jamk.fi

Pilvipalvelut

Contriboard Labra2

Ryhmä 3
Alexander Andreev
Rami Ojala
Ilari Rajala
Asko Ropponen

Laboratorioraportti
TTTW0430 - Pilvipalvelut, Jarmo Viinikanoja
18.11.2019
Tieto- ja viestintätekniikan tutkinto-ohjelma

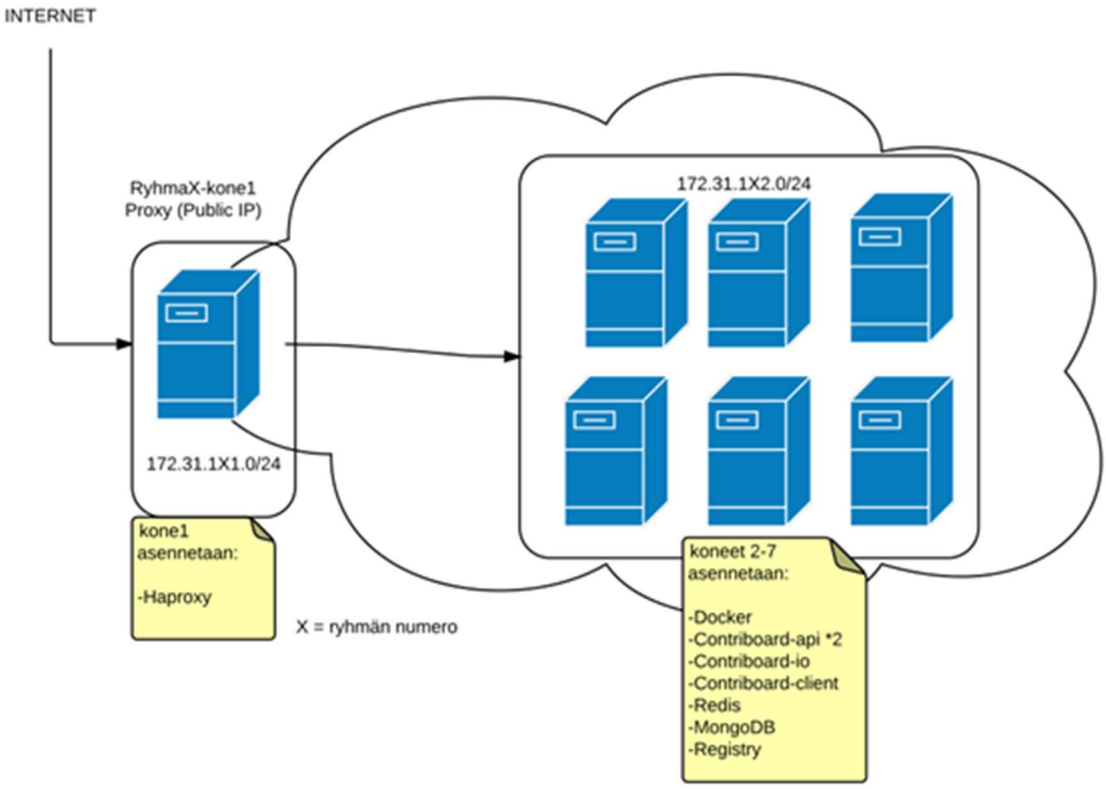
Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Sisältö

1 Johdanto

Pilvipalvelut laboratorio kaksi työnä tehtiin Contriboard palvelun pystytys Amazon Web Services pilvipalveluun siten, että sovelluksen komponentit jaettiin eri koneille. Laboratoriotyössä käytettiin laboratoriotyön ohjeistusta ja opettaja Viinikanojan asiantuntevia vinkkejä. Työssä käytettiin myös Elastic Load Balancing AWS järjestelmää.

2 Tehtävänanto



Kuvio 1 Tehtävänannon mukainen ympäristö.

- Asenna yhdelle koneelle Docker Registry ja laita Contriboardin komponentit sille niin, että kun niitä asennetaan Dockerilla imaget haetaan omasta Docker registrystä.

- Asenna Contriboard toimimaan neljällä koneella niin, että jokaisella koneella on yksi komponentti paitsi Redis ja Mongo, jotka ovat samalla koneella.

- Lisää kaksi API-konetta, toinen samalle koneelle, kuin missä on jo API ja toinen eri koneelle ja jaa liikenne näille API-koneille käyttäen ELB:tä.

- Muuta HAProxyn asetuksia niin, että kuorma tasataan API-koneiden kesken ja todenna että kuormantasaus toimii. Tasaa liikenne myös sisäverkossa API-koneille.

- Tietoturva: Muuttakaa security grouppien asetuksia niin, että vain tarvittava liikenne sallitaan vain teidän ryhmänne subnetistä + proxylle tietenkin täytyy sallia jotain kaikkialta. Sulkekaa myös ylimääräiset portit.

- Lisätkää ryhmänne avain HAProxy koneelle niin, että saatte yhteyden muihin koneisiin antamalla käskyn ssh *koneen-ip tai esim osan nimi (api, io, mongo jne.)*.

- Tehkää kuormantasaus myös IO-komponentille, eli asentakaa jollekin koneelle toinen IO-kontti (vaikka samalle missä toinenkin on) ja tasatkaa kuorma IO-konttien kesken. Kiinnittäkää huomiota siihen, että IO:lle jutellaan käyttäen websockettia, joka on tilallinen yhteys.

- Todentakaa että, jos resetoitte HAProxyn, sen jälkeen, kun yksi käyttäjä muodostaa yhteyden palveluun, pitäisi kaikkien pyyntöjen päätyä yhdelle IO-koneelle, vaikka backendissä näkyy useampi IO toiminnassa. Jos tulee toiselta koneelta yhteys, pitäisi sitten sen päätyä toiselle koneelle.

- Suunnitelkaa kuinka nykyisellä kokoonpanolla voisimme varautua siihen, että jos Mongo kone hajoaa, tietokanta olisi muuallakin tallella.

