jamk.fi

Korkean saavutettavuuden kuormantasaaja klusteri

Ville Pohjola

Opinnäytetyö
Toukokuu 2020
Tekniikan ala
Insinööri (AMK), Tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Pohjola, Ville	Opinnäytetyö, AMK	Kuukausi Vuosi
	Sivumäärä	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Korkean saavutettavuuden kuormantasaaja klusteri

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Mika Rantonen, Jani Immonen

Toimeksiantaja(t)
Pinja Oy

Tiivistelmä

Opinnätetyön toimeksiantajana toimi Pinja Oy, joka on suomalainen digitalisaation ja teollisuuden uudistamisen kumppani. Tavoitteena oli tutkia Korkean saatavuuden kuormantasausjärjestelmiä ja tuottaa Pinja Oy:lle tällainen kuormantasaajaklusteri hankitun tiedon perusteella. Työn toteutustapa oli kuormantasausta tekevän klusterin osalta ennalta määritelty Pinjalta jo löytyvän, Corosyncillä ja Pacemakerilla toteutetun, kuormantasaajaklusterin kaltaiseksi. Tästä syystä työn tutkimustehtävänä oli löytää jo olemassa olevasta ratkaisusta parannuskohteita ominaisuuksien ja hallittavuuden kannalta.

Kuormantasaajaklusteri toteutettiin rakentamalla Pinjan virtualisointiympäristöön Centos palvelimista klusteri käyttämällä Corosync ja Pacemaker klusterointiteknologioita. Klusterin hallintatyökaluksi valittiin Ansible ja tälle luotiin tarvittavat Playbookit. Lisäksi klusterin erityisvaatimuksena oli kyetä käsittelemään Letsencryptin sertifikaatteja. Tätä varten luotiin Shell-skripti ja Ansible playbook. Tuloksena saatiin konfiguraatioiden osalta hallittavampi ja versionhallintaa tukeva järjestelmä, jossa useita Pinjan edellisen kuormantasausklusterin puutteita oli korjattu ja jonka ylläpitoa saatiin yksinkertaistettua.

Syy miksi, kuormantasaajaklusterin hallintaan valittiin Ansible oli, että se on helposti jatkokehitettävissä. Muun muassa kaikki hallintaan liittyvät rutiinitoimet kuormantasaajalla voitaisiin korvata automatisoimalla ne Ansiblella. Samoin Letsencryptin sertifikaattien hankkimista ja hallintaa voisi Ansiblen avulla jatkokehittää.

Avainsanat (asiasanat)

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Pohjola, Ville	Bachelor's thesis	Month Year
		Language of publication:
	Number of pages	Permission for web publication: x

Title of publication
High Availability Load-balancer Cluster

Degree programme
Data Network Technology

Supervisor(s)
Mika Rantonen, Jani Immonen

Assigned by
Pinja Oy

Abstract

The thesis was assigned by Pinja Oy which is a Finnish partner for digitalization and reforming of industry. The goal was to study high availability load balancing systems and to provide Pinja Oy a load balancing cluster with the acquired knowledge from these studies and trough experience from Pinjas earlier load balancing cluster. The manner of creating the load balancing cluster was set in advance to be like the load balancing cluster, created with Corosync and Pacemaker, Pinja already had. For this reason, the investigative mission of the thesis was to find points of improvement from the load balancing cluster when it comes to creating and maintaining the cluster and what are its abilities.

The load balancing cluster was made by creating a cluster from Centos servers with Corosync and Pacemaker clustering technologies to Pinja's virtual environment. Ansible was decided to be the management tool for the cluster and necessary playbooks were created for it. In addition, the special requirement for the cluster was the ability to handle Letsencrypt certificates. For this reason, a Shell-script and another Ansible playbook was created. As a result, a more manageable and easier maintained system was created where many of the Pinja's old load balancing cluster's shortcomings were fixed.

The reason for picking Ansible as a management tool was that it can be easily further developed. For example, the many routine management task on the load balancer could be automated using Ansible. Also, acquirement and the management of the Letsencrypt certificates could be further developed.

Keywords/tags (subjectshttp://vesa.lib.helsinki.fi/)

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

ACL	Access Control List
DMZ	Demilitarized Zone
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ID	Identifier
IP	Internet Protocol
ISGR	Internet Security Research Group
JSON	JavaScript Object Notation
SLA	Service-Level Agreement
SSH	Secure Shell
SSL	Secure Sockets Layer
STONITH	Shoot The Other Node In The Head
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UUID	Universally Unique Identifier
VLAN	Virtual Local Area Network
YAML	Yet Another Markup Language, YAML Ain’t Markup Language

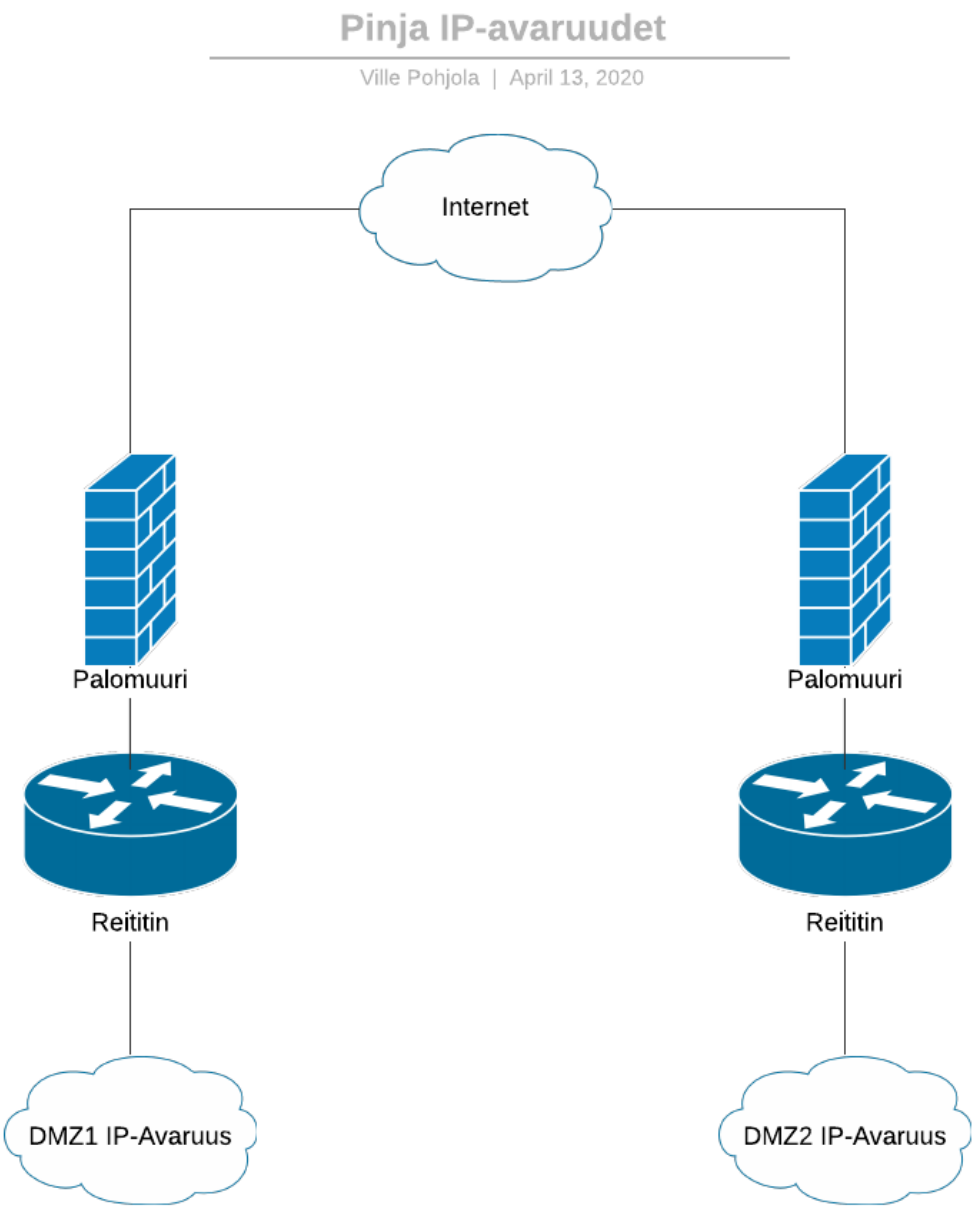
1 Johdanto

1.1 Yritys

Opinnäytetyön toimeksiantaja on suomalainen digitalisaation ja teollisuuden uudistamisen kumppani Pinja. Pinja tarjoaa näkemystä liiketoiminnan haasteisiin ja näkökulmaa tulevaisuuden teknisiin ratkaisuihin sekä vie tästä syntyneet ideat käytäntöön. Suomessa lähes 500 henkilöä työllistävä Pinja syntyi, kun Protacon, ARROW, SWD, Descal, Netwell, Vision Systems sekä Powen yhdistyivät. Pinjan pääkonttori sijaitsee Jyväskylässä. Pinja tuottaa asiakkailleen digitalisaation elinkaaripalveluita, johon kuuluvat muun muassa: ohjelmistokehitys, ylläpito, pilvialusta-, ict-, tietoturva- ja tukipalvelut. Alun perin vuonna 1990 perustettu Pinja, toimii yli kymmenessä toimipisteessä ympäri maata ja palvelee asiakkaita yli kolmessakymmenessä maassa. Vuonna 2019 Pinjan liikevaihto oli noin 40 miljoonaa euroa. (Pinjan tarina 2020.)

1.2 Toimeksianto

Toimeksiantona oli asentaa Pinjan tuotantoympäristöön korkean saatavuuden kuormantasaajaklusteri (High availability load balancing cluster) palvelemaan julkisesta verkosta tulevaa HTTP (HyperText Transfer Protocol) ja TCP (Transmission Control Protocol) liikennettä. Kuten kuviosta 1 selviää, Pinjalla on tuotantoympäristössä olemassa kaksi julkisen verkon IP-avaruutta (Internet Protocol), jotka on jaettu kahteen loogiseen ympäristöön IP-avaruuden perusteella (Ks. Kuvio 1). IP-avaruuksilla DMZ1 (Demilitarized Zone) ja DMZ2 on kullakin oma palomuuri. Pinjalla on olemassa kuormantasaajaklusteri palvelemaan DMZ1:tä, mutta on tullut tarve tarjota kuormantasausta myös DMZ2 verkon IP-osoitteille.



Kuvio 1. Pinjan IP-avaruudet.

1.3 Vaatimusmäärittely

Kuormantasaajan tarpeen ilmettyä tehtiin nopea vertailu fyysisten kuormantasaajien, kaupallisten virtualisointiympäristöön asennettavien palveluiden ja DMZ1:n kuormantasaajan kaltaisen avoimen lähdekoodin kuormantasaajan välillä. Lopulta toimeksiantaja päätyi avoimen lähdekoodin kuormantasaajaan sen edullisuuden vuoksi ja koska siitä löytyy yrityksessä jo aiempaa kokemusta.

Ylläpidettävyyden kannalta päätettiin rakentaa erillinen klusteri ja tehdä siihen parannuksia, sen sijaan, että liikenne olisi tasattu DMZ1 verkon kuormantasaajalla.

Kuormantasaajan tuli olla arkkitehtuuriltaan sellainen, että yhden kuormantasaaja komponentin rikkoutuminen ei vaikuta kuormantasaajan toimivuuteen. Kuormantasaajan tuli suoriutua vähintään 100Mb/s (Megabittiä sekunnissa) liikenteestä tilanteessa, jossa yksi tai useampi kuormantasaaja komponenteista on rikki tai saavuttamattomissa. Sen piti kyetä tasaamaan SSL-salattua (Secure Sockets Layer) HTTP ja TCP liikennettä ja pystyä käsittelemään Letsencrypt:n tarjoamia sertifikaatteja.

