Lappeenrannan teknillinen yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma

Kandidaatintyö

Nidal Abu Raed

Jatkuva toimitus ja Docker

Työn tarkastaja(t): D.Sc. (Tech.) Ari Haapponen

Työn ohjaaja(t): D.Sc. (Tech.) Ari Haapponen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Nidal Abu Raed

Jatkuva toimitus ja Docker

Kandidaatintyö
2018

42 sivua , 19 kuva, 1 taulukko, 1 liite

Työn tarkastajat: D.Sc. (Tech.) Ari Happonen

Hakusanat: jatkuva toimitus, Docker, konttiratkaisut, virtualisointi
Keywords: containerization, cloud-services, virtualization

Tässä tutkielmassa tarkastellaan mitä tarkoitetaan kontti teknologialla, ja miten jatkuva toimitus hyödyntää kyseistä tekniikkaa sekä mitä mahdollisuuksia kontti ratkaisuilla on tarjota, ja mitä haasteita se tuo mukanaan. Lisäksi aiemmin käytettyjä virtuaalikoneita analysoidaan ja verrataan konttiratkaisuihin. Työssä toteutettiin 2048 -pelin konttikuljetus Google Cloud Platform -alustalle havainnollistamaan kuljetusprosessia. Tutkimus tuloksena konttien tehtävä on eristää sovellus itsenäiseen yksikköön joka voidaan suorittaa missä vain ympäristössä, jolloin sovellusten toimittaminen helpottuu ja se voidaan automatisoida. Virtuaalikoneiden käyttöä on korvattu konteilla konttien kevyen luonteen vuoksi. Konttiratkaisut tarjoavat pääasiallisesti siirrettävyyttä, modulaarisuutta ja tehokkuutta.

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Nidal Abu Raed

Continuous development/deployment with Docker

Bachelor's Thesis

42 pages, 19 figures, 1 tables, 1 appendices

Examiners : D.Sc. (Tech.) Ari Happonen

Keywords: continuous delivery, Docker, containerization, virtualization

This study investigates what is container technology and how does continuous delivery relate to this matter. Also this study exposes the opportunities and challenges that comes along with these technologies. In addition, previously used alternatives, such as virtual machines, were analysed and compared to recently trending container solutions. In this study, containerization of 2048 -game was implemented to demonstrate the delivery process of an application to cloud application platform (Google Cloud). As findings, containers are used to isolate application into independent units that can executed in any environment that support these technologies, which makes the delivery process of applications easier and possibility to be automated. The lightweight nature of containers have replaced the use of virtual machines in most cases. Key benefits of containers are portability, modularity and efficiency.

ALKUSANAT

Työ on tehty Lappeenrannan Teknillisessä Yliopistossa, ja haluan kiittää kaikki jotka ovat olleet tukenani tätä työtä tehdessä. Erityisesti haluan kiittää Ari Happosta työn ohjaamisesta.

SISÄLLYSLUETTELO

LIITTEET

SYMBOLI- JA LYHENNELUETTELO

CPU	Central Processing Unit
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
PaaS	Platform as a Service
SDK	Software Development Kit

1 JOHDANTO

Tässä opinnäytetyössä tarkastellaan mitä tietotekniikassa tarkoitetaan kontti teknologialla, ja miten jatkuva toimitus hyödyntää kyseistä tekniikkaa. Työssä tutkitaan mitä mahdollisuuksia kontti ratkaisuilla on tarjota, sekä mitä haasteita se tuo mukanaan. Konttiratkaisuja verrataan myös aiempiin olemassa oleviin ratkaisuihin. Lisäksi työssä toteutettiin sovelluksen konttikuljetus pilvipalvelu -alustalle. Lopuksi esitellään johtopäätöksiä ja pohdintaa aiheesta.

1.1 Tausta

Valitsin työni aiheen omasta kiinnostuksesta konttiteknologiaan ja halusta tutustua Google Cloud Platform -alustan käyttöön. Lisäksi ohjelmistotuotteiden jatkuva toimitus asiakkaille vaikutti mielenkiintoiselta ja tarpeelliselta ymmärtää. Aihealueista kuulin ensimmäistä kertaa kanssaopiskelijoilta, ja asiantuntijoilta jotka työskenteli kyseisten tekniikoiden parissa. Myös Docker “hype” sosiaalisessa mediassa kannusti tutkimaan, mitä vaikutuksia sillä on ohjelmistokehityksen kannalta ja tutustumaan aikaisempiin ratkaisuihin. Nykypäivänä IT-alan yritykset harjoittavat ohjelmistokehityksen jatkuvaa toimittamista, tarkoituksena automatisoida päivityksien julkaisuprosessi, mikä on kiinnostavaa myös pelien kehittämisen kannalta.

1.2 Tavoitteet ja rajaukset

Kandidaatintyön tavoitteena on kirjallisuuskatsauksen kautta ymmärtää mitä konttiteknologia ja jatkuva toimitus on. Tavoitteena on siis ymmärtää konttien ja jatkuvan toimituksen periaatteet, sekä niiden tuomia mahdollisuuksia ja haasteita. Lisäksi työssä analysoidaan konttiratkaisuja edeltäviä ratkaisuja, tavoitteena vertailla ratkaisujen vahvuuksia ja heikkouksia keskenään. Kandidaatintyössä toteutetaan myös 2048 -pelin konttikuljetus Google Cloud Platform -pilvipalveluun. Konttikuljetuksen päätavoitteena on

havainnollistaa kuinka sovellukset käytännössä pakataan kontteihin ja nostetaan pilvipalveluun pyörimään ihmisten käyttöön. Lisäksi tavoitteena on teoreettisesti havainnollistaa kuinka jatkuva toimitusputki toteutetaan GCP -alustalla. Konttikuljetus toteutetaan Docker-kontteja hyödyntäen. Konttikuljetus mahdollistaa 2048 -pelin jakamisen kenelle tahansa pelattavaksi Internet-selaimella. Pelin ohjelmointikielenä käytettiin JavaScript -ohjelmointikieltä.