3 Teoreettiset lähtökohdat

3.1 AWS

Contriboard palvelu pystytetään Amazon Web Services palveluun, johon opettaja Viinikanoja oli jo valmiiksi luonut tunnukset ja avaimet. Työssä käytetään Amazon EC2 (Elastic Compute Cloud), joka tarjoaa muutettavissa olevaa laskenta kapasiteettiä pilvipalvelussa. EC2 koneissa käytetään korkea taajuisia Intelin Xeon prosessoreita ja niissä on tasapainotettu laskenta muisti ja verkkoresurssit. Työssä käytettävissä t2.micro koneissa on 3,3GHz skaalautuva prosessori teho, muistia yksi gigatavu ja tallennus Amazon EBS. (Elastic Block Store, Amazon EC2 Instance IP Addressing).

Tietoturvallisuuden vuoksi vain kahdelle koneelle (ulkoverkko) on annettu mahdollisuus päästä suoraan internettiin julkisilla IP-osoitteilla. Näille koneille asennetaan HAProxy, joiden kautta muut private IP koneet (sisäverkko) keskustelevat toistensa kanssa ja ovat yhteydessä internettiin.

SSH yhteydellä ulkoverkon koneeseen saa ryhmälle valmiiksi määritetyllä PEM avaimella käyttäen PuTTY terminaali emulaattoria. Ulkokoneen kautta saadaan taas SSH yhteys muodostettua sisäverkon koneisiin käyttäen niiden private IP-osoitteita.

3.2 Docker

Contriboard asennetaan käyttäen Docker kontteja. Docker on joukko PaaS (Platform as a Service) tuotteita ja ne käyttävät käyttöjärjestelmä tason virtualisointia ohjelmiston ja siihen tarvittavien kirjastojen ja riippuvuuksien toimittamiseen, joita sanotaan konteiksi. Niillä pystytään helposti jakamaan toiselle koneelle identtinen ohjelmisto ympäristöineen. (What is a Container).

3.3 HAProxy

Kuorman tasaamisessa laboratorio työssä käytetään HAProxya. HAProxy on TCP / http kuormantasaus- ja välityspalvelin, jonka avulla verkkopalvelin jakaa saapuvat pyynnöt useille päätepisteille. Tämä on hyödyllinen tapauksissa, joissa liian monet samanaikaiset yhteydet ylikyllästävät yhden palvelimen ominaisuuksia. Yhdelle palvelimelle yhteyden muodostamisen sijaan, joka käsittelee kaikkia pyyntöjä, client muodostaa yhteyden HAProxy-instanssiin, joka käyttää käänteistä välityspalvelinta lähettämään pyynnön yhdelle käytettävissä olevista päätepisteistä, kuormantasaus algoritmin perusteella (How to Use HAProxy for Load Balancing.)

4 Työn kulku

4.1 AWS

Toisen laboratoriotyön aikana, meillä on jo valmiina ensimmäisen laboratoriotyön aikana luotu ympäristö. Ainoa tehty muutos, hankittu Elastic IP, joten julkinen IP, pysyy samana koneiden uudelleenkäynnistyksen jälkeen.

4.2 Contriboard

4.2.1 Oman Docker Registryn luominen

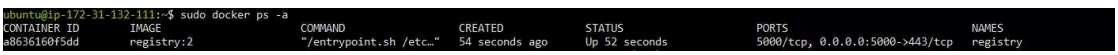
Jotta Docker registryä voitaisiin käyttää, sillä pitäisi olla tunnetun tahon myöntämä varmenne (en. Certificate). Testitapauksessa sitä voi käyttää myös ilman kyseistä varmennetta, joko pelkällä http-yhteydellä, tai itse allekirjoitetulla varmenteella (Test an insecure registry.)

Loimme siis oman varmenteen (Kuvio 2). Varmenteen luonnissa millään muulla ei tässä tapauksessa ole väliä, kuin että Common Name-kenttään (FQDN) laitetaan jotain järkevää. Tätä käytetään, jotta varmenne ei ole sidottu IP-osoitteeseen, joka voi muuttua.

Kuvio 2 Varmenteen luonti.

Sitten käynnistimme rekisterin (Kuviot 3 ja 4). Registryä käynnistäessä annetaan ympäristömuuttujina äsken luodun varmenteen sijainti.

Kuvio 3 Docker Registryn käynnistys.



Kuvio 4 Docker Registry käynnissä.

Seuraavaksi lisäämme rekisteriin haluamamme paketit (Kuviot 5 ja 6).

```
ubuntu@ip-172-31-132-111:~$ sudo docker image ls
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
redis                     latest              01a52b3b5cd1        3 weeks ago         98.2MB
registry                  2                   f32a97de94e1        7 months ago        25.8MB
mongo                     3.2                 fb885d89ea5c        11 months ago       300MB
n4sjamk/teamboard-io      latest              cfa2b17ffd55        3 years ago         327MB
n4sjamk/teamboard-api     latest              3be5ab65756a        3 years ago         489MB
n4sjamk/teamboard-client  latest              f4762f2de240        3 years ago         844MB
ubuntu@ip-172-31-132-111:~$ sudo docker image tag redis localhost:5000/redis
ubuntu@ip-172-31-132-111:~$ sudo docker push localhost:5000/redis
The push refers to repository [localhost:5000/redis]
6f3fa587ec88: Pushed
58ffe58b3e94: Pushed
c7fcc133516e: Pushed
a0e3cc85530d: Pushed
96550adba4f9: Pushed
2db44bce66cd: Pushed
latest: digest: sha256:4332745b2116859f4b77ff9dbc57ad9f5327dfe5e65807e064bdd2fe7c3cbfef size: 1572
```

Kuvio 5 Docker-pakettien lisäys rekisteriin.

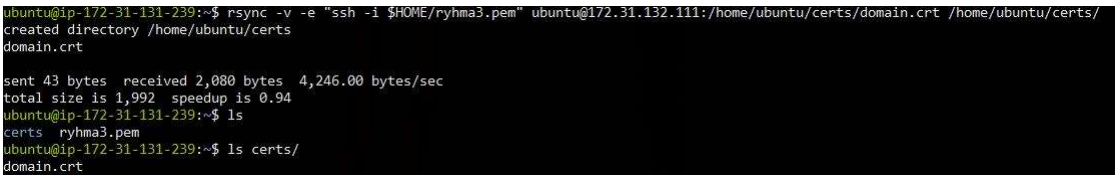
```
ubuntu@ip-172-31-132-111:~$ sudo docker image tag mongo:3.2 localhost:5000/mongo
ubuntu@ip-172-31-132-111:~$ sudo docker push localhost:5000/mongo
The push refers to repository [localhost:5000/mongo]
7eaf69109a22: Pushed
b436f480c034: Pushed
f6a5611931ed: Pushed
2bcf250f2488: Pushed
fcd5eec06559: Pushed
1f5a9fb2648f: Pushed
38ccb1166c8a: Pushed
ed81bb40beff: Pushed
75c203162075: Pushed
9d3049f87bb2: Pushed
337a2e6463ae: Pushed
latest: digest: sha256:3176c386ff76d928893a4be19d6b33a4ed64032266335d9dfa6e3103c152b719 size: 2614
ubuntu@ip-172-31-132-111:~$ sudo docker image tag n4sjamk/teamboard-io localhost:5000/teamboard-io
ubuntu@ip-172-31-132-111:~$ sudo docker push localhost:5000/teamboard-io
The push refers to repository [localhost:5000/teamboard-io]
5f70bf18a086: Pushed
ab45967bd1d0: Pushed
f93f4be07c8f: Pushed
4c0b80774d6b: Pushed
127bd5dfbd90: Pushed
2069b8a7f871: Pushed
f007f46cf986: Pushed
fd0f1d8a7fd0: Pushed
11ddcbabd68c: Pushed
0d81735d8272: Pushed
982549bd6b32: Pushed
8698b31c92d5: Pushed
latest: digest: sha256:aabc45677caa8e39300d08316cdff228db78a1ab2aebac595f5d18002743a4a2 size: 3449
ubuntu@ip-172-31-132-111:~$ sudo docker image tag n4sjamk/teamboard-api localhost:5000/teamboard-api
ubuntu@ip-172-31-132-111:~$ sudo docker push localhost:5000/teamboard-api
The push refers to repository [localhost:5000/teamboard-api]
5f70bf18a086: Mounted from teamboard-io
5b9bbc427137: Pushed
29a9377ac298: Pushed
dc634a91fe14: Pushed
c485ddbc9ce3: Pushed
aa235c9cc64f: Pushed
506e96b4fda4: Pushed
3062c38aeb3d: Pushed
5f7f0e5d6f5f: Pushed
0d81735d8272: Mounted from teamboard-io
982549bd6b32: Mounted from teamboard-io
8698b31c92d5: Mounted from teamboard-io
latest: digest: sha256:45f42e0ad815e9533b484708670424d2a922fd6d1f10002dfb41a9ca38dcbdba size: 3449
```

Kuvio 6 Docker-pakettien lisäys rekisteriin.

Kuvio 7 Docker-pakettien lisäys rekisteriin.

4.2.2 Varmenteiden jakaminen

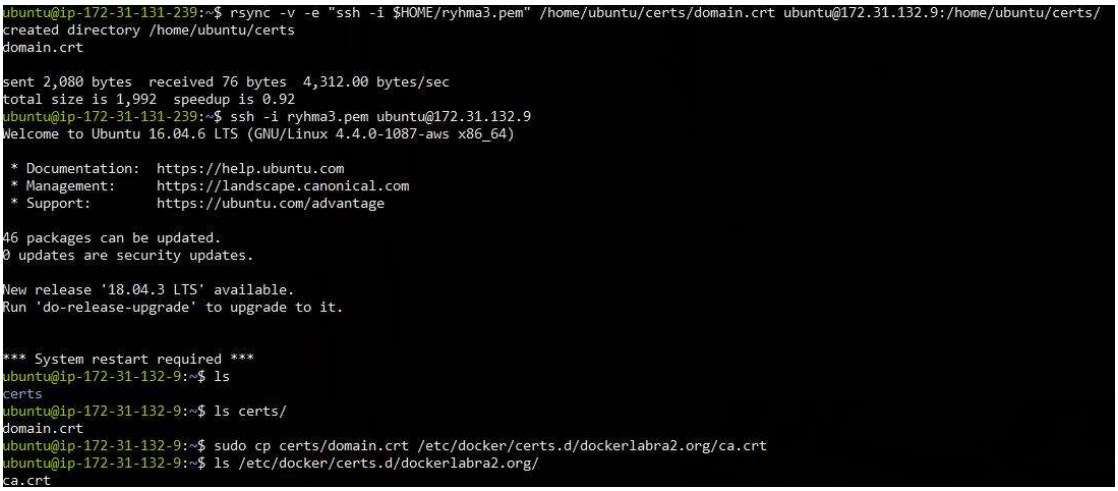
Jotta rekisteriä voidaan käyttää, täytyy koneiden, jotka hakevat Docker-paketteja rekisteristä luottaa rekisteriin. Tämä tarkoittaa, että sen luomamme varmenne täytyy siirtää jokaiselle koneelle, joka tahtoo käyttää rekisteriä. Ensimmäiseksi siirsimme edellisessä kappaleessa luomamme domain.crt -tiedoston koneelle, jolla on pääsy muille ryhmämme koneille, koska se kone, jolla Docker Registry sijaitsee, ei ole koneiden väliseen SSH-yhteyteen vaadittavaa avainta. Sieltä lähetimme domain.crt tiedoston jokaiselle koneelle, joka sitä tarvitsi (Kuvio 9). Eli jokainen kone, joka ajaa Contriboard-komponenttia.



Kuvio 8 domain.crt siirtäminen.

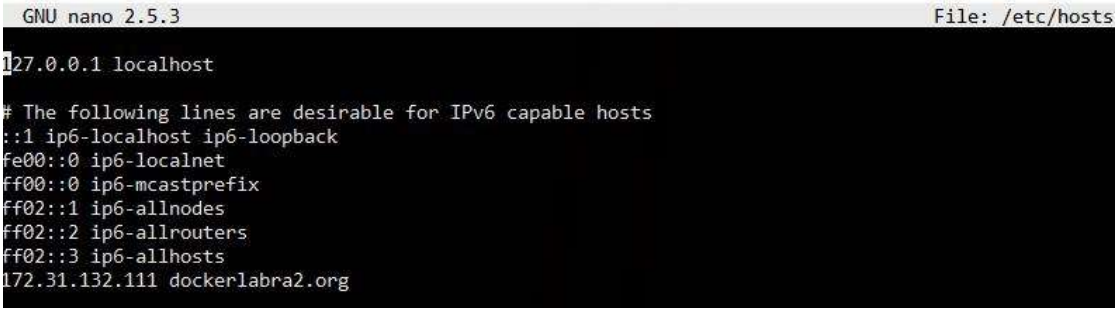
Koska meillä ei ollut oikeuksia sijoittaa domaiun.crt-varmennetiedostoa oikeaan paikkaan suoraan rsync:illä ssh:n yli, jouduimme siirtämään sen ensin kotikansioon kohdekoneella. Sitten otimme manuaalisesti SSH yhteyden koneeseen, ja siirsimme tiedoston oikeaan paikkaan (Kuvio 9). Tässä käytämme varmenteen luonnissa määriteltyä Common Name-arvoa, ja tiedoston nimi on ca.crt, eikä domain.crt (meidän tapauksessa dockerlabra2.org).

HUOM! Kuvista poiketen, polun täytyy sisältää myös portti, mikä on määritelty registryn käynnistäessä, eli meidän tapauksessa tiedoston sijainti on ”/etc/docker/certs.d/dockerlabra2.org:5000/ca.crt”.



Kuvio 9 domain.crt -tiedoston sijoittaminen.

Sitten muokkaamme /etc/hosts -tiedostoa. Lisäämme sinne rivin, joka ohjaa aiemmin annetun Common Name-arvon (dockerlabra2.org), tarkoittamaan sen koneen IP-osoitetta, jolla Docker Registry sijaitsee (Kuvio 10).



Kuvio 10 /etc/hosts.

4.2.3 Konttien käynnistys

Konttien käynnistämien sujuu nyt samalla lailla kuin normaalistikin, mutta Docker käyttää nyt myös omaa rekisteriämme. Koska Contriboard on nyt hajautettu useammalle koneelle, täytyy myös tehdä vaadittavat ympäristömuuttuja tiedostot (client.txt, api.txt ja io.txt), niille koneille jotka ko. komponentteja ajavat. Myös HAProxyn asetustiedostosta (Kuvio 11), määritellään API, Client ja IO koneiden osoitteet osoittamaan niille koneille missä komponentit sijaitsevat. Komponentit käynnistetään tietyssä järjestyksessä, suositeltavaa on käynnistää Redis ja Mongo ennen IO

ja API-kontteja. Client-kontin järjestyksellä ei ole väliä, jos asetukset ovat oikein. Kuvista poiketen kannattaisi myös käyttää dockerin komentorivi argumenttia ”—restart=always”, jotta kontti käynnistyy uudestaan aina koneen käynnistyessä.

```
backend API
balance roundrobin
    timeout server 30000
    timeout connect 4000
    server API1 172.31.132.85:9001 weight 1 maxconn 2048 check

backend WS
balance leastconn
    timeout server 30000
    timeout connect 4000
    server WebSocket1 172.31.132.30:9002 weight 1 maxconn 2048 check

backend CLIENT
        server client 172.31.132.9:80 weight 1 maxconn 2048 check
```

Kuvio 11 HAProxy asetukset.

Ensimmäiseksi Client komponentti. Tässä vaiheessa tarkistetaan, että HAProxy koneen julkinen IP-osoite on muuttumaton, ja sitten lisäämme sen client.txt-tiedostoon (Kuvio 12). Tämä tiedosto siis sijaitsee sillä koneella missä Contriboardin Client -komponentti ajetaan.

```
NODE_ENV=production
IO_URL=http://52.209.227.5
IO_PORT=80
API_URL=http://52.209.227.5/api
API_PORT=80
```

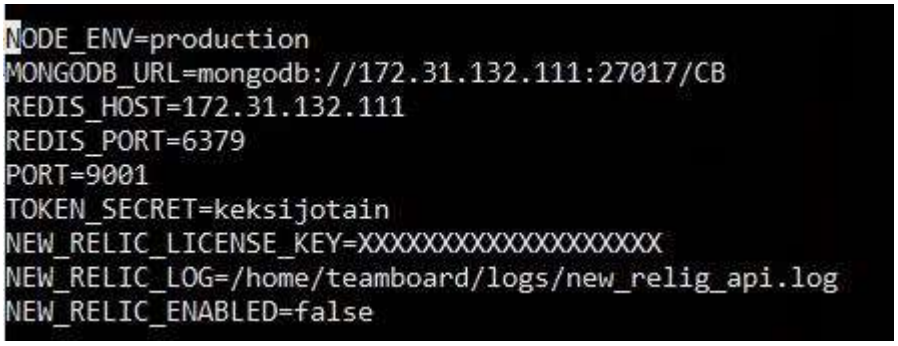
Kuvio 12 client.txt.

Sitten voimme käynnistää Clientin (Kuvio 13). Käynnistyskomennossa määritetään kontin kuvan nimeksi se paketti, joka on tallennettu, omaan rekisteriimme. Näin docker hakee sen sieltä.

Kuvio 13 Client -komponentin käynnistys.

Redis ja Mongo käynnistetään samalla koneella kuin Registry. Niiden sijainti ei ole muuttunut ensimmäisestä laboratoriotyöstä.

API:n käynnistäminen (Kuvio 15) sujuu samoin, mutta api.txt tiedostoa täytyy muuttaa, koska API sijaitsee nyt omalla koneellaan. API:n täytyy siis tietää MongoDB:n ja Redis-komponentin koneen osoite (Kuvio 14).



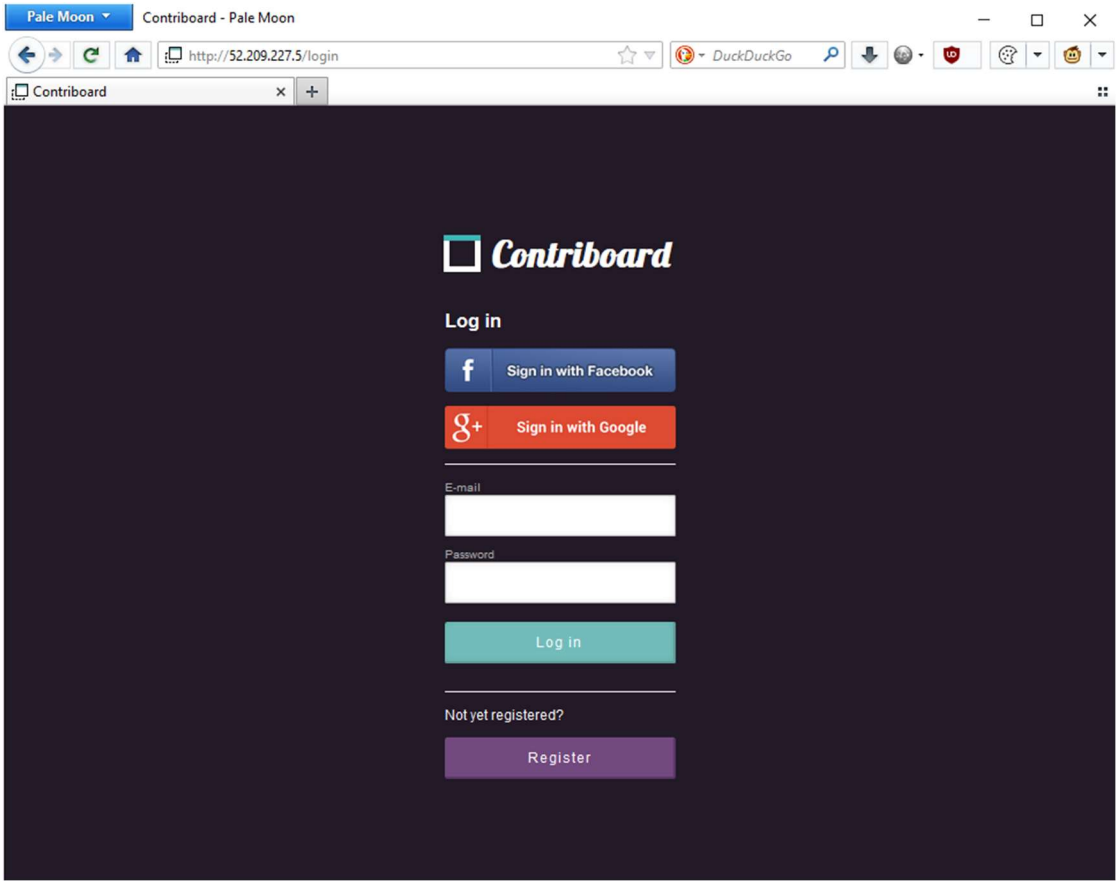
Kuvio 14 api.txt.

Kuvio 15 API -kontin käynnistys.

IO-kontin io.txt asetustiedostosta täytyy muuttaa, että REDIS_HOST-asetus osoittaa siihen koneeseen missä Redis-kontti pyörii. Sitten ajetaan taas samalla lailla kuin muutkin (Kuvio 16).

Kuvio 16 IO-kontin ajaminen.

Sitten voidaankin avata se selaimessa, menemällä HAProxy koneen IP-osoitteeseen. Jos kaikki on oikein, pitäisi näkyä kirjautumissivu (Kuvio 17).



Kuvio 17 Contriboard selaimessa.

4.3 Kuormantasaus

Jotta palvelu kestäisi monta käyttäjää, täytyy kuormaa hajauttaa usean koneen kesken. Liikenne näihin sitten tasataan joko Amazonin Elastic Load Balancerilla tai HAProxyllä.

Hajautetaan API ja IO konttien kuormaa. Ajetaan niistä uudet instanssit. API-konttien määrää lisätään kahdella, ja IO-kontteja yhdellä. Kaksi API-konttia pyörii samalla palvelimella (Kuvio 18), ja yksi ajetaan eri palvelimella (Kuvio 19). IO-kontteja ajetaan kaksi samalla palvelimella (Kuvio 20).

```
ubuntu@ip-172-31-132-85:~$ sudo docker ps -a
CONTAINER ID        IMAGE                                   COMMAND                  CREATED             STATUS              PORTS
                    NAMES
d265130ca959        dockerlabra2.org:5000/teamboard-api     "/bin/sh -c '/usr/bi…"   2 weeks ago         Up 37 minutes       0.0.0
.0:9002->9001/tcp   contriboard-api2
05ddc618b00d        dockerlabra2.org:5000/teamboard-api     "/bin/sh -c '/usr/bi…"   3 weeks ago         Up 37 minutes       0.0.0
.0:9001->9001/tcp   contriboard-api
ubuntu@ip-172-31-132-85:~$
```

Kuvio 18 Kaksi API-konttia samalla koneella

```
ubuntu@ip-172-31-132-119:~$ sudo docker ps -a
CONTAINER ID        IMAGE                                   COMMAND                  CREATED             STATUS              PORTS
                    NAMES
d906e4d524a3        dockerlabra2.org:5000/teamboard-api     "/bin/sh -c '/usr/bi…"   2 weeks ago         Up 37 minutes       0.0.0
.0:9001->9001/tcp   contriboard-api
ubuntu@ip-172-31-132-119:~$
```

Kuvio 19 Kolmas API-kontti eri koneella

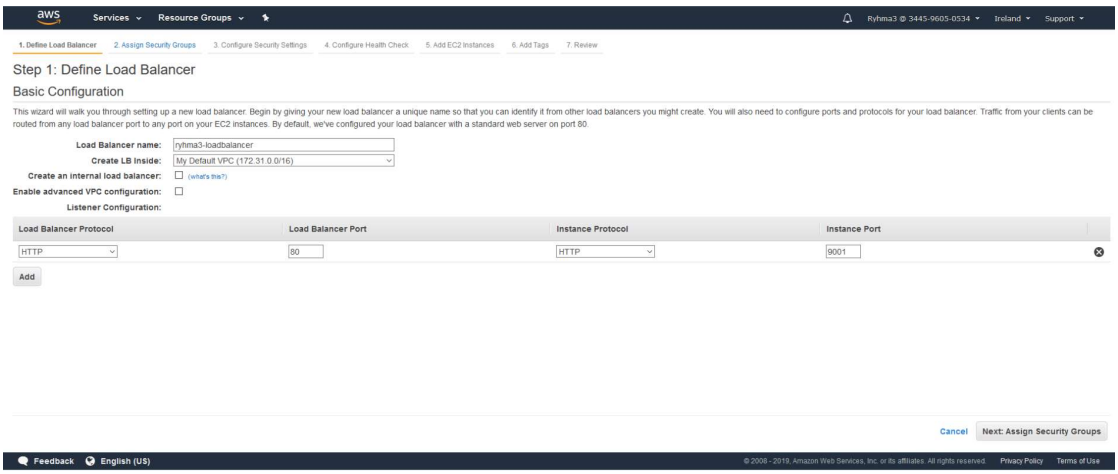
```
ubuntu@ip-172-31-132-30:~$ sudo docker run -d --env-file=/envit/io.txt -p 9003:9002 --expose=9002 -v /tmp:/home/teamboard/logs -
-name contriboard-io2 dockerlabra2.org:5000/teamboard-io
93d7ff8612f3c72321282e165137d90768b6a1981eb9b0d99c413e51ff8de7fb
ubuntu@ip-172-31-132-30:~$ sudo docker ps -a
CONTAINER ID        IMAGE                                   COMMAND                  CREATED             STATUS              PORTS
                    NAMES
93d7ff8612f3        dockerlabra2.org:5000/teamboard-io      "/bin/sh -c '/usr/bi…"   12 seconds ago      Up 11 seconds       0.0.0.
0:9003->9002/tcp    contriboard-io2
75f2f74f36bb        dockerlabra2.org:5000/teamboard-io      "/bin/sh -c '/usr/bi…"   2 weeks ago         Up 2 minutes        0.0.0.
0:9002->9002/tcp    contriboard-io
```

Kuvio 20 IO-konttien kahdennus

4.3.1 Elastic Load Balancer

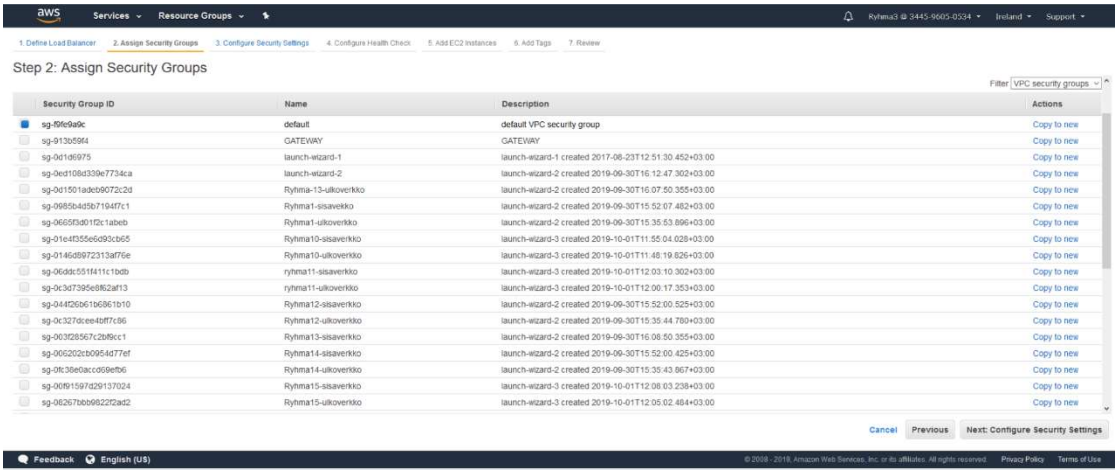
Elastic Load Balancer eli ELB on Amazon Web Servicen tarjoama kuormantasaaja, joka jakaa tulevan liikenteen tasaisesti valituille EC2-instansseille. Palvelu otetaan käyttöön AWS käyttöliittymän kautta. Kokeilimme kuormantasausta tätä kautta vain API:lle.

Ensin annetaan nimi uudelle kuormantasaajalle (Kuvio 21).



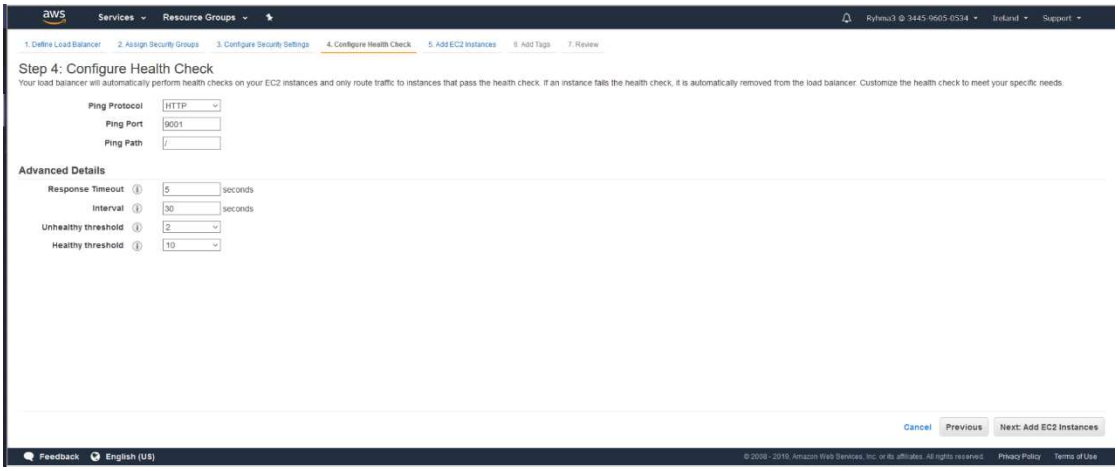
Kuvio 21 ELB Askel 1.

Määritellään Security Groups (Kuvio 22).



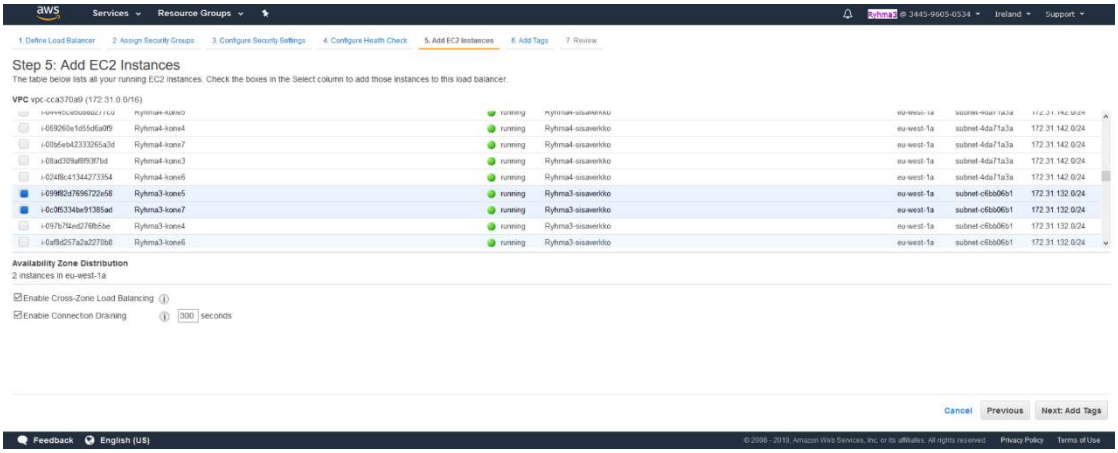
Kuvio 22 ELB Askel 2.

Märitellään Health Check (Kuvio 23).

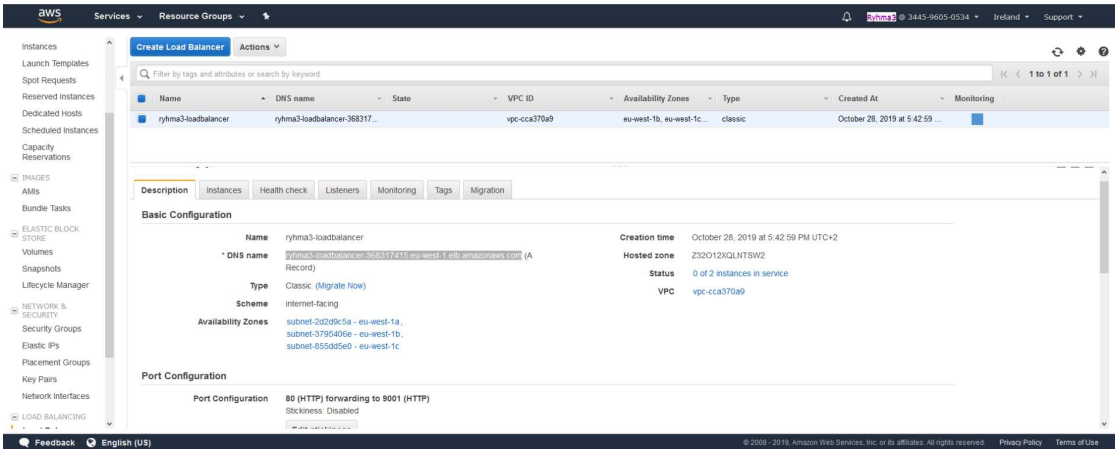


Kuvio 23 ELB Askel 4.

Lisätään instanssit kuormatasaajaan (Kuvio 24).



Kuvio 24 ELB Askel 5.



Kuvio 25 ELB DNS name.

4.3.2 HAProxy

Ennen HAProxya käyttäöönottoa otetettiin ELB pois käytöstä. HAProxyn asetuksiin lisättiin API:n kuormantasaus, joka onnistui lisäämällä muutama rivi HAProxyn konfiguraatio tiedostoihin (Kuvio 26).

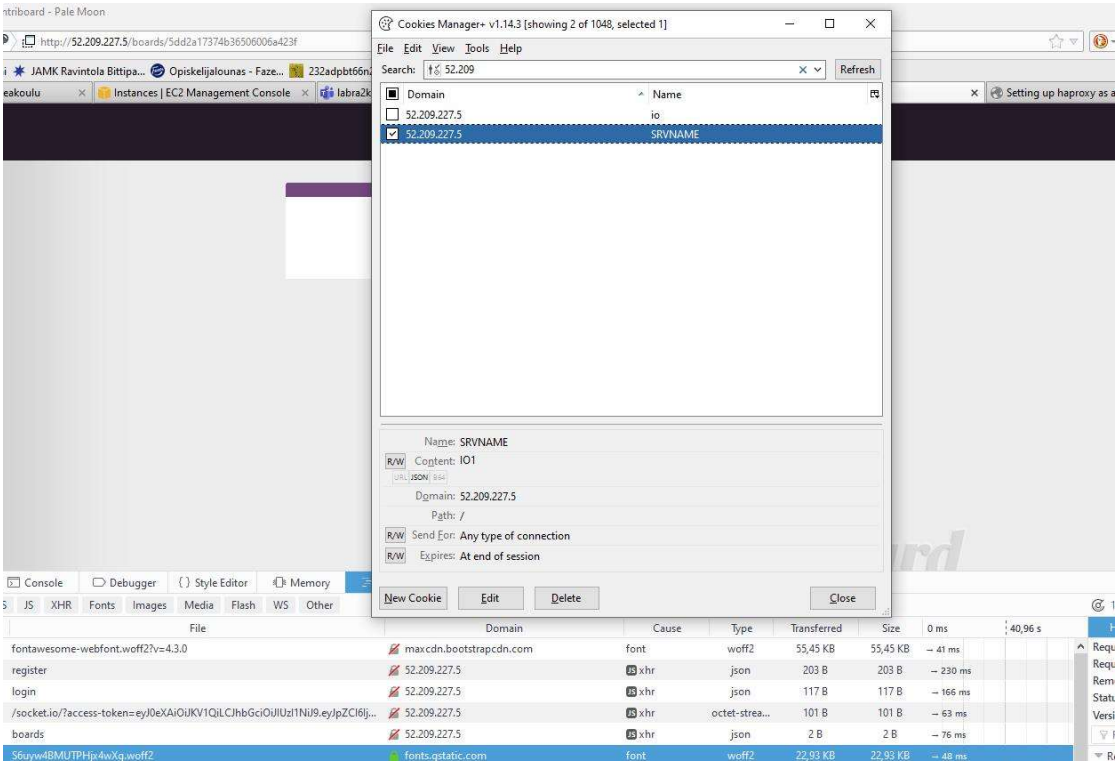
```
backend API
balance roundrobin
    timeout server 30000
    timeout connect 4000
    server API1 172.31.132.85:9001 weight 1 maxconn 2048 check
    server API2 172.31.132.85:9002 weight 1 maxconn 2048 check
    server API3 172.31.132.119:9001 weight 1 maxconn 2048 check
```

Kuvio 26 HAProxy config API kuormantasaus.

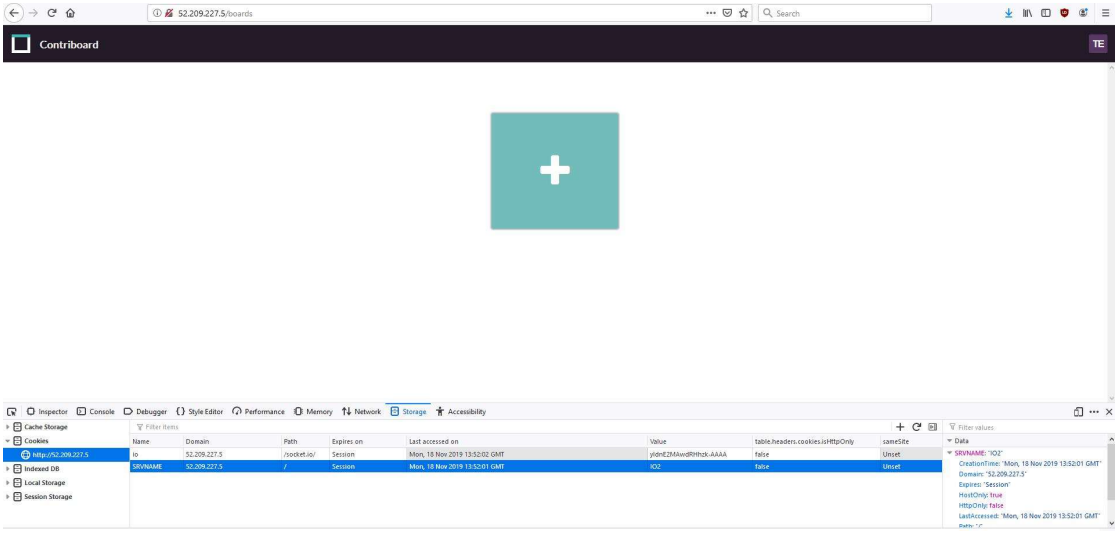
Nämä rivit lisättyä boottasimme HAProxyn ja kuormantasaus oli valmis API-konttien osalta. IO-konttien osalta kuormantasauksen täytyy aina ohjata saman clientin yhteys samalle koneelle (Sticky Load Balancing). HAProxyn asetustiedostossa asetetaan toinen palvelin ja niille asetus ”cookie <nimi>”. Myös lisätään asetus ”cookie SRVNAME insert”. Tämä antaa selaimelle keksin missä arvona sen serverin ”cookie <nimi>”. Sitten kun selain ottaa HAProxy:yn yhteyden, HAProxy osaa tämän keksin perusteella ohjata yhteyden oikeaan IO-palvelimeen (Kuviot 28 ja 29).

```
backend WS
balance leastconn
    timeout server 30000
    timeout connect 4000
    cookie SRVNAME insert
    server WebSocket1 172.31.132.30:9002 weight 1 maxconn 2048 cookie IO1 check
    server WebSocket2 172.31.132.30:9003 weight 1 maxconn 2048 cookie IO2 check
```

Kuvio 27 IO -konttien load balance.



Kuvio 28 Sticky Load Balance slaimessa.



Kuvio 29 Eri koneella menee eri IO-serverille.

5 Pohdinta

Työn teko meni sukkelaan, sillä kuivaharjoittelu muistiinpanot sekä labra yksi raportti auttoi palauttamaan asioita mieleen. Päätimme jo alussa, että Docker rekisteri tulee toimimaan Kone kolmosen päällä, sillä jokainen Docker container oli jo sillä koneella. Tällä tavoin oman rekisterin pystyttäminen tapahtui todella helposti ja jokainen image oli vain Tagia ja pushia vailla valmis.

Otimme rekisterin sertifikaatit kone kolmesta ja siirsimme ne rsync SSH:n avulla kone ykköselle, josta siirsimme ne eteenpäin sisäverkon muihin koneisiin, jonka avulla Docker pull komento saatiin toimimaan myös omaan rekisteriin. Ensimmäistä Docker run komentoa ajettaessa huomasimme, että emme saaneet yhteyttä Docker rekisteri koneeseen, jolloin ylistetty opettajamme Viinikanoja lopulta onnistui auttamaan meitä kertomalla sen, että sisäverkko tarvitsi uuden säännön, joka salli liikenteen Docker rekisteriin. Lisäsimme Amazon Security Group:iin säännön, joka salli sisäisen liikenteen kyseisessä verkossa.

Docker rekisterin toimimisen jälkeen, aloimme pystyttämään komponentteja eri Amazonin koneille. Tämä sujui suhteellisen kivuttomasti taas vanhoja muistiinpanoja välillä tarkastellen. Saimme koko järjestelmän lopulta suhteellisen nopeasti toimimaan, mutta Contriboardin reaaliaikaisuus oli hajalla, joka johtui siitä, että emme osien asetusten muutosten jäljiltä olleet koskaan käynnistäneet kontteja uudestaan.

Reaaliaikaisuus saatiin lopulta toimimaan oikein käynnistämällä IO ja API kontit uudestaan.

HAProxy kuormantasaus API konttien kesken oli pystyssä jo ennen kuin Elastic Load Balancea edes katsottiin. Otimme kuitenkin HAProxy kuormantasauksen pois päältä, kun teimme ELB kuormantasausta. ELB käyttöön otossa huomasimme, että se on täysin turha ja joutava kapistus sekä, että HAProxy on äärettömästi parempi.

Lähteet

Amazon EC2 Instance IP Addressing. Artikkeli docs.aws.amazon.com sivustolla. Viitattu 28.10.2019. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html#concepts-private-addresses.

How to Use HAProxy for Load Balancing. HAProxyn kuorman tasauksen käyttöohje linode.com sivustolla. Viitattu 11.11.2019. https://www.linode.com/docs/uptime/loadbalancing/how-to-use-haproxy-for-load-balancing/.

Test an insecure registry. Docker dokumentaatio. Viitattu 11.11.2019. https://docs.docker.com/registry/insecure/#use-self-signed-certificates.

What is a Container. Artikkeli docker.com sivustolla. Viitattu 28.10.2019. https://www.docker.com/resources/what-container.