1.4 Tutkimusmenetelmät

Tutkimuksen tarkoituksena oli etsiä aiheeseen liittyvää materiaalia alan johtavilta toimijoilta heidän omista artikkeleistaan ja suorittaa vertailua eri kuormantasaus menetelmien välillä sekä tarkastella näiden menetelmien sopivuutta, peilaten yrityksen tarpeisiin ja kustannuksiin. Lisäksi käytettiin hyväksi jo olemassa olevaa tietotaitoa ja nämä yhdessä muodostivat tutkimuksen tietoperustan. Tämän aineiston perusteella muodostettiin käytänne (best practise), jota lähdettiin toteuttamaan, tarpeisiin soveltaen, käytäntöön. Tutkimuksessa on viitteitä enemmän laadullisesta tutkimuksesta, kuin määrällisestä, sillä se on luonteeltaan aineistopohjaista eikä sen koko täytä määrällisen tutkimuksen kriteereitä, eikä tulosta voi tilastollisesti yleistää. (Anita Saaranen-Kauppinen & Anna Puusniekka. 2006c)

Tutkimuksessa on viitteitä Ankkuroidusta teoriasta (Grounded theory), sillä siinä tarkastellaan kokonaisuutta osakomponenttien summana. Lisäksi tutkimus on luonteeltaan aineistolähtöinen ja sitä hankitaan, kunnes aineisto saturoituu ja ei tuo enää uutta tietoa. Aineistosta on tähän tutkimukseen teoriaksi valittu kattavin yksittäinen tietolähde. (Anita Saaranen-Kauppinen & Anna Puusniekka. 2006b)

Tutkimuksessa on myös Diskurssianalyysin piirteitä sillä siinä vertaillaan eri lähdeaineistojen tekstejä ja pyritään näiden pohjalta luomaan käytänteitä ja

hahmottamaan merkityssuhteita kokonaisuuden kannalta. Oleellista on myös, miten faktat aineistossa rakentuvat ja kuinka ne esitellään. (Anita Saaranen-Kauppinen & Anna Puusniekka. 2006a)

2 Korkean saatavuuden kuormantasaus klusteri

2.1 Korkea saatavuus

2.1.1 Yleistä korkeasta saatavuudesta

Korkealla saatavuudella (High Availability) tarkoitetaan järjestelmää tai järjestelmän komponenttia, joka on jatkuvasti toimintakykyinen tai saatavilla. Tätä voidaan mitata siten, että suhteutetaan saatavuutta ajallisesti siihen, että palvelu olisi 100% aina saatavilla. Tietotekniikassa usein pyritään saavuttamaan, niin sanottu, viiden yhdeksikön (99.999%) saatavuus. Tähän päästäkseen järjestelmän ja sen komponentien tulee olla hyvin testattuja ennen käyttöönottoa. Korkean saatavuuden suunnittelu keskittyy usein varajärjestelmien ympärille, koska järjestelmät ja verkot koostuvat useista osista. (High availability 2019.)

Saatavuutta lasketaan tyypillisesti vähentämällä keskimääräisen täyden kuukauden minuuteista minuutit, kun järjestelmä ei ole ollut saatavilla, jakamalla tämä erotus keskimääräisen täyden kuukauden minuuteilla ja sen jälkeen kertomalla sadalla. Tämä tarkoittaa, että saavuttaakseen viiden yhdeksikön saatavuuden palvelu tai järjestelmä saa olla keskimääräisessä kuukaudessa enintään 26 sekuntia tavoittamattomissa. (How availability is measured 2019.)

Korkean saatavuuden järjestelmän olisi hyvä kyetä nopeasti toipumaan häiriöistä ja minimoida loppukäyttäjään vaikuttavat tapahtumat. Parhaita käytänteitä korkean saatavuuden kannalta ovat muun muassa:

- Eliminoida mahdolliset yksittäiset pisteet, joiden häiriö estää tai keskeyttää koko palvelun tai järjestelmän käytön.

- Taata järjestelmän varmistukset, että järjestelmä on helposti ja nopeasti palautettavissa.
- Käyttää kuormantasausta palvelu- ja verkkoliikenteessä.
- Valvoa järjestelmää poikkeustilanteiden varalta ja järjestää hälytykset.
- Hajauttaa järjestelmä maantieteellisesti useaan lokaatioon, sähkökatkojen, poliittisten tilanteiden tai luonnon aiheuttamien häiriöiden varalta. (How to achieve high availability 2019.)

Varmistusten ottaminen on erityisen tärkeää, kun halutaan, että järjestelmä on nopeasti palautettavissa järjestelmäkatastrofin jälkeen ja, kun halutaan välttää datan häviöitä. (Mt.)

2.1.2 Korkean saatavuuden mallit

Korkean saatavuuden saavuttamiseksi on päälajeittain olemassa neljä erilaista mallia (High availability models). Näistä voidaan valita resursseihin ja tarvittavan SLA-tason (Service-Level Agreement) saavuttamiseen sopiva malli. Mallit on kuvattuna node-tasolla, mutta ne voidaan toteuttaa myös kokonaisen järjestelmän tasolla.

Kuormantasaus

Kuormantasausmallissa, sekä primääri, että sekundaari node tai nodet osallistuvat järjestelmän pyyntöjen prosessointiin yhtä aikaa. Tällä saavutetaan täysi katkottomuus palvelussa tai katko on niin lyhyt, että käyttäjä ei sitä huomaa. (High availability models 2020.)

Kuuma valmius

Kuuma valmius -mallissa sekundaari nodeen on asennettu kaikki tarvittavat ohjelmistot ja komponentit. Sekundaari node on myös käynnistetty ja valmiina vastaanottamaan dataa, kun primääri node kaatuu. Tällä saavutetaan järjestelmähäiriön toipumisajaksi joitain sekunteja. (Mt.)

Lämmin valmius

Lämpimän valmiuden tilassa olevalla sekundaari nodella on samat ominaisuudet, kun kuumalla nodella, mutta lämpimässä valmiudessa ohjelmistot ja palvelut

käynnistetään vasta, kun primääri node kaatuu. Tässä mallissa järjestelmähäiriön toipumisaikaan lisätään näiden palveluiden käynnistymiseen menevä aika. (Mt.)

Kylmä valmius

Kylmä valmius tarkoittaa, että sekundaari nodelle asennetaan ja konfiguroidaan palvelut ja ohjelmistot vasta, kun primääri node kaatuu ensimmäistä kertaa. Myöhemmin, jos primääri node jälleen kaatuu sekundaari node käynnistetään uudelleen. Tässä mallissa järjestelmähäiriön kesto on sekundaari noden asentamiseen ja myöhemmin käynnistämiseen menevä aika. (Mt.)

2.1.3 Klusterointitopologiat (Clustering topologies)

Klusteritopologiat luokitellaan niiden tarjoaman korkean saatavuuden tason mukaan. Topologialla voit määrittää redundanssitason, jonka tarvitset ohjelmisto- tai laitevirheen yhteydessä ja klusterin hallinnointiin voidaan käyttää erinäistä klusterinhallinta ohjelmistoa. Näiden klusterinhallinta ohjelmistojen avulla voit vähentää järjestelmän palautumisaikaa. (High availability models 2020.)

N + 1

N + 1 topologiassa on yksi ylimääräinen sekundaari node, joka on valmis ottamaan vastaan kaatuneen noden roolin. Jos primääri nodeilla ajetaan eri ohjelmistoja keskenään, varmistavan noden pitää sisältää kunkin primääri noden rooliin vaadittavat ohjelmistot ja konfiguraatiot. Yhden palvelun tai primääri noden klusterissa tätä voidaan käyttää yksinkertaisena aktiivi-passiivi roolituksena. (Mt.)

N + M

Yhden sekundaarinoden ollessa riittämätön turvaamaan järjestelmän saatavuus häiriön sattuessa voidaan asentaa sekundaari nodeja useampi tuomaan redundanssia järjestelmään. Tässä topologiassa on tulee miettiä mikä on sopiva redundanssitaso järjestelmän luotettavuuden kannalta, koska jokainen node lisää kustannuksia. (Mt.)

N-to-1

Tässä topologiassa sekundaari node toimii väliaikaisena korvikkeena, kunnes primääri node saadaan palautettua toimintakuntoon. Primääri noden palautuessa palvelut ja instassit pitää uudelleen aktivoida, että korkeasaatavuus saadaan palautettua. (Mt.)

N-to-N

N-to-N klusteritopologiassa klusteri jakaa kaatuneen noden palvelut klusterin muiden nodejen kesken. Tällä saavutetaan se, että klusteri ei tarvitse toimeettomia sekundaari nodeja, mutta aktiivisilla nodeilla pitää olla riittävä kapaisteetti ottaa vastaan kaatuneen noden palvelut ja kuorma. (Mt.)

2.1.4 Corosync

Corosync Cluster Engine on järjestelmäryhmien kommunikointityökalu, jota käytetään korkean saatavuuden kehyksenä projekteille, kuten Pacemaker ja Asterisk. Siinä on ominaisuuksia joilla voidaan tarjota korkeaa saatavuutta applikaatioille. Se tarjoaa C-applikaatio-ohjelmointirajapinnoille ominaisuuksia, kuten:

- Suljettu prosessiryhmien kommunikointimalli
- Yksikertaisen saatavuushallinnan, joka käynnistää kaatuneen applikaation
- Statistiikka- ja konfiguraatiotietokannan, josta voi hakea ja vastaannottaa muutosnotifikaatioita ja informaatiota
- Päätösvaltajärjestelmän, joka notifioi applikaatioita, kun päätösvalta saavutetaan tai menetetään. (The Corosync Cluster Engine 2020.)

Corosync:iä käytetään yleisesti muodostamaan korkean saatavuuden klusteri yhdessä Pacemakerin kanssa. (Mt.)

2.1.5 Pacemaker

Pacemaker on korkean saatavuuden klusterin hallintatyökalu, jolla hallitaan klusterissa pyöriviä resursseja. Se saavuttaa korkean saatavuuden huomaamalla ja toipumalla node- ja resurssi-tason häiriöistä käyttämällä Corosyncin tarjoamia kommunikointityökaluja. Se pystyy tekemään tätä luotettavasti käytännössä minkä tahansa kokoisissa klustereissa ja tarjoaa mallin, joka antaa järjestelmän ylläpitäjälle hyvät työkalut hallinoida ja roolittaa klusterin resursseja. (Pacemaker 2018.)

2.1.6 Fencing / STONITH (Shoot the other node in the head)

Fencing on oleellinen korkean saatavuuden komponentti, joka usein jää muuta kuormantasaajaklusteria vähemmälle huomiolle, koska se ei juurikaan anna mitään käyttäjälle näkyvää toiminnallisuuta. Fencinglillä tarkoitetaan ominaisuutta klusterissa, jossa valvotaan nodien tilaa ja huomattaessa poikkeavuutta reagoidaan tähän poikkeavaan tilaan jollain tavalla. Kun yhden noden tilaa ei voida varmistaa, tehdään tälle Fencing-operaatio. Tällä koitetaan välttää aivohalkiotila (Split-brain), jossa klusteri jakaantuu kahteen, tai useampaan, toiminnalliseen klusteriin. (Fencing and STONITH 2020.) Esimerkki: yksi kolmesta N-to-N klusterin nodesta menettää yhteyden muuhun klusteriin. Tämän noden näkökulmasta muut klusterin nodet ovat alhaalla ja tämä klusterista erkaantunut node yrittää ottaa näiden palvelut itselleen. Samoin muut kaksi nodea yrittävät ottaa yhteytensä menettäneen noden palvelut itselleen. Tällöin joudutaan tilanteeseen, jossa jokainen palvelu on käynnistetty klusterissa kaksi kertaa. IP-resurssin tapauksessa kaksi palvelinta vastaisi samaan IP-osoitteeseen, jolloin palvelun saatavuus voi estyä osittain tai jopa kokonaan.

Nodetason Fencing

Noden tasolla fencing tarkoittaa, että klusteri yrittää estää epäonnistuneen noden käynnistämästä palveluita ollenkaan. Tämä toteutetaan yleensä STONITH (Shoot The Other Node In The Head) toiminnallisuudella. STONITH:ssa muut nodet poistavat toimimattoman noden klusterissa pakottamalla tälle uudelleenkäynnistyksen. (Mt.)

Resurssitason Fencing

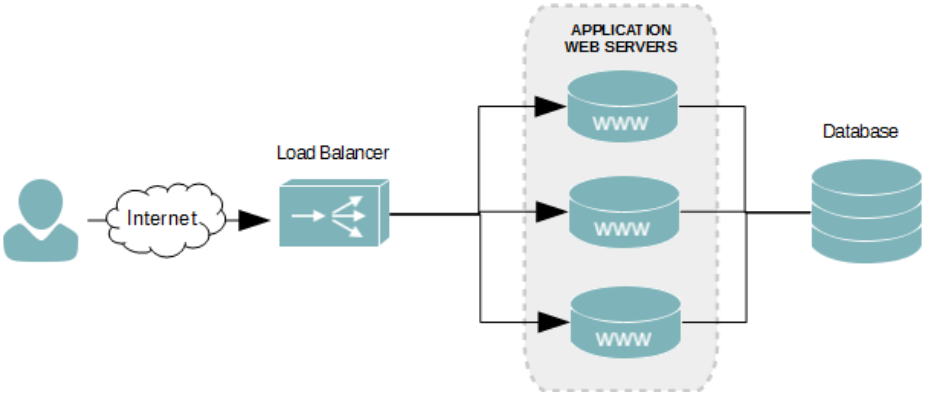
Resurssitason fencingillä tarkoitetaan fencingiä, jossa epäonnistuneelta nodelta estetään pääsy klusteroituihin resursseihin, kuten esimerkiksi katkaistaan yhteys kytkimeen ja tätä kautta jaettuihin levyihin. Tällä varmistetaan, että vain yksi klusterin nodeista voi käydä kirjoittamassa johonkin palveluun liittyvät muutokset levylle.

2.2 Kuormantasaus

2.2.1 Yleistä kuormantasauksesta

Kuormantasauksella tarkoitetaan tietotekniikassa verkkoliikenteen jakamista useamman vastaanottavan taustapalvelimen kesken. Verkkosivujen täytyy usein palvella parhaimmillaan jopa miljoonia käyttäjiä yhtä aikaisesti nopeasti ja luotettavasti. Data voi olla esimerkiksi kuvia ja videoita, jolloin liikennettä voi generoitua huomattavia määriä. Tällöin otetaan käyttöön erinäisiä metodeja, kuorman tasaamiseksi useammalle vastaanottavalle palvelimelle.