1.3 Työn rakenne

Aiheen käsittely aloitetaan johdanto-osuuden jälkeen luvussa kaksi teorialla. Kolmannessa luvussa esitellään sovelluksen konttikuljetus, eli työn toteutusosa. Luvussa käsitellään toteutukseen käytettäviä tekniikoita ja työkaluja. Neljännessä luvussa analysoidaan aiempia olemassa olevia ratkaisuja niiden vahvuuksia ja heikkouksia vertailemalla. Viidennessä luvussa esitellään omaa yleistä pohdintaa, ja mietteitä siitä, miten työssä onnistuttiin päätavoitteiden suhteen. Luvussa käydään myös läpi konseptien tulevaisuutta, ja sen tuomia mahdollisuuksia. Työn viimeisessä luvussa tehdään yhteenveto työssä esitellyistä asioista.

2 KIRJALLISUUSKATSAUS KONTTITEKNOLOGIAN TEORIASTA JA KÄYTÖSTÄ

Työn tekemiseen vaadittiin yleinen käsitys pilvilaskennan eri osa-alueilta, joista tärkeimpiä olivat sovellus kontit (*engl. application containers*) ja pilvisovellusalustat (*engl. Cloud Application Platform*). Lisäksi vaadittiin ymmärrys mitä tarkoitetaan jatkuvalla toimituksella ohjelmistokehityksessä. Tässä luvussa esitellään työhön oleellisesti liittyviä aihealueita niiden teorian kautta.

Konttiteknologia on ollut kasvavassa suosiossa jo vuosikymmenen, ja se on mahdollistanut käyttöjärjestelmän virtualisoinnin ja jakamaan saman instanssin käyttöjärjestelmästä [1]. Konttitekniikka on käytännössä osa virtualisointitekniikkaa, joka tarkoittaa jonkin asian virtualisointia, kuten fyysisen laitteiston, varaston tai tietokoneverkon resursseja [2]. Virtualisointi mahdollistaa yhden fyysisen koneen käyttämään useita itsenäisiä ja eristettyjä virtuaalikoneita, virtualisoimalla prosessori, muisti, levy ja verkko -resurssit vastaaviksi virtuaalisiksi resursseiksi. Virtuaalikoneista enemmän luvussa 4.

Ensimmäinen täysin toteutettu linux konttikuljetus metodi, LXC, kehitettiin vuonna 2007 [3], jonka jälkeen ihmiset alkoivat ajatella säiliöitä ylimääräisenä eristys prosessina, jolla voidaan minimoida virtuaalikoneista koituvia kustannuksia [4]. Docker puolestaan ilmestyi ensimmäisen kerran vuonna 2013, ja se tarjosi kokonaista ekosysteemiä konttien hallintaa. Tuolloin Docker tuki ainoastaan linux käyttöjärjestelmäytimiä, mutta tänä päivänä se soveltuu myös Windows käyttöjärjestelmille Hyper-V (Virtual Machine manager) ominaisuuksineen.

2.1 Docker-konttiratkaisun periaatteet

Tosiasiassa termi “kontti” on vain abstrakti konsepti kuvaamaan miten tietyt tietokoneen ominaisuudet toimivat yhdessä. Docker hyödyntää seuraavia Linux-ytimien ominaisuuksia [4]:

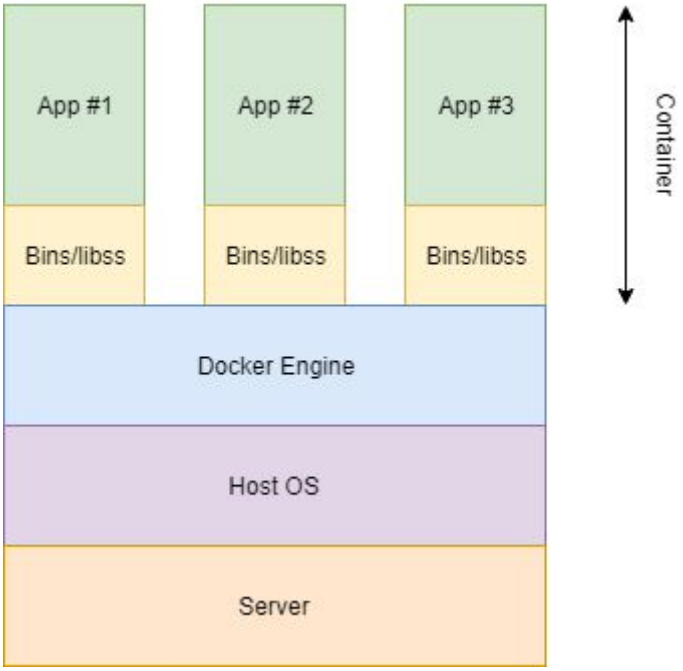
- Nimiavaruuksia (*engl. namespaces*)
- Kontrolliryhmiä (*engl. cgroups tai control groups*)
- Union File System

Docker käyttää linux-ytimen tarjoamia eri nimiavaruuksia yhdessä pääasiallisesti konttien väliseen eristämiseen ja itse konttien luomiseen. Nimiavaruudet, joita docker käyttää ovat *pid, uts, mnt, ipc, net* ja *USER*. Jokaisella nimiavaruudella on oma tehtävä konttien eristämisessä [1][4].

