jamk.fi

Korkean saatavuuden tietokantaratkaisu

Sami Autio

Opinnäytetyö
Toukokuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Autio, Sami	Opinnäytetyö, AMK	Toukokuu 2018
	Sivumäärä 52	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Korkean saatavuuden tietokantaratkaisu

Tutkinto-ohjelma

Ohjelmistotekniikka

Työn ohjaaja(t)

Ari Rantala

Toimeksiantaja(t)

Kaaos Unlimited Oy

Tiivistelmä

Nykyajan maailmassa digitalisaation kehittyessä loppukäyttäjät vaativat palveluilta enemmän kuin koskaan ennen. Palveluiden täytyy olla nopeita, helppokäyttöisiä sekä aina saatavilla. Näiden lisäksi palvelujen odotetaan nykyään myös olevan globaalisti toimivia. Palvelun ollessa globaalisti toimiva, myös käyttäjiä on ympäri maailmaa useilla eri aikavyöhykkeillä. Tällöin huoltokatkoja aiheuttavia ylläpitotoimenpiteitä on mahdotonta ajoittaa sopiviin ajankohtiin.

Tietokantapalvelimet luonteensa vuoksi ovat olleet yksi ongelma-alue korkean saatavuuden tarjoamisessa. Työssä tutkittiin ja kokeiltiin olisiko Patroni-nimeä kantava avoimeen lähdekoodiin perustuva projekti vastaus tähän ongelmaan. Konkreettisena työnä tutustuttiin Patroni-projektiin sekä pystytettiin ja konfiguroitiin tietokantaklusteri sitä käyttäen. Tietokantaklusterille luotiin myös korkean saatavuuden kuormantasaus käyttäen HAProxyja.

Tuloksissa testattiin tietokantaklusterin sekä kuormantasauksen automaattista vikasietoisuuskykyä. Testaus kohdistui automaattisten vikasietoisuusmekanismien toimivuuksien todentamiseen. Tuloksena saatiin tieto järjestelmän kyvystä muuttaa klusterissa olevien komponenttien rooleja vikatilanteessa.

Työssä päädyttiin johtopäätökseen, jossa Patroni olisi mahdollinen ratkaisu tietokantojen orkestrointiin korkean saatavuuden järjestelmässä. Luotettavuuden varmistamiseksi täytyisi vielä testata järjestelmää tuotantoympäristöön verrattavassa ympäristössä.

Avainsanat (asiasanat)

Patroni, tietokanta, tietokantaklusteri, postgreSQL, korkea saatavuus, vikasietoisuus

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Autio, Sami	Bachelor's thesis	May 2018
		Language of publication: Finnish
	Number of pages: 52	Permission for web publication: x

Title of publication

Database solution with high availability

Degree programme

Software Engineering

Supervisor(s)

Rantala, Ari

Assigned by

Kaaos Unlimited Oy

Abstract

Today, with the development of digitalization, end users require more than ever from services. Services needs to be fast, easy to use and always available. In addition, services are expected to function globally. When a service is globally functioning, its users are divided into multiple time zones around the world. Due to this, service downtimes are also impossible to schedule at appropriate times.

Due to their nature, database servers have been one problem area in providing high availability. The solution investigated and experimented if an open source project, Patroni, could provide the answer to this problem. The concrete part of this thesis was to familiarize onself with Patroni and set up a database cluster using it. Highly available load balancing using HAProxies was also created for the database cluster.

Database cluster and load balancing were tested for automatic fault tolerance. The testing focused on verifying an automatic failover mechanism. Information was obtained about the ability of the system to change roles of the components in a failover situation.

The thesis concluded that Patroni could be a possible solution for orchestrating databases in a high availability system. To ensure reliability, the solution should be further tested in an environment equivalent to the production environment.

Keywords/tags (subjects)

Patroni, database, database cluster, postgreSQL, high availability, fault tolerance

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Sanasto

Ansible

Automaatiotyökalu, jolla voidaan automatisoida muun muassa konfiguraatioiden hallintaa ja ohjelmistojen käyttöönottoa.

Apache

Apache Foundation:in ylläpitämä HTTP-palvelinohjelmisto.

API

Ohjelmointirajapinta (application programming interface), jolla palvelut voivat keskustella keskenään.

CentOS

Red Hat Enterprise Linux käyttöjärjestelmästä johdettu avoimeen lähdekoodiin perustuva käyttöjärjestelmä.

Debian

Debian-projektin ylläpitämä Linux-pohjainen käyttöjärjestelmä.

Django

Korkean tason Python web-sovelluskehys, jota ylläpitää Django Software Foundation.

Github

Ohjelmistojen kehitysalusta, joka tarjoaa versionhallintaa pilvipalveluna.

IP-osoite

Verkko-osoite, jota käytetään tietoliikennepakettien ohjaamiseen.

Legacy

Vanhaksi käynyt ohjelmisto tai osa sitä. Työssä tällä viitataan vanhaan järjestelmään.

MySQL

Avoimeen lähdekoodiin perustuva tietokantamoottori.

Nginx

Ilmainen, avoimeen lähdekoodiin perustuva ja suorituskyvyltään korkea HTTP-palvelinohjelmisto.

Perl

Ohjelmointikieliperhe, johon kuuluu Perl 5 ja Perl 6. Työssä tällä viitataan Perl 5 kieleen.

Phoenix

Työnimi työssä viitatulle Legacyn korvaavalle uudelle järjestelmälle.

Python

Itsenäisen voittoa tavoittelemattoman Python Software Foundation organisaation ylläpitämä ohjelmointikieli.

Vagrant

Työkalu virtuaalikoneiden rakentamiseen ja ylläpitoon.

Zalando

Saksassa 2008 perustettu yritys ja nykyään Euroopan alueella toimiva muotivaatteisiin erikoistunut verkkokauppa.

1 Lähtökohdat

1.1 Taustaa

Opinnäytetyön aihe tuli ajankohtaiseksi, kun toimeksiantaja (ks. Luku 1.2) tarvitsi modernin ratkaisun tietokannoille siirtyessään tuottamaan palvelua vanhan järjestelmän (Legacy) sijaan uudella järjestelmällä (Phoenix). Järjestelmää käytetään maailmanlaajuisesti rekrytointiprosessien yhteydessä, jolloin uusi modernisoitu järjestelmä tulisi saada mahdollisimman huoltokatkottomaksi.

Aika-ajoin tulee tarve päivittää tuotantopalvelimia tai tehdä joitain muita ylläpitotehtäviä, jotka Legacy-järjestelmässä vaativat huoltokatkon. Huoltokatkot ovat epämieluisia palvelun kaikille osapuolille.

Loppukäyttäjille nämä ilmenevät hetkinä, jolloin palvelua ei voi käyttää. Tämä voi pahimmassa tapauksessa johtaa siihen, ettei työnantaja ja mahdollisesti potentiaalinen työntekijä tule ikinä kohtaamaan toisiaan. Vaikka työntekijä palaisikin tekemään hakemuksensa myöhemmin, tästä voi olla jo syntynyt haittaa työnantajaimagolle.

Palveluntarjoajan näkökulmasta huoltokatkot ovat hankalia toteuttaa varsinkin nopealla aikataululla, koska näistä voi joutua sopimaan asiakkaiden kanssa ensin. Joidenkin asiakkaiden kanssa voi myös olla tehty palvelutasosopimuksia (service level agreement), jotka saattavat rajoittaa katkojen pituutta tai määrää.

Tietokantapalvelimet ovat yksi osa-alue Legacy-järjestelmässä, jotka näitä kyseisiä huoltokatkoja aiheuttavat. Tulevassa Phoenix-järjestelmässä tähän osa-alueeseen tarvittiin muutos.

Korkean saatavuuden tietokantaratkaisuille on nykyään olemassa tarjolla muutamia erilaisia vaihtoehtoja, joista tässä työssä on päädytty käsittelemään Patroni-nimistä tietokantojen orkestrointityökalua.

1.2 Toimeksiantaja

Opinnäytetyön toimeksiantajana toimi Kaaos Unlimited Oy (Kaaos), joka on Jyväskylästä lähtöisin oleva ohjelmistotalo. Kaaos työllistää viisi työntekijää ja lukeutuu näin

ollen pieneksi yritykseksi. Pienestä koostaan huolimatta sen asiakkaina on järjestötoimijoita aina julkisen pörssin noteeraamiin globaalisti toimiviin konserneihin asti.

1.3 Nykyinen järjestelmä

Kaaoksen päätuote on Artist-niminen rekrytointijärjestelmä. Artist on 2000-luvun alussa alkunsa saanut järjestelmä, joka on toteutettu käyttäen siihen aikaan saatavilla olevilla tekniikoilla. Artist-järjestelmä on Perl-ohjelmointikielellä toteutettu websovellus, jossa palvelinohjelmistoina toimivat Apache ja Nginx. Tietokantaratkaisuna järjestelmässä toimii kahdennettu MySQL-tietokanta, joiden välille on luotu replikointi.

Tietokannoille ei ole toteutettu minkäänlaista automatisoitua vikasietoisuuskykyä (automated failover). Palautuminen virhetilasta vaatisi joko virtuaalikoneen palauttamisen varmuuskopioista, tai vaihtoehtoisesti täytyisi toinen palvelin konfiguroida palvelemaan manuaalisin keinoin. Päätyminen tähän tilaan ja siitä manuaalisin keinoin palautuminen on tietysti aikaa vievää. Se voi myös tapahtua milloin tahansa, kuten normaalin työajan ulkopuolella, jolloin reagointikyky on vielä heikompaa.

1.4 Tuleva järjestelmä

Tuleva järjestelmä toteutetaan Python-ohjelmointikielellä Django-sovelluskehystä hyödyntäen. Vaikka Django tukeekin MySQL-tietokantaa natiivisti, on kyseisessä tietokantamoottorissa silti omat puutteensa. Puutteet tulevat muun muassa JSON-datatyypin tuesta. Tästä syystä Phoenix-järjestelmässä on päädytty käyttämään postgreSQL-tietokantamoottoria.

Tuleva järjestelmä on tarkoitus saada täysin vikasietoiseksi (fault tolerance), eli kaikkien sen infrastruktuurikomponenttien tulisi tällöin olla vähintään kahdennettu. Ongelmatilanteessa järjestelmän tulisi myös pystyä reagoimaan sekä palautumaan toimintakykyiseksi automaattisesti.

1.5 Tavoitteet ja rajaukset

Työn tavoitteena oli luoda erityisesti Phoenix-järjestelmää varten korkean saatavuuden tietokantaratkaisu. Ratkaisun täytyisi olla palvelumuotoinen ja uudelleen käytettävissä, eikä vain kiinteä osa uutta järjestelmää. Palvelumuotoisuudella ja uudelleen käytettävyydellä tarkoitetaan tässä yhteydessä sitä, että sovellus joka toteutettavaa ratkaisua käyttäisi, voisi olla käytännössä mikä tahansa. Ratkaisun lopputuloksena olisi siis tietokantaklusteri, joka olisi saatavilla yhden IP-osoitteen takana.