Kuormantasaaja toimii, kuin vuoronumeroautomaatti, joka ohjaa asiakkaat järjestykseesä vapaana olevalle asiakaspalvelijalle, joka vastaavasti käy tietokannasta hakemassa asiakkaan haluaman tiedon tai tekemässä sinne muutoksen. Kuormantasauksen toiminta on esitetty kuviossa 2. Kuormantasaaja siis jakaa kuormaa palvelupyyntöjä vastaanottavien palvelinten kesken siten, että palvelu olisi mahdollisimman tehokkaasti saatavilla. Kuormantasaaja antaa myös mahdollisuuden lisätä ja vähentää palvelimia jakamaan kuormaa kuorman määrän mukaan, ja auttaa tällä takaamaan korkean saatavuuden. Esimerkiksi tiedettyinä ruuhka-aikoina voidaan kapasiteettiä lisätä ja sen ulkopuolella vähentää. Kuormantasaajalla liikennettä voidaan myös siirtää palvelimelta toiselle. Tällä voidaan mahdollistaa rullaten päivittäminen (rolling update), missä palvelimet päivitetään yksi kerrallaan kuormantasaajalla, liikennettä palvelimelta toiselle siirtämällä ilman, että palvelu katkeaa tai, että tämä päivittäminen näkyy asiakkaalle. (What Is Load Balancing 2020.)



Kuvio 2. Kuormantasauksen toiminta. (litespeed-cluster 2019.)

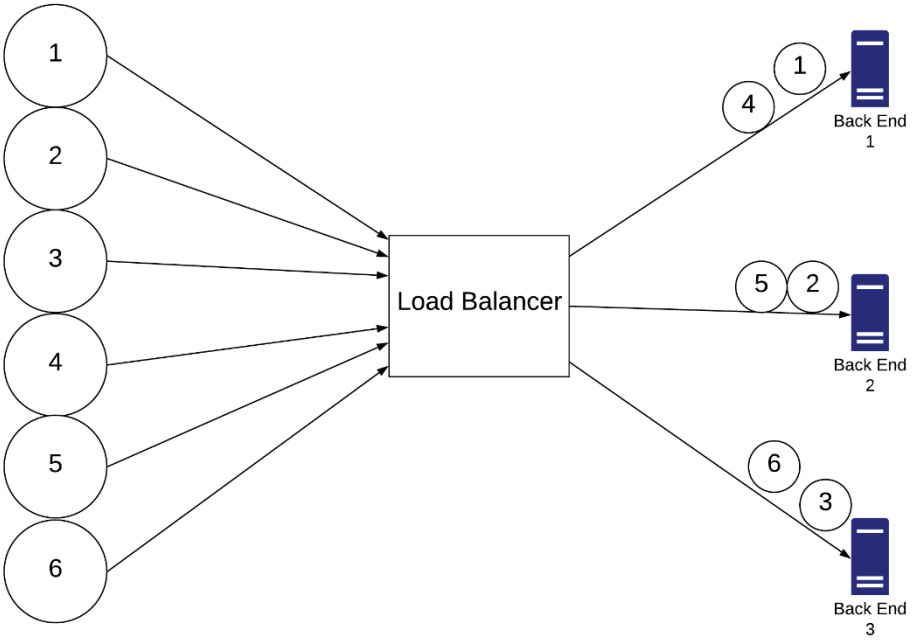
Kuormantasaajia on sekä laitteisto-, että ohjelmistopohjaisia. Laitteistopohjaisten kuormantasaajien etuna ovat usein kuormantasaukselle optimoitu ja pyhitetty laite, joka pystyy tätä myötä suoriutumaan suuremmasta kuormasta, kuin ohjelmistopohjainen kuormantasaaja. Ohjelmistopohjaisen kuormantasaajan etuja ovat muunneltavuus, joustavuus ja edullisuus. Ohjelmistopohjaisen kuormantasaajan voi asentaa haluamaansa laitteseen tai virtuaaliympäristöön. (Mt.)

2.2.2 Kuormantasausalgoritmit

Kuormaa tasattaessa voidaan käyttää erimuotoisia algoritmejä tarpeen mukaan. Eri algoritmit tarjoavat muokattavuutta kuormantasaukseen ja asymmetrisen liikenteen ohjauksen palvelimille. Kuormantasausalgoritmejä ovat muun muassa kiertovuorottelu, painotettu kiertovuorottelu, vähiten yhteyksiä, painotettu vähiten yhteyksiä ja lähde IP tarkiste. (Load Balancing Tecniques 2020.)

Kiertovuorottelu (round robin)

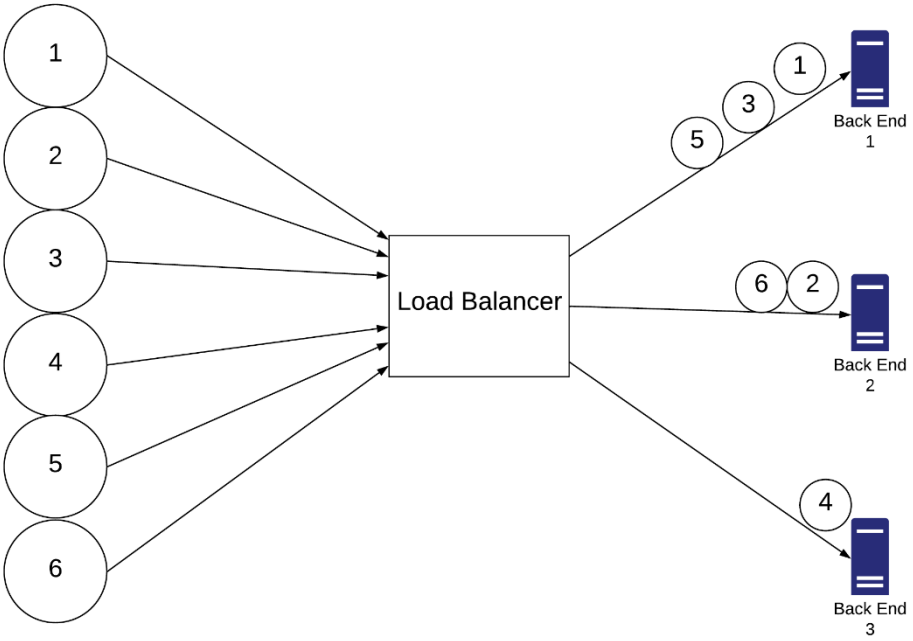
”Kiertovuorottelussa” kuorma jaetaan palvelimille järjestelmällisesti vuorotellen. Palvelimet ottavat siten aina yhtä paljon yhteyksiä vastaan. (Load Balancing Tecniques 2020.) Tästä esimerkki on nähtävissä kuviosta 3.



Kuvio 3. ”Kiertovuorottelu” alrogitmin toiminta.

Painotettu kiertovuorottelu (weighted round robin)

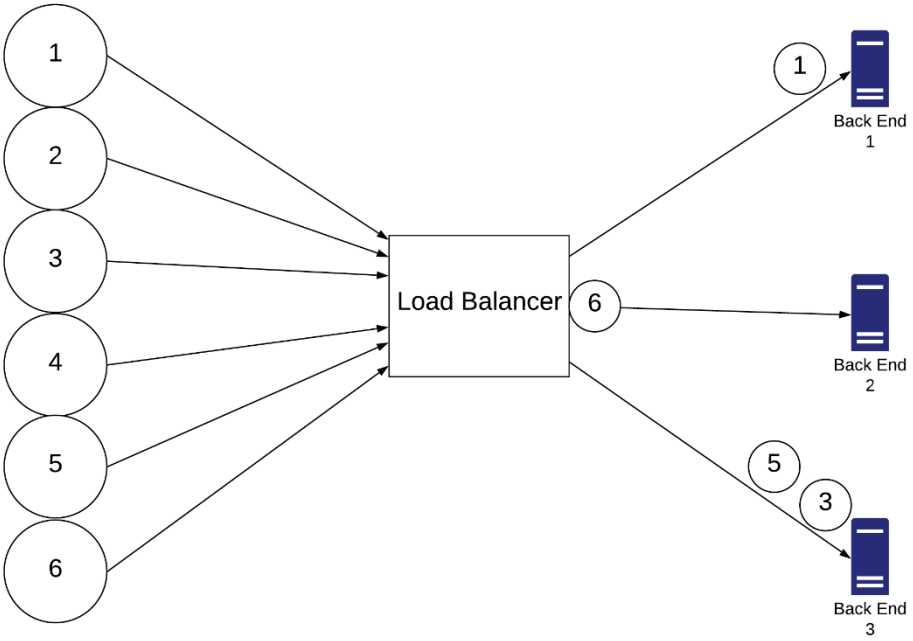
Muuten sama, kuin ”kiertovuorottelu”, mutta palvelimia voidaan painottaa niin, että ne ottavat vastaan vaihtelevan määrän paketteja. Tällä voidaan ottaa huomioon palvelimien muut tehtävät, tai jos halutaan antaa joillekin palvelimille enemmän resursseja. Kuvion 4 esimerkissä Back End 1 palvelimen painotukseksi on annettu 2 ja muiden palvelimien painotus on 1. Tästä seuraa se, että Back End 1 saa yhtä paljon kuormaa kuin muut palvelimet yhteensä (2 = 1 + 1). (Load Balancing Tecniques 2020.)



Kuvio 4. ”Painotetun kiertovuorottelu algoritmin” toiminta.

Vähiten yhteyksiä (least connection)

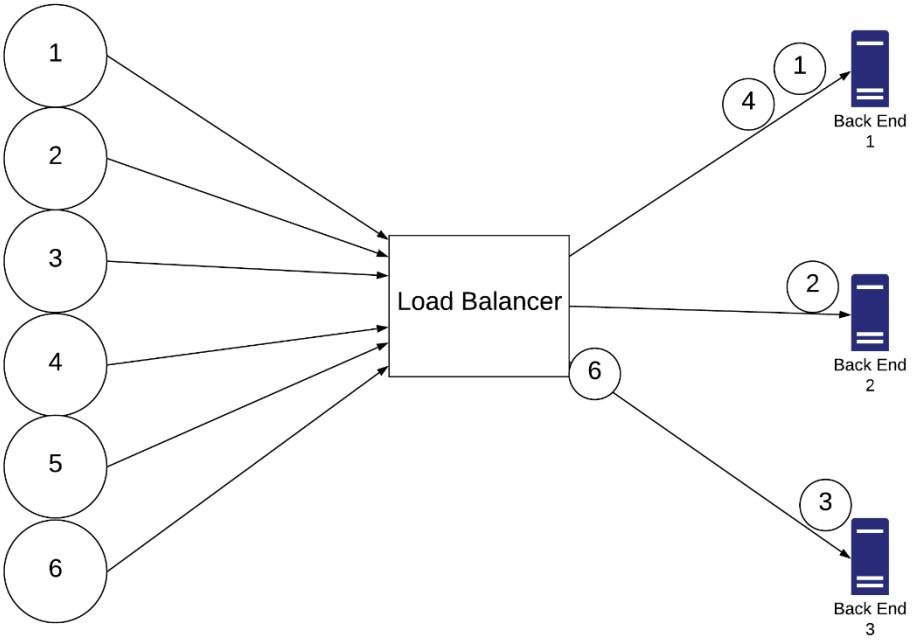
”Vähiten yhteyksiä”, on algoritmi, jossa palvelin missä on vähiten auki olevia yhteyksiä, vastaanottaa seuraavan yhteyden. Tällä algoritmillä voidaan ottaa huomioon kunkin palvelimen sen hetkinen kuorma. Kuviossa 5 on kuvattu tilanne missä Back End 2 palvelimella on vähemmän yhteyksiä auki kuin Back End 1:llä tai Back End 3:lla, joten seuraava yhteys ohjataan kuormantasaajalta sinne. (Load Balancing Tecniques 2020.)



Kuvio 5. ”Vähiten yhteyksiä” algoritmin toiminta.

Painotettu vähiten yhteyksiä (weighted least connection)

Tämä algoritmi on muuten sama, kuin ”vähiten yhteyksiä”, mutta kahdella tai useammalla ollessa sama määrä yhteyksiä, korkeimmin painotettu vastaanottaa yhteyden. Näin saadaan lisää hallittavuutta yhteyksien ohjaamiseen. Kuviossa 6 on kuvattu tilanne, missä kahdella palvelimella, Back End 2 ja Back End 3, on yhtäpaljon yhteyksiä auki. Back End 3:lle on kuitenkin annettu painoksi 2, kun Back End 2:n paino on 1. Seuraava paketti ohjataan näistä kahdesta painavemmalle eli Back End 3:lle. (Load Balancing Tecniques 2020.)



Kuvio 6. ”Painotettu vähiten yhteyksiä” algoritmin toiminta.

Lähde IP tarkiste (Source IP Hash)

Tässä algoritmissä luodaan lähde ja kohde IP-osoitteista uniikki tarkisteavain. Tätä avainta käytetään ohjaamaan liikenne aina samalle palvelimelle niin kauan, kun palvelin on kykenevä ottamaan vastaan liikennettä. Jos palvelimien määrä muuttuu, tarkiste lasketaan uudelleen. (Load Balancing Tecniques 2020.)

2.2.3 HAProxy (High Availability Proxy)

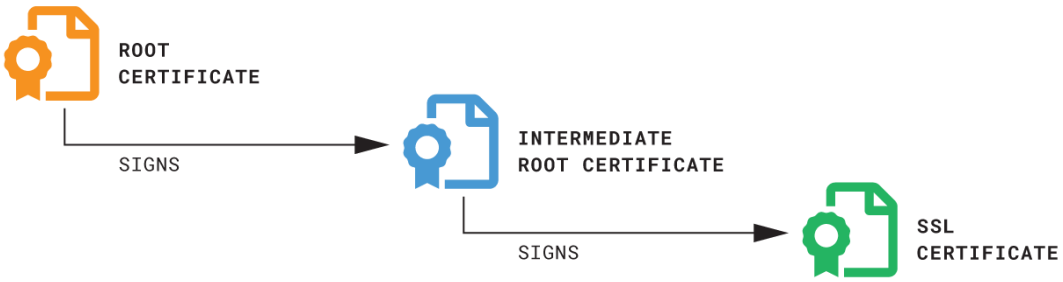
HAProxy on ilmainen ja luotettava avoimen lähdekoodin palvelu, jota pääasiassa käytetään TCP ja HTTP liikenteen kuormantasaukseen tai ohjaamiseen. Se on erittäin kevyt ja laajasti käytetty. Sen tehtävä on jakaa verkkoliikennettä eteenpäin taustajärjestelmille (Back End) ja toimia itse eduspalvelimena (Front End) TCP ja HTTP liikenteelle. Se kykenee salaamaan ja purkamaan SSL-liikennettä (Secure Sockets Layer) ja sillä voidaan tehdä käytännepohjaista (policy based) liikenteen ohjausta ACL:ien (Access Control List) avulla. Sitä julkaistaan useimmille linux jakeluille ja pilvialustat, kuten Google Cloud Platform ja Amazon Web Services, tarjoavat sitä oletuksena. (The Reliable, High Performance TCP/HTTP Load Balancer 2020.)

2.2.4 SSL / TLS

SSL tai nykyään TLS (transport layer security) on standardoitu teknologia tietoturvallisten yhteyksien luomiseen palvelimien tai palveluiden ja niitä käyttävän asiakkasohjelman tai asiakkaan välille. Tavallisesti tällaista yhteyttä käytetään esimerkiksi verkkoselaimessa, kun avataan tietoturvallinen verkkosivu. Ilman SSL-salausta liikenne kulkee verkon yli tekstimuotoisena. Jos potentiaalinen hyökkääjä pystyy kuuntelemaan tätä liikennettä, on kaikki informaatio helposti luettavissa ja käytettävissä. SSL liikenne vaatii, että palvelua tarjoava taho on asettanut palvelunsa käyttöön SSL-sertifikaatin ja, että asiakasohjelma luottaa tähän sertifikaattiin. (What is Secure Sockets Layer2020.)

SSL-Sertifikaatti

SSL-Sertifikaatti on luottamukseen perustuva sertifikaatti, jota käytetään SSL yhteyksien muodostamiseen. SSL-sertifikaatin voi luoda joko itse, tai ostaa luotetulta toimijalta heidän luoma sertifikaatti ja asettaa palvelin käyttämään sitä. SSL-Sertifikaatteja varten on luotu sertifikaatti auktoriteetti (Certificate Authority) hierarkia, jossa joillekin luotettaville toimijoille on annettu oikeus myöntää laajemmin luotettuja sertifikaatteja. Tällä on mahdollistettu, että esimerkiksi verkkosivu voi käyttää juurisertifikaatti auktoriteettinään (Root Certificate Authority) tällaista toimijaa ja antaa käyttäjille pääsyn sivustolle ilman, että he erikseen luottavat sivuston ylläpitäjän itseluomaansa sertifikaattiin, vaan lista luotetuista sertifikaateista on implementoitu selaimeen. Juurisertifikaatti auktoriteetit myöntävät myös välittäjä sertifikaatteja niin, että luotetut välittäjät voivat taas luoda palvelimille ja palveluille sertifikaatteja, ilman, että käytetään juurisertifikaattia yksinään näiden luomiseen. Tämä lisää sertifikaatin turvallisuutta. Kuviossa 7 on kuvattu tästä muodostuva sertifikaattiketju (Certificate Chain). (What is a Certificate Authority 2020.)

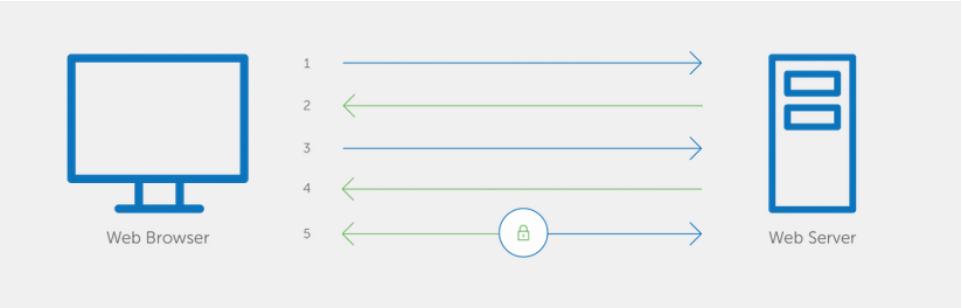


Kuvio 7. SSL-Sertifikaattiketju. (What is the SSL Certificate Chain?)

SSL-Kättely

Kun yritetään avata SSL-liikennettä asiakasohjelma, kuten verkkoselain, ja palvelin avaavat SSL-yhteyden käyttäen SSL-kättelyä (SSL-Handshake). Kättelyssä luodaan yhteydelle istuntoavain (Session key), koska salaus käyttäen privaatti- ja julkiavainta on laskennan kannalta työläs prosessi. Kuviossa 8 näkyy kättelyn eri vaiheet esitettynä SSL-yhteydessä, joka muodostetaan verkkoselaimen ja verkkopalvelimen välille. (Ks. Kuvio 8) Nämä vaiheet ovat seuraavat:

1. Verkkoselaimen ensimmäistä kertaa yhdistäessä verkkopalvelimeen, verkkoselain pyytää palvelua identifioimaan itsensä.
2. Verkkopalvelin lähettää verkkoselaimelle kopion SSL-sertifikaatistaan sisällyttäen siihen palvelimen julkisen avaimen.
3. Tämän jälkeen Verkkoselain katsoo sertifikaatin juuren ja vertaa sitä omien luotettujen juurisertifikaattien listaansa ja että se on yhä voimassa ja sertifikaatti on myönnetty juuri tälle verkkosivulle. Mikäli sertifikaatti on luotettava, verkkoselain luo symmetrisen istuntoavaimen, käyttäen verkkopalvelimen lähettämää julkista avainta ja lähettää sen verkkopalvelimelle.
4. Verkkopalvelin purkaa salauksen verkkoselaimen lähettämästä istuntoavaimesta ja lähettää kuittauksen verkkoselaimelle, että istunto on valmis alkamaan käyttäen luotua istuntoavainta.
5. Nyt verkkoselain ja verkkopalvelin voivat lähettää salattua liikennettä käyttäen tätä istuntoavainta. (How Does the SSL Certificate Create a Secure Connection?)



Kuvio 8. SSL-kättely. (Mt.)

2.2.5 Let’s Encrypt

Let’s Encrypt on ISRG:n (Internet Security Research Group) tarjoama ilmainen ja automatisoitu SSL-sertifikaatti auktoriteetti, joka antaa mahdollisuuden käyttää SSL salausta ilmaiseksi. Let’s Encryptin periaatteita ovat:

- Ilmaisuus
- Automaattisuus
- Tietoturvallisuus
- Läpinäkyvyys
- Avoimuus
- Yhteisöllisyys

Let’s Encrypt Tarjoaa kaikille Domain (Toimialue) nimen omistavalle taholle luotettavan SSL-sertifioinnin ilmaiseksi ja antaa palvelimen hankkia ja uusia sertifikaatit automaattisesti. Se tarjoaa alustan ottaa TLS:n parhaat käytänteet käyttöön ja kaikki sen tarjoamat ja kumoamat SSL-sertifikaatit ovat julkisesti tarkastettavissa. Sen automatisointiprotokolla on julkisesti saatavilla ja muiden käytettävissä ja Let’s Encrypt pyrkii toimimaan yhteisönsä kanssa hyvässä yhteistyössä kehittyäkseen. (About Let’s Encrypt.)

2.3 Keskitetty hallinta (Centralized Management)

2.3.1 Yleistä keskitetystä hallinnasta

Keskitetyllä hallinnalla tarkoitetaan tietotekniikassa järjestelmää, joka toimii koordinaattorina muille ympäristön järjestelmille tai niiden osakomponenteille. Yksi esimerkki tällaisesta järjestelmästä on keskitetty käyttäjienhallinta, jossa käyttäjiä hallinoidaan keskitetysti yhdestä järjestelmästä ja josta tieto jaetaan muille järjestelmille. Tällä mahdollistetaan se, että halutut käyttäjät voivat kirjautua samoilla kirjautumistiedoilla useampaan järjestelmään ja, että käyttäjään tai sen oikeuksiin tehdyt muutokset tapahtuvat välittömästi kaikille keskitettyä hallintaa käyttäville järjestelmille tai palveluille. Tämä on yleistettävissä muihinkin tietueisiin, kuin käyttäjiin, esimerkiksi konfiguraatioihin. Etuja keskitetyn hallinan tietojärjestelmissä on:

- Ylläpidettävyys
- Tietoturva
- Kontrolloitavuus
- Idempotenssius (tuottaa aina saman tuloksen)
- Skaalautuvuus

Keskitettyyn hallintaan löytyy tarpeesta riippuen useita erilaisia työkaluja. Jotkut järjestelmät käyttävät agentteja, jotka käyvät aina vetämässä (pull method) uudet tiedot määritettyinä ajankohtina. Toinen vaihtoehto on pusku mallinen (push model), jossa keskuspalvelin työntää uuden tiedon hallittaville järjestelmille tai palvelimille. (Bravo, M. 2018. Why use configuration management tools.)

2.3.2 Ansible

Ansible on Red Hatin sponsoroima avoimen lähdekoodin yhteisöprojekti tietojärjestelmien luontiin, konfigurointiin ja orkestrointiin. Se ei vaadi kohteena olevaan palvelimeen agentteja vaan kommunikointi tapahtuu SSH:n (Secure Shell) tai

muun tuetun yhteyden yli, joita voidaan tarvittaessa lisätä eri liitännäisillä. Se ei myöskään vaadi keskitettyä ohjauspalvelinta, josta haluttuja kohdepalvelimia käskytetään. Ainoat vaatimukset kohdepalvelimelle ovat yhteys käskyttävään palvelimeen ja python 2.6 tai uudempi tai python 3.5 tai uudempi. Käskyttävän palvelimen vaatimuksiin kuuluu tämän lisäksi, että sen käyttöjärjestelmä ei saa olla Windows. (Installing Ansible 2020.)

Ansiblea ohjataan yksinkertaisella YAML (Yet Another Markup Language) kielellä. Sitä voidaan käyttää erinäisten järjestelmäkomopnenttien, kuten infrastruktuurin, verkkojen, konttien ja Pilvialustojen, luontiin ja konfigurointiin.YAML:lla rakennetaan Ansibleen “pelikirjoja”, joilla voidaan konfiguroida, käyttöönottaa ja orkestroida järjestelmiä aina yhdestä koneesta kokonaisiin ympäristöihin. (How Ansible Works 2020.)

Pelikirjat (Playbooks)

Pelikirjoissa asetetaan kohdekoneille eri rooleja joita voi olla esimerkiksi tietokanta- tai edustapalvelin. Roolissa konfiguroidaan tähän rooliin kuuluvat toimet, kuten tietokannan asennus ja konfigurointi. Pelikirja sekä roolit voivat koostua useasta vaiheesta. Toiminnot voivat koostua erilaisista liitännäisistä, joilla on oma tarkoitus, kuten asentaa paketteja tai luoda kansiorakenteita. Ansibleen voidaan tuottaa liitännäisiä millä tahansa ohjelmointikielellä joka kykenee palauttamaan JSON:ia. (Working With Playbooks 2020.)

Ansible Tower

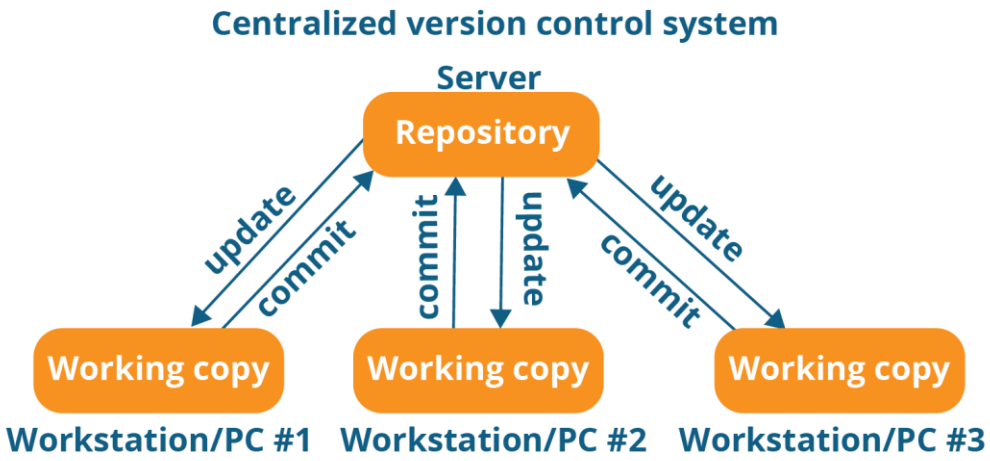
Ansible Tower on maksullinen Red Hatin verkkopohjainen järjestelmä Ansiblen managerointiin. Se tarjoaa helppokäyttöisen käyttöliittymän konfigurointien ja asennusten hallintaan ja sinne voidaan tehdä esimerkiksi ajastettuja toimintoja. Ansible Towerista on myös avoimen lähdekoodin ilmaisversio AWX. (Red Hat Ansible Tower 2020.)

2.3.3 Jinja2

Jinja2 on yksi käytetyimmistä mallinnusmoottoreista (Template Engine) Pythonille. Se on Djangon mallinnusjärjestelmän inspiroima. Jinja2 laajentaa tätä tehokkaammilla työkaluilla ja niiden lisäksi mahdollistaa hiekkalaatikossa ajamisen ja optionaalisen automaattisen ajon sulkemisen applikatioille joille turvallisuus on ensisijaista. Pohjautuu Unicodeen. Jinjaa käytetään Djangon lisäksi, esimerkiksi Ansiblessa template pohjien luomiseen. (Jinja 2020.)

2.3.4 Git

Git on Linux Torvaldsin perustama avoimen lähdekoodin versionhallintajärjestelmä. Sen avulla voi hallita, ylläpitää ja julkaista projekteja. Pääosin Gittiä käytetään koodin kanssa, mutta sillä pystyy hallinnoimaan lähes mitä tahansa tekstiin perustuvia tiedostotyyppejä. Toisin kuin edeltäjissään Gitissä koko repositorio kopioidaan lokaaliin järjestelmään ja siihen tehdään muutoksia. Kuviossa 9 kuvattu tämä rakenne ja toiminta. Tämän jälkeen muutoksia verrataan keskuspalvelimen repositorioon ja mahdollisesti työnnetään (push) sinne. (What Exactly Is GitHub Anyway? 2020)



Kuvio 9. Gitin käyttö. (Lubański, M. 2019.)

2.3.5 Github

Github on Git repositorion hostaus palvelu. Se lisää kuitenkin käyttöön tukun omia ominaisuuksiaan ja verkkoselain pohjaisen graafisen käyttöliittymän. Githubissa avoimet repositoriot ovat ilmaisia, mutta yksityiset repositoriot ja osa ominaisuuksista maksullisia. Github tarjoaa työnkulun (Workflow) automaatiota kääntämisestä testaukseen ja julkaisuun. Se tarjoaa tukea koodin tarkistukseen ja lähettää hälytyksiä repositorioiden hallitsijoille, jotta he voivat paremmin reagoida haavoittuvuuksiin. Github tarjoaa työkaluja koodin tarkistuksiin ja tukee näillä tiimien kykykä tehdä parempaa koodaustyötä. Github tarjoaa laajennuksia ohjelmistoympäristöihin, joiden avulla voit käyttää Githubissa sijaitsevaa repositoriota niistä suoraan. Lisäksi Githubiin on mahdollista tuottaa koodiin ja projektiin liittyvä dokumentaatio suoraan Githubin tarjoamaan Wikiin. (How developers work 2020.)

3 Suunnitelma

3.1 Palvelinarkkitehtuuri

Kuormantasaajaklusterin malliksi otettiin Pinjan DMZ1 IP-avaruudessa jo olemassa ollut kuormantasaaja klusteri ja tehtiin siihen parannuksia ominaisuuksien ja hallittavuuden kannalta. Yksi ominaisuus mikä kokonaan puuttuu DMZ1 tasaajasta on kyky käsitellä Letsencryptin sertifikaatteja.

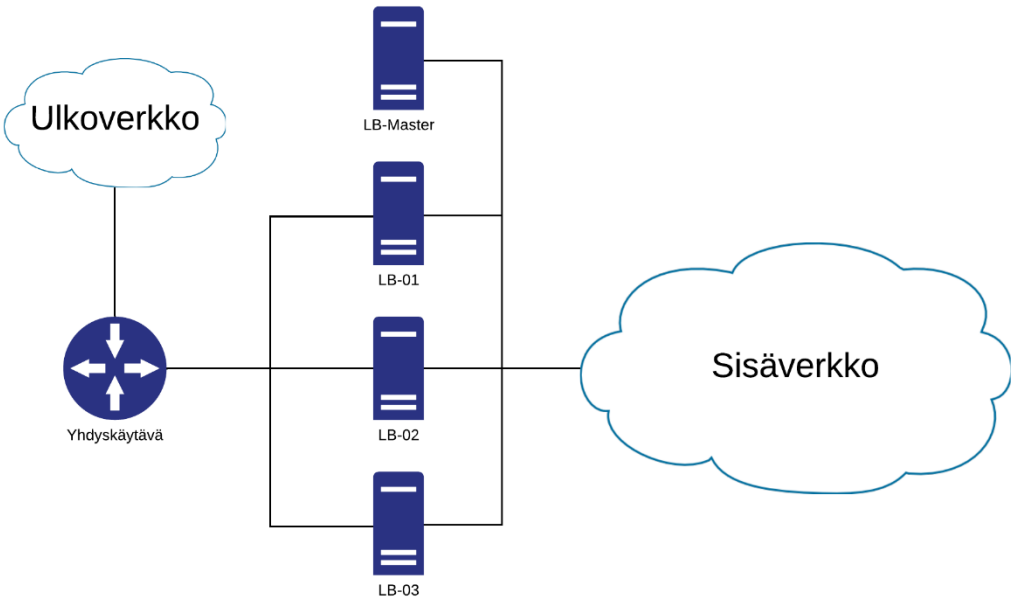
DMZ2-Kuormantasaajan palvelinten määrän päättämiseksi käytiin katsomassa Pinjan valvonnasta DMZ1 kuormantasaajan liikennemääriä. Kun verrattiin kuviossa 10 esiteltyä DMZ1-kuormantasaajan nodejen käytettyjen TCP sockettien määriä nähtiin, että palvelimilla ei olla lähelläkään yksittäisen palvelimen teoreettista maksimia (65535) käytettyjen sockettien määrissä. Pahimmassa tapauksessa, jos yksi palvelin joutuisi ottamaan vastaan koko kuormantasaajan liikenteen sillä olisi käytössä edelleen alle 5000 sockettia.

Kuvio 10. DMZ1 kuormantasaajan tcp sockettien käyttö.

Seuraavaksi katsottiin DMZ1 kuormantasaajalle saapuvaa TCP liikennettä puolen vuoden ajalta, mikä on nähtävissä kuviossa 11. Tästä voitiin nähdä, kuinka liikenne keskittyy lähinnä lb-02:lle ja yksittäinen korkea piikki uloittuu noin 80Mb/s. Koko klusterin tavanomainen huippuliikenne päivässä oli alle 50Mb/s. ss

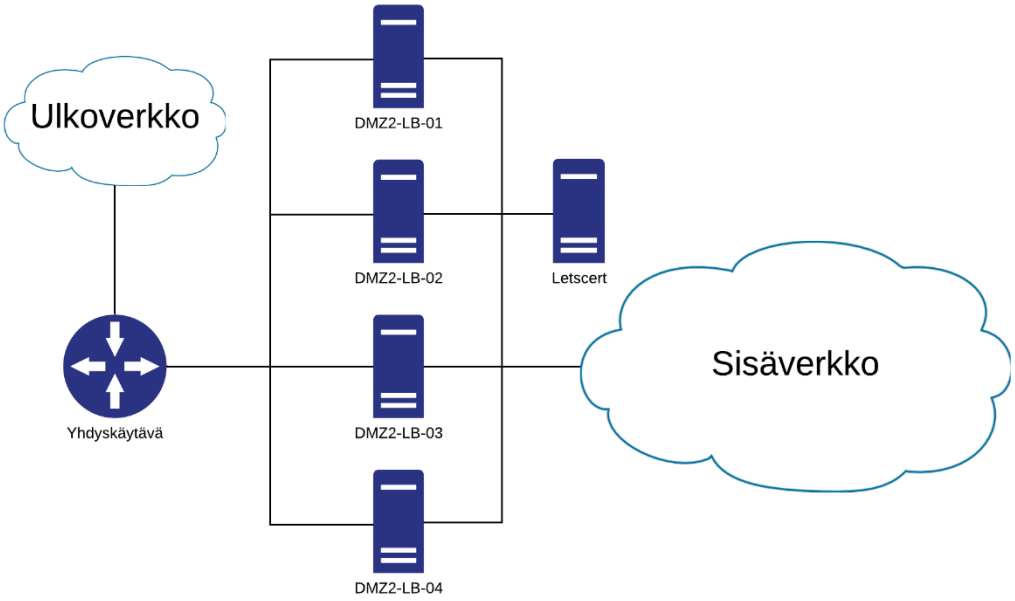
Kuvio 11. DMZ1-kuormantasaajan liikenne 6kk.

Kuviossa 12 on kuvattu DMZ1-kuormantasaaja, joka koostuu yhdestä hallintapalvelimesta ja kolmesta kuormaa tasaavasta nodesta. Palvelimet on klusteroitu keskenään Corosync ja Pacemaker ohjelmistoilla. Hallintapalvelin sisältää kaikki Corosyncin ja Pacemakerin resurssien käyttämät konfiguraatiotiedostot, josta ne synkronoidaan kuormaatasaaville lb-01, lb-02 ja lb-03 nodeille. Tässä ratkaisussa hallintapalvelin ei vastaanota ulkoverkosta tulevaa liikennettä ja kuormaa.



Kuvio 12. DMZ1-kuormantasaaja.

DMZ2:n kuormantasaus poikkeaa, kuviossa 13 esitettyyn tapaan, DMZ1:n kuormantasaajasta siten, että hallintapalvelin puuttuu. Sen sijaan, palvelimet konfiguroidaan synkronoidusti Ansiblen avulla. Näiden neljän palvelimen lisäksi luodaan erillinen Letscert-palvelin, joka käsittelee Letsencryptin sertifikaattien uusinnat ja luonnin.



Kuvio 13. DMZ2-kuormantasaaja.

Jos peilataan DMZ1 kuormantasaajan liikenteeseen ja otetaan sieltä kaikkein suurimmat hetkelliset kuormat, pitäisi DMZ2 kuormantasaajan suoriutua tuosta pyydetystä 100Mb/s liikenteestä. Kun katsotaan tuota aiemmin todettua liikennepiikkiä lb-02 kohdalla ja oletetaan, että yksittäinen node saturoituu 80Mb/s kohdalla, neljä nodea pystyy tällöin suoriutumaan 320Mb/s liikenteestä. Tässä tapauksessa tuo pyydetty 100Mb/s toteutuu vaikka kaksi nodea olisi yhtä aikaa tavoittamattomissa.

3.2 Palvelimet

3.2.1 Kuormantasaajaklusteri

Kuormantasausta tekevät palvelimet ovat Pinjan templatesta luotuja Centos 7 linux koneita. Näistä koneista luodaan klusteri Corosync ja Pacemaker ohjelmilla ja konfiguroidaan fencing. Jokaiselle koneelle asennetaan myös HAProxy kuormantasausta tekeviä Pacemaker-resursseja varten.

3.2.2 Letscert-palvelin

Letscert-palvelimen tehtävänä on toimia backendinä (taustapalvelin) Letsencryptin sertifikaattien uusimista varten. Palvelimelle asennetaan Certbot-ohjelmisto sekä luodaan Shell-skripti ja Ansible Playbook, jolla sitä (certbot) ja sertifikaatteja hallinoidaan. Skripti seuraa sertifikaattien tilaa, luo Ansiblelle tarvittavat tiedostot ja Ansible toimittaa uusitut sertifikaatit kuormantasaajaklusterin palvelimille sekä lataa uudet sertifikaatit käyttöön.

Sertifikaattien uusintaa varten pitää luoda Letsencryptin sertifikaattia käyttävään HAProxy konfiguraatioon ACL, jossa sertifikaatin uusimiseen liittyvä liikenne ohjataan Letscert palvelimelle. Liikenne voidaan tunnistaa ”/.well-known” polusta, joka poimitaan ACL:n ”path_beg” vivun avulla.

3.3 Hallinta

DMZ2:n kuormantasaajan hallinta poikkeaa DMZ1:n vastaavasta siten, että DMZ1:llä oli luotu jokaista erillistä VLAN:ia (Virtual Local Area Network) varten jokaiselle palvelimelle omat virtuaalirajapinnat. Uudella tasaajalla virtuaalirajapinta luodaan resurssiksi Pacemakeriin. Tällä säästetään sisäverkon tapauksessa IP-osoitteita, kun jokaisella klusterin palvelimella ei tarvitse olla omaa kiinteää IP-osoitetta vaan ip on kytketty resurssiin, joka liikkuu klusterissa.

Sen sijaan, että skriptillä synkronoidaan konfiguraatiot erilliseltä hallintapalvelimelta, hallinta tapahtuu keskitetysti Ansiblen avulla siten, että ylläpitäjä käy hakemassa github repositoriosta tarvittavat Ansible-playbookit ja tekee niihin konfiguraatio muutokset. Tämän jälkeen ajetaan kyseinen playbook, joka vie konfiguraatio muutokset kuormantasaajaklusterin palvelimille. Tämä tehdään mielellään keskitetyltä terminaali palvelimelta, mutta voidaan tehdä mistä vain, mistä on yhteys kuormantasaajaklusterin palvelimiin. Kun halutut muutokset on saatu kuormantasaajaklusterin palvelimille, käyttäjä puskee tekemänsä muutokset github repositorioon.

Tämän jälkeen itse kuormantasaaja palvelimilla, ylläpitäjän täytyy käydä jollakin näistä palvelimista luomassa tai päivittämässä tarvittavat resurssit. Pacemaker-resurssien luonnin kannalta ei ole väliä millä kuormantasaajaklusterin palvelimista ne tehdään.

Uutta Letenscrypt sertifikaattia varten pitää ensin luoda HAProxy konfiguraatio, missä liikenne ohjataan kokonaan letscert palvelimelle. Tämän jälkeen luodaan kuormantasaajalle vaadittavat Pacemaker-resurssit, että liikenne voi kulkea kuormantasaajan kautta. Seuraavaksi käydään Letscert palvelimella ajamassa sertifikaattien luomista ja ylläpitämistä varten luotu skripti millä saadaan sertifikaatit kuormantasaajaklusterin palvelimille. Vielä täytyy käydä tekemässä konfiguraatio muutokset HAProxy konfiguraatioon Ansiblen avulla ja vaihtaa ne käyttöön. Tämän jälkeen sertifikaattien uusinta tapahtuu automaattisesti.

4 Toteutus

Kuormantasaaja klusterin toteutus aloitettiin asentamalla neljä centos 7 palvelinta Pinjan virtuaaliympäristöön. Palvelimet nimettiin dmz2-lb-01, dmz2-lb-02, dmz2-lb-03 ja dmz2-lb-04. Kullekin palvelimelle annettiin yksi julkiseen verkkoon suunnattu rajapinta ja yksi sisäverkkoon suunnattu rajapinta. Palvelimet keskustelevat keskenään tämän sisäverkon rajapinnan kautta. Näille asennettiin viimeisimmät päivitykset ja konfiguroitiin /etc/hosts tiedostoon palvelinten hostnamet kuvion 14 mukaisesti.

```
10.209.4.101    dmz2-lb-01.protacon.com dmz2-lb-01.kvks.protacon.com dmz2-lb-01
10.209.4.102    dmz2-lb-02.protacon.com dmz2-lb-02.kvks.protacon.com dmz2-lb-02
10.209.4.103    dmz2-lb-03.protacon.com dmz2-lb-03.kvks.protacon.com dmz2-lb-03
10.209.4.104    dmz2-lb-04.protacon.com dmz2-lb-04.kvks.protacon.com dmz2-lb-04
```

Kuvio 14. Hosts konfiguraatio.

4.1 Klusterointi

Klusterointi tehtiin asentamalla corosync ja Pacemaker palvelut sekä niihin liittyvät fence-agentit kaikille palvelimille sekä laitettiin nämä palvelut tilaan, jossa ne käynnistyvät palvelimen käynnistymisen yhteydessä. Kuviossa 15 näkyy nämä komennot ja näiden lisäksi, kuinka käynnistettiin Pacemaker ja luotiin .ssh kansio ja sinne authorized_keys, jonne vietiin dmz2-lb-01:n ssh-avain, hallinnointia varten. Viimeisenä annetaan salasana ”hacluster” käyttäjälle, jonka pitää olla kaikilla klusterin koneilla sama.

```
$# yum install corosync pacemaker pcs fence-agents-all -y
$# for service in pcsd corosync pacemaker; do systemctl enable $service; done
$# systemctl start pcsd
$# mkdir .ssh
$# vim .ssh/authorized_keys
$# passwd hacluster
```

Kuvio 15. Klusterointi asennukset.

Kuvioissa 16 näkyy komennot, joilla luotiin ensin yhteinen autentikointi, johon tarvittiin tätä ”hacluster” käyttäjää sekä pystytetään itse Pacemaker klusteri nimelle ptc_dmz2_lb_cluster ja nimetään sen jäsenet. Tässä vaiheessa klusterin tila nähdään komennolla “pcs status” ja jälkimmäisen kuvion 17 tulosteessa tämä on nähtävissä.

```
$# pcs cluster auth dmz2-lb-01 dmz2-lb-02 dmz2-lb-03 dmz2-lb-04
$# pcs cluster setup --name ptc dmz2 lb cluster dmz2-lb-01 dmz2-lb-02 dmz2-lb-03 dmz2-lb-04
```

Kuvio 16. Klusterin autentikointi.

```
[root@dmz2-lb-04 ~]# pcs status
Cluster name: dmz2_lb_cluster
Stack: corosync
Current DC: dmz2-lb-02 (version 1.1.19-8.el7_6.4-c3c624ea3d) - partition with quorum
Last updated: Thu Apr  2 15:24:26 2020
Last change: Fri Jun 14 14:31:19 2019 by root via cibadmin on dmz2-lb-01

4 nodes configured
0 resources configured

Online: [ dmz2-lb-01 dmz2-lb-02 dmz2-lb-03 dmz2-lb-04 ]
```

Kuvio 17. Klusterin tila juuri käynnistettynä.

Tämän jälkeen haettiin kunkin palvelimen vmware ID (Identifier) komennolla /usr/sbin/dmidecode ja tästä parsitaan itse UUID (Universally Unique Identifier). Kuviossa 18 on esitelty tämä komento palvelimella dmz2-lb-04 ja sen tuloste. Tätä UUID:tä tarvitaan fencingin järjestämiseen.

```
[root@dmz2-lb-04 ~]# /usr/sbin/dmidecode | grep UUID
        UUID: 423906e2-bb13-46cb-87eb-1ce5c12a538a
```

Kuvio 18. Palvelimen UUID:n selvitys.

Fencing luotiin kullekin palvelimelle seuraavalla komennolla:

pcs stonith create fence_dmz2-lb-01 fence_vmware_soap pcmk_host_list="dmz2-lb-01" port="4239bb41-0408-b932-afbb-728b580a6d5d" ipaddr="vcenter02.kvks.protacon.com" ssl_insecure=1 power_timeout=20 login=stonithha@vsphere.local passwd=<salasana> login_timeout=60 pcmk_monitor_timeout=60s

Komennossa vaihtuu palvelimen nimi ja “port” parametriksi asetetaan aina edellisellä komennolla haettu palvelinta vastaava UUID. Komennossa luodaan resurssi, joka ottaa yhteyden Vcenteriin ja komentaa siellä käyttäjää stonithha, joka on luotu tätä fencing toimintoa varten. Stontihha käyttäjän oikeudet on asetettu VMware ympäristössä siten, että se kykenee uudelleen käynnistämään palvelimen tarvittaessa.

4.2 Ansible Playbook

Hallinnointia varten luotiin yksinkertainen, kuvion 19 mukainen, ansible playbook. Ensin ansible luo tarvittavan kansiorakenteen vars-tiedoston perusteella ja tämän jälkeen luo saman vars-tiedoston perusteella templaten pohjalta HAProxy konfiguraation.

```
---
- name: include variables
  include_vars: "dmz2-lb-vars.yml"

#- debug:
    #var: dmz2_lb

- name: create haproxy chroot folders
  file:
    path: /usr/share/haproxy-{{ item.name }}
    state: directory
  with_items: "{{ dmz2_env }}"

- name: haproxy config file
  template:
    src: "templates/haproxy-site.cfg.j2"
    dest: "/etc/haproxy/haproxy-{{ item.name }}.cfg"
  with_items: "{{ dmz2_env }}"
```

Kuvio 19. Ansible playbook.

Kuviossa 20 on esitelty HAProxyn konfiguraatio jinja2 templaten ensimäinen osa. Tässä konfiguroidaan instanssin pidfilet, chrootit ja socketit sekä annetaan joitain oletus konfiguraatioita. Kuviossa myös määritellään tilastosivun konfiguroinnit ja kuinka sinne autentikoidutaan.

```
# this config needs haproxy-1.1.28 or haproxy-1.2.1

global
        # 10.209.8.6 - ha-master
        # 10.209.0.140 - lb-master
        log 10.209.0.140        local2
        pidfile         /var/run/haproxy-{{ item.name }}.pid
        chroot          /usr/share/haproxy-{{ item.name }}
        stats socket    /var/run/haproxy-{{ item.name }}.stat mode 777
        user            haproxy
        group           haproxy
        daemon

defaults
{% set protocol = item.protocol | default(http) %}
        log     global
        mode    {{ protocol }}
        option  {{ protocol }}log
        option  dontlognull
        retries 3
        option  redispatch
        maxconn 1000
        timeout connect 50s
        timeout client 50s
        timeout server 120s

listen {{ item.name }}-stats {{ item.status_ip | default(item.frontends["frontend_1"].front_ip) }}:44444
        stats admin if TRUE
        stats hide-version
        stats uri /tilanne
        stats auth {{ item.name }}:{{ item.name }}-stats
        stats show-desc Kuormantasaaja {{ item.name }}
        stats realm {{ item.name }}\ test\ stats
        stats refresh 5
        mode http
        stats enable
```

Kuvio 20. HAProxyn oletuskonfiguraatiot.

Kuviossa 21 on esitelty, kuinka templatessa määritellään HAProxyyn frontend konfiguraatiot. Tässä on myös annettu optiona Access-listojen käyttö, jos halutaan ohjata useampaan eri taustapalvelimeen (backend) liikennettä. Tätä tarvitaan esimerkiksi Letsencryptin sertifikaattien kanssa. Samoin “Options”:ien käyttö on mahdollistettu.

```
{% set frontends = item.frontends %}
{% for frontend in frontends %}
{% set front = frontends[frontend] %}
{% set name = front.name %}
########{{ name | regex_replace('.','#') }}########
#       {{ name | regex_replace('.',' ') }}       #
#       {{ name }}       #
#       {{ name | regex_replace('.',' ') }}       #
########{{ name | regex_replace('.','#') }}########

frontend  {{ name | lower | replace(" ","-") }}-frontend
{% set binds = front.binds %}
{% for bind in binds %}
        bind {{ front.front_ip }}:{{ binds[bind] }}
{% endfor %}

        mode {{ item.protocol }}
        option {{ item.protocol }}log
{% if front.options is defined %}
{% set options = front.options %}
{% for option in options %}
        {{ options[option] }}
{% endfor %}
{% endif %}
{% if front.acls is defined %}
{% set acls = front.acls %}
{% for acl in acls %}
        acl {{ acls[acl].name }} {{ acls[acl].parameters }}
{% endfor %}
{% endif %}
{% for backend in front.backends %}
        use_backend {{ front.backends[backend] | default(name) }}
{% endfor %}
{% endfor %}

{% set backends = item.backends %}
{% for backend in backends %}
backend {{ backends[backend].name | default(name) }}
```

Kuvio 21. HAProxyn frontend konfiguraatiot.

Lopuksi määritellään templateen HAProxy konfiguraation taustapalvelimet. Kuviossa 22 on esitelty tämä toiminto.

```
{% set backend_servers = backends[backend].backend_servers %}
{% for backend_server in backend_servers %}
       server {{ backend_servers[backend_server].hostname }} {{ backend_servers[backend_server].host_ip }}:{{ backend_servers[backend_server].port }}
{% endfor %}
{% endfor %}
```

Kuvio 22. backend konfiguroinnin luonti.

Konfigurointia varten luotiin vars-tiedosto ja sinne esimerkki konfiguraatio, miten HAProxy konfiguraatio luodaan. Tämä konfiguraatio esimerkki on esitelty kuviossa 23 ja konfiguraatio on aina YAML tyyppistä. Objektit, joita voi määritellä useamman,

kuin yhden, on esitetty aina monikossa ja sen jälkeen numeroituna näiden alle voi asettaa useampia tietueita. Numero järjestys on optionaalinen. Esimerkiksi, jos tarvitaan useampi, kuin yksi frontend niin ne voi eritellä omiksi objekteikseen ja niiden sisään tarvittavat lisä konfiguraatiot, kunhan sisennykset on tehty oikealle tasolle.

```
dmz2_env:
#- name: "example"
#  status_ip: 12.123.123.123
#  protocol: tcp
#  frontends:
#    frontend_1:
#      name: "example"
#      front_ip: "12.123.123.123"
#      haproxy_balance: "roundrobin"
#      binds:
#        bind1: 123
#        bind2: 456
#      options:
#        option1: "option example"
#      acls: #optional
#        acl1:
#          name: acl_example
#          parameters: "-i example.ex"
#          backend: back_example
#      backends:
#        backend1: "back_example if acl_example" #acl example only "back_examble" if no acl defined
#  backends:
#    backend1:
#      name: back_example
#      options:
#        option1: "option example"
#      backend_servers:
#        backend_server1:
#          hostname: "example1.ex"
#          host_ip: "10.0.0.123"
#          port: 123
```

Kuvio 23. Esimerkki HAProxy konfiguraatiota varten ansiblen vars tiedostosta.

Letsencryptin sertifikaatteja käyttäessä oli tärkeää, että HAProxyllä ohjattiin sertifikaattien uusimiseen liittyvä liikenne Letscert palvelimelle. Kuviossa 24 on esimerkki konfiguraatio tällaisesta palvelusta. Oleellista oli ACL:n luonti ja, että backendeistä löytyy Letscert-palvelin.

```
- name: "letscert"
  status_ip: [redacted]
  protocol: tcp
  haproxy_balance: "roundrobin"
  frontends:
    frontend_1:
      name: "letscert-front"
      front_ip: [redacted]
      binds:
        bind1: 80
        bind2: "443 ssl crt /etc/ssl/letsencrypt/letsencrypt.protacon.fi/letsencrypt.protacon.fi.pem"
      options:
        option1: "default_backend backend_letstest"
      acls: #optional
        acl1:
          name: letsencrypt_certbot_proxy
          parameters: "path_beg /.well-known"
      backends:
        backend1: "backend_letscert if letsencrypt_certbot_proxy"
  backends:
    backend1:
      name: backend_letstest
      backend_servers:
        backend_server1:
          hostname: "letstest.protacon.com"
          host_ip: "10.[redacted]125"
          backend_ports:
          port: 80
    backend2:
      name: backend_letscert
      backend_servers:
        backend_server1:
          hostname: "letscert.protacon.com"
          host_ip: "10.[redacted].9"
          backend_ports:
          port: 80
```

Kuvio 24. Letsencrypt esimerkki konfiguraatio.

4.3 Kuormantasaaja resurssit

Resursseja varten asennettiin palvelimelle HAProxy tekemään kuormantasausta. Yum paketinhallinnalla asennus onnistuu helposti kuvion 25 mukaisesti.

```
[root@dmz2-lb-01 ~]# yum install haproxy
```

Kuvio 25. HAProxyn asennus.

Pacemakeriin tehtiin kutakin palvelua varten viisi Pacemaker-resurssia. Ensin luotiin sisäverkon rajapintaan virtuaalirajapinta. Tämä onnistuu kuvion 26 mukaisella komennolla. Eriteltynä:

- ”pcs resource create vlan-letscert-test” määritellään, että luodaan resurssi ja, että sen nimi on ”vlan-letscert-test”.
- ”ocf:heartbeat:iface-vlan” määritellään käytettävä moduuli.
- ”vlan_interface=ens256 vlan_id=352” määritellään moduulin mukaisesti mihin rajapintaan virtuaalirajapinta luodaan ja mikä vlan id sille annetaan.
- ”op monitor timeout=”20s” interval=”10s” määritellään optionaaliset monitoroinnit ja health checkit resurssille.
- ”—group cluster-letscert” Luodaan oma ryhmä letscert resursseja varten.

```
[root@dmz2-lb-01 ~]# pcs resource create vlan-letscert-test ocf:heartbeat:iface-vlan vlan_interface=ens256 \
> vlan_id=352 op monitor timeout="20s" interval="10s" --group cluster-letscert
```

Kuvio 26. Virtuaalirajapinta resurssin luonti.

Seuraavaksi annettiin juuri luodulle virtuaalirajapinnalle ip. Tämä tehtiin kuvion 27 mukaisesti luomalla IP-osoiteresurssi Pacemakeriin. Tämän resurssin rakenne oli hyvin samantyyppinen, kuin virtuaalirajapinan resurssin. Ainoastaan käytettävä moduuli vaihtui ja siihen määriteltiin IP-osoite, mihin rajapintaan se asetetaan ja sille sopiva verkkomaski.

```
[root@dmz2-lb-01 ~]# pcs resource create ip-letscert-test \
> IPaddr2 ip=10.209.34.130 nic="ens256.352" cidr_netmask="28" --group cluster-letscert
```

Kuvio 27. Sisäverkon IP-osoiteresurssin luominen.

Samalla tavalla luotiin myös julkisenverkon IP-osoittelle Pacemaker-resurssi. Kuviosta 28 voi nähdä muutokset missä IP-osoitteeksi on annettu julkiverkon osoite ja kyseiseen avaruuteen sopiva verkkomaski.

```
[root@dmz2-lb-01 ~]# pcs resource create ip-front-letscert-test \
> IPaddr2 ip=1[redacted] nic="ens224" cidr_netmask="24" --group cluster-letscert
```

Kuvio 28. Julkiverkon IP-osoiteresurssin luominen.

Seuraavaksi luotiin asymmetristä reititystä varten oletusyhdyskäytäväresurssi. Tätä tarvittiin, koska oletuksena kuormantasaaja käyttää verkkoliikennöintiin julkisen-IP:n puuttuessa sisäverkon rajapintaa. Tästä seuraa, että julkisenverkon IP:n saaminen ei luo automaattisesti palvelimelle oletusyhdyskäytävää julkisenverkon IP:lle ja tähän IP-osoitteeseen tulevan liikenteen paluuliikenne suuntautuu sisäverkkoon. Tämän resurssin luominen on nähtävissä kuviosta 29. Tähän määritellään haluttu rajapinta, oletusyhdyskäytävä, käytettävä reititystaulu ja protokolla.

```
[root@dmz2-lb-01 ~]# pcs resource create def-letscert-test ocf:heartbeat:Route destination="default" \
> device="ens224" gateway="1[redacted].1" table="100" family="ip4" --group cluster-letscert
```

Kuvio 29. Oletusyhdyskäytäväresurssin luominen.

Lopuksi luotiin HAProxyresurssi Pacemakeriin. Tähän resurssiin osoitettiin moduulille vain polku haluttuun HAProxy konfiguraatio tiedostoon, mikä on nähtävissä kuviosta 30.

```
[root@dmz2-lb-01 ~]# pcs resource create haproxy-letscert-test ocf:heartbeat:haproxy \
> conffile="/etc/haproxy/haproxy-letscert.cfg" --group cluster-letscert
```

Kuvio 30. HAProxy resurssin luominen.

Nyt voitiin nähdä “pcs status” komennolla luomamme resurssit. Kuviossa 31 on esitelty tämä näkymä. Siitä nähtiin, että resurssit on onnistuneesti luotu omaan ryhmäänsä ”cluster-letscert” ja nämä resurssit liikkuvat nyt ryhmänä tarivttaessa kuormantasauspalvelimelta toiselle. Käynnistettäessä resurssit käynnistyivät dmz2-lv-02 palvelimelle.

```
Resource Group: cluster-letscert
    vlan-letscert-test (ocf::heartbeat:iface-vlan):    Started dmz2-lb-02
    ip-letscert-test   (ocf::heartbeat:IPaddr2):       Started dmz2-lb-02
    ip-front-letscert-test     (ocf::heartbeat:IPaddr2):       Started dmz2-lb-02
    def-letscert-test  (ocf::heartbeat:Route): Started dmz2-lb-02
    haproxy-letscert-test      (ocf::heartbeat:haproxy):       Started dmz2-lb-02
```

Kuvio 31. Kuormantasaaja resurssi ryhmä.

4.4 Letscert-palvelin

Letsencryptin sertifikaattien hallinnointia varten asennettiin Centos 7 palvelin samasta templatesta, kuin klusterin Centos 7 palvelimet. Palvelimella tarvitsemme Letencryptin sertifikaattien hankkimista ja uusimista varten Certbot ohjelmaa. Tämän asennus tapahtui yksinkertaisesti ja on esitelty kuviossa 32.

```
[root@letscert ~]# yum install certbot
```

Kuvio 32. Certbotin asennus.

Certbotin käyttö on esitelty kuviossa 33. Siinä luodaan esimerkki sertifikaatti letsencrypt.protacon.fi nimiselle palvelimelle. Tätä varten tulee avata portit 80 ja 443 palomuurilta ja julkisessa nimipalvelussa täytyy toimialuenimen osoittaa sellaista IP:tä minkä takaa palvelin löytyy.

```
[root@letscert ~]# certbot certonly --standalone -d letsencrypt.protacon.fi
```

Kuvio 33. Esimerkki sertifikaatin luonti.

Sertifikaattien ylläpidon automatisoimiseksi tehtiin Shell-skripti ja Ansible playbook. Näiden tehtävänä oli luoda ja uusia sertifikaatteja ja toimittaa ne hallitusti Ansiblen

avulla kuormantasausta tekeville palvelimille ja asettaa ne siellä käyttöön. Kuviossa 34 on esitelty Shell-skriptin osa, missä:

1. Luodaan lista nykyisten olemassa olevien sertifikaattien md5 tarkisteista.
2. Luodaan kaksi muuta apumuuttujaa.
3. Ajetaan certbot renew, joka uusii uusimisiässä olevat sertifikaatit.
4. Katsotaan onko annettu domainia argumenttina skriptiin, jolloin yritetään luoda tälle domainille uusi sertifikaatti.
5. Luodaan listat uusista sertifikaateista ja niiden poluista.

```
#!/bin/bash

#certbot dmz2-automation tool, Vilpo

#read current serts to variables and renew with certbots
#read again certs md5sums to variable to see possible changes of certs

current_certs=($(md5sum /etc/letsencrypt/live/*/fullchain* | awk '{print $1}'))
sertlist=($(ls /etc/letsencrypt/live))
echo "${sertlist[@]}"
renewed_certs=()


certbot renew > /dev/null 2>&1

if [ -n "$1" ]
then
certbot certonly --standalone -d $1 --non-interactive --agree-tos -m ptcit@pronetti.com
fi

new_certs=($(md5sum /etc/letsencrypt/live/*/fullchain* | awk '{print $1}'))
new_path=($(md5sum /etc/letsencrypt/live/*/fullchain* | awk '{print $2}'))
```

Kuvio 34. Shell-skriptin apumuuttujien luonti ja sertifikaattien uusiminen.

Kuviossa 35 on esitelty skriptin osa missä vertaillaan skriptin alussa luotua md5 tarkistetta ja sertifikaattien uusimisen jälkeistä tarkistetta ja näiden perusteella luodaan lista sertifikaateista, jotka on muuttuneet.

```
#check if certs are renewed and append changed certs names to array
for (( i = 0; i < ${#current_certs[@]}; i++ ))
        do

        if [ "${current_certs[$i]}" = "${new_certs[$i]}" ]
        then
                #add to list of new certs.
                pathname="${new_path[$i]}"
                renewed_certs+=($(basename $(dirname $pathname)))
        fi
done
```

Kuvio 35. sertifikattien tarkisteiden vertailu ja muutosten listaus.

Tämän jälkeen luodaan uusista ja uusituista sertifikaateista lista. Tästä muodostetaan Ansiblea varten vars-tiedosto ja ajetaan Ansible playbook niin, kuin on esitelty kuviossa 36.

```
#See if there is new certs and create folders for them and if totally new add them to array for ansible
for sert in ${sertlist[@]}
        do
        if [[ $sert != "README" ]]
        then
                if [ ! -d /etc/ssl/letsencrypt/$sert ]
                then
                        renewed_certs+=($sert)
                fi
        fi
done

echo "new_certs:" > /users/letscert-ansible/roles/dmz2-letsencrypt/vars/new_certs.yml
for (( i = 0; i < ${#current_certs[@]}; i++ ))
        do


printf "%s\n" "- name: cert$i"  "  cert: ${renewed_certs[$i]}" >> /users/letscert-ansible/roles/dmz2-letsencrypt/vars/new_certs.yml
        done

        ansible-playbook /users/letscert-ansible/dmz2-letsencrypt.yml -i /users/letscert-ansible/hosts/dmz2-lb.hosts
```

Kuvio 36. Ansiblen vars-tiedoston luonti ja playbookin ajo.

Kuviossa 37 on esitelty skriptin esimerkki sertifikaateista luoma Ansiblen vars-tiedosto. Tämän tiedoston perusteella Ansible playbook tekee muutokset kuormantasaajalle.

```
new_certs:
- name: cert0
  cert: hopo-test-cert.fi
- name: cert1
  cert: letsencrypt.protacon.fi
/users/letscert-ansible/roles/dmz2-letsencrypt/vars/new_certs.yml (END)
```

Kuvio 37. Ansiblen vars-tiedosto.

Ansible playbookin ensinmäiset taskit (toiminnot) on esitelty kuviossa 38. Taskissa ensin luetaan sisään tämä skriptin luoma vars tiedosto. Tämän jälkeen tehdään lista HAProxy konfiguraatioista, missä esintyvät muuttuneet sertifikaatit.

```
---
# tasks file for dmz2-letsencrypt
- name: include variables
  include_vars: "new_certs.yml"

- name: get haproxy instance names
  shell: grep -R "{{ item.cert }}" /etc/haproxy | cat | awk '{print $1}'
  with_items: "{{ new_certs }}"
  register: find_result
  ignore_errors: yes
```

Kuvio 38. HAProxy listan luonti muuttuneiden sertifikaattien perusteella.

Seuraavaksi viedään sertifikaatit kuormantasaajalle ja lopuksi ladataan konfiguraatiot HAProxy instansseissa uudelleen, juuri luodun listan perusteella. Nämä taskit on esitelty kuviossa 39.

```
- name: copy sertificates
  copy:
    src: /etc/ssl/letsencrypt
    dest: /etc/ssl

- name: reload Haproxy
  shell: haproxy -f /etc/haproxy/{{ item | basename | splitext | first }}.cfg \
         -p /var/run/{{ item | basename | splitext | first }}.pid \
         -sf $(cat /var/run/{{ item | basename | splitext | first }}.pid)
  with_items: "{{ find_result.results | map(attribute='stdout_lines') | list  }}"
```

Kuvio 39. Sertifikaattien vienti ja HAProxyjen uudelleen lataus.

4.5 Ylläpito

Lopuksi tehtiin klusterin ylläpitoa varten oma Ansible playbook helpottamaan klusterin ylläpitoa. Normaalisti Pinjalla palvelimet päivitetään keskitetysti Spacewalk-

järjestelmästä. Tämä kuitenkin tarkottaisi kuormantasaajan kohdalla sitä, että kaikki palvelimet päivittyisivät yhtä aikaa mikä tarkoittaisi, että palvelu on palvelimen uudelleen käynnistymisen ajan tavoittamattomissa. Rullaten päivittämistä (Rolling update) varten luotiin kuviossa 40 esitelty playbook missä palvelimet päivitetään ja uudelleen käynnistetään yksikerrallaan siten, että seuraavan palvelimen päivityssykli ei käynnisty ennen, kuin edellinen on valmis. Tällä saadan pidettyä palvelu aina saatavilla ja aiheutuu vain käyttäjälle huomaamaton katko, kun palveluresurssi siirtyy Pacemakerissa nodelta toiselle uudelleenkäynnistymisen yhteydessä.

```
---
- name: Dmz2 Load Balancer configuration
  hosts: lb
  serial: 1

  tasks:
  - name: test
    shell: hostname

  - name: yum update
    yum: name=* state=latest

  - name: reboot
    reboot:
```

Kuvio 40. Rolling update playbook.

5 Pohdinta

Tavoite oli lähteä toteuttamaan avoimeen lähdekoodiin perustuvaa korkean saatavuuden kuormantasaaja klusteria jo olemassa olleen DMZ1 kuormantasaaja klusterin pohjalta. Tarkoitus oli korjata osaltaan tämän aiemman kuormantasaajan puutteita ja tuoda uusia ominaisuuksia ja etenkin hallittavuutta kuormantasaajaklusteriin. Tarkoitus oli yksinkertaistaa hallittavuutta sellaiselle tasolle, että itse työkalut olisivat intuitiivisia ja helpommin ylläpidettäviä. Tähän päästiin ainakin ylläpidettävyyden kannalta. Tämän kaltaisen avoimenlähdekoodin

ohjelmistoihin perustuvan klusterin parhaita puolia on muunneltavuus, jota käytettiin tässä työssä laajasti hyödyksi, mikä näkyy esimerkiksi valittuina työkaluina ja miten järjestelmä suoriutuu Letsencryptin sertifikaateista. Lopputuloksena saatiin tuotettua Pinjan ja sen asiakkaiden käyttöön toimiva kuormantasausjärjestelmä, joka on mahdollista toimittaa myös asiakkaiden omiin virtualiympäristöihin.

DMZ1 kuormantasaajalle uutta palvelua tekevän pitää tietää huomattava määrä kuormantasaajan toiminnasta ja sen konfiguroimisesta. Yksi sen suurimpia puutteita on palautteen puute virheen tehdessä. Esimerkiksi, jos kuormantasaajaresurssi epäonnistuu, kuten HAProxy instanssin käynnistyminen, tästä ei saada mitään lokia miksi epäonnistuminen tapahtuu. Tätä lähdettiin korjaamaan Ansible playbookilla, joka pitäisi huolta, että kaikki tarvittavat konfiguraatiot ja kansiorakenteet olisivat paikallaan ja keskenään saman muotoisia. Näin saataisiin yksinkertaistettua dokumentaatiota ja ylläpitoa.

Suurin haaste oli Letsencryptin sertifikaattien käytön mahdollistaminen. Tämä oli vaatimuksena, koska yksi kuormantasausta tarvitseva asiakas käyttää palvelimellaan yli 130 Letsencryptin sertifikaattia. Sertifikaattien luontiin ja hallintaan käytettävä certbot olettaa, että sertifikaattia käytetään samalla palvelimella certbotin kanssa, joten sertifikaattien synkronointiin kuormantasaajalle piti luoda skriptejä ja selvittää millä keinoin saisi sertifikaattien uusimiseen kohdistuvan liikenteen eriteltyä palvelimen normaalista HTTPS liikenteestä. Tähän löydettiin ratkaisu HAProxyn access-listoista, jolla saatiin eroteltua sertifikaatin uusimiseen liittyvä liikenne muusta liikenteestä.

Hukkaresursseja saatiin vähennettyä siten, että DMZ1-kuormantasaajan hallintanode otettiin mukaan kuormantasausta tekeväski nodeksi. Samoin saatiin yksinkertaistettua mahdollista laajentamistarvetta siten, että koneen asennuksen jälkeen ainoastaan Ansible playbookeihin pitää lisätä ylimääräinen node, jota niiden tulee hallita. Seuraava playbookin ajo toimittaisi resurssit uudelle nodelle.

Päätös klusterin luomisesta ja muodosta saatiin toukokuussa 2019 ja yliheitto kaavailtiin heinäkuun 2019 loppuun. Tämä tarkoitti, että toteutukseen jäi aikaa reilu

kuukausi, kun ottaa kesälomat huomioon. Lisäksi toteutusta tehtiin muiden töiden ohella. Aikataulu tarkoitti, että jatkokehitettävää jäi ja testaus jäi puutteelliseksi. Esimerkiksi Letscert-palvelimen skriptistä on huomattavissa potentiaalinen virhe, koska uusittujen sertifikaattien lista tehdään vasta uuden sertifikaatin luomisen jälkeen, joka voi muuttaa listan järjestystä. Seuraus on, että kuormantasaajalla HAProxy instansseja ladataan turhaan, mikä ei ole suureksi haitaksi.

Ansible valittiin työkaluksi juuri sen takia, että on helppo jatkokehittää tulevaisuudessa. Ensisijainen korjaus olisi tehdä Ansible playbookiin kansiorakenteen luonti HAProxy-instanssien konfiguraatiotiedostoista löytyviä kansioita varten. Tällä saataisiin vähennettyä kuormantasaajaklusterissa tehtyjä töitä. Samoin Pacemaker-resurssien luonti olisi hyvä automatisoida Ansiblen tehtäväksi. Myös Letsencryptin sertifikaatinluontiprosessin voisi automatisoida. Nykyisellään se on tarpeettoman monimutkainen. Näiden jälkeen olisi hyvä katsoa miten HAProxy konfiguraatiot määritellään Ansiblen vars-tiedostoon. Toteutukseen päätynyt tapa oli ensinmäinen itteraatio ja siinä on varmasti kehitettävää, että saataisiin konfigurointi intuitiivisemmaksi ja käyttäjäystävällisemmäksi. Esimerkiksi Shell-skripti, joka esittee oleelliset kysymykset ja tekee vastauksiin virheiden tarkistukset voisi parantaa käytettävyytä konfiguroinnin osalta. Sama skripti voisi haluttaessa myös ajaa Ansible-playbookin millä saataisiin vietyä tehdyt konfiguraatiomuutokset heti kuormantasaajalle. Ansiblella voisi myös hallinoida palveluiden huoltosivuja. Jos palvelu on jostainsyystä tällähetkellä tavoittamattomissa, se näkyy käyttäjälle palvelun täydellisenä puuttumisena. Sen sijaan votaisiin kuormantasaajalla pitää eri palveluiden huoltosivuja, jotka näkyvät, kun palvelu ei muuten ole saatavissa tai Pacemakerin kuormantasausresurssi on alhaalla.

Ansiblen ulkopuolinen välitön kehityskohde olisi poistaa yksi node klusterista. Klusterin suorituskyky riittää mainiosti kaavailtuihin tarpeisiin myös kolmella nodella ja yhden noden poistamisella päästäisiin parittomaan määrään, joka olisi korkean saatavuuden kannalta parempi ratkaisu. Tämä tarkoittaisi päätösvallan kannalta sellaisen riskin minimoimista missä klusteri jakautuu kahteen toiminnalliseen pariin ja syntyy aivohalkio. Tästä seuraisi se, että palvelimet tekisivät toisilleen Fencing-operaatioita toistuvasti. Koska Pinjan (silloisen Protaconin) konesaliliiketoiminta

myytiin Ficololle syksyllä 2019, vmwareen perustuva Fencing rikkoutui, kun vmwareen tehty “stongithha” käyttäjä poistui. Tämän seurauksena Fencing tulisi konfiguroida käyttämään jotain muuta agenttia, kuin vmwareen perustuvaa. Tämä tulisi tehdä myös vanhemmalle DMZ1 kuormantasaajalle, sillä sitä koskee sama tilanne.

Lähteet

About Let’s Encrypt, Viitattu 12.4.2020. https://letsencrypt.org/about/

Anita Saaranen-Kauppinen & Anna Puusniekka. 2006a. Analyysi. KvaliMOTV - Menetelmäopetuksen tietovaranto.Tampere: Yhteiskuntatieteellinen tietoarkisto Viitattu 5.4.2020. https://www.fsd.tuni.fi/menetelmaopetus/kvali/L7_3_6_3.html

Anita Saaranen-Kauppinen & Anna Puusniekka. 2006b. Grounded theory. KvaliMOTV - Menetelmäopetuksen tietovaranto. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 5.4.2020. https://www.fsd.tuni.fi/menetelmaopetus/kvali/L5_2.html

Anita Saaranen-Kauppinen & Anna Puusniekka. 2006c. Mitä laadullinen tutkimus on. KvaliMOTV - Menetelmäopetuksen tietovaranto.Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 5.4.2020. https://www.fsd.tuni.fi/menetelmaopetus/kvali/L1_2.html

Fencing and STONITH. 2020. Verkkosivut. Viitattu 11.4.2020. https://documentation.suse.com/sle-ha/15-SP1/html/SLE-HA-all/cha-ha-fencing.html

High availability. 2019. Verkkosivut. Viitattu 28.3.2020. https://searchdatacenter.techtarget.com/definition/high-availability

High availability models. 2020.Verkkosivut. Viitattu 29.3.2020. https://www.ibm.com/support/knowledgecenter/SSANHD_7.5.1/com.ibm.mbs.doc/gp_highavail/c_cluster_config_models.html

How Ansible Works. 2020.Verkkosivut. Viitattu 11.4.2020. https://www.ansible.com/overview/how-ansible-works

How availability is measured. 2019. Verkkosivut. Viitattu 28.3.2020. https://searchdatacenter.techtarget.com/definition/high-availability

How developers work. 2020. Verkkosivut. Viitattu 13.4.2020. https://github.com/features

How Does the SSL Certificate Create a Secure Connection. 2020. Verkkosivut. Viitattu 12.4.2020. https://www.digicert.com/ssl/

How to achieve high availability. 2019. Verkkosivut Viitattu 28.3.2020. https://searchdatacenter.techtarget.com/definition/high-availability

Installing Ansible. 2020. Verkkosivut. Viitattu 11.4.2020. https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#managed-node-requirements

litespeed-cluster. 2019. Kuvio. Viitattu 13.4.2020 https://www.scalahosting.com/blog/wp-content/uploads/2019/05/litespeed-cluster.png

Load Balancing Techniques. 2020. Verkkosivu. Viitattu 13.4.2020. https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques/

Lubański, M. 2019. Centralized version control. Verkkosivu. Viitattu 13.4.2020 https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0

Pacemaker. 2018. Julkaisijan verkkodokumentaatio. Viitattu 29.3.2020. https://wiki.clusterlabs.org/wiki/Pacemaker

Pinjan tarina. 2020. Yrityksen verkkosivut. Viitattu 28.3.2020. https://www.pinja.com/me-olemme-pinja/

Red Hat Ansible Tower. 2020. Yrityksen verkkosivut. Viitattu 13.4.2020. https://www.ansible.com/products/tower

The Corosync Cluster Engine. 2020. Verkkosivut. Viitattu 29.3.2020. http://corosync.github.io/corosync/

The Reliable, High Performance TCP/HTTP Load Balancer. 2020. Julkaisijan verkkosivut. Viitattu 13.4.2020. http://www.haproxy.org/

What is a Certificate Authority. 2020. Verkkosivut. Viitattu 12.4.2020. https://www.ssl.com/faqs/what-is-a-certificate-authority/

What is Load Balancing? 2020. Verkkosivu. Viitattu 13.4.2020. https://www.nginx.com/resources/glossary/load-balancing/

What is Secure Soccets Layer? 2020. Verkkosivut. Viitattu 12.4.2020. https://www.digicert.com/ssl/

What is the SSL-certificate Chain? 2020. Verkkosivut. Viitattu 12.4.2020. https://support.stackpath.com/hc/en-us/articles/360023757932-SSL-Certificates-Explained

Bravo, M. 2018. Why use configuration management tools. Viitattu 13.4.2020. https://opensource.com/article/18/12/configuration-management-tools

Working With Playbooks. 2020. Verkkosivut. Viitattu 11.4.2020. https://docs.ansible.com/ansible/latest/user_guide/playbooks.html