridge csrx
root@u          /# docker network connect right_bridge csrx
```

Kuvio 22. Konttipalomuurin käynnistäminen

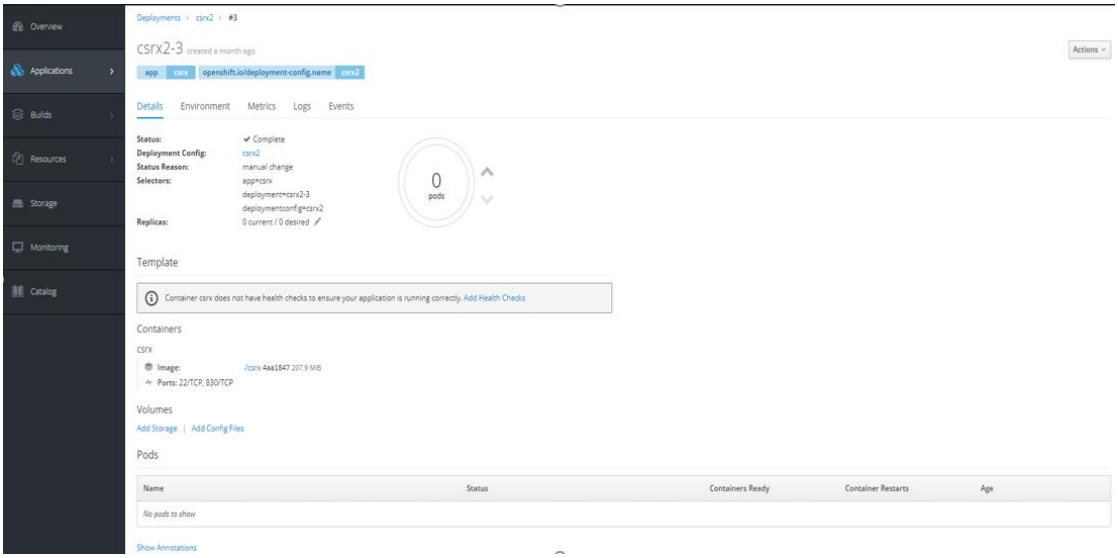
Kontin onnistuneen käynnistymisen jälkeen täytyy tarkastaa, että kontti toimii oikein ja kontissa pyörii oikeat prosessit. Prosesseina täytyisi olla ainakin Junos käyttöjärjestelmän normaaleita prosesseja, kuten nsd, srxpfe ja mgd. Samalla myös tarkistettiin, että hallintarajapinta on saanut IP-osoitteen onnistuneesti. Kuviossa 23 on tarkastettu kontin prosessit ja hallintarajapinnan IP-osoite.



Kuvio 23. Kontin prosessien ja IP-osoitteen tarkastelu

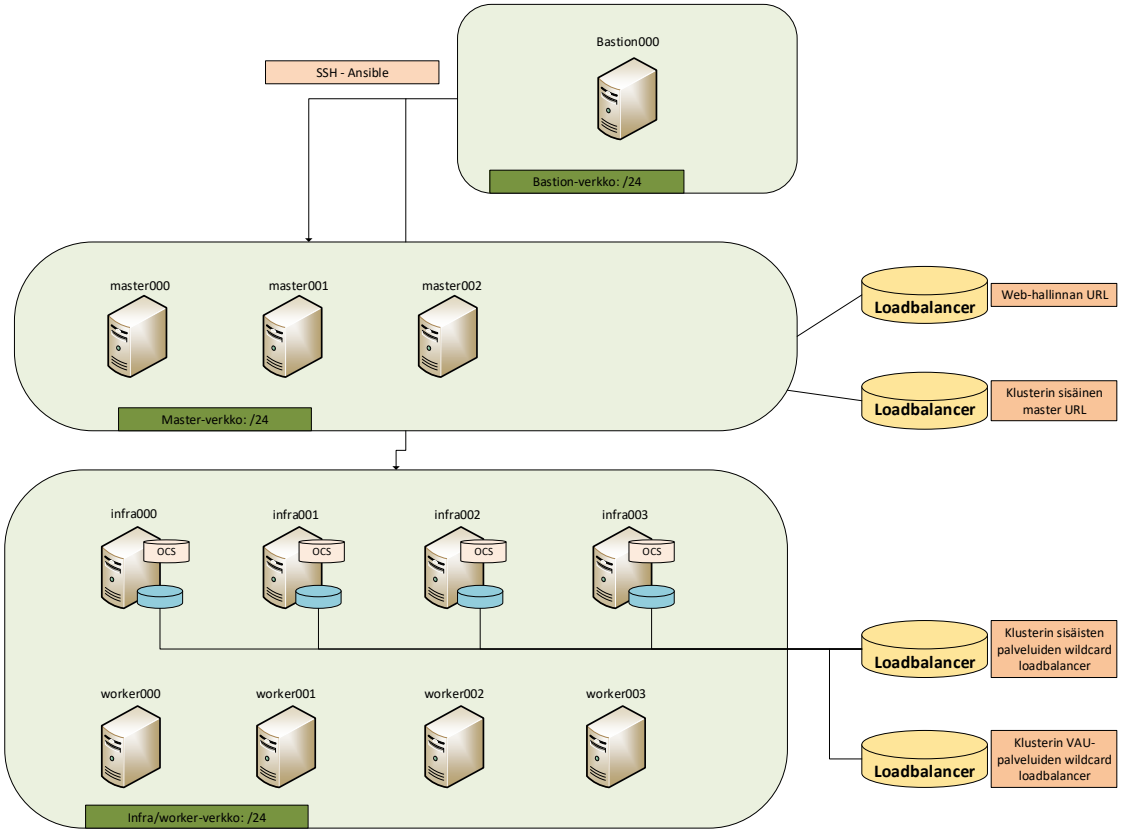
8.3 Openshift Container Platform

Yrityksessä oli hiljattain otettu käyttöön Openshift Container Platform, jossa konttipalomuuria haluttiin koestaa. Kehittäjän näkökulmasta ympäristöä käytetään selaimen käyttöliittymällä. Käyttöliittymässä pystyy helposti lisäämään konttien kopioiden määrää, tarkastamaan lokeja sekä kontin yleisiä tapahtumia. Kehittäjän silmin käyttöliittymä on erittäin yksinkertainen, koska kaikki tarvittavat toiminnot löytyvät helposti ja tarvittaessa kontin tai deploymentin YAML-tiedostojakin pystyy muokkaamaan halutusti. Kuviossa 24 on nähtävissä käyttöliittymän yksinkertaisuus, ja että kuinka podien määrää voidaan helposti muuttaa yhdellä painalluksella.



Kuvio 24. Openshift Container Platform käyttöliittymä.

Ympäristö oli valmiiksi rakennettu, joten opinnäytetyötä varten järjestelmän ylläpitäjä loi uudet tunnukset ja oman projektin. Käyttäjätunnuksille lisättiin admin-oikeudet projektiin. Projekti on klusterin oma abstrakti kokonaisuus ja projekti ajaa podeja eli kontteja comput- merkityillä nodeilla eli worker nodeilla. Infra-nodet ajavat erinäisiä infrakomponentteja, kuten grafanaa, alermanageria ja cluster logging podeja. Käytetyn Openshift konttiajoympäristön topologia on kuvattu kuviossa 25.



Kuvio 25. Openshift konttiajoympäristön topologia

Openshift Container Platformilla on mahdollista käyttää Openshiftin omaa privaatti rekisteriä tai hakea levykuvia muista saatavilla olevista rekistereistä. Oikeuksien takia päädyttiin käyttämään yrityksessä käytössä olevaa yksityistä rekisteriä, johon aikaisemmassa vaiheessa levykuva saatiin onnistuneesti vietyä.

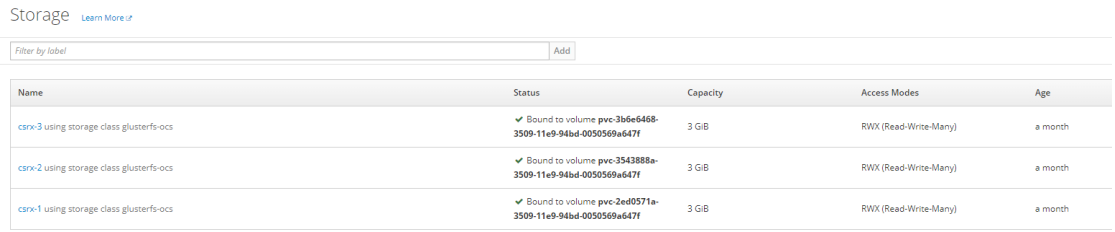
Ennen levykuvan käyttämistä omassa projektissa Openshiftissä, täytyi luoda secret-objekti, jonka avulla pystytään hakemaan rekisteristä levykuvia projektiin. Secret-objekti tarjoaa mekanismin, jolla voidaan säilyttää arkaluonteisia tietoja. Secret-objektin luomisen jälkeen päästiin ensimmäistä kertaa ottamaan levykuva OCP:ssa käyttöön Deploy image -toiminnon avulla. Deploy käynnistyi hyvin ja se yritti saada podia

eli konttia myös käynnistymään. Kontin käynnistymisen yhteydessä törmättiin ensimmäiseen virheilmoitukseen, kun kontti ei käynnistynyt ollenkaan. Käyttöliittymän lokeilta löydetty virheilmoitus on nähtävissä kuviossa 26.

```
1  /etc/rc.local: line 13: /var/local/.csrxenv: No such file or directory
```

Kuvio 26. OCP:n ensimmäinen virheilmoitus

Virheilmoitus viittasi alustavasti joko puutteellisiin loogisten levyjen (engl. volume) mounttaamiseen tai oikeuksiin. Ongelman selättämiseksi päädyttiin luomaan pysyväiset tiedontallennusvälineet (engl. storage) projektille. Kuviossa 27 on nähtävissä luodut pysyvät tiedontallennusvälineet projektille.



Kuvio 27. Pysyvät tiedontallennusvälineet

Pysyvien tiedontallennusvälineiden luomisen jälkeen muokattiin Deployn YAML-tiedostoa, jossa pystyttiin ottamaan käyttöön aikaisemmin luodut pysyvät tiedontallennusvälineet. Samaan YAML-tiedostoon lisättiin myös privileged-oikeudet, jotka myös järjestelmänylläpitäjä lisäsi projektille, koska valmistajan mukaan cSRX -kontti tarvitsee toimiakseen –privileged oikeudet. Kuviossa 28 on näytetty muokattu YAML-tiedosto.

Kuvio 28. Deploymentin muokattu YAML-tiedosto

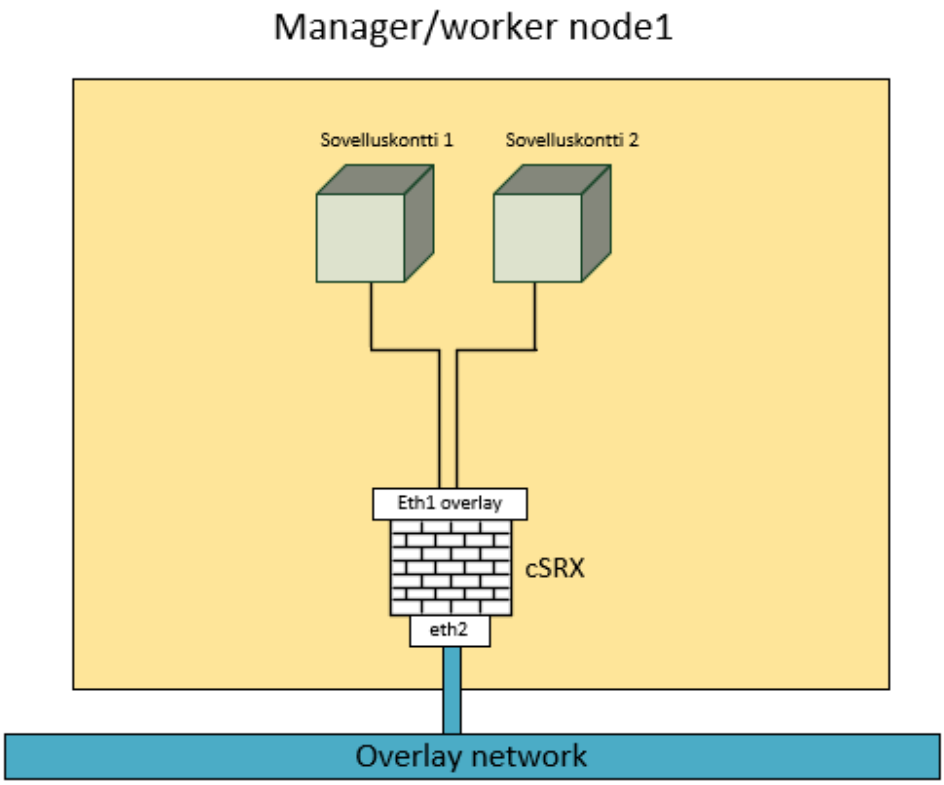
Muutosten jälkeen kontin käynnistyminen eteni seuraavaan virheilmoitukseen, joka on nähtävissä kuviossa 29. Virheilmoitus viittaisi suoraan levykuvassa olevaan rc.local -skriptiin. Rc.local on jakeluista löytyvä skripti, joka oletuksena ajetaan käynnistyksen loppuvaiheessa. Kyseisen rc-local -skriptin pystyi etsimään kontin hakemistoista, josta huomattiin, että virheilmoituksen ilmoittama rivi 16 viittasi skriptin komentoon ’echo ”/var/crash/coredump.%e.%p > /proc/sys/kernel/core_pattern’. Komento yrittää siis tulostaa tiedot järjestelmän kaatumisista sellaiseen sijaintiin, johon ei oikeuksia ole riittävästi, vaikka Openshift projektilla on täydet admin -ja privileged oikeudet.

Kuvio 29. OCP:n toinen virheilmoitus

Projektille oli annettu jo kaikki käyttöoikeudet, jotta se voi kontteja ajaa, joten virheilmoitusta oli erittäin hankala alkaa selvittämään. Konttia yritettiin saada käynnistymään myös järjestelmänylläpitäjän oikeuksilla, jos jokin oikeus olisikin mahdollisesti unohtunut lisätä, mutta päädyttiin samaan virheilmoitukseen. Tässä vaiheessa päätettiin kysyä tuotteen valmistajalta, että onko tuotetta testattu Openshift-ympäristössä. Tuotteen valmistaja vastasi, että tuotetta ei ole testattu kyseisessä ympäristössä ja sen toimivuutta ei voida taata. Tämän tiedon jälkeen toimeksiantajan päätöksellä päätettiin siirtyä toiseen ympäristöön, koska vaihtoehdoiksi olisi jäänyt levykuvan puukottaminen ja tätä ei haluttu tehdä, koska kyseessä on valmis tuote, jota ei haluta liikaa puukottaa.

8.4 Docker Swarm & Compose

Epäonnistuneiden Openshift testausten jälkeen päätettiin siirtyä Docker Swarm -ympäristöön. Yrityksessä on testiympäristönä käytössä Docker Swarm, mutta testipalvelinten käyttötarpeen vuoksi päätettiin konttipalomuuria testata perinteisillä virtuaalikoneilla, joissa käyttöjärjestelmänä toimi Xubuntu 18.04. Ensimmäiseksi päädyttiin käyttämään yhtä virtuaalikonetta, joka toimii samanaikaisesti manager -ja worker nodena Swarmissa, mutta tulevaisuudessa worker nodeja olisi useampi, joilla jokaisella pyörisi oma konttipalomuurinsa. Kuviossa 30 on kuvailtu Swarmin raaka looginen topologia. Topologiassa konttien kaikki liikenne pyritään käyttämään konttipalomuurin kautta, jolloin konttipalomuurin eth1 rajapinta toimii overlay-ajurina ja eth2 rajapinta toimii untrust-alueena ulkoverkkoon päin. Muut jatkossa lisättävät worker nodet pystyvät keskustelemaan manager noden ja muiden worker nodejen kanssa alemman tason overlay verkon avulla. Verkko-osoitteita ei loogiseen topologiaan vielä suunniteltu, koska konttipalomuurin toimintaa klusterissa ei ollut vielä todettu. Verkko-osoitteet olivat tavoitteena suunnitella myöhemmässä vaiheessa, kunhan konttipalomuurin toiminta on todennettu ensiksi klusterissa, koska todennäköisesti saatettiin törmätä ongelmiin kontin käynnistämisessä.

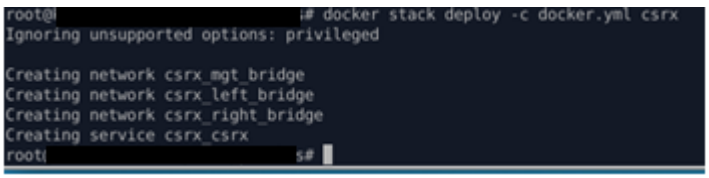


Kuvio 30. Swarmin looginen topologia

CSRX konttipalomuurille voi määrittää useita parametrejä, joten paras keino konttipalomuurin käyttöönottoon oli luoda yksi Docker compose-tiedosto, jossa pystytään määrittämään tarpeelliset parametrit ja verkkorajapintojen lisäämiset. Compose-tiedoston avulla saadaan nopeasti ja helposti luotua kontti ja sille tarvittavat määritykset, kuten Dockerin verkkorajapinnat. Kuviossa 31 on näytetty työssä käytetty compose-tiedosto, joka on kirjoitettu YAML-kielellä. Tiedostossa on määritetty tarvittavat verkkorajapinnat, liitetty kontti niihin rajapintoihin ja määritetty myös muut parametrit kontille.

Kuvio 31. Konttipalomuurin Docker-compose tiedosto

Compose-tiedostoa käyttäen kontti otetaan käyttöön swarmiin docker stack deploy -komennolla, jolloin kontti muodostuu swarmiin. Konttipalomuurin käyttöönotossa törmättiin kuitenkin heti ensimmäiseen virheeseen, kun käynnistyksen yhteydessä huomattiin, ettei swarm tue privileged-tilaa. Tuotteen valmistaja on määrittänyt, että kontti tulisi ajaa aina privileged-tilassa. Kuviossa 32 on nähtävissä virheilmoitus ja verkkorajapintojen muodostuminen.



Kuvio 32. Docker deploy stack -komento

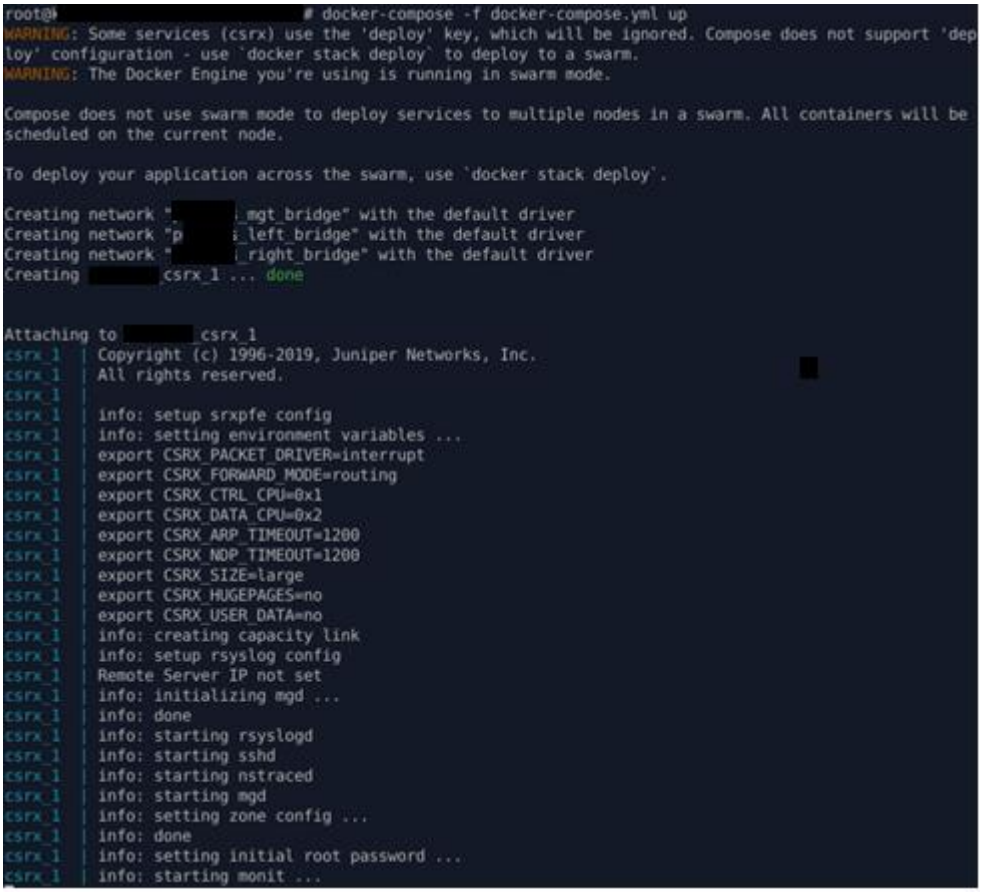
Palvelu ei muodostanut ainuttakaan konttia johtuen privileged-tilan yhteensopivuusongelmasta swarmin kanssa. Palvelun lokeja tutkimalla huomattiin, että virheilmoitukset viittaavat rc.local skriptiin ja siihen, ettei kontti saa tarpeellisia oikeuksia. Tällä

hetkellä Docker swarm ei vielä tule privileged-tilaa, joten kontin ajaminen on mahdotonta privileged-tilassa. Virallista korjausta privileged-tilan sallimiseen swarmissa ei tällä hetkellä ole olemassa, joten minkäänlaisia epävirallisia korjauksia ei päätetty tehdä, koska tuotantoympäristöönkään epävirallisia korjauksia ei tulevaisuudessa ei olisi mahdollista tehdä. Kuviossa 33 on nähtävissä lokeilta löydetyt virheet, jotka viittaavat suoraan privileged-tilan puuttumiseen.

```
docker service logs csrx_csrx
| /etc/rc.local: line 16: /proc/sys/kernel/core_pattern: Read-only file system
| Copyright (c) 1996-2019, Juniper Networks, Inc.
| All rights reserved.
|
| Copyright (c) 1996-2019, Juniper Networks, Inc.
| All rights reserved.
|
| /etc/rc.local: line 16: /proc/sys/kernel/core_pattern: Read-only file system
| Copyright (c) 1996-2019, Juniper Networks, Inc.
| All rights reserved.
|
| /etc/rc.local: line 16: /proc/sys/kernel/core_pattern: Read-only file system
| Copyright (c) 1996-2019, Juniper Networks, Inc.
| All rights reserved.
|
| /etc/rc.local: line 16: /proc/sys/kernel/core_pattern: Read-only file system
```

Kuvio 33. Swarmin lokeilta löydetty virheilmoitus

Konttipalomuuria yritettiin swarmissa käynnistää myös ilman privileged-tilaa, mutta tällöin törmättiin samaan ongelmaan, että kontti ei käynnistynyt. Ilman privileged-tilaa lokeillekaan ei muodostunut minkäänlaisia virheilmoituksia. Compose-tiedoston toiminta varmistettiin kuitenkin käynnistämällä kontti swarmin ulkopuolella komennolla docker-compose -f docker-compose.yaml up. Kuviossa 34 on komennon syöte ja lokeille ei myöskään tullut minkäänlaisia virheilmoituksia sekä kontti käynnistyi ongelmitta.



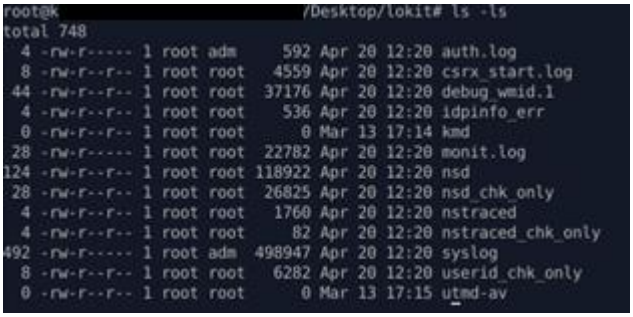
Kuvio 34. Docker-composella kontin käynnistys

Swarmin ja konttipalomuurin yhteensopivuusongelmasta oltiin yhteydessä tuotteen valmistajaan, jotka totesivat saman, että konttia pystytään ajamaan swarmin ulkopuolella, mutta ei klusterin sisällä johtuen kontin vaatimasta privileged-tilan käytöstä. Tuotteen valmistaja pyrkii kehittämään tuotetta tällä hetkellä siten, että sitä olisi mahdollista ajaa ilman privileged-tilaa. Ilman privileged-tilaa konttipalomuuria voitaisiin käyttää layer 2 -palomuurina siten, että layer 3 hallintayhteys säilyisi.

8.5 Lokitus

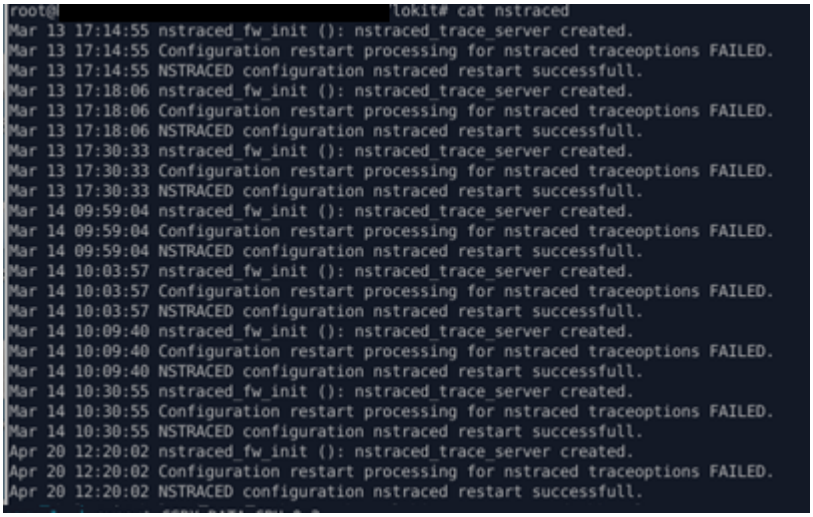
CSRX konttipalomuurissa lokitiedostot tallennetaan oletuksena /var/log -hakemistoon. Oletuksena lokitiedostoja pidetään yllä ainoastaan, mikäli ulkoisia massamuistin sisältöä ei ole otettu käyttöjärjestelmän käyttöön /var/log -hakemistolle, joten cSRX kontin tuhoutuessa myös lokitiedostot hävitetään. Pysyvän lokitiedoston pystyy ottamaan käyttöön kontin käynnistymisen yhteydessä lisäämällä pysyväisen lokitiedoston mahdollistavat parametrit compose -tiedostoon. Konttipalomuuri muodostaa

automaattisesti 13 eri lokitiedostoa, joista kerrotaan tarkemmin omissa aliotsikoissaan. Kuviossa 35 on cSRX konttipalomuurin automaattisesti luomat lokitiedostot.



Kuvio 35. Konttipalomuurin lokitiedostot

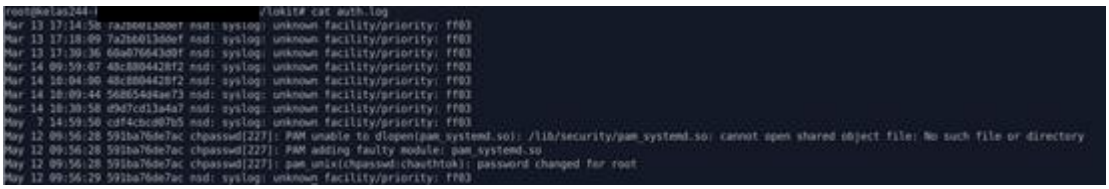
Lokitiedostojen pysyvyyden määrittelevän parametrin toimivuutta testattiin poistamalla kontti ja tarkistamalla, että säilyykö lokitiedostot ja niiden sisältö vielä ennallaan. Samalla pystyttiin testaamaan myös, että kirjoittaako konttipalomuuri yli vanhat lokitiedostot, kun se laitetaan takaisin ajoon compose-tiedostolla. Kuviossa 36 lokitiedostoa lukemalla pystyttiin päättelemään, että konttipalomuuri ei yli kirjoita vanhoja lokitiedostoja, koska lokitiedostosta löytyy vielä päivämäärän aikaisemman päivämäärän lokitietoja, vaikka kontti poistettiin päivämäärien välissä.



Kuvio 36. Lokitiedostojen pysyvyys

8.5.1 Auth.log lokitiedosto

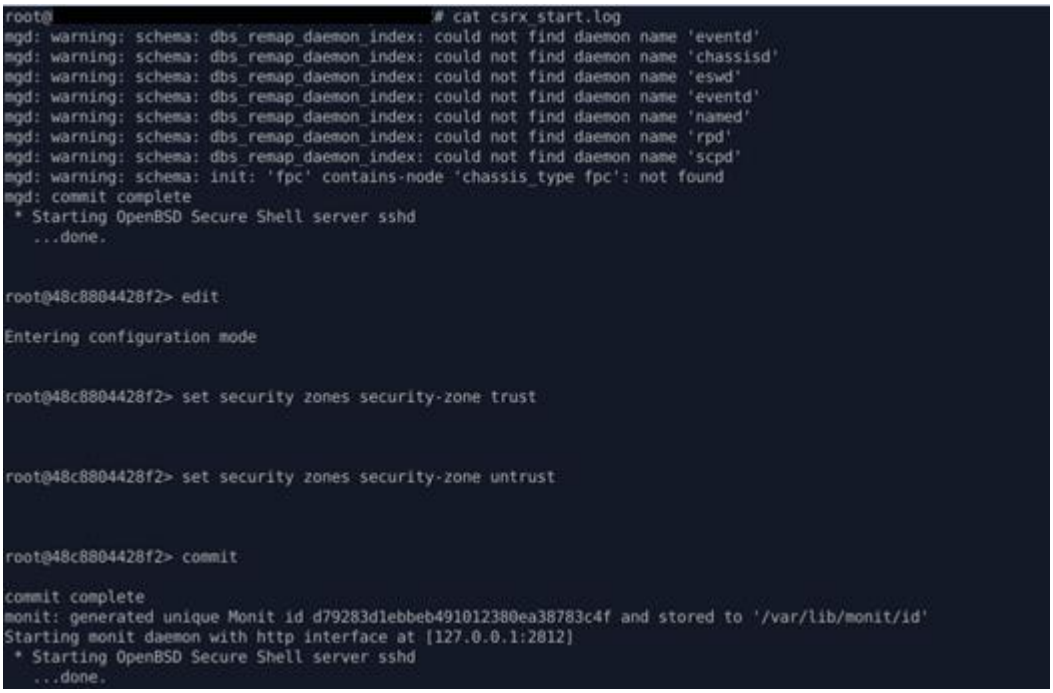
Yleisesti auth.log lokitiedoston tulisi lokittaa tietoa käyttäjien todentamisesta. Kuviossa 37 on auth.log lokitiedoston sisältö, josta voidaan huomata, että ainakin salasanan vaihdoksesta tulee tieto lokitiedostoon. Mitään virallisia dokumentteja lokitiedoston tarkasti tallentamasta tiedosto ei ollut käytettävissä.



Kuvio 37. Auth.log lokitiedosto

8.5.2 Csrx_start.log lokitiedosto

Lokitiedoston tarkasta sisällöstä ei löytynyt Juniperin dokumenteista tarkkaa tietoa, mutta lokitiedosto ainakin lokittaa mgd-prosessin eli hallinta daemonin toimintaa, sekä syötettyjä komentoja. Kuviossa 38 on näytetty osa lokitiedoston sisällöstä.



Kuvio 38. Csrx_start.log lokitiedosto

8.5.3 Debug_wmid.1 lokitiedosto

Debug_wmid.1 on lokitiedosto userFW (user firewall) -ominaisuudelle. UserFW:n määrittelemät palomuurisäännöt sallivat järjestelmänvalvojien hallita käyttäjien pääsyä verkkoon rooliensa perusteella. UserFW:n wmic -prosessi luo ja tallentaa lokiviestit lokitiedostoon. Debug_wmid -lokitiedosto luodaan, kun wmic -prosessi käynnistetään ensimmäistä kertaa. Kuviossa 39 on esitetty katkelma debug_wmid.1 lokitiedostosta.

```
root@                    Desktop/lokit# cat debug_wmid.1
[2019/03/13 15:15:00,  0]   wmid: Debug class number of 'event_log': 20
[2019/03/13 15:15:00,  0]   wmid: Debug class number of 'pc_probe': 21
[2019/03/13 15:15:00,  0]   the debug level string is "-1 event_log:0 pc_probe:0"
[2019/03/13 15:15:01,  0]   get slave running mode!
[2019/03/13 15:15:06,  0]   wmid: Debug class number of 'event_log': 20
[2019/03/13 15:15:06,  0]   wmid: Debug class number of 'pc_probe': 21
[2019/03/13 15:15:06,  0]   the debug level string is "-1 event_log:0 pc_probe:0"
[2019/03/13 15:15:07,  0]   clear configure request!
[2019/03/13 15:15:07,  0]   get start indication!
[2019/03/13 15:15:07,  0]   ##########################################################
[2019/03/13 15:15:07,  0]   WMID service startup!
[2019/03/13 15:15:07,  0]   ##########################################################
[2019/03/13 15:15:07,  0]   ##########################################################
[2019/03/13 15:15:07,  0]   WMID wait configuration!
[2019/03/13 15:15:07,  0]   ##########################################################
[2019/03/13 15:15:07,  0]   the pc probe thread number is 100.
[2019/03/13 15:15:07,  0]   global probe wql:
[2019/03/13 15:15:07,  0]   select * from win32_logonsession where logontype = 2 or logontype = 10
[2019/03/13 15:15:07,  0]   global eventlog wql:
[2019/03/13 15:15:07,  0]   select * from win32_ntlogevent where logfile = 'security' and eventtype = 4 
ier = 674 or eventidentifier = 4624 or eventidentifier = 4768 or eventidentifier = 4769 or eventidentifi
[2019/03/13 15:15:07,  0]   get null dc configuration!
[2019/03/13 15:15:07,  0]   none ip filter configuration.
[2019/03/13 15:15:07,  0]   get perf tune configuration.
[2019/03/13 15:15:07,  0]   the probe rate: 100, eventlog size: 100000 dcerpc timeout: 10
[2019/03/13 15:15:07,  0]   ************************************************************
[2019/03/13 15:15:07,  0]   global setting:
[2019/03/13 15:15:07,  0]   global debug level: 0, trace option: 0
[2019/03/13 15:15:07,  0]   ************************************************************
[2019/03/13 15:15:09,  0]   wmid: Debug class number of 'event_log': 20
[2019/03/13 15:15:09,  0]   wmid: Debug class number of 'pc_probe': 21
[2019/03/13 15:15:09,  0]   the debug level string is "-1 event_log:0 pc_probe:0"
[2019/03/13 15:15:10,  0]   clear configure request!
[2019/03/13 15:15:10,  0]   get start indication!
[2019/03/13 15:15:10,  0]   ##########################################################
[2019/03/13 15:15:10,  0]   WMID service startup!
[2019/03/13 15:15:10,  0]   ##########################################################
[2019/03/13 15:15:10,  0]   ##########################################################
[2019/03/13 15:15:10,  0]   WMID wait configuration!
[2019/03/13 15:15:10,  0]   ##########################################################
[2019/03/13 15:15:10,  0]   the pc probe thread number is 100.
[2019/03/13 15:15:10,  0]   global probe wql:
[2019/03/13 15:15:10,  0]   select * from win32_logonsession where logontype = 2 or logontype = 10
[2019/03/13 15:15:10,  0]   global eventlog wql:
[2019/03/13 15:15:10,  0]   select * from win32_ntlogevent where logfile = 'security' and eventtype = 4 
ier = 674 or eventidentifier = 4624 or eventidentifier = 4768 or eventidentifier = 4769 or eventidentifi
[2019/03/13 15:15:10,  0]   get null dc configuration!
[2019/03/13 15:15:10,  0]   none ip filter configuration.
[2019/03/13 15:15:10,  0]   get perf tune configuration.
[2019/03/13 15:15:10,  0]   the probe rate: 100, eventlog size: 100000 dcerpc timeout: 10
[2019/03/13 15:15:10,  0]   ************************************************************
[2019/03/13 15:15:10,  0]   global setting:
[2019/03/13 15:15:10,  0]   global debug level: 0, trace option: 0
[2019/03/13 15:15:10,  0]   ************************************************************
[2019/03/13 15:15:11,  0]   wmid: Debug class number of 'event_log': 20
```

Kuvio 39. Debug_wmid.1 lokitiedosto

8.5.4 Idpinfo_err lokitiedosto

IDP:n (Intrusion Detection and Prevention) ominaisuuden käyttöönoton yhteydessä konttipalomuuri muodostaa idpinfo_err nimisen lokitiedoston, johon tallennetaan tapahtua IDP:stä, kuten viat ja virheilmoitukset. Järjestelmätietojen lisäksi IDP:n pystyy konfiguroimaan siten, että se luo myös tapahtumalokeja hyökkäyksistä samalle lokitiedostolle. IDP luo tapahtumalokit, kun jokin tapahtuma vastaa IDP-käytännön sääntöä. IDP:lle voidaan myös määrittää kirjautumissääntö, jolloin konttipalomuuri luo lokimerkinnän jokaiselle tapahtumalle, joka vastaa sääntöä. Kuviossa 40 on lokitiedosto kokonaisuudessaan juuri käyttöönotetulta konttipalomuurilta, jossa IDP:tä ei ole otettu käyttöön.

```
root@                    Desktop/lokit# cat idpinfo_err
[15:14:59] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[15:18:10] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[15:30:37] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[07:59:08] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[08:04:01] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[08:09:45] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[08:30:59] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
[09:20:07] Error: sc_klib_subs_clear_status: sc_dev_ioctl() failed
```

Kuvio 40. Idpinfo_err lokitiedosto

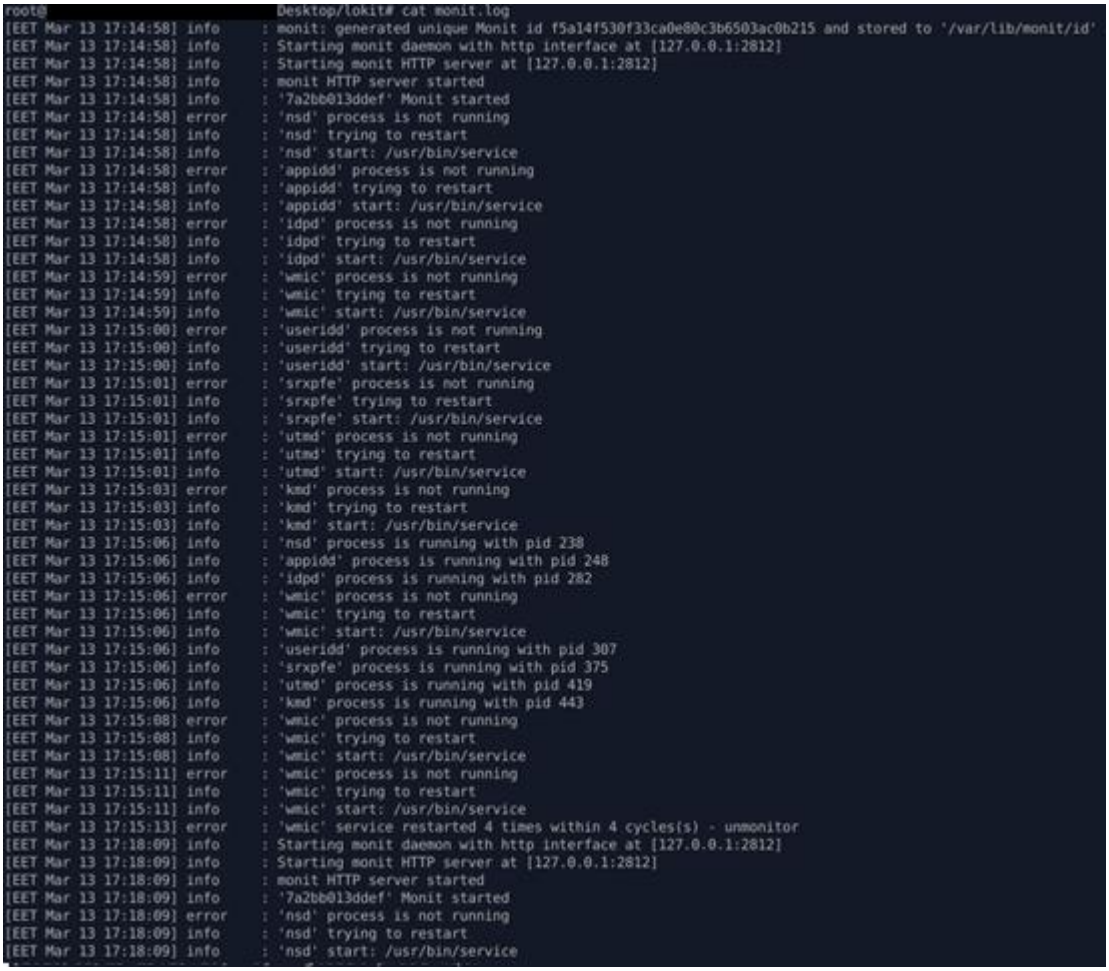
8.5.5 Kmd lokitiedosto

KMD-lokitiedostoon tallentuu IKE:n (Internet Key Exchange) virheilmoituksia. IKE on IPsec:n (Internet Protocol Security) standardiprotokolla, jota käytetään varmistaman VPN:n neuvottelut ja verkkoyhteyden suojaaminen. KMD-lokitiedoston pystyy myös konfiguroimaan näyttämään pelkästään VPN:n tilaviestejä. Konttipalomuuri muodostaa KMD-lokitiedoston automaattisesti käynnistyksen yhteydessä. Kuvakaappausta ei lokitiedostosta otettu, koska toiminnallisuutta, jota se lokittaa ei ollut otettu käyttöön.

8.5.6 Monit.log lokitiedosto

Monit.log lokitiedosto on konttipalomuurin prossien seuraamisen tarkoitettu lokitiedosto. Se pitää sisällään virheilmoitukset ja tilamuutokset cSRX:n prosesseista, kuten

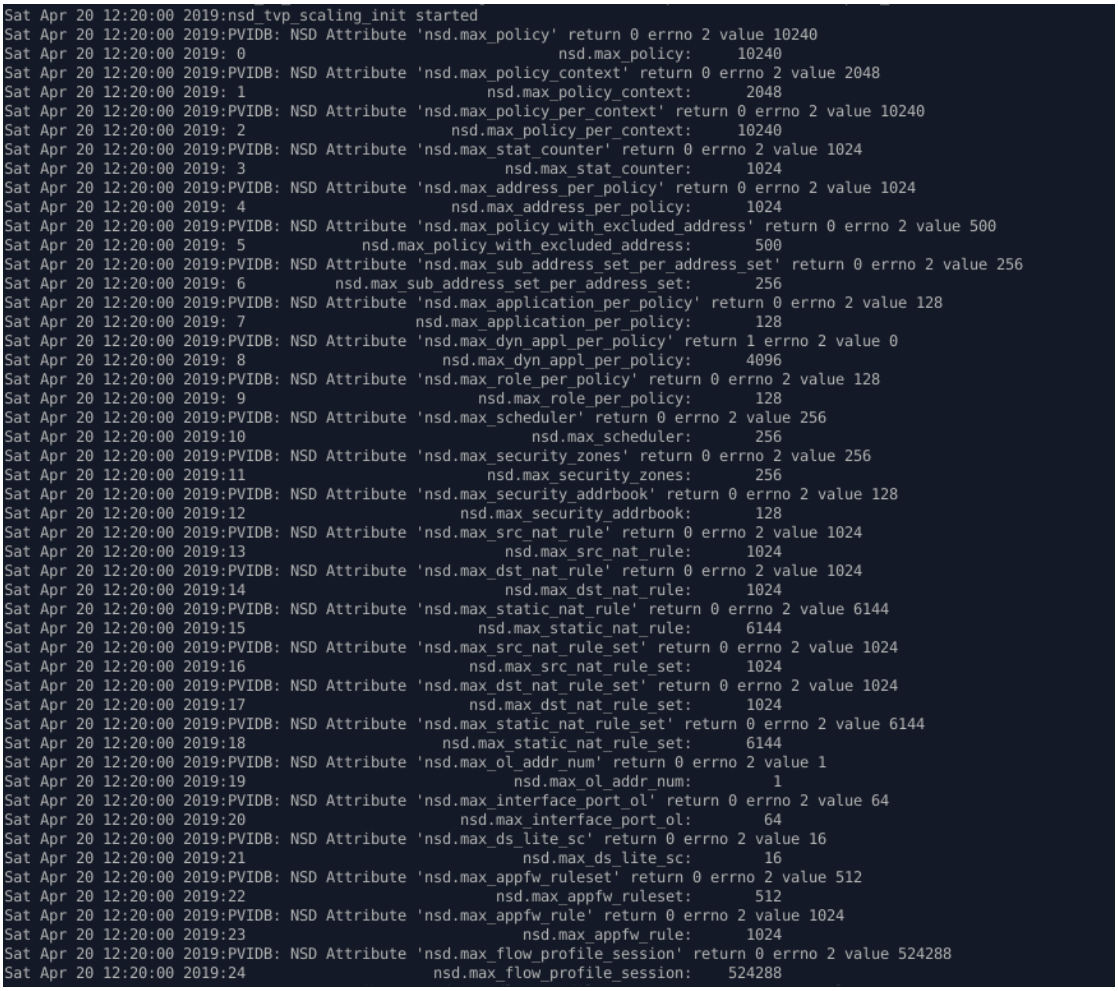
srxpfe, wmic, ipdf ja nsd. Lokitiedostosta pystyy selvittämään prosessien nykyiset tilat ja niiden muutokset, joka tekee vianselvityksestä helpompaa. Kuviossa 41 on esitetty katkelma monit.log lokitiedostosta, josta voidaan nähdä aikaisemmin mainittuja prosesseja.



Kuvio 41. Monit.log lokitiedosto

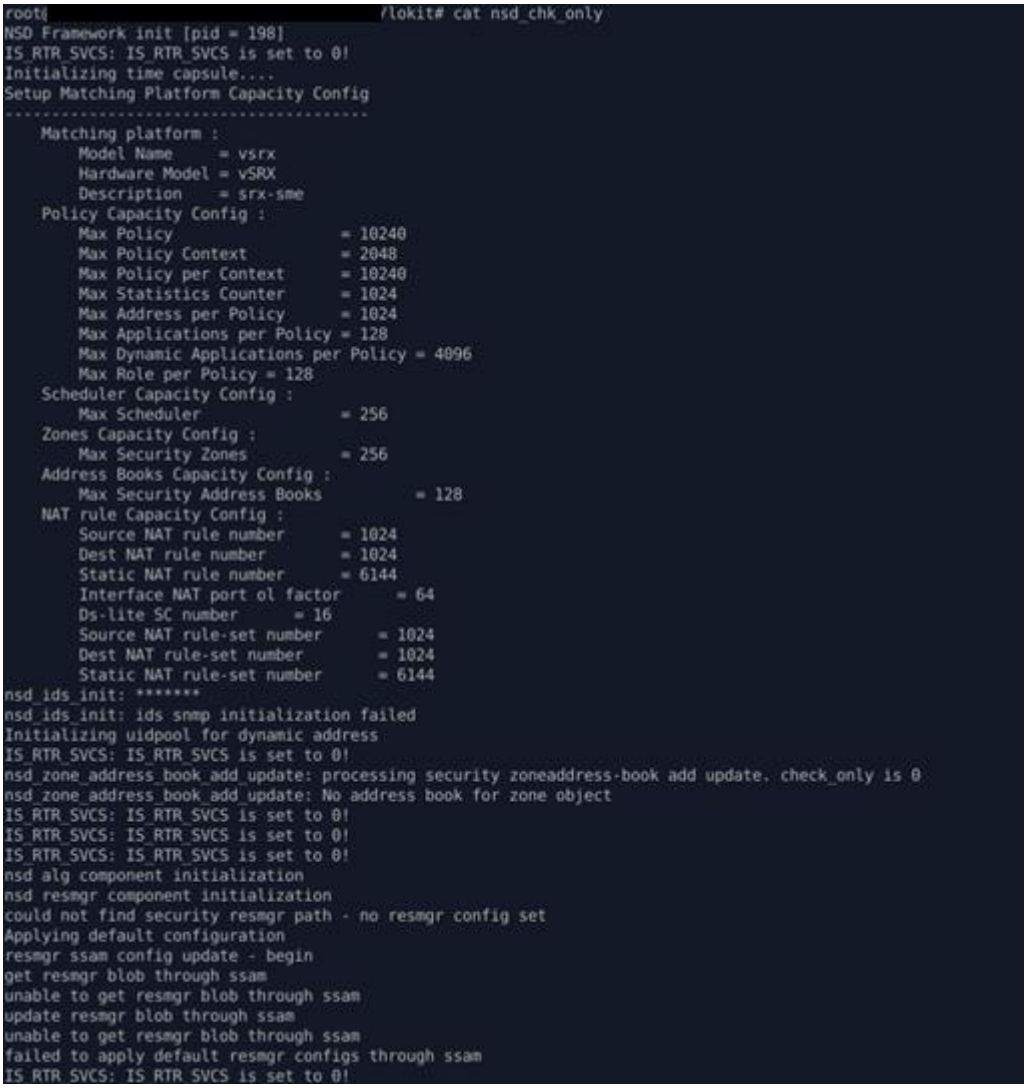
8.5.7 Nsd ja nsd_chk_onlyt lokitiedostot

NSD (Network Security Daemon) -lokitiedostoon tallennetaan NSD:n tilaviestejä ja virheilmoituksia. Lokitiedostossa on eritelty tarkasti jokaisen NSD attribuutin arvoja ja tiloja. NSD hallitsee palomuurin konfigurointeja, joita käytetään Junos ohjelmistolla. Se lukee alueita, käytänteitä, reititysinstansseja ja tarkastaa niiden toimintaa sekä tekee suoritustarkastuksia. Kuviossa 42 on juuri käynnistetyn cSRX konttipalomuurin nsd -lokitiedosto.



Kuvio 42. Nsd lokitiedosto

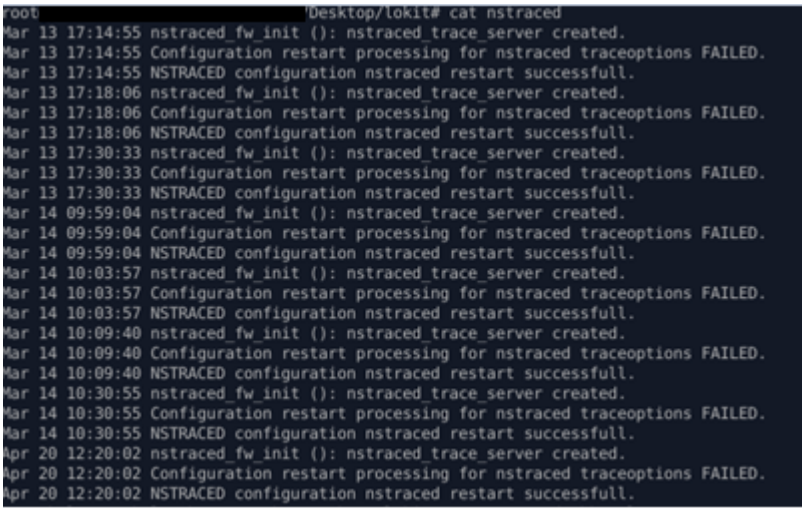
Nsd_chk_only lokitiedosto tarjoaa tietoa alustan kapasiteettirajoituksista. Kyseinen lokitiedosto yli kirjoitetaan aina, kun konttipalomuurilla tehdään ’commit check’ eli komentojen tarkistus. Lokitiedosto sisältää myös tarkempia tietoja mahdollisista virheistä järjestelmässä. Kuviossa 43 on nsd_chk_only lokitiedoston sisältö juuri käynnistetyltä cSRX konttipalomuurilta.



Kuvio 43. Nsd_chk_only lokitiedosto

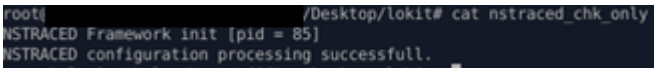
8.5.8 Nstraced ja nstraced_chk_only lokitiedostot

Nstraced lokitiedosto näyttää nstraced prosessin tarkempia tietoja ja myös nstraced konfiguroinnin tilaviestejä. Kuviossa 44 on nstraced lokitiedoston sisältö konttipalo-muurista, jossa nstraced -toimintoa ei ole konfiguroitu.



Kuvio 44. Nstraced lokitiedosto

Nstraced_chk_only lokitiedosto näyttää nstraced prosessin tilamuutokset. Lokitiedosto on yleisesti lyhyt, joka on havaittavissa kuviossa 45.



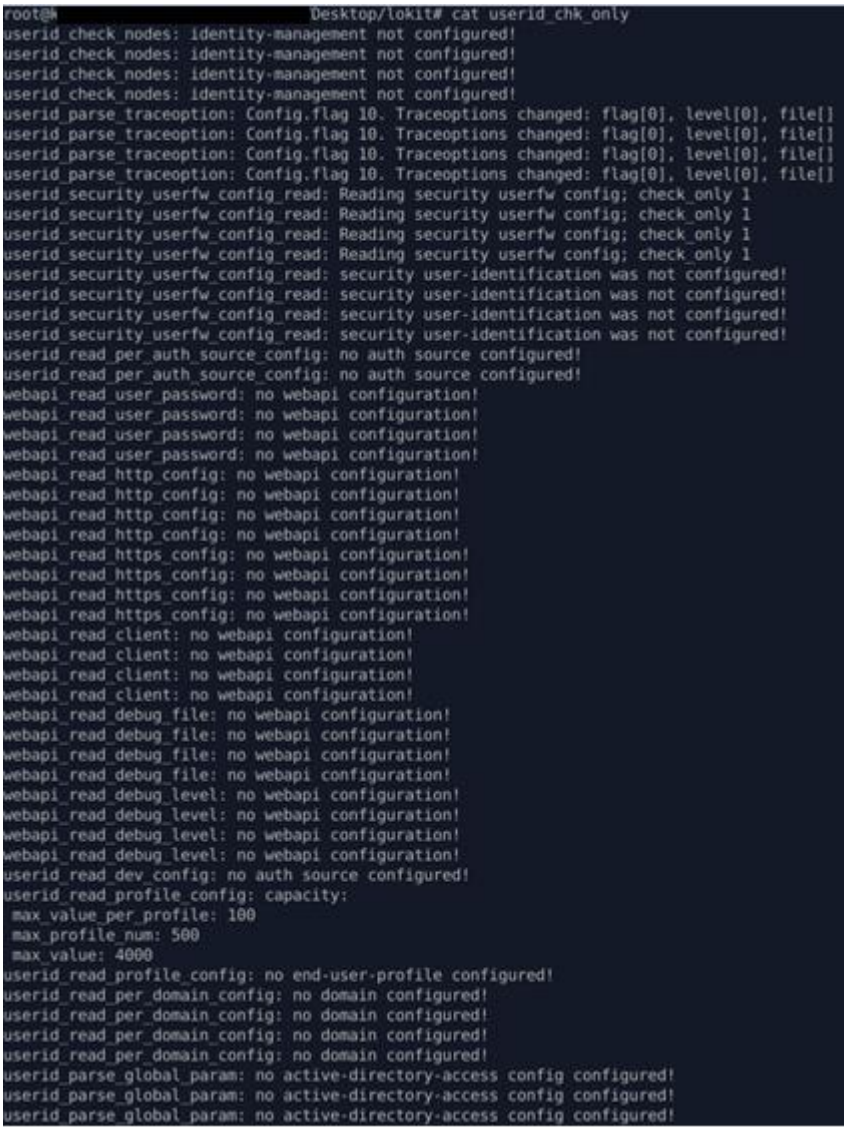
Kuvio 45. Nstraced_chk_only lokitiedosto

8.5.9 Syslog lokitiedosto

Järjestelmän lokitus (engl. system logging) on oletuksena konttipalomuurissa käytössä. Syslog lokitiedosto on suurin ja se sisältää eniten tietoa, mutta sen tulkitseminen voi olla myös erittäin haastavaa. Tarvittaessa syslogin voi konfiguroida tallentamaan tiedon kaikista suoritetuista komennoista konttipalomuurissa. Syslog onkin yleisesti tärkein lokitiedosto, mutta sen lukeminen on erittäin työlästä, joten syslogin lukemista varten suositellaan käytettäväksi erillistä syslog palvelinta, jonka avulla syslogia pystytään lukemaan esimerkiksi selain käyttöliittymän kautta.

8.5.10 Userid_chk_only lokitiedosto

Userid prosessi ylläpitää userid_chk_only lokitiedostoa. Lokitiedostoon tallennetaan tietoa userid:n prosessista ja siihen liittyvistä tilaviesteistä. Lokitiedosto muodostetaan automaattisesti konttipalomuurin käynnistyksessä, mutta lokitiedostossa ei ole juurikaan tähdellistä tietoa ennen, kuin userid on otettu käyttöön. Kuviossa 46 on esitetty osa userid_chk_only lokitiedostosta.



Kuvio 46. Userid_chk_only lokitiedosto

8.5.11 Utmd-av lokitiedosto

Konttipalomuuri muodostaa automaattisesti jäljitys (engl. trace) asetuksen UTM (Unified threat management) antivirus ominaisuudelle, jolloin antivirus moottori muodostaa automaattisesti utmd-av lokitiedoston. Lokitiedoston viestit muodostavat UTM:n prosessi, joka suojelee verkkoa yleisimmiltä hyökkäyksiltä. Lokitiedostoa käytetään vaativampaan UTM antiviruksen vianselvitykseen.

9 Yhteenveto

9.1 Pohdinta

Opinnäytetyön tavoitteena oli tutustua konttipalomuuriin ja tutkia sen mahdollista käyttöä kahdessa erilaisessa klusterissa. Alkuperäisenä suunnitelmana oli pyrkiä ainoastaan käyttämään Openshift konttiajoympäristöä, mutta konttipalomuurin yhteensopivuus ongelman vuoksi alkuperäisen testiympäristön käyttämisestä luovuttiin. Tuotteen valmistaja ei myöskään luvannut minkäänlaista tukea kyseiseen ympäristöön, joka myös omalta osaltaan auttoi päätöksessä siirtyä toiseen ympäristöön. Toiseksi ympäristöksi valikoitui Docker swarm, joka on ollut laajemmassa testikäytössä yrityksessä jo aikaisemmin. Swarm ympäristö päädyttiin rakentamaan virtuaalikoneilla, koska kaikki testipalvelimet yrityksessä olivat muussa käytössä. Valmistaja lupaili alustavasti, että konttipalomuurin pitäisi toimia swarmissa moitteetta, mutta loppujen lopuksi tuote ei toiminutkaan toimeksiantajan haluamalla tavalla klusterin sisällä, vaan ainoastaan klusterin ulkopuolella.

Konttipalomuuri olisi saatu todennäköisesti toimimaan klusterin ulkopuolella ja valmistaja oli myös todennut sen toimivuuden, mutta toimeksiantaja linjasi, että konttipalomuuri halutaan toimivaksi klusterin sisällä, jotta konttipalomuuri saataisiin oikeassa tuotantoympäristössä esimerkiksi kuormantasauksen piiriin helposti. Konttipalomuurin käytöllä olisi haluttu saada tietoa, että mitkä kontit keskustelevat keskenään, koska esimerkiksi kaikkien konttien ei ole tarpeellista keskustella tietokantakontille ja konttipalomuurin avulla tämän kaltaiset ongelmat pystyttäisiin helposti ennalta eh-

käisemään. Tämän hetkisessä swarm testiympäristössä konttien liikenteestä ei juurikaan saada tarpeeksi tietoa tietoliikennenäkökulmasta ja tätä asiaa olisi konttipalomuurilla haluttu parantaa.

Konttipalomuurin käyttöönottamista testiympäristöissä vaikeutti dokumentoinnin puuttuminen, koska ainoat dokumentit konttipalomuurista kertoivat päällisin puolin konttipalomuurituotteen ominaisuuksista ja, että kuinka konttipalomuuri perinteisellä Dockerilla saadaan käyntiin. Tuotteena konttipalomuuri on suhteellisen uusi, koska tuote on julkaistu kesällä 2018. Tuotteen valmistaja pyrki kuitenkin tukemaan opinnäytetyön edistymistä opastamalla sähköpostitse, mutta juurikaan tukea konttipalomuurin käyttöön klusteriympäristön sisällä ei löytynyt. Konttipalomuuraus on suhteellisen uusi asia ja varsinkin kyseisen konttipalomuurituotteen käytöstä ei ole tutkimuksia vielä tehty.

Opinnäytetyön alkuperäinen prosessi muuttui suuresti alkuperäisestä suunnitelmasta, koska työssä törmättiin useisiin odottamattomiin ongelmiin. Ongelmat veivät paljon aikaa, jonka vuoksi muutosskenaarioiden avulla konttipalomuurin ominaisuuksien testaus jäi kokonaan pois opinnäytetyöstä. Mielestäni tämä ei ollut kuitenkaan huono asia, koska opinnäytetyössä saatiin kuitenkin paljo lisätietoa tuotteesta ja sen yhteensopivuudesta erilaisissa konttiklusteri ympäristöissä.

9.2 Jatkokehitys

Opinnäytetyössä käytetyllä konttipalomuurituotteella ei ole nykyisellä toiminnallaan varsinaista käyttöä yrityksen tuotantoympäristössä. Openshiftin tukea tuskin lähiaikoina on vielä tulossa, koska konttipalomuurituotteen valmistajan mukaan tuotetta ei ole vielä kertaakaan testattu Openshift tai Kubernetes -ympäristöissä. Valmistaja on kuitenkin luvannut kehittää tuotettaan, jotta sitä pystyttäisiin jatkossa käyttämään klusterin sisällä ainakin Docker swarmissa. Valmistajan kanssa keskusteltiin mallista, jossa konttipalomuurin verkkorajapinta olisikin ”transparent” eli sillä ei olisi ollenkaan IP-osoitetta, vaan se toimisi ”bump-in-a-wire” -mallilla. Valmistajan mukaan konttipalomuurista voitaisiin poistaa privileged -tilan vaatimus, mikäli konttipalomuuria voitaisiin ajaa ”transparent” -tilassa, joka tarkoittaa, että konttipalomuurilla ei olisi IP-osoitetta datapolulla. Konttipalomuuri toimisi siis vain layer 2 palomuurina

sovelluskontin välissä, mutta layer 3 hallintayhteys konttipalomuurille säilyisi edelleen. Juniperin cSRX konttipalomuurituotteen testaamista tullaan jatkamaan yrityksessä vielä tulevaisuudessa, kunhan valmistaja on ensin tehnyt lupaamansa korjaukset tuotteeseen, jotta se toimisi myös halutusti klusterin sisällä.

Lähteet

Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M. & Steinder, M. 2015. Docker Containers across Multiple Clouds and Data Centers. IEE sähköinen kirjasto 10.12.2015. Viitattu 23.1.2019. https://ieeexplore-ieee org.ezproxy.jamk.fi:2443/document/7431433

Bernstein, D. 2014. Containers anc Cloud: From LXC to Docker to Kubernetes. IEE sähköinen kirjasto 1.9.2014. Viitattu 7.2.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/7036275

Changing the Size of a cSRX Container. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 11.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/task/multi-task/security-csrx-linux-server-docker-csrx-size.html

Chelladhurai, J., Chelliah, P. & Kumar, S. 2016. Securing Docker Containers from Denial of Service (DoS) Attacks. IEEE sähköinen kirjasto 2.7.2016. Viitattu 13.1.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/7557545

Church, M. 2016. Understading Docker networking drivers and their use cases. Julkaistu 19.12.2016. Viitattu 14.3.2019. https://blog.docker.com/2016/12/understanding-docker-networking-drivers-use-cases/

Cimpanu, C., 2018. 17 Backdoored Docker images removed from Docker hub. Julkaistu 13.7.2018. Viitattu 13.3.2019. https://www.bleepingcomputer.com/news/security/17-backdoored-docker-images-removed-from-docker-hub/

Clincy, V. & Shahriar, H. 2018. Web Application Firewall: Network Security Models and Configuration. IEE sähköinen kirjasto. Viitattu 28.2.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/8377769

Configuring CPU Affinity for a cSRX Container. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 11.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/task/multi-task/security-csrx-linux-server-docker-cpu-affinity.html

Configuring Traffic Forwarding on a cSRX Container. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 12.7.2018. Viitattu 13.3.2019 https://www.juniper.net/documentation/en_US/csrx/topics/task/multi-task/security-csrx-linux-server-docker-forwarding-mode.html

cSRX Environment Variables Overview. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 12.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/reference/general/security-csrx-environment-variables.html

Desikan, T., Gummaraju, J. & Turner, Y. 2015. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. Banyanops websivut. Viitattu 10.1.2019. https://www.banyanops.com/blog/analyzing-docker-hub/

Docker overview. 2017. Artikkeli Docker docs-verkkosivulla. Viitattu 19.1.2019. https://docs.docker.com/v17.09/engine/docker-overview/

Dockerfile reference. N.d. Artikkeli Docker docs-verkkosivulla. Viitattu 19.1.2019. https://docs.docker.com/engine/reference/builder/

Docker Registry. N.d. Artikkeli Docker-docs verkkosivuilla. Viitattu 19.1.2019. https://docs.docker.com/registry/

Docker Security Vulnerabilities. N.d. CVE-haavoittuvuustietokanta. Viitattu 13.3.2019. https://www.cvedetails.com/vulnerability-list/vendor_id-13534/product_id-28125/Docker-Docker.html

Duan, G. N.d. Next Generation Firewall vs. Container Firewall. Artikkeli NeuVectorin verkkosivuilla. Viitattu 28.2.2019. https://neuvector.com/network-security/next-generation-firewall-vs-container-firewall/

Duan, G. & Huang, F. N.d. How to Deploy a Docker Container Firewall. Viitattu 15.1.2019. https://neuvector.com/docker-security/how-to-deploy-a-docker-container-firewall/

Duan, G. N.d. Webb application firewall vs. Container firewall. Viitattu 14.3.2019. https://neuvector.com/network-security/web-application-firewall-vs-container-firewall/

Enabling Persistent Log File Storage to a Linux Host Directory. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 11.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/task/configuration/security-csrx-persistent-log-files.html

Get Started, Part 4: Swarms. N.d. Artikkeli Dockerin sivuilla. Viitattu 13.3.2019. https://docs.docker.com/get-started/part4/

HyungJik, L & JeunWoo, L. 2010. Design for management software of desktop virtualization solutions. IEEE sähköinen kirjasto 17.10.2010. Viitattu 11.1.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/5674779

Jatin, A. 2017. Container Technologies Overview. Julkaistu 8.7.2017. Viitattu 23.1.2019. https://dzone.com/articles/container-technologies-overview

Jimenez, I., Maltzahn, C., Lofstead J., Moody, A., Mohror, K., Arpaci-Dusseau, R. & Arpaci-Dusseau, A. 2016. Characterizing and Reducing Cross-Platform Performance Variability Using OS-Level Virtualization. IEE sähköinen kirjasto 23.3.2016. Viitattu 14.3.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/7529983

Kuoppala, E. 2017. Tietoturva Kubernetes-konttiympäristössä. Opinnäytetyö. Jyväskylän ammattikorkeakoulu, tietotekniikan tutkinto-ohjelma. Viitattu 13.3.2019. https://www.theseus.fi/bitstream/handle/10024/139793/Kuoppala_Eerik.pdf?sequence=1&isAllowed=y

Network Virtualization. N.d. Artikkeli Techopedian verkkosivuilla. Viitattu 19.1.2019. https://www.techopedia.com/definition/655/network-virtualization

Neupane, J., Haddad, R. & Chen, L. 2018. Next Generation Firewall for Network Security: A Survey. IEEE sähköinen kirjasto 19.4.2018. Viitattu 14.3.2019. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/8478973

Nilesh Mojidra. 2016. Stateful vs. Stateless Firewalls. Cybrary yrityksen verkkosivut 29.6.2016. Viitattu 4.2.2019. https://www.cybrary.it/0p3n/stateful-vs-stateless-firewalls

Organisaatio. N.d. Kansaneläkelaitoksen verkkosivut. Viitattu 13.11.2018 https://www.kela.fi/organisaatio

Overview. N.d. Artikkeli Openshiftin verkkosivuilla. Viitattu 13.3.2019. https://docs.openshift.com/container-platform/3.11/architecture/index.html

Overview of Docker Compose. N.d. Artikkeli Dockerin verkkosivuilla. Viitattu 13.3.2019. https://docs.docker.com/compose/overview/

Portnoy, M. 2016. Virtualization Essentials Second Edition. Indianapolis, Indiana: John Wiley & Soncs, Inc.

Server Virtualization. N.d. Artikkeli Techopedian verkkosivuilla. Viitattu 19.1.2019. https://www.techopedia.com/definition/688/server-virtualization

Specifying the packet I/O Driver for a cSRX container. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 12.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/task/multi-task/security-csrx-linux-server-docker-packet_io_driver.html

Understand container communication. N.d. Artikkeli Dockerin verkkosivuilla. Viitattu 14.3.2019. https://docs.docker.com/v17.09/engine/userguide/networking/default_network/container-communication/#container-communication-between-hosts

Understanding cSRX with a Bare-Metal Linux Server. 2018. Artikkeli Juniper Networksin sähköisessä kirjastossa. Julkaistu 11.7.2018. Viitattu 13.3.2019. https://www.juniper.net/documentation/en_US/csrx/topics/concept/security-csrx-docker-overview.html

Virtualization. N.d. Artikkeli VMwaren verkkosivuilla. Viitattu 10.1.2019. https://www.vmware.com/solutions/virtualization.html#how-it-works

What is Firewall?. N.d. Artikkeli Ciscon verkkosivuilla. Viitattu 4.2.2019. https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html

Wilkins, S. 2013. Stateful Firewall Fundamentals: A Better, Easier, More Secure Firewall. Julkaistu 9.1.2013. Viitattu 4.2.2019. https://www.pluralsight.com/blog/it-ops/stateful-firewall-fundamentals