Ratkaisun täytyy myös ottaa huomioon eri toimintaympäristöt (ks. Luku 2.2) ja olla yhteensopiva niiden kanssa. Työssä myös rajattiin käsiteltäväksi toimintaympäristöksi ainoastaan kehitysympäristö. Vaikkakaan erot ympäristöjen välillä eivät ole suuret, niin jokaisessa on silti oma verkkoinfrastruktuurinsa.

Näiden tavoitteiden saavuttamiseksi oli työssä ennalta määrätty tutkittavaksi työkaluksi Patroni-niminen tietokantojen orkestrointityökalu. Työssä ei siis vertailtu eri työkaluja vaan keskityttiin vain kyseiseen työkaluun, sekä sen tuomiin ja siitä puuttuviin ominaisuuksiin.

2 Vaatimusmäärittely

2.1 Yleistä

Vaatimusmäärittelyssä käydään läpi kaikki ennalta tiedostetut vaatimukset ratkaisulle. Tämä määrittely tukee ratkaisuun liittyvien vaihtoehtojen tutkimista ja vertailua; sekä toimii myös ohjaavana apuvälineenä ratkaisua toteuttaessa.

Yleisiä vaatimuksia ratkaisulle oli asetettu liittyen lähdekoodin saatavuuteen, ylläpitoon ja muiden työkalujen kanssa yhteensopivuuteen. Lähdekoodin täytyi olla avoimesti saatavilla (open source), ja järjestelmän täytyi olla ylläpidettävissä kokonaisuudessaan itse ilman kolmannen osapuolen palveluita (self hosted). Toteutuksen täytyi myös olla hallittavissa Vagrant- ja Ansible-työkalujen avulla. Vagrant-yhteensopivuus liittyy vain kehitysympäristöön, kun taas Ansible yhteensopivuus kaikkiin toimintaympäristöihin. Vagrant- ja Ansible-skriptit ovat yksinkertaistamisen vuoksi jätetty työstä pois ja konfiguroinnit tapahtuvat manuaalisin keinoin.

Työssä palvelinten lukumäärää oli rajoitettu (ks. Luku 2.3), mutta toteutuksen täytyi jatkoa ajatellen olla skaalattavissa (ks. Luku 3.5).

Lisäksi toteutuksen luonteestakin johtuen muun muassa palvelinten päivittäminen täytyi olla joko täysin tai ainakin lähes täysin huoltokatkotonta. Palvelussa saa esiintyä hyvin lyhyt hetkellinen hitaus, mutta kokonaista katkoa ei saa syntyä.

2.2 Toimintaympäristöt

Toimintaympäristöllä tarkoitetaan työssä palvelimiin, niiden sijaintiin sekä niiden verkkomäärityksiin kohdistuvia kokonaisuuksia. Näitä voivat olla muun muassa paikallisella työasemalla oleva virtualisoitu toimintaympäristö tai kolmannen osapuolen fyysiset tai virtuaaliset palvelinympäristöt.

Toimintaympäristöjä työssä käsiteltävälle tietokantaratkaisulle on kolme. Ensimmäinen ympäristö, johon myöskin tämä työ painottuu, on kehitysympäristö. Kehitysympäristö kattaa paikalliselle työasemalle virtualisoituun ympäristöön rakennetun kokonaisuuden.

Toinen ympäristö on yrityksen sisäinen pilvi-infrastruktuuri, joka koostuu fyysisillä palvelimilla pyörivistä virtuaalikoneista. Tämä ympäristö toimii myös kokeilu- ja testausalustana uusille infrastruktuurimuutoksille ja -päivityksille.

Viimeinen ympäristö on tuotantoympäristö, joka koostuu kolmannen osapuolen tarjoamista virtuaalikoneista ja muista palveluista. Kolmannella osapuolella tarkoitetaan mitä tahansa yleisesti tunnettua pilvipalvelun tarjoajaa, kuten muun muassa AWS, Upcloud, Rackspace tai jokin muu vastaava palvelun tarjoaja. Tuotantoympäristö on se ympäristö, jossa itse asiakkaiden käytössä oleva palvelu sijaitsee.

2.3 Palvelimet

Palvelimille oli asetettu muutama vaatimus, jotka koskivat lähinnä käyttöjärjestelmiä ja verkkoasetuksia. Palvelinten määrä oli myös tämän työn osalta rajoitettu viiteen (5) palvelimeen. Nämä viisi palvelinta jakautuivat niin, että kolme toimi tietokantapalvelimina ja kaksi kuormantasaajina.

Palvelinkäyttöjärjestelmäksi vaatimuksena oli uusin vakaa versio CentOS käyttöjärjestelmästä, joka työtä tehdessä oli versionumeroltaan 7. Tämä vaatimus perustui lähinnä käyttöjärjestelmän elinkaaren pituuteen, joka CentOS käyttöjärjestelmillä on pidempi kuin esimerkiksi Debian-käyttöjärjestelmillä. Debian käyttöjärjestelmillä täysi tuki kestää kolme vuotta ja ylläpitopäivityksiä tulee viiden vuoden ajan (Debian Releases n.d.). CentOS-käyttöjärjestelmillä puolestaan täysi tuki kestää viisi vuotta ja ylläpitopäivityksiä tulee CentOS 7 -versiossa yhdeksän vuoden ajan (CentOS Product Specifications n.d.).

Verkkoasetuksien vaatimukset tulivat pääasiallisesti toimintaympäristöjen infrastruktuurien arkkitehtuurisista rajoitteista, sekä osaksi hallintaan ja orkestrointiin käytettyjen työkalujen pohjalta. Käytännössä orkestrointityökalujen kohdalla nämä pitivät sisällään kovennetut hallintarajapinnat, joiden kautta palveluja voidaan ylläpitää ja hallita.

2.4 Tietokantamoottori

Legacy-järjestelmässä on käytössä tietokantamoottorina MySQL versio 5.1. Phoenix-järjestelmän tarpeisiin MySQL-tietokantamoottori ei ole enää riittävä, vaan tulevaan järjestelmään vaatimuksena on tietokantamoottorin oltava PostgreSQL. PostgreSQL-versioksi toimeksiantaja oli asettanut vaatimukseksi uusimman vakaan version. Työn tekemisen hetkellä uusin vakaa versio oli versionumeroltaan 10.

3 Suunnittelu ja teknologiavalinnat

3.1 Yleistä

PostgreSQL-tietokantamoottorille korkean saatavuuden toteuttamiseksi on olemassa myös muita ratkaisuja, mutta yhdessä toimeksiantajan kanssa oli päädytty tutkimaan tarkemmin vain yhtä mahdollista ratkaisua, ja tästä syystä muut ratkaisut ja niiden vertailu on jätetty työn ulkopuolelle.

Työssä käsiteltäväksi ratkaisuvaihtoehdoksi toimeksiantajan aikaisemmin tekemän työn perusteella valittiin Patroni-niminen tietokantojen orkestrointityökalu, koska se vaikutti täyttävän ratkaisulle asetetut vaatimukset. Se on suunniteltu käytettäväksi

PostgreSQL-tietokantamoottorin kanssa. Patroni myös sisältää valmiudet siihen, että sillä voidaan toteuttaa korkea saatavuus käyttäen itse rakennettua kuormantasaus ratkaisua.

3.2 Arkkitehtuuri

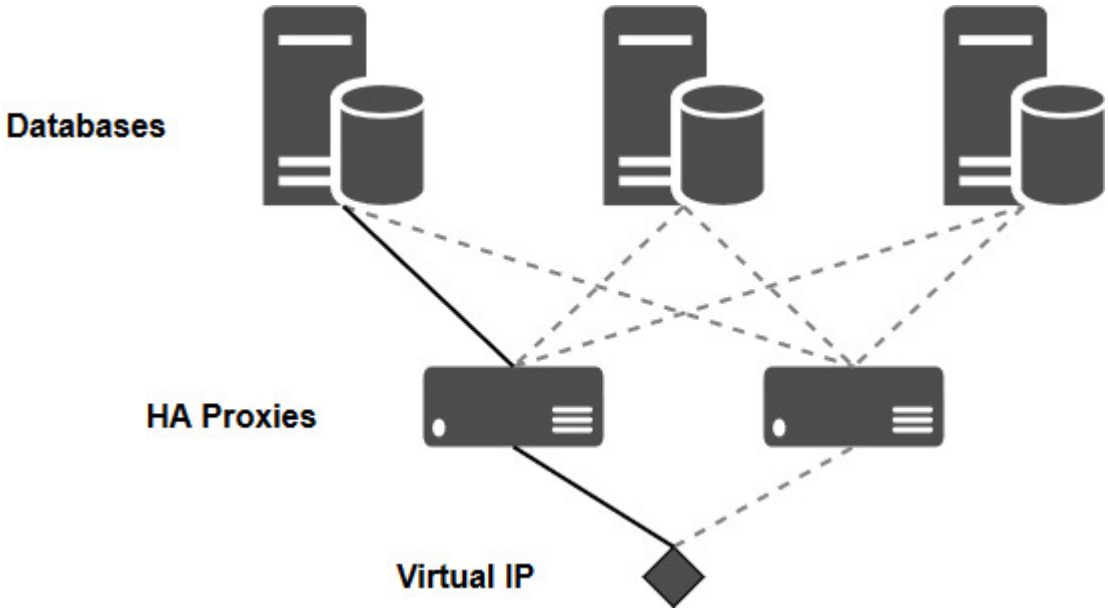
Ratkaisussa oli kolme tietokantapalvelinta, jotka muodostavat yhdessä samalla myös etcd-klusterin. Asiakasliikenteen ohjaus näille palvelimille tapahtuu kuormantasauksen kautta. Kuormantasaus oli ratkaisussa kahdennettu, eli sitä varten oli varattu kaksi palvelinta (ks. Taulukko 1).

Taulukko 1. Palvelimien resurssit ja palvelut

Nimi	Ytimien lkm	Muisti (gb)	Palvelut	Portit
caha01	1	0,5	HAProxy, Keepalived	5000, 7000
caha02	1	0,5	HAProxy, Keepalived	5000, 7000
cadb01	1	1	Patroni, PostgreSQL, Etcd,	2380, 5432, 8008
cadb02	1	1	Patroni, PostgreSQL, Etcd,	2380, 5432, 8008
cadb03	1	1	Patroni, PostgreSQL, Etcd,	2380, 5432, 8008

Asiakasliikenne kulkee klusterissa aina vain yhtä reittiä pitkin. Tämä tarkoittaa sitä, että tietokantapalvelimista ja kuormantasauspalvelimista on aina vain yksi molemmista käsittelemässä asiakasliikennettä. Mikäli käsittelevä palvelin päätyy toimintakyvyttömään tilaan, niin reitti muuttuu automaattisesti. Vikatilanteen syntyessä tietokantapalvelimen reititys vaihtuu toiselle toimintakykyiselle tietokantapalvelimelle.

Kuormantasauspalvelimen kohdalla reititys vaihtuu kiertämään toisen kuormantasauspalvelimen kautta (ks. Kuvio 1).



Kuvio 1. Arkkitehtuuri

3.3 Palomuuri

Palomuurin pohjimmainen tarkoitus on suodattaa tietoliikennettä verkkojen välillä (Palomuuri 2015). Sitä voidaan myös jossain tapauksissa käyttää liikenteen ohjaamiseen. Linuxin ytimessä on ollut jo pitkään Netfilter-niminen järjestelmä, joka mahdollistaa pakettien muokkaamisen ja pysäyttämisen. Tätä järjestelmää ohjataan käyttäen jotain saatavilla olevaa palomuurisovellusta. (Netfilter 2015.)

CentOS 7 -käyttöjärjestelmässä vakiona palomuurisovelluksena toimii FirewallD. FirewallD on dynaaminen palomuuri, joka tarkoittaa sitä, että sen asetuksia voidaan muuttaa suoraan ohjelmaa ajaessa ilman tarvetta palomuurin uudelleenkäynnistämiselle. Kyseisessä palomuuriohjelmassa on myös ominaisuus, jolla voidaan luoda verkkoalueita (network zone). Nämä verkkoalueet voidaan liittää tiettyihin yhteyksiin tai rajapintoihin ja niille voidaan määrittää eri luottamustasoja. (FirewallD 2016.)

Toinen palomuurisovellus on CentOS-käyttöjärjestelmissäkin aikaisemmin vakiona ollut Iptables. Iptables on yksinkertaisempi palomuurisovellus kuin FirewallD. Siinä

säännöt luodaan ketjuihin, joiden mukaan pakettien liikennettä ohjataan ja rajoitetaan (Iptables 2016).

Näistä kahdesta palomuurisovelluksesta oli päädytty käyttämään Iptablesia. Tähän valintaan päädyttiin toimeksiantajan asettamien ohjeistuksien pohjalta. Lopullisen ratkaisun näkökulmasta tällä valinnalla ei ollut toiminnallista vaikutusta.

3.4 Patroni

Patroni on alun perin Zalandon sisäinen projekti, joka myöhemmin vuonna 2015 julkaistiin avoimena lähdekoodina GitHub-palveluun. Se on oman tulkintansa mukaan sapluuna (template), jolla voidaan luoda kustomoitu korkean saatavuuden ratkaisu PostgreSQL-tietokannoille (Patroni introduction n.d.).

Patroni perustuu Governor projektiin (Patroni n.d.) ja perii näin ollen sen ominaisuudet tuoden lisäksi omiaan. Se on myöskin syrjäyttänyt alkuperäisen Governor-projektin (Governor n.d.).

Patroni-klusteria ohjataan patronictl-työkalulla, joka tulee Patronin pip-paketin asentamisen yhteydessä. Patronictl-työkalulla on mahdollista saada tietoa klusterin komponenteista ja niiden tiloista, sekä vaikuttaa itse niihin. Työkalu tukee muun muassa manuaalista leaderin vaihtoa (switchover, manual failover).

3.5 Palvelinten skaalaus

Skaalaus tarkoittaa palvelun resurssien suurentamista tai pienentämistä. Suurentamisesta puhuttaessa tarkoitetaan yleensä ylöspäin skaalaamista ja vastaavasti pienentämisestä puhuttaessa alaspäin skaalaamista.

Skaalausta ylöspäin voi tehdä kahdella tavalla, joista yleensä käytetään nimityksiä vertikaalinen ja horisontaalinen. Vertikaalisessa skaalauksessa palvelimien lukumäärä pysyy samana, mutta niihin lisätään tehoja kuten suoritinytimien määrää tai muistia. Horisontaalisessa skaalauksessa lisätään itse palvelimien määrää tuoden sitä kautta lisää resursseja palvelun käytettäväksi.

Horisontaalisessa skaalauksessa myös itse palvelu täytyy olla suunniteltu niin, että se tukee tätä vaihtoehtoa. Tutkinnan alla olevassa Patronissa tämä on otettu huomioon niin, että palvelimia lisäämällä voidaan parantaa järjestelmän virhesietoisuuskykyä.

3.6 Replikointi

3.6.1 Yleistä

Replikointi tarkoittaa tietokantojen välistä automaattisesti tapahtuvaa kopiointia. PostgreSQL-tietokantamoottori tukee muutamaa eri replikointitekniikkaa kuten muun muassa tiedostopohjaista (file-based) lokitiedostojen siirtoa (Log-Shipping Standby Servers n.d.) ja suoratoistoreplikointia (streaming replication), jossa transaktiolokeja siirretään toiselle palvelimelle heti kun niitä tulee (Streaming Replication n.d.).

3.6.2 Symmetrisyys

Patroni käyttää PostgreSQL-tietokantamoottorin suoratoistoreplikointia (Patroni replication modes n.d.). Suoratoistoreplikaatio on Patronissa mahdollista toteuttaa kahdella eri tavalla. Vakiokonfiguraatioilla käytössä on asymmetrinen replikaatio, jossa saatavuus on priorisoitu. Asymmetrisellä konfiguraatiolla vastaus transaktion onnistumisesta tulee heti, kun leader-tietokanta on suorittanut sen kirjoittamisen. Tämä vaihtoehto saattaa aiheuttaa tietojen katoamista tilanteessa, jossa leader-tietokanta päätyy toimintakyvyttömään tilaan, jonka seurauksena Patroni ylentää (promote) toissijaisen (secondary) palvelimen uudeksi leaderiksi (Patroni replication modes n.d.).

Toinen vaihtoehto on konfiguroida Patroni käyttämään symmetristä replikaatiota. Symmetrisessä replikaatiossa on priorisoitu eheys. Eheyden varmistamiseksi vastaus transaktion onnistumisesta tulee vasta, kun toissijainenkin (secondary) palvelin on varmistanut kirjoitusoperaation onnistuneeksi. Varmistus tulee aina tietyltä toissijaiselta tietokannalta, jonka Patroni määrittää automaattisesti. Tietoa tästä toissijaisesti tietokannasta ylläpidetään avain–arvo-paritietokannassa (ks. Luku 3.7.1). Leader-tie-

tokannan joutuessa vikatilaan Patroni ylentäisi kyseisen toissijaisen tietokannan uudeksi leaderiksi, jolloin tiedon eheydestä voidaan olla varmoja. (Patroni replication modes n.d.).

Työssä konfiguraatioita ei ole muutettu symmetrisyyden kohdalla, joten käytössä on asymmetrinen replikointi. Tuotantoympäristön olosuhteissa näistä kahdesta vaihtoehdosta päädyttiin käyttämään symmetristä replikaatiota, koska eheyden osuus on silloin kaikkein tärkeintä. Lisäksi suorituskyvyn oletettiin olevan tarpeisiin nähden tarpeeksi hyvä myös sillä vaihtoehdolla.

3.6.3 Replikointijärjestys

Toteutuksessa on kolme tietokantapalvelinta, joista vain yksi voi olla leader-roolissa kerrallaan. Jäljelle jäävien kahden toissijaisen seuraajan replikointijärjestys voidaan konfiguroida kahdella eri tavalla. Ensimmäinen vaihtoehto on asettaa molemmat seuraamaan aina sen hetkistä leader-tietokantaa (ks. Kuvio 2). Tällöin virhetilanteen syntyessä olisi teoriassa kaksi saman tasoista vaihtoehtoa seuraavaksi leader-tietokannaksi.

Kuvio 2. Molemmat toissijaiset tietokannat seuraavat leader-tietokantaa

Toinen vaihtoehto on asettaa replikointijärjestys vesiputousmalliseksi (cascading) (ks. Kuvio 3). Tällöin leader-solmun joutuessa virhetilaan olisi suoraan sitä seuraava seuraaja ensisijaisesti uusi leader-rooliin ylennettävä.

Kuvio 3. Tietokannat vesiputousmallissa

Hyöty vesiputousmallista tulee suurempien klustereiden kohdalla. Tällöin voidaan asettaa vesiputousmallisia replikointisuhteita ja tällä tavalla vähentää rasitusta leader-palvelimelta ohjaamalla suorat yhteydet muille palvelimille (Log-Shipping Standby Servers n.d.).

Toteutettava ratkaisu oli kolmen palvelimen kokonaisuus, jolloin vesiputousmallista ei ollut juurikaan hyötyä. Tästä syystä replikoinnissa päädyttiin käyttämään ensimmäistä vaihtoehtoa, jossa toissijaiset palvelimet replikoivat suoraan leader-palvelimelta.

3.7 Korkea saatavuus

Saatavuudella tarkoitetaan aikaa, jolloin palvelu on saatavilla, sekä sitä kuinka kauan järjestelmältä kestää vastata käyttäjän tekemään pyyntöön (Heidi, E. 2016.). Korkean saatavuuden järjestelmissä on pyritty minimoimaan aika, jolloin palvelu ei olisi saatavilla.

3.7.1 Avain–arvo-parivarastointi

Avain–arvo-paritietokannat ovat yksinkertaisia tietokantoja, joissa jokaista avainta kohti on määritetty yksi arvo. Tätä avaimen ja arvon yhteyttä kutsutaan avain–arvo-pariksi. (What is a Key-Value Store? n.d.)

Ratkaisussa tarvittiin avain–arvo-paritietokantaa tallentamaan tietoa klusterin osista ja niiden tiloista. Näitä tietoja tarvitaan klusterin ylläpitämiseen, jossa on erittäin tär-

keää pystyä muodostamaan luotettava kokonaistilannekuva millä tahansa ajanhetkellä. Kyseisiin tietoihin nojaten tehdään muun muassa automatisoituja ratkaisuja ongelmatilanteisiin reagoidessa.

Patroni tukee kahta eri avain–arvo-paritietokantaa, jotka ovat etcd ja Consul. Molemmat ovat ilmaisia ja avoimeen lähdekoodiin perustuvia. Etcd tuotteen takaa löytyy CoreOS niminen yritys ja Consul tuotteen takaa puolestaan HashiCorp. Molemmat käyttävät konsensuksen saavuttamiseen Raft-algoritmia (ks. Luku 3.7.3). Suurin ero näiden kahden välillä tulee ominaisuuksissa määrässä, joissa Consul on kokonaisvaltaisempi ratkaisu tarjoten laajemman määrän ominaisuuksia. Consul tarjoaa muun muassa palveluiden löytämisen (service discovery) ja tuen klusterin hajauttamiseen eri palvelinkeskuksiin (Introduction to Consul n.d.). Etcd vaatii näiden ominaisuuksien saavuttamiseen kolmannen osapuolen ratkaisuja tuekseen.

Työssä toteutettu ratkaisu ei tarvinnut lisäominaisuuksia, joita Consul olisi tarjonnut, joten molemmat vaihtoehdot olivat hyvin saman tasoisia. Etcd tuoden mukanaan ratkaisun näkökulmasta vähemmän ylimääräisiä ominaisuuksia tekee siitä yksinkertaisemman ja valittiin tästä syystä käytettäväksi ratkaisussa.

3.7.2 Kuormantasaus

Kuormantasaus on oleellinen osa korkean saatavuuden infrastruktuureissa. Kuormantasausta käytetään parantamaan suorituskykyä ja luotettavuutta jakamalla työkuormaa usealle palvelimelle (Anderson, M. 2017). Lisäksi tällä tavalla saadaan luotua järjestelmälle vikasietoisuuskykyä eliminoimalla mahdollisia yksittäisiä vikaantumispaikkoja (single point of failure). Toteutuksessa kuormantasaus toteutettiinkin juuri vikasietoisuuskyvyn näkökulmasta.

HAProxy on ilmainen avoimeen lähdekoodiin perustuva kuormantasausohjelmisto TCP- ja HTTP-pohjaisille sovelluksille (HAProxy Description n.d.). HAProxy on suosittu ratkaisu kuormantasaukseen (Anicas, M. 2014) ja toimeksiantajalle ennestään tuttu. Lisäksi Patroni käyttää HAProxya myös esimerkeissään. Näistä syistä päädyttiin käyttämään HAProxya osana ratkaisua.

Yksittäisellä HAProxy-palvelimella ei voida toteuttaa korkean saatavuuden ratkaisua, sillä se tuottaisi yksittäisen vikaantumispaikan. Tämä tarkoittaa sitä, että mikäli palvelin päätyisi toimintakyvyttömään tilaan, niin myös ratkaisun saatavuus menetettäisiin samalla. Korkean saatavuuden toteuttamiseksi täytyi HAProxy-palvelin kahdentaa.

Vikasietoisuuden toteuttamiseksi kuormantasaus tarvitsee lisäkseen liikenteen ohjaamista. Liikenteen ohjausta varten ei erityisesti etsitty tai vertailtu vaihtoehtoja, sillä tiedossa oli alusta alkaen kaksi vaihtoehtoa. Vaihtoehdot olivat Keepalived ja Heartbeat. Vaihtoehdoista valikoitui käytettäväksi ratkaisuksi Keepalived, koska tästä oli kokemusta jo valmiiksi, joten se tiedettiin varmasti toimivaksi valinnaksi.

Keepalived on ilmainen ja avoimeen lähdekoodiin perustuva reititysohjelmisto, jonka tehtävänä on luoda tukeva pohja kuormantasaukselle ja korkealle saatavuudelle Linux ja Linuxiin pohjautuvissa ympäristöissä (What is Keepalived? n.d.). Keepalived-ohjelmiston vastuulla on varmistaa liikenteen ohjaus aina toimintakykyiselle HAProxy-palvelimelle.

3.7.3 Saatavuuden orkestrointi

Ratkaisussa voi olla vain yksi pääasiallinen kuormantasaus- ja tietokantapalvelin kerrallaan, jotta se toimisi odotetusti ja hallitusti. Tästä syystä näiden palvelimien tuottamien palveluiden orkestroiminen on erittäin tärkeässä roolissa.

Vikasietoisuuteen kykenevän järjestelmän täytyy pystyä määrittämään palveluidensa tilaa itsenäisesti ja luotettavasti. Ratkaisussa käytettiin vikasietoisuuteen kykenevää orkestrointia tietokantojen sekä myös kuormantasaajien kanssa.

Tietokantojen orkestrointi toteutettiin käyttäen Patroni-työkalua, joka puolestaan nojautuu toimenpiteissään etcd avain–arvo-paritietokantaan. Etcd pitää tietoa järjestelmän komponenttien tiloista. Luotettavuuden näille tiedoille tuo algoritmi, jolla arvoja ylläpidetään.

Algoritmi jota käytetään arvojen asettamiseen ja muuttamiseen etcd-tietokannassa on Raft. Raft on konsensus algoritmi, joka on suunniteltu olevan helposti ymmärrettävä (What is Raft? n.d.). Päätös arvojen asettamisesta ja muuttamisesta perustuu etcd-tietokantojen konsensukseen, jolla voidaan varmistaa tietojen luotettavuus.

Konsensuksen muodostamiseksi vaaditaan enemmistön yksimielisyys. Tämä tarkoittaa sitä, että kolmen etcd-tietokannan klusterista kaksi täytyy olla toimintakykyisiä, jotta konsensus on mahdollista toteutua.

Kuormantasauksen orkestrointiin käytettiin Keepalived-reititysohjelmistoa. Keepalived-ohjelmiston tehtävänä oli seurata kuormantasauspalvelimien tilaa. Mikäli ensisijainen palvelin päätyy toimintakyvyttömäksi niin vaihtaa Keepalived kelluvan ip-osoitteen (floating ip) toiselle palvelimelle, jolloin toissijainen palvelin ottaa reitityksen vastuulleen ja nousee näin ollen ensisijaisen palvelimen rooliin.

4 Toteutus

4.1 Huomioita toteutuksesta

Jokaiselle tietokantapalvelimelle asennettiin PostgreSQL-tietokantamoottorin lisäksi etcd. Etcd-klusterin palvelut olisi suuremman kokoluokan toteutuksissa hyvä olla eriytettynä niille varatuille palvelimille paremman suorituskyvyn takaamiseksi. Tässä ratkaisussa palvelinten määrää oli kuitenkin rajoitettu, minkä vuoksi ne sijoitettiin tietokantapalvelimille.

Kuormantasauspalvelimille asennettiin HAProxy-kuormantasausohjelmistoksi. Näille palvelimille asennettiin myös Keepalived korkean saatavuuden toteuttamiseksi.

4.2 Alustavat toimenpiteet

Tietokanta- ja kuormantasauspalvelimilla on osa alustavista toimenpiteistä yhtenäisiä. Näihin lukeutuvat alustavat päivittämiset ja palomuurisovelluksen vaihtaminen.

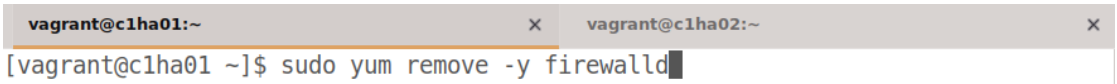
Kaikissa toteutuksen palvelimissa käytetiin Centos 7 -käyttöjärjestelmiä, jotka oli hankittu valmiina Vagrant-paketteina. Nämä paketit eivät kuitenkaan olleet täysin ajan tasalla, joten aluksi tietokantapalvelimet päivitettiin ajan tasalle (ks. Kuvio 4). Tämä sama päivitys tehtiin myös kuormantasauspalvelimille (ks. Kuvio 5).

Kuvio 4. Tietokantapalvelimien päivittäminen

Kuvio 5. Kuormantasauspalvelimien päivittäminen

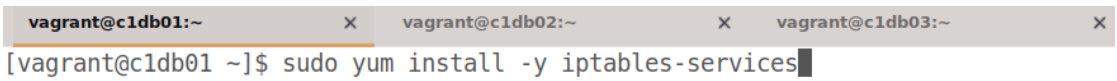
Toteutuksen teknologiavalinnoissa päädyttiin käyttämään iptables-palomuuriohjelmistoa käyttöjärjestelmän mukana tulevan FirewallD-ohjelmiston sijaan. Tämän muutoksen toteuttamiseksi täytyi ensiksi FirewallD poistaa tietokantapalvelimilta (ks. Kuvio 6). Sama päätös koskee myös kuormantasauspalvelimia, jolloin poistetaan FirewallD myös niiltä (ks. Kuvio 7).

Kuvio 6. FirewallD poistaminen tietokantapalvelimilta



Kuvio 7. FirewallD poistaminen kuormantasauspalvelimilta

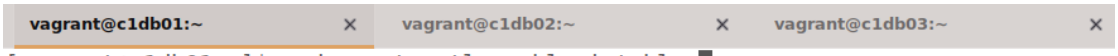
Kun FirewallD oli poistettu kaikilta tietokantapalvelimilta, niin voitiin niille asentaa iptables (Kuvio 8). Samalla tavalla asennettiin iptables myös kuormantasauspalvelimille (ks. Kuvio 9).



Kuvio 8. Iptables asentaminen tietokantapalvelimille

Kuvio 9. Iptables asentaminen kuormantasauspalvelimille

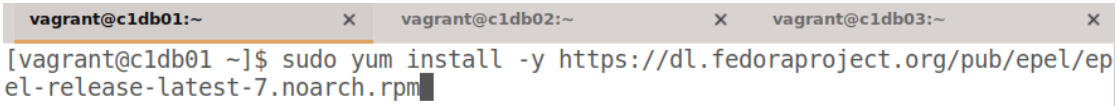
Iptables-ohjelmiston asennuksen jälkeen täytyi muistaa myös asettaa se käynnistymään käyttöjärjestelmän käynnistyksen yhteydessä (ks. Kuvio 10, Kuvio 11).



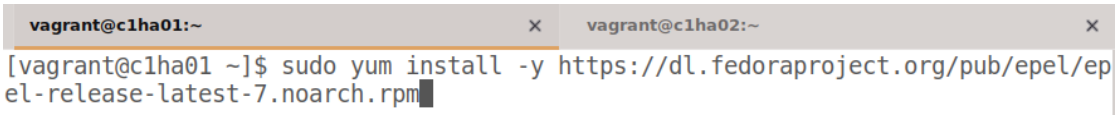
Kuvio 10. Iptables asettaminen käynnistymään automaattisesti tietokantapalvelimilla

Kuvio 11. Iptables asettaminen käynnistymään automaattisesti kuormantasauspalvelimilla

Kaikille palvelimille lisättiin myös epel-repositorio (ks. Kuvio 12, Kuvio 13). Tietokantapalvelimet tarvitsivat tätä Python-työkalujen asentamista varten. Kuormantasauspalvelimet eivät tämän työn kohdalla tätä tarvitsisi, mutta lisättiin se myös niille, koska toimeksiantajalla oli myöhemmin tarve tähän.



Kuvio 12. Epel-repositorion lisääminen tietokantapalvelimille



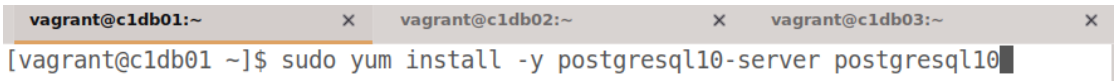
Kuvio 13. Epel-repositorion lisääminen kuormantasauspalvelimille

4.3 Tietokantapalvelimet

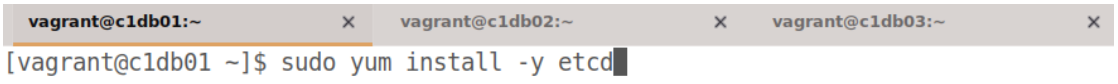
Tietokantapalvelimille täytyi lisätä repositorio PostgreSQL varten (ks. Kuvio 14).

Kuvio 14. PostgreSQL-repositorion lisääminen

Repositorion lisäämisen jälkeen voitiin PostgreSQL asentaa normaalisti yum-työkalua käyttäen (ks. Kuvio 15). Kyseisellä työkalulla voitiin myös asentaa Etcd-ohjelmisto (ks. Kuvio 16).



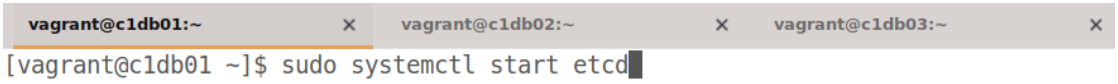
Kuvio 15. PostgreSQL-tietokannan asentaminen



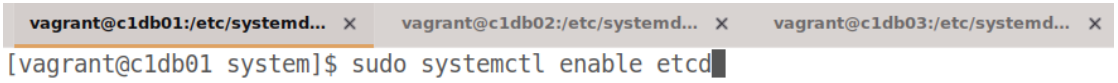
Kuvio 16. Etcd-tietokannan asentaminen

Tietokantapalvelimet käyttävät etcd:tä pitääkseen yllä tietoa klusterin tilasta. Etcd sisältää muun muassa jatkuvasti päivittyvän tiedon siitä, mikä tietokantapalvelimista on leader-roolissa.

Etcd-palvelun (service) konfiguraatioihin (ks. Liite 1) määriteltiin normaalit palveluasetukset sekä etcd-tietokannan konfiguraatiotiedoston sijainti. Etcd-tietokannan konfiguraatiot sisälsivät tiedon muun muassa kuunneltavista IP-osoitteista ja muista etcd-tietokantaklusterin sijainneista (ks. Liite 2). Konfiguraatioiden ollessa paikallaan, voitiin palvelu käynnistää (ks. Kuvio 17). Palvelun onnistuneen käynnistämisen jälkeen täytyi muistaa etcd myös asettaa käynnistymään automaattisesti (ks. Kuvio 18).



Kuvio 17. Etcd-palvelun käynnistäminen



Kuvio 18. Etcd-palvelun asettaminen käynnistymään automaattisesti

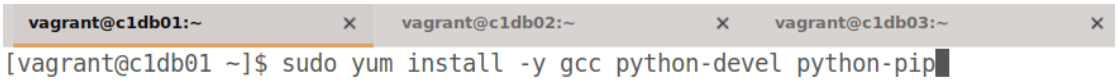
Patroni itsessään on Python-kielellä toteutettu orkestrointityökalu, joka ohjaa yhtä PosgreSQL-tietokantasolmua (node) klusterissa. Patroni tekee kyselyjä avain–arvo-paritietokantaklusterille (key-value storage) ja määrittää kyselyjen tuloksien perusteella PostgreSQL-tietokannan asetuksia dynaamisesti.

Kuormantasauksen suuntaan Patroni sisältää API-ratkaisun, josta HAProxy pystyy kyselemään tilatietoja. API:n palauttamista viesteistä HAProxy saa tiedon, mikä tietokantapalvelimista on leader-roolissa ja osaa ohjata liikenteen sille.

Patroni- ja PostgreSQL-instanssit ovat yleensä samalla solmulla, koska ne ovat suorassa vaikutuksessa toisiinsa. Avain–arvo-paritietokantaklusteri voi olla erillinen kokonaisuus, mutta työssä käsiteltävä ratkaisu oli suorituskyvyltään suhteellisen vaatimaton, joten myös ne sisältyvät samoihin klusterin solmuihin.

Solmuista vain yksi voi olla kerrallaan leader-roolissa. Leader-roolin solmu on aktiivinen solmu, joka käsittelee tulevat kyselyt ja vastaa niihin. Muut solmut ovat replikoivassa asemassa. Replikoivassa asemassa olevat solmut pitävät itsensä samassa tasossa leader-solmun kanssa, mutta eivät käsittele tai vastaa kyselyihin, eli toimivat passiivisessa tilassa. Kaikki nämä solmut klusterissa ovat silti saman arvoisia siinä mielessä, että mikäli leader-roolin solmu päätyisi vikatilaan, mikä tahansa toissijaisista solmuista voisi ottaa vastuulleen kyseisen roolin.

Patronin asentamista ja käyttämistä varten tarvitsi ensiksi asentaa sen riippuvuudet (dependencies). Kyseinen ohjelmisto asennettiin käyttäen pip-työkalua, jolloin tämä täytyi myös asentaa muiden riippuvuuksien mukana (ks. Kuvio 19).



Kuvio 19. Patronin riippuvuuksien asentaminen

Pip ja Setuptools täytyivät olla päivitetty, jotta Patronin asentaminen niitä käyttäen on mahdollista. Näiden päivittäminen tapahtui pip-työkalua käyttäen (ks. Kuvio 20).

Kuvio 20. Pip ja Setuptools päivittäminen

Pip ja Setuptools työkalujen päivittämisen jälkeen Patroni oli mahdollista asentaa. Patronista haluttiin versio 1.4.2, jolloin tämä oli määriteltävä asennuksen yhteydessä. Lisäksi Patroni tukee etcd ja Consul avain–arvo-paritietokantoja, jolloin täytyi valita, kumpaa näistä haluttiin käyttää. Toteutuksessa käytettiin etcd-tietokantaa, jolloin tämä myös määriteltiin asentamisen yhteydessä (ks. Kuvio 21).

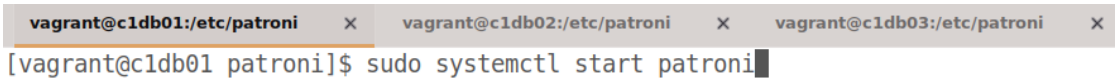
Kuvio 21. Patronin asentaminen

Patroni-palvelun konfiguraatioihin (ks. Liite 3) määriteltiin normaalit palveluasetukset sekä konfiguraatiotiedoston sijainti. Lisäksi näihin konfiguraatioihin määriteltiin käyttäjä ja ryhmä, jonka oikeuksilla palvelu käynnistetään.

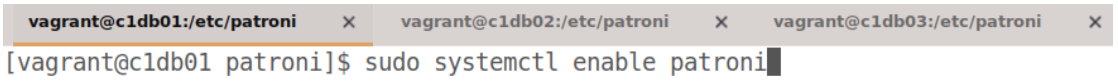
Patroni-ohjelmiston konfiguraatioille luotiin kansio, johon tietokantoihin liittyvät konfiguraatiot luotiin (ks. Kuvio 22).

Kuvio 22. Patroni konfiguraatiokansion luominen

Patronin konfiguraatiot sisälsivät tarvittavat tiedot etcd-klusteriin yhdistämistä varten, PostgreSQL-tietokantaan liittyvät tiedot sekä Patronin omia orkestrointiin liittyviä tietoja (ks. Liite 4). Nämä konfiguraatiot olivat samoja kaikilla tietokantapalvelimilla, lukuun ottamatta yksilöiviä nimiä ja IP-osoitteita. Konfiguraatioiden asettamisen jälkeen voitiin Patroni-prosessi käynnistää ja asettaa se myös käynnistymään jatkossa automaattisesti (ks. Kuvio 23, Kuvio 24).

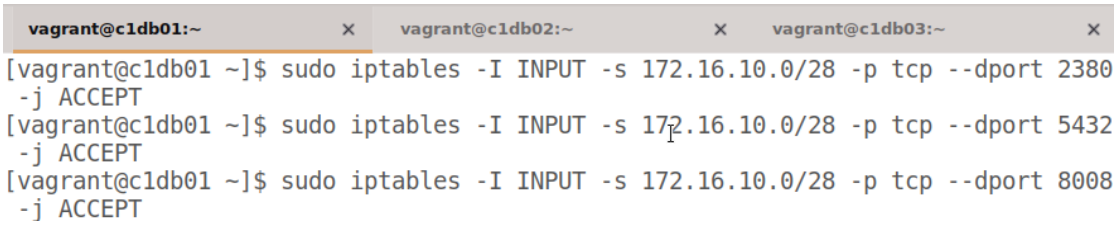


Kuvio 23. Patroni-prosessin käynnistäminen



Kuvio 24. Patronin asettaminen käynnistymään automaattisesti

Tietokantapalvelimille täytyi myös tehdä lisäyksiä palomuurisääntöihin, jotta palvelut voivat kommunikoida keskenään (ks. Kuvio 25).



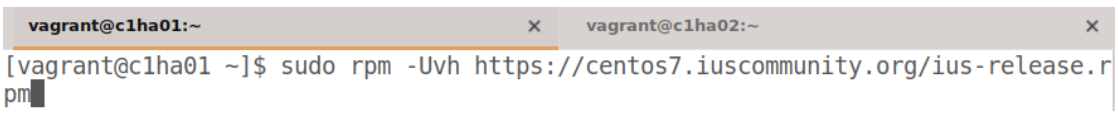
Kuvio 25. Tietokantapalvelimen palomuuriasetukset

Palomuurisääntöjen lisäämisen ja niiden toimivuuden todentamisen jälkeen täytyi muistaa asettaa ne pysyviksi (ks. Kuvio 26).

Kuvio 26. Tietokantapalvelimen palomuuriasetuksien tallentaminen

4.4 Kuormantasauspalvelimet

HAproxyn asentamista varten lisättiin kuormantasauspalvelimille IUS Community -repositorio (ks. Kuvio 27). Repositorion lisäämisen jälkeen voitiin HAProxy asentaa yum-työkalua käyttäen (ks. Kuvio 28).



Kuvio 27. IUS Community repositorion lisääminen

Kuvio 28. HAProxyn asentaminen

Korkean saatavuuden toteuttamiseksi asennettiin kuormantasauspalvelimille Keepalived-ohjelmisto. Keepalived on riippuvainen psmisc-paketista, jolloin se täytyi asentaa myös (ks. Kuvio 29).

Kuvio 29. Keepalived ja sen riippuvuuksien asentaminen

SELinuxia varten oli olemassa vakiokäytännöt HAProxy-ohjelmiston rajoittamiseksi. Näitä rajoituksia muutettiin niin, että niissä sallitaan HAProxyn ottaa yhteys kaikkiin TCP-portteihin (ks. Kuvio 30).

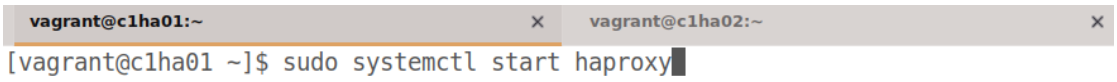
Kuvio 30. SELinux käytäntöjen muuttaminen HAProxya varten

HAProxy-konfiguraatioihin täytyi määritellä sijainti, johon kuormantasaukseen liittyvää statistiikkaa kerätään. Tämä sijainti täytyi myös olla olemassa, joten sitä varten lisättiin tiedosto (ks. Kuvio 31).

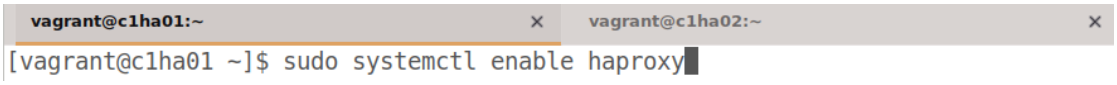
Kuvio 31. HAProxy statistiikkatiedoston luominen

HAProxyjen konfiguraatioihin määriteltiin kuunneltavaksi kaksi IP-osoitetta, joista toinen oli statistiikkasivua varten ja toinen virtuaalinen IP-osoite (ks. Liite 5). Virtuaalinen IP-osoite ohjaa tietokantaliikennettä.

Konfiguraatioiden asettamisen jälkeen käynnistettiin HAProxy-prosessi ja tämän jälkeen asetettiin se käynnistymään automaattisesti (ks. Kuvio 32, Kuvio 33).

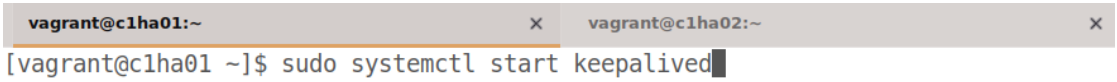


Kuvio 32. HAProxy-prosessin käynnistäminen

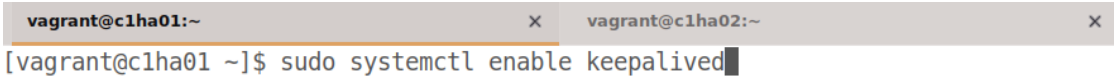


Kuvio 33. HAProxyn asettaminen käynnistymään automaattisesti

Keepalived-konfiguraatioihin määriteltiin muun muassa seurannassa pidettävät verkkorajapinnat ja tieto alkuperäisestä tilasta sekä tärkeysjärjestyksestä (ks. Liite 6). Konfiguraatioiden asettamisen jälkeen käynnistettiin Keepalived-prosessi ja asetettiin se käynnistymään jatkossa automaattisesti (ks. Kuvio 34, Kuvio 35).



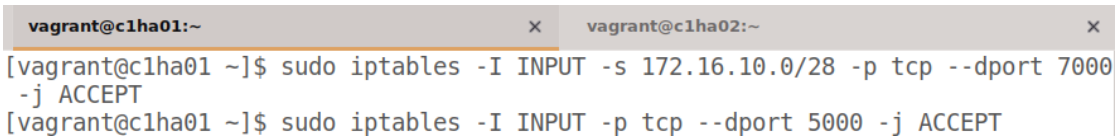
Kuvio 34. Keepalived-prosessin käynnistäminen



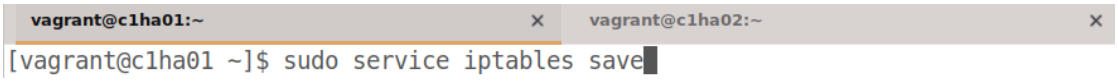
Kuvio 35. Keepalived asettaminen käynnistymään automaattisesti

Kuormantasauspalvelimien palomuurin täytyy sallia liikenne tietokantayhteyksiä varten, sekä statistiikkasivun katselua varten. Nämä palomuurisäännöt lisättiin käyttäen

iptables-palomuuriohjelmistoa (ks. Kuvio 36). Palomuurisääntöjen asettamisen jälkeen täytyi myös muistaa tehdä näistä pysyviä (permanent), jotta ne eivät pyyhkiydy pois palvelimen uudelleen käynnistyessä (ks. Kuvio 37).



Kuvio 36. Kuormantasauspalvelimen palomuurisäännöt



Kuvio 37. Kuormantasauspalvelimen palomuuriasetusten tallentaminen

5 Tulokset

5.1 Tietokantapalvelinten vikasietoisuus

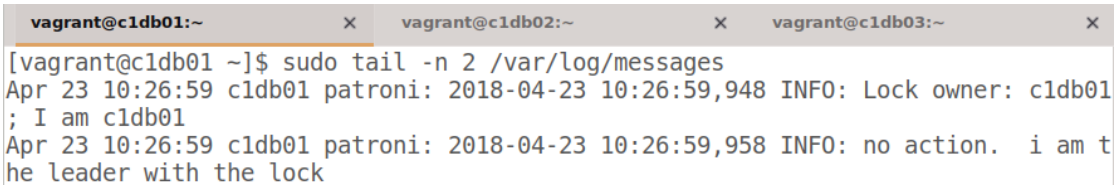
Toimintaympäristöksi työhön oli rajattu kehitysympäristö, jolloin testaamissa otettiin kantaa vain vikasietoisuusmekanismin toimintaan. Viiveiden ja tarkempien analyysien tekeminen olisi ollut tässä ympäristössä käytännössä turhaa, koska kyseinen ympäristö ei vastannut tuotantoympäristöä.

Lähtötilanteessa klusteri oli täysin toimintakykyinen, c1db01 oli leader-roolissa ja muut seurasivat sitä (ks. Kuvio 38).

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
Frontend				0	1	-	1	1	2 000	1			0	0	0	0	0					OPEN
c1db01	0	0	-	0	1		1	1	100	1	1	5s	0	0		0		0	0	0	0	32s UP
c1db02	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	31s DOWN
c1db03	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	30s DOWN

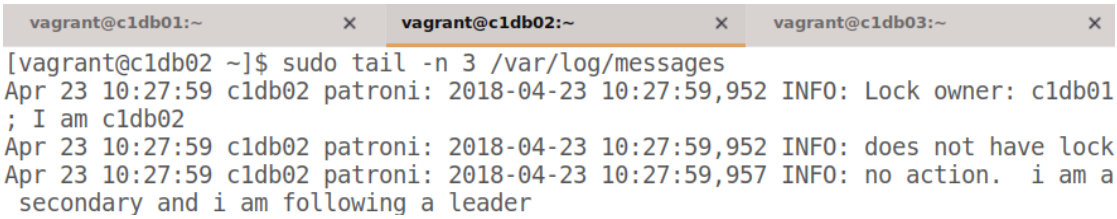
Kuvio 38. Kuormantasaus tietokantapalvelimien testauksen alussa

Etcd sisältää lukkoarvon, joka pitää sisällään tiedon leader-roolissa olevasta palvelimesta. Lähtötilanteessa lukon omisti c1db01-palvelin (ks. Kuvio 39).



Kuvio 39. c1db01 leader-roolissa

Muut palvelimet näkivät saman tiedon tehdessään kyselyjä etcd-tietokantaan ja näin ollen olivat tilassa, jossa seurasivat leaderia (ks. Kuvio 40).



Kuvio 40. c1db02 secondary-roolissa

Testauksessa sammutettiin kokonaan c1db01-palvelin virhetilanteen simuloimiseksi. Lopputulos kyseisellä mekanismilla olisi sama, mikäli virhetilanne syntyisi verkkoyhteyden katkeamisesta tai pelkän etcd- tai Patroni-palvelun (service) kaatumisesta.

Palvelimien sammutettua se ei ollut enää päivittämässä leader-lukon arvoa etcd-tietokantaan, jolloin se vapautui käytettäväksi toiselle palvelimelle. Lukon saa asetettua vain ensimmäinen, joka tässä tapauksessa oli c1db02-palvelin. c1db02 saatuaan asetettua lukon ylensi (promote) itsensä uudeksi leaderiksi (ks. Kuvio 41). Ylennyksen jälkeen c1db02 toimi täysin samalla tavalla kuin lähtötilanteessa c1db01 (ks. Kuvio 42).

```
Apr 23 10:30:42 c1db02 patroni: 2018-04-23 10:30:42,423 INFO: promoted self to l
eader by acquiring session lock
Apr 23 10:30:42 c1db02 patroni: server promoting
Apr 23 10:30:42 c1db02 patroni: 2018-04-23 10:30:42,430 INFO: cleared rewind sta
te after becoming the leader
```

Kuvio 41. c1db02 automaattinen ylennys

```
autiosam@ ×   vagrant@c1db02:~ ×   vagrant@c1db03:~ ×
[vagrant@c1db02 ~]$ sudo tail -n 2 /var/log/messages
Apr 23 10:32:53 c1db02 patroni: 2018-04-23 10:32:53,460 INFO: Lock owner: c1db02
; I am c1db02
Apr 23 10:32:53 c1db02 patroni: 2018-04-23 10:32:53,468 INFO: no action.  i am t
he leader with the lock
```

Kuvio 42. c1db02 leader-roolissa

Kuormantasauspalvelimet huomasivat, ettei ensisijainen tietokantapalvelin ollut enää toimintakykyinen ja lopetti liikenteen ohjaamisen sille (ks. Kuvio 43).

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
Frontend				0	1	-	1	1	2 000	1			0	0	0	0	0					OPEN
c1db01	0	0	-	0	1		1	1	100	1	1	2m3s	0	0		0		0	0	0	0	2m30s UP 2/3 ↓
c1db02	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	2m29s DOWN
c1db03	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	2m28s DOWN

Kuvio 43. Liikenteen ohjaamisen lopettaminen c1db01-solmulle

Kun Patroni oli ylentänyt taustalla uuden ensisijaisen tietokantapalvelimen, niin huomasi kuormantasaus tämän myös ja alkoi uudelleen ohjaamaan liikennettä tälle palvelimelle (ks. Kuvio 44).

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
Frontend				0	1	-	0	1	2 000	1			133	452	0	0	0					OPEN
c1db01	0	0	-	0	1		0	1	100	1	1	2m28s	133	452		0		0	0	0	0	19s DOWN
c1db02	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	2m54s DOWN 1/2 ↑
c1db03	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	2m53s DOWN

Kuvio 44. Liikenteen ohjaamisen aloittaminen c1db02-solmulle

Testauksessa käytetyillä asetuksilla järjestelmä palautui toimintakykyiseksi noin kahdessakymmenessä sekunnissa (ks. Kuvio 45). Tämän jälkeen tietokantayhteydet yhdistettiin uudestaan toimintakykyiselle palvelimelle (ks. Kuvio 46).

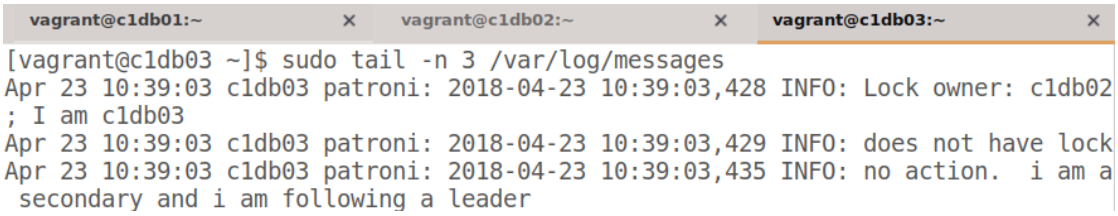
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
Frontend				0	1	-	0	1	2 000	1			133	452	0	0	0					OPEN
c1db01	0	0	-	0	1		0	1	100	1	1	2m32s	133	452		0		0	0	0	0	23s DOWN
c1db02	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	1s UP
c1db03	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	2m57s DOWN

Kuvio 45. Tietokanta palautunut toimintakykyiseksi

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
Frontend				0	1	-	1	1	2 000	2			133	452	0	0	0					OPEN
c1db01	0	0	-	0	1		0	1	100	1	1	7m37s	133	452		0		0	0	0	0	5m28s DOWN
c1db02	0	0	-	0	1		1	1	100	1	1	1m17s	0	0		0		0	0	0	0	5m6s UP
c1db03	0	0	-	0	0		0	0	100	0	0	?	0	0		0		0	0	0	0	8m2s DOWN

Kuvio 46. Tietokantayhteyksien uudelleen luominen

Muut tietokantapalvelimet näkivät kyselyillään c1db02-palvelimen olevan nyt leader-roolissa ja vaihtoivat näin ollen seuraamaan sitä (ks. Kuvio 47).



Kuvio 47. c1db03 secondary-roolissa

Myös alas ajettu tietokantapalvelin palatessaan takaisin näki lukon uuden omistajan ja alkoi seuraamaa sitä (ks. Kuvio 48).

```
vagrant@c1db01:~ ×  vagrant@c1db02:~ ×  vagrant@c1db03:~ ×
[vagrant@c1db01 ~]$ sudo tail -n 3 /var/log/messages
Apr 23 10:40:03 c1db01 patroni: 2018-04-23 10:40:03,466 INFO: Lock owner: c1db02
; I am c1db01
Apr 23 10:40:03 c1db01 patroni: 2018-04-23 10:40:03,466 INFO: does not have lock
Apr 23 10:40:03 c1db01 patroni: 2018-04-23 10:40:03,474 INFO: no action.  i am a
 secondary and i am following a leader
```

Kuvio 48. c1db01 secondary-roolissa

5.2 Kuormantasauspalvelinten vikasietoisuus

Kuormantasauspalvelinten testauksessa alkutilanteessa c1ha01-palvelin oli ensisijaisena ja näin ollen sillä oli virtuaalinen IP-osoite (ks. Kuvio 49).

```
vagrant@c1ha01:~ ×  vagrant@c1ha02:~ ×
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:25:51:d9 brd ff:ff:ff:ff:ff:ff
    inet 10.16.1.1/21 brd 10.16.7.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet 10.16.1.10/21 brd 10.16.7.255 scope global secondary eth1
       valid lft forever preferred lft forever
```

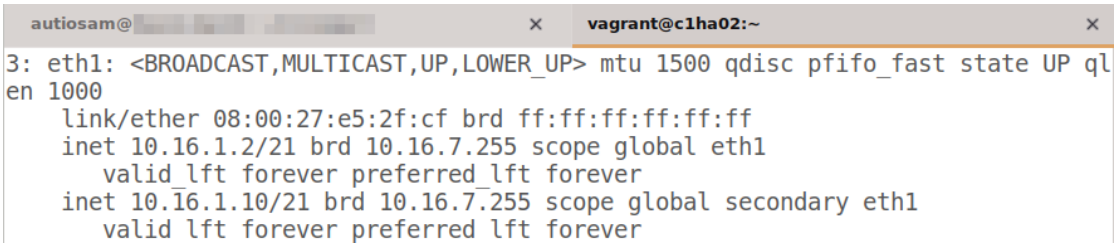
Kuvio 49. c1ha01 julkinen rajapinta kuormantasauksen testauksen alussa

Testissä sammutettiin kokonaan c1ha01-palvelin, jonka c1ha02 huomasi ja reagoi tähän ylentämällä itsensä master-rooliin. Tämän lisäksi c1ha02 asetti virtuaalisen IP-osoitteen itselleen ja alkoi mainostaa sitä verkkoon (ks. Kuvio 50).

```
autiosam@ ×  vagrant@c1ha02:~ ×
Apr 24 05:41:32 c1ha02 Keepalived_vrrp[3675]: VRRP_Instance(VI_localphoenix) Tra
nsition to MASTER STATE
Apr 24 05:41:33 c1ha02 Keepalived_vrrp[3675]: VRRP_Instance(VI_localphoenix) Ent
ering MASTER STATE
Apr 24 05:41:33 c1ha02 Keepalived_vrrp[3675]: VRRP_Instance(VI_localphoenix) set
ting protocol VIPs.
Apr 24 05:41:33 c1ha02 Keepalived_vrrp[3675]: Sending gratuitous ARP on eth1 for
 10.16.1.10
```

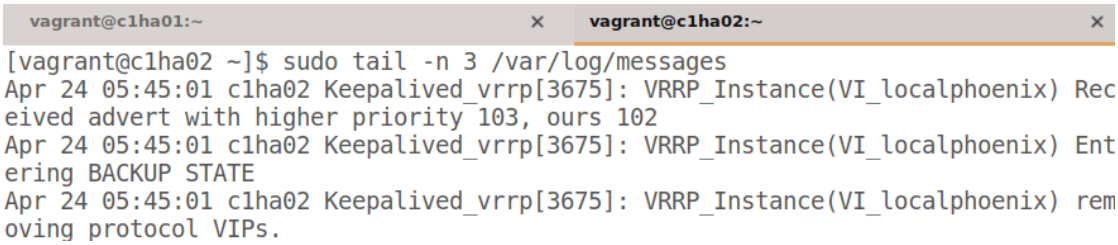
Kuvio 50. c1ha02 automaattinen ylennys

Virtuaalinen IP-osoite oli tämän jälkeen näkyvissä tarkastellessa c1ha02-palvelimen verkkorajapintoja (ks. Kuvio 51).



Kuvio 51. c1ha02 julkinen rajapinta ylennyksen jälkeen

Sammutetun palvelimen tullessa takaisin toimintakykyiseen tilaan c1ha02 huomasi tämän ja palautui takaisin backup-rooliin sekä poisti virtuaalisen IP-osoitteen itsel-tään (ks. Kuvio 52).



Kuvio 52. c1ha02 palautuminen takaisin backup-tilaan

6 Pohdinta

6.1 Tulosten tarkastelu

Tavoitteena opinnäytetyölle oli kokeilla Patronia käytännön tasolla Phoenix-järjestel-män kehittämistä varten, saaden siitä kokemusta ja tietoa. Kokeilu tapahtui pystyttä-mällä tietokantaklusteri käyttäen Patronia ja HAProxyja. Patronia käytettiin kluste-rissa tietokantojen orkestrointiin. HAProxyt toimivat kuormantasauksessa ja ne konfi-guroitiin myös vikasietoisiksi käyttäen Keepalived-työkalua.

Tietokantaklusterin vikasietoisuusmekanismi todennettiin toimivaksi. Toteutettu ratkaisu osoittautui potentiaaliseksi vaihtoehdoksi toimeksiantajan ongelmaan tietokantojen korkean saatavuuden toteuttamiselle.

Laajempaa testausta ei lähdetty toteuttamaan, sillä siitä saatu tieto ei olisi ollut verrattavissa tuotantoympäristön olosuhteisiin. Ratkaisun luotettavuuden varmistamiseksi täytyisi siis työssä toteutettua tietokantaklusteria testata tuotantoympäristöön verrattavassa ympäristössä.

Patronin käyttöönotossa täytyi turvautua osittain omiin selvityksiin ja kokeilujen pohjalta tehtyihin havaintoihin, sillä Patronin oma dokumentaatio ei kata kaikkia ominaisuuksia ja on myös yleisesti hyvin suppea. Tällöin työ pohjautuu osittain myös omiin kokemuksen kautta tulleisiin näkemyksiin, jolloin osa työn teknisistä yksityiskohdista saattaa päteä vain tässä toteutuksessa.

Opinnäytetyön tulokset ovat kuitenkin käyttökelpoisia toimeksiantajalle ja tuloksia voi hyödyntää kartoittaessa vaihtoehtoja tietokantaratkaisulle. Tuloksista on myös mahdollista saada ohjeistusta Patronilla toteutetun tietokantaklusterin pystyttämisessä.

6.2 Kehitysideat

6.2.1 Testaus

Työssä keskityttiin vain kehitysympäristöön, eikä tämän takia pystytty tekemään paljoa tuotantoympäristöön verrattavaa testausta. Mekanismi virhesietoisuuden takaamiseksi todennettiin toimivaksi, mutta luotettavuus todellisessa käytössä selviäisi vain laajemmilla testauksilla.

Paremman luotettavuuden takaamiseksi tulisi tehdä testausta ympäristössä, jossa verkkotopologia olisi verrattavissa tuotantoympäristöön. Testausta täytyisi tehdä tuotanto-olosuhteisiin odotettavalla ja arvioidulla rasituksella, jolloin voitaisiin saada realistisempaa tietoa viiveistä ja datan eheydestä.

6.2.2 Tietoturva

Työssä käsiteltiin tietoturvaa lähinnä vain palomuuriasetuksien kohdalla. Tuotantoon soveltuvaksi ratkaisuksi täytyisi tietoturvaa tarkastella laajemmin. Tietoturvaa tulisi tarkastella kokonaisuuden kaikilla tasoilla eli verkko-, palvelin- ja sovellustasoilla.

Tietoturvan koventamiseksi olisi hyvä lisätä muun muassa bastion-palvelin (bastion host), ja VPN-putkitukset joilla voitaisiin erottaa ja suojata ympäristön hallintaliikenne ja sisäinen liikenne varmemmin julkisesta internetistä.

6.2.3 Kontitus

Aiheeseen tutustuessa vastaan tuli myös keskustelua Patronin kontituksesta. Palveluiden kontitusta on toimeksiantaja jo aloittanutkin muiden projektien yhteydessä. Patronin kontittaminen näin ollen voisi olla luonteva suunta jatkokehitykselle tulevaisuudessa.

Zalandolla on myös aktiivinen projekti tätä varten nimeltään Spilo. Spilo on docker-levykuva (docker image), joka sisältää Patronin ja postgreSQL-tietokantamoottorin. Esimerkiksi Spiloa käyttämällä voisi ratkaisua lähteä viemään kontitettuun ympäristöön.

Lähteet

Anderson, M. 2017. Artikkeli Digital Ocean yhteisösivustolla. Viitattu 17.4.2018. https://www.digitalocean.com/community/tutorials/what-is-load-balancing

Anicas, M. 2014. Artikkeli Digital Ocean yhteisösivustolla. Viitattu 17.4.2018. https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

CentOS Product Specifications. N.d. CentOS projektin kotisivut. Viitattu 10.4.2018. https://wiki.centos.org/About/Product

Debian Releases. N.d. Debian projektin kotisivut. Viitattu 10.4.2018. https://wiki.debian.org/DebianReleases

FirewallD. 2016. Verkkojulkaisu linux wikisivustolla. Viitattu 22.3.2018. https://www.linux.fi/wiki/FirewallD

Governor. N.d. Governor projektin versionhallinta. Viitattu 26.3.2018. https://github.com/compose/governor

HAProxy Description. N.d. HAProxy tuotesivusto. Viitattu 17.4.2018. http://www.haproxy.org/#desc

Heidi, E. 2016. What is High Availability?. Viitattu 11.4.2018. https://www.digitalocean.com/community/tutorials/what-is-high-availability

Introduction to Consul. N.d. Consul tuotesivusto. Viitattu 17.4.2018. https://www.consul.io/intro/index.html

Iptables. 2016. Verkkojulkaisu linux wikisivustolla. Viitattu 22.3.2018. https://www.linux.fi/wiki/Iptables

Log-Shipping Standby Servers. N.d. PostgreSQL dokumentaatio. Viitattu 31.3.2018. https://www.postgresql.org/docs/10/static/warm-standby.html

Netfilter. 2015. Verkkojulkaisu linux wikisivustolla. Viitattu 22.3.2018. https://www.linux.fi/wiki/Netfilter

Palomuuri. 2015. Verkkojulkaisu linux wikisivustolla. Viitattu 22.3.2018. https://www.linux.fi/wiki/Palomuuri

Patroni. N.d. Patroni projektin versionhallinta. Viitattu 26.3.2018. https://github.com/zalando/patroni

Patroni introduction. N.d. Patroni projektin wikisivusto. Viitattu 26.3.2018. https://patroni.readthedocs.io/en/latest/index.html

Patroni replication modes. N.d. Patroni projektin wikisivusto. Viitattu 28.3.2018. https://patroni.readthedocs.io/en/latest/replication_modes.html

Streaming Replication. N.d. PostgreSQL dokumentaatio. Viitattu 31.3.2018. https://www.postgresql.org/docs/10/static/warm-standby.html

What is a Key-Value Store?. N.d. Aerospike tuotesivusto. Viitattu 11.4.2018. https://www.aerospike.com/what-is-a-key-value-store/

What is Keepalived?. N.d. Keepalived tuotesivusto. Viitattu 17.4.2018. http://www.keepalived.org/

What is Raft?. N.d. Raft algoritmin kotisivut. Viitattu 21.4.2018. https://raft.github.io/

Liitteet

Liite 1. Etcd service konfiguraatiot

```
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
User=etcd
Type=notify
Environment=ETCD_DATA_DIR=/var/lib/etcd
Environment=ETCD_NAME=%m
ExecStart=/usr/bin/etcd --config-file /etc/etcd/etcd.yml
Restart=on-failure
RestartSec=10s
WorkingDirectory=/var/lib/etcd/
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
```

Liite 2. Etcd konfiguraatiot

```
# Human-readable name for this member.
name: etcd.c1db01

# Path to the data directory.
data-dir: /var/lib/etcd/patroni.etcd

# Path to the dedicated wal directory.
wal-dir:

# Number of committed transactions to trigger a snapshot to disk.
snapshot-count: 10000

# Time (in milliseconds) of a heartbeat interval.
heartbeat-interval: 100

# Time (in milliseconds) for an election to timeout.
election-timeout: 1000

# Raise alarms when backend size exceeds the given quota. 0 means use the
# default quota.
quota-backend-bytes: 0

# List of comma separated URLs to listen on for peer traffic.
listen-peer-urls: http://172.16.10.11:2380

# List of comma separated URLs to listen on for client traffic.
listen-client-urls: http://172.16.10.11:2379,http://127.0.0.1:2379

# Maximum number of snapshot files to retain (0 is unlimited).
max-snapshots: 5

# Maximum number of wal files to retain (0 is unlimited).
max-wals: 5

# Comma-separated white list of origins for CORS (cross-origin resource sharing).
cors:

# List of this member's peer URLs to advertise to the rest of the cluster.
# The URLs needed to be a comma-separated list.
initial-advertise-peer-urls: http://172.16.10.11:2380

# List of this member's client URLs to advertise to the public.
# The URLs needed to be a comma-separated list.
advertise-client-urls: http://172.16.10.11:2379
```

```
# Discovery URL used to bootstrap the cluster.
discovery:

# Valid values include 'exit', 'proxy'
discovery-fallback: 'proxy'

# HTTP proxy to use for traffic to discovery service.
discovery-proxy:

# DNS domain used to bootstrap initial cluster.
discovery-srv:

# Initial cluster configuration for bootstrapping.

initial-cluster:
etcd.c1db01=http://172.16.10.11:2380,etcd.c1db02=http://172.16.10.12:2380,etcd.
c1db03=http://172.16.10.13:2380

# Initial cluster token for the etcd cluster during bootstrap.
initial-cluster-token: 'etcd-cluster-token-vboxcluster'

# Initial cluster state ('new' or 'existing').
initial-cluster-state: 'new'

# Reject reconfiguration requests that would cause quorum loss.
strict-reconfig-check: false

# Accept etcd V2 client requests
enable-v2: true

# Enable runtime profiling data via HTTP server
enable-pprof: true

# Valid values include 'on', 'readonly', 'off'
proxy: 'off'

# Time (in milliseconds) an endpoint will be held in a failed state.
proxy-failure-wait: 5000

# Time (in milliseconds) of the endpoints refresh interval.
proxy-refresh-interval: 30000

# Time (in milliseconds) for a dial to timeout.
proxy-dial-timeout: 1000

# Time (in milliseconds) for a write to timeout.
proxy-write-timeout: 5000

# Time (in milliseconds) for a read to timeout.
```

```
proxy-read-timeout: 0

client-transport-security:
  # DEPRECATED: Path to the client server TLS CA file.
  ca-file:

  # Path to the client server TLS cert file.
  cert-file:

  # Path to the client server TLS key file.
  key-file:

  # Enable client cert authentication.
  client-cert-auth: false

  # Path to the client server TLS trusted CA cert file.
  trusted-ca-file:

  # Client TLS using generated certificates
  auto-tls: false

peer-transport-security:
  # DEPRECATED: Path to the peer server TLS CA file.
  ca-file:

  # Path to the peer server TLS cert file.
  cert-file:

  # Path to the peer server TLS key file.
  key-file:

  # Enable peer client cert authentication.
  peer-client-cert-auth: false

  # Path to the peer server TLS trusted CA cert file.
  trusted-ca-file:

  # Peer TLS using generated certificates.
  auto-tls: false

# Enable debug-level logging for etcd.
debug: false

# Specify a particular log level for each etcd package (eg: 'etcd-
main=CRITICAL,etcdserver=DEBUG'.
log-package-levels:

# Specify 'stdout' or 'stderr' to skip journald logging even when running under sys-
temd.
```

```
log-output: default

# Force to create a new one member cluster.
force-new-cluster: false
```

Liite 3. Patroni service konfiguraatiot

```
[Unit]
Description=Runners to orchestrate a high-availability PostgreSQL
After=syslog.target network.target

[Service]
Type=simple

User=postgres
Group=postgres

# Where to send early-startup messages from the server
# This is normally controlled by the global default set by systemd
# StandardOutput=syslog

ExecStart=/bin/patroni /etc/patroni/postgres.yml

# only kill the patroni process, not it's children, so it will gracefully stop postgres
KillMode=process

# Give a reasonable amount of time for the server to start up/shut down
TimeoutSec=30

# Do not restart the service if it crashes, we want to manually inspect database on
failure
Restart=no

[Install]
WantedBy=multi-user.target
```

Liite 4. Patroni konfiguraatiot

```
scope: localphoenix
namespace: localphoenix/service/
name: c1db01

restapi:
 listen: 172.16.10.11:8008
 connect_address: 172.16.10.11:8008

etcd:
 host: 127.0.0.1:2379

bootstrap:
 # this section will be written into Etcd:/<namespace>/<scope>/config after initializ-
ing new cluster
 # and all other cluster members will use it as a `global configuration`
 dcs:
  ttl: 30
  loop_wait: 10
  retry_timeout: 10
  maximum_lag_on_failover: 1048576
  postgresql:
   use_pg_rewind: true
   parameters:

 # some desired options for 'initdb'
 initdb:  # Note: It needs to be a list (some options need values, others are switches)
 - encoding: UTF8
 - data-checksums

 pg_hba:  # Add following lines to pg_hba.conf after running 'initdb'
 - host replication replicator 172.16.10.0/28 md5
 - host all all 0.0.0.0/0 md5

postgresql:
 listen: 172.16.10.11:5432
 connect_address: 172.16.10.11:5432
 data_dir: /var/lib/pgsql/10/data/localphoenix
 bin_dir: /usr/pgsql-10/bin
 pgpass: /tmp/pgpass
 authentication:
  replication:
   username: replicator
   password: <password>
  superuser:
   username: postgres
   password: <password>
```

```
  parameters:
    unix_socket_directories: '.'

tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
```

Liite 5. HAProxy konfiguraatiot

```
global
   maxconn 100

defaults
   log global
   mode tcp
   retries 2
   timeout client 30m
   timeout connect 4s
   timeout server 30m
   timeout check 5s

listen stats
   mode http
   bind 10.16.1.1:7000
   stats enable
   stats uri /

listen localphoenix_backend
   bind 10.16.1.10:5000
   option httpchk
   http-check expect status 200
   default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
   server c1db01 172.16.10.11:5432 maxconn 100 check port 8008
   server c1db02 172.16.10.12:5432 maxconn 100 check port 8008
   server c1db03 172.16.10.13:5432 maxconn 100 check port 8008
```

Liite 6. Keepalived konfiguraatiot

```
vrrp_script chk_haproxy {
  script "killall -0 haproxy" # check the haproxy process
  interval 2 # every 2 seconds
  weight 2 # add 2 points if OK
}

vrrp_instance VI_localphoenix {
  interface eth2 # interface to monitor
  unicast_src_ip 172.16.10.1
  unicast_peer {
   172.16.10.2
  }
  state MASTER
  priority 101
  virtual_router_id 51
  virtual_ipaddress {
    10.16.1.10/21 brd 10.16.7.255 dev eth1
  }
  track_script {
    chk_haproxy
  }
}
```