1. PID eli prosessi-id pitää huolen, että toisen kontin prosessi ei vaikuta toiseen.
2. UTS (UNIX Timesharing System) sallii prosessin tunnistamaan järjestelmän tunnisteita (kuten verkkotunnuksen nimen), ja näin mahdollistaa konteille omat riippumattomat NIS-verkkotunnukset ja palvelimen nimen.
3. MNT tehtävänä on tuoda konteille kuva omasta tiedostojärjestelmästä, jotta prosesseilla on nimiavaruuksien liitoskohdissa eri käsitys tiedostojärjestelmän hierarkiasta.
4. IPC tarkoittaa prosessien välistä viestintää. IPC-nimiavaruus siis vastaa IPC-resurssien eristämisestä konttien sisällä suoritettujen prosessien välillä.
5. NET takaa verkon eristyksen. Se tarjoaa kontille oman järjestelmän verkkopinon, joka koostuu muun muassa IP-osoitteesta, porteista, omista verkkolaitteista, jne.
6. USER vastaa käyttäjien eristämisestä. Se sallii konttien omata eri näkemys *userID:n* ja *groupID:n* alueesta, mikä takaa että kontin sisällä pysyy root-käyttäjä oikeudet vaikka kontin ulkopuolella saattaa olla oikeudeton käyttäjä.

Kontrolliryhmiä docker hyödyntää resurssienhallintaan. Kontrolliryhmien tavoite on taata että kontti käyttää vain niitä resursseja joita se tarvitsee, eikä yksi kontti voi kaataa kokonaista järjestelmää alas. Kontrolliryhmät mahdollistaa tietokoneen resurssien (CPU, muisti, verkko, jne.) tehokkaan jakamisen konttien välillä. Union File System puolestaan voidaan ajatella olevan virtuaalinen tiedostojärjestelmä, joka on eri tiedostojen kerrosten pino. Kyseisellä tiedostojärjestelmällä on kaksi pääasiallista etua: (1) konttien luominen ja suorittaminen ei vaadi aina tiedostojen täydellistä kopioimista, ja tekee näin kontti instanssien luomisesta nopeaa ja halpaa, (2) muutoksien tekeminen on nopeaa, sillä muutos päivitetään vain siihen kerrokseen jossa muutos tapahtui.

Toisin kuin virtuaalikoneet, kontit perustuvat käyttöjärjestelmä-tason virtualisointiin [5]. Docker-kontin kevyt luonne mahdollistaa usean kontin samanaikaisen suorittamisen samalla palvelimella tai virtuaalikoneella. Kuvan 1 diagrammista näkyy kuinka jokaiselle kontille on varattu oma eristetty “käyttäjätila”, joka sallii usean kontin suorittamisen yhdellä isäntäkoneella. Diagrammista myös näkyy kuinka käyttöjärjestelmä tason arkkitehtuuri jakautuu jokaiselle kontille, ainoat osat jotka on luotu “raa’alla voimalla” ovat *bins/libs* -lohkot. Tästä syystä kontit ovat todella kevyitä yksiköitä.



Kuva 1. docker-kontti diagrammi

Docker-kontteja voi kuka tahansa käyttää hyödyksi siirrettävien sovelluksien pakkaamiseen ja toimitukseen. Pakatut kontit voi sellaisenaan ladata mihin tahansa pilvipalveluun (pilvisovellus alustaan) pyörimään. Tyypillisesti kyseiset pilvipalvelut luokitellaan PaaS -tyyppisiksi (Platform as a Service). PaaS-termillä tarkoitetaan kokonaista kehitys- ja käyttöönottoympäristöä, joka on varustettu joukolla resursseja pilvessä, joiden avulla voidaan tarjota yksinkertaisia pilvipohjaisia sovelluksia aina monimutkaisiin yrityssovelluksiin [6]. PaaS -palveluntarjoaja tyypillisesti tarjoaa jonkinlaista moottoria konttien organisointiin. Konttien organisointiin kuuluvat erityisesti seuraavat tehtävät: aikataulutus (sijoittelu, replikointi/skaalaus, uudelleensuunnittelu, päivitykset), resurssien hallinta (muisti, CPU, volyymit, kuvat, portit, IP) ja palveluiden hallinta (eli konttien järjestäminen leimojen, ryhmien, nimiavaruuksien, kuormitus tasapainotuksen tai valmius tarkastuksen avulla) [7].

2.2 Mahdollisuudet ja haasteet

Kontti on standardisoitu ohjelmistoyksikkö, jonka tehtävä on eristää sovellus ja sen riippuvuussuhteet itsenäiseen yksikköön joka voidaan suorittaa missä vain ympäristössä [4]. Kontit poistavat fyysisten laitteistojen tarpeen ja tehostaa laskentatehon käyttöä, eli energiaa ja kustannuksia säästyy merkittävästi. Tuotannon käyttöönotto on helppoa ja vaivatonta, koska kehitysympäristö on sama kuin tuotantoympäristö, eli kun ohjelmistokehittäjät esimerkiksi tuottavat uutta koodia sovellukseen, ja kun se viedään tuotantoympäristöön, päivitys aina "toimii" [8]. Konttien salliminen ei vain helpota ohjelmistojen käyttöönottoa ja kehitystä, mutta se tekee siitä myös nopeampaa. Näin ollen yritykset kykenevät vastaamaan asiakkaiden nopeaan kysynnän vaihteluun sekä kyky palvella suuria määriä käyttäjiä paranee huomattavasti.

Docker helpottaa sovellusten pilkkomista pienempiin toiminnallisiin osiin kontteihin. Esimerkiksi jollekin sovellukselle voidaan halutaan luoda tietokanta yhteen konttiin ja palvelin toiselle sekä itse Node.js sovellus kolmanteen, dockerin avulla pystytään helposti linkittämään kontit luomaan yhtenäisen sovelluksen, jota on helppo skaalata ja komponentteja päivitellä itsenäisesti tulevaisuudessa.

Kontit varaavat resursseja tarpeen mukaisesti, mikä tarkoittaa että laskenta resurssien käyttö on tehokasta. Useita kontteja voidaan suorittaa samanaikaisesti yhden palvelimen päällä. Näin ollen varmistetaan myös se että, yhden instanssin häiriö ei vaikuta koko systeemin kaatumiseen.

Docker-kontit edesauttavat sovellusten siirrettävyyttä. Konttien avulla voidaan koota sovellukset kerran ja muodostaa niistä kontti-kuva, mikä voidaan suorittaa missä vain ympäristössä, mikä tukee dockeria. Näin ollen infrastruktuurissa vapaasti liikkuminen on vaivatonta ja nopeaa.

Lisäksi Dockerin käyttäjien etuna on yhä kasvavassa määrin oleva Docker Hub, mikä voidaan ajatella olevan docker-kuvien "kauppana". Docker Hub on pilvipohjainen repository, jonne Docker yhteisön jäsenet voivat luoda docker-kuvia ja jakaa niitä julkisesti

muiden käytettäväksi. Docker Hub:sta on todella helppoa etsiä omille tarpeilleen vastaavia kontti-kuvia, joita voi käyttää sellaisenaan, esimerkiksi DevOps-asiantuntija saattaa hetkessä ladata nginx -verkkopalvelimen kontissa olevan sovelluksen käyttöön.

Suuri huolenaihe Dockerin suhteen on turvallisuuskysymykset, sillä tunkeutujat voivat helposti kaapata super-käyttäjä oikeudet [1]. Tämä johtuu siitä, että kontit käyttävät isäntäkoneen käyttöjärjestelmää jolloin konttien sisällä suoritetut operaatiot ovat läpinäkyviä isäntäkoneella.

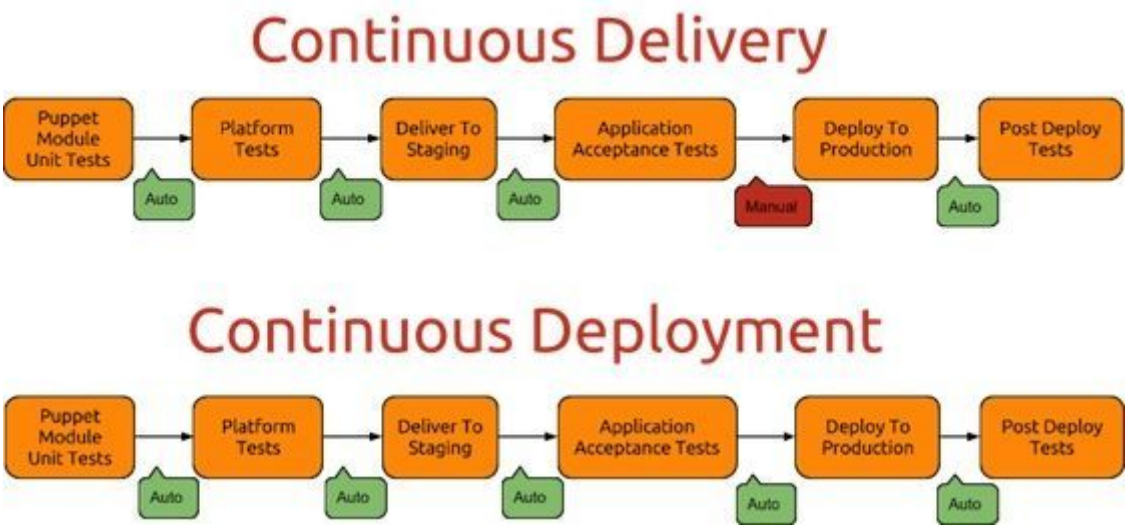
PaaS-tyyppisten pilvisovellus alustojen avulla pystytään välttämään kustannuksia, ohjelmistolisenssien ostamisen ja hallinnan monimutkaisuutta, sekä ohjelmistojen perustana olevia infrastruktuuri valintoja ja väliohjelmistoja. Lisäksi erillisiä kehitystyökaluja ja muita resursseja ei tarvitse sen koommin määrittää. Käytännössä voidaan vain keskittyä sovellusten ja palveluiden kehittämiseen ja hallintaan, ja pilvipalvelun tarjoaja pitää huolen kaikesta muusta [6].

2.3 Docker osana jatkuvaa toimitusta

Jatkuva toimitus (*engl. continuous delivery*) on osa DevOps ohjelmistokehitys käytäntöä, jossa koodi muutokset automaattisesti kootaan, testataan ja valmistellaan tuotantoon [9]. Tapauksessa jossa jatkuva toimitus on toteutettu oikein, ohjelmistokehittäjillä on aina käyttöönotto-valmis koottu artefakti, joka on läpäissyt testausvaiheen. Jatkuva toimitus mahdollistaa ohjelmistokehittäjiä automatisoimaan testausvaiheen, jotta aikaa jää enemmän sovelluksen päivityksien tarkastamiseen ennen sen käyttöönottoa asiakkaille.
Jatkuvalla toimituksella on merkittäviä hyötyjä ohjelmistokehityksessä. Sen avulla mahdollistetaan ohjelmistojulkaisu prosessin automatisointi, parannetaan ohjelmistokehittäjien tuottavuutta, tunnistetaan bugit nopeammin sekä toimitetaan päivityksiä nopeammin.

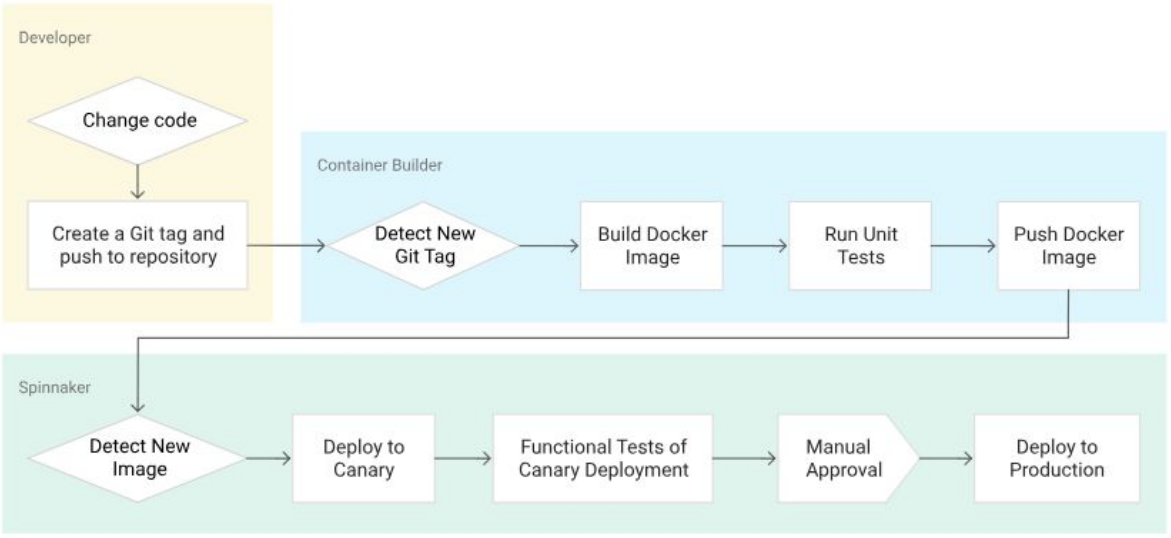
Termit jatkuva toimitus ja jatkuva käyttöönotto saatetaan sekoittaa keskenään, oleellinen ero näillä kahdella termillä kuitenkin on manuaalinen hyväksyntä päivittää tuotantoon.

Kuva 2 havainnollistaa molempien käytäntöjen prosessivaiheiden automatisaatiota, kuvasta näkee jatkuvan toimituksen eron jatkuvaan käyttöönottoon.



Kuva 2. Jatkuva toimitus ja jatkuva käyttöönotto

Jotta ohjelmistojen jatkuva toimittaminen on käytännössä mahdollista, täytyy toimitusprosessille luoda erillinen toimitus putki [10]. Toimitusputken tavoitteena on automatisoida koodin kokoaminen, testaaminen ja lisääminen sovellukseen. Koodi muutokset tulee automaattisesti virrata putkea pitkin edellä mainittujen vaiheiden läpi aina tuotantoon asti. Kuva 3 havainnollistaa minkälaisiin osiin GCP:lla toteutettu toimitusputki jaetaan.



Kuva 3. GCP:llä toteutetun toimitusputken toimintaperiaate [10]

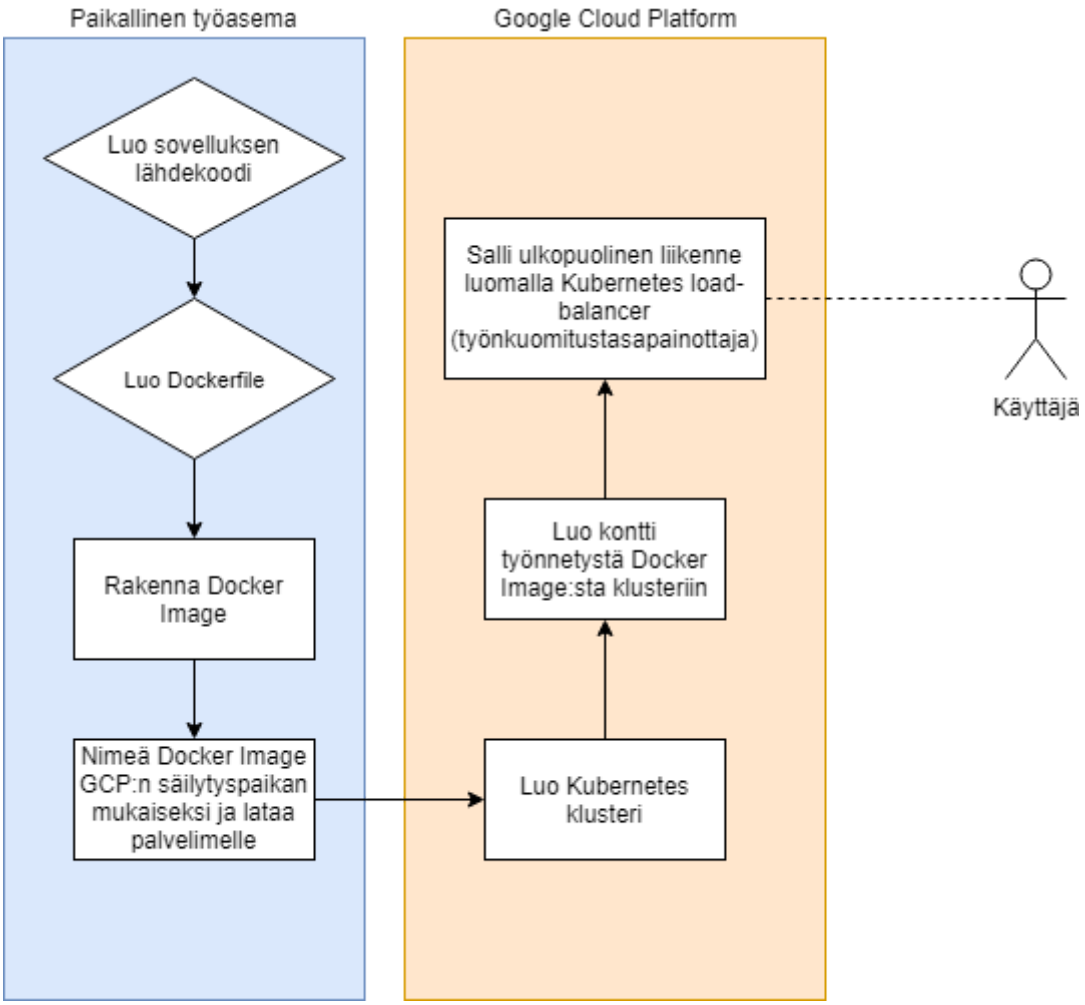
GCP:n Container Builder määritellään havaitsemaan koodi muutokset sovelluksen lähdekoodissa, joka kerta kun uusi tagi ladataan git-arkistoon. Lataaminen saa aikaan container builderin rakentamaan uuden docker-konttikuvan sekä lataamaan sen GCP:n omaan konttirekisteriin automaattisesti. Konttirekisteriin saapuneet konttikuvat havaitaan automaattisesti Spinnaker-työkalun avulla, joka puolestaan kuljettaa ne suoritus ympäristöön, suorittaa testaukset ja levittää kontit tuotantoympäristöön. Testausvaiheen jälkeen Spinnaker kuitenkin odottaa manuaalista hyväksyntää ennen konttien levittämistä tuotantoympäristöön.

3 SOVELLUKSEN DOCKER-KONTTIKULJETUS: GOOGLE CLOUD PLATFORM

Tällä hetkellä julkisesti tunnetuin ja käytetyin konttitekniikka markkinoilla on Docker, joka on “melkein” jo saavuttanut standardi aseman eri kontti tyyppien keskuudessa [7]. Docker sisältää myös oman klusterointi- ja aikataulutus työkalun docker-konteille, Docker Swarmin. Muita olemassa olevia vaihtoehtoja konttien organisointiin ja hallintaan on esimerkiksi Kubernetes, Amazon AWS ECS, Microsoft Azure Service Fabric. Jokaisella edellä mainitulla työkalulla pystytään hallinnoimaan docker-kontteja.

Kandidaatintyössä toteutettiin 2048 -pelin docker-konttikuljetus Google Cloud Platform -alustalle, joka mahdollistaa pelin paljastamisen Internetiin kenen tahansa pelattavaksi. Konttikuljetuksen päätavoitteena on havainnollistaa kuinka helposti sovellusten käyttöönotto toteutuu kyseisellä tekniikalla.

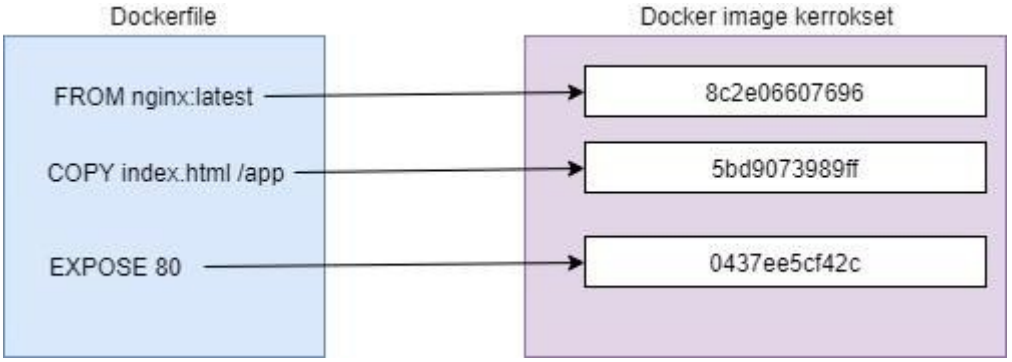
Tässä luvussa esitellään kontin toteutusprosessi ohjelmistokehittäjän näkökulmasta ja sen nostaminen ja hallinnointi Google Cloud -alustalla. Työssä käytettiin Docker Toolbox -ympäristöä kontti-kuvien rakentamiseen, Google Cloud SDK -työkaluja docker kuvien työntämiseen pilveen ja Kubernetes-moottoria konttien hallinnointiin. Itse 2048 -peli toteutettiin JavaScript -ohjelmointikielellä.



Kuva 4. Konttikuljetus korkea prosessikuvaus

3.1 Docker image: Dockerfile

Pääasiallisesti docker-kontti rakennetaan docker image:sta, joka käytännössä koostuu useista kerroksista [11]. Jokainen kerros edustaa tiettyä käskyä, jotka on määritelty tiedostoon nimeltä Dockerfile. Kuvassa 5 on havainnollistettu, miten Dockerfile tiedostosta rakennetaan docker image kerroksittain.



Kuva 5. Esimerkki: Docker imagen rakentuminen Dockerfile:stä

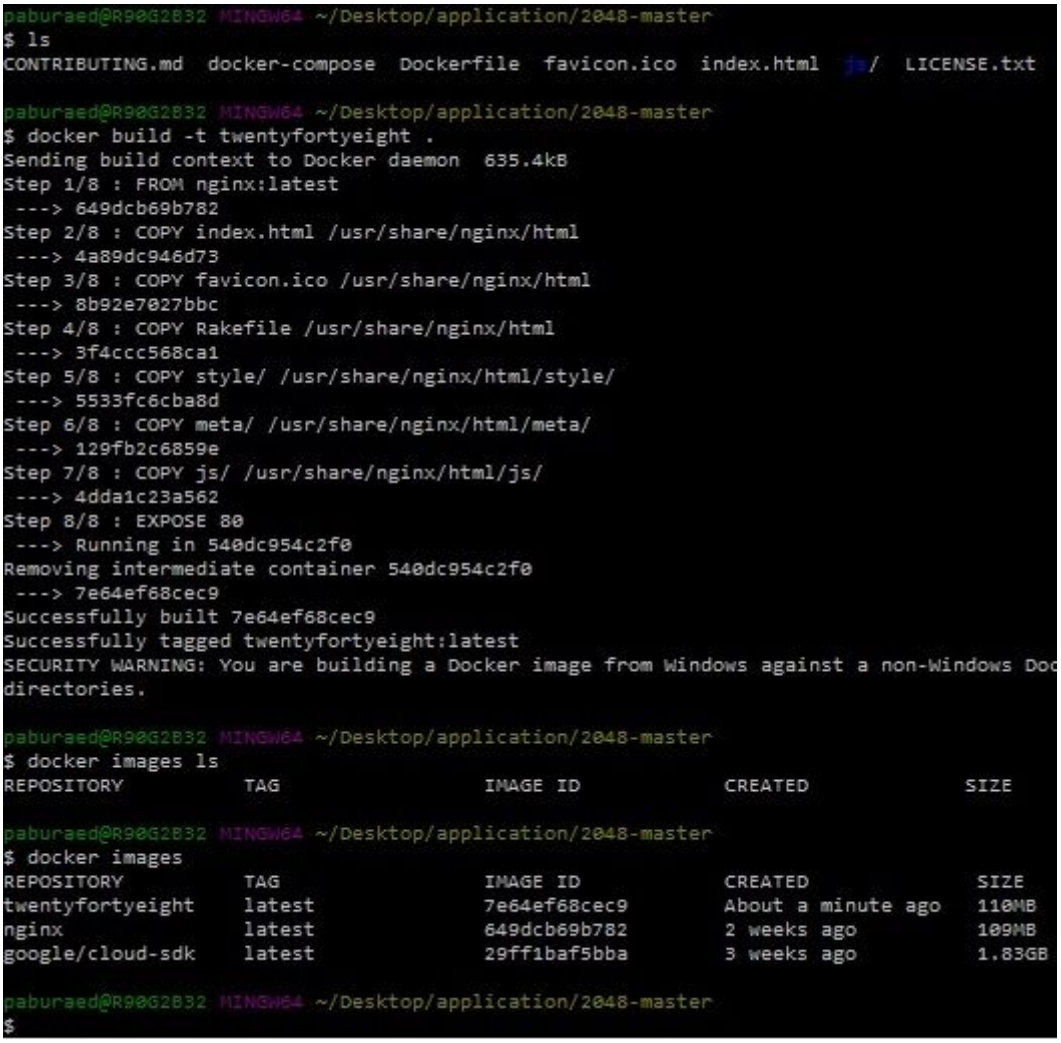
Dockerfile on käytännössä siis koodi, joka koostuu peräkkäin listatuista toimenpiteistä ja argumenteista, joiden tarkoitus on luoda uusi kuva (docker image). Perinteisesti Dockerfile alkaa **FROM** -käskyllä, josta rakentamis prosessi alkaa [12]. Kuvan esimerkki Dockerfile koostuu kolmesta käskystä. Ensimmäisellä rivillä luodaan kerros *nginx:latest* -kuvasta, joka luo nginx-verkkopalvelimen konttiin sovelluksen käyttöön. Toisella rivillä nykyisestä työhakemistosta kopioidaan **COPY** -käskyllä *index.html* -tiedosto /app hakemistoon. Kolmannella rivillä konttia ohjataan **EXPOSE** -käskyllä kuuntelemaan porttia *80,* mahdollistamaan yhteyden muodostamista sisällä olevaan prosessiin.

Kun Dockerfile on määritelty, voidaan sen avulla Docker Image muodostaa yksinkertaisesti seuraavalla syntaksilla:

docker build -t [NIMI/TAG] .

Edellä mainittu syntaksi rakentaa taustaprosessina Docker Imagen käyttäjän antamalla nimellä/tägillä.

Kandidaatintyön konttia varten luotiin 2048-pelille oma Dockerfile, joka on tarkemmin määritelty liitteessä 1. Kuvassa 6 näkyy kuinka pelille luodaan oma docker-kuva nimellä:tagilla *twentyfortyeight:latest*.



Kuva 6. Docker Image luonti 2048-pelille

Docker-kuvaa luodessa on huomioitava, että työhakemistona toimii sovelluksen kansio, ja että myös Dockerfile sijaitsee siellä.

3.1.1 Gcloud container registry: Docker Image Push

Google Cloud tarjoaa SDK-nimellä toimivia (Software Development Kit) työkaluja, joilla päästään käsiksi muun muassa Googlen laskentamoottoriin ja pilvivarastoon komentoriviltä [13]. Kyseiset työkalut helpottavat sovelluskehittäjiä automatisoimaan yksitoikkoisia arkipäiväisiä tehtäviä infrastruktuurin hallitsemiseksi. Oleellisin työssä käytetty kirjasto oli *gcloud*. Gcloud on komentorivipohjainen kirjasto, joka puhuu suoraan Googlen laskentamoottorin kanssa [14], sen avulla docker-kuva voidaan työntää Googlen konttirekisteriin. Toisaalta sanotaan, että Google Cloud SDK:n käyttö paikallisesti omalla koneella saattaa herättää turvallisuus epäilyksiä, sillä pääsyjärjestelmässä on mahdollisuus määrittää muuttujat julkisesti luettaviksi [15].

Ensimmäinen kontin docker-kuva ladattiin Googlen konttirekisteriin gcloud -työkalulla. Gcloud vaatii kirjautumisen ennen yhteyden muodostamista GCP:n kanssa. Kirjautuminen suoritetaan seuraavalla syntaksilla:

gcloud init

Syntaksin suorittaminen pyytää syöttämään google cloud käyttäjätunnuksen ja salasanan. Kirjautumisen onnistuttua gcloud pyytää asettaa projektin ja alueen/vyöhykkeen. Asetusten syötettyä gcloud on valmiina käyttöön. Yhteyden muodostettua docker-kuva voidaan parilla komennolla nostaa Googlen konttirekisteriin. Jotta uusi docker-kuva voidaan työntää rekisteriin, se tulee tägätä jonkin alueen rekisterin nimellä, GCP projekti ID:llä ja kuvan nimellä seuraavassa formaatissa:

[HOSTNAME]/[PROJECT-ID]/[IMAGE]

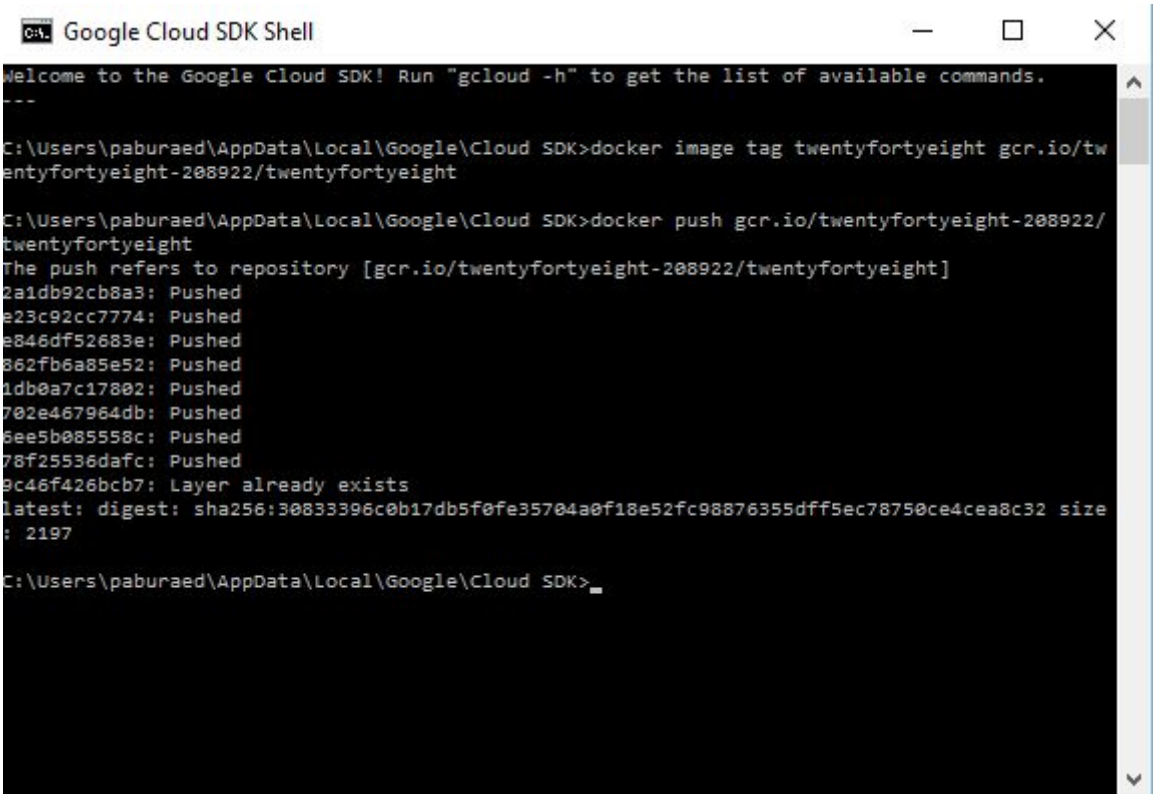
Syntaksi, joka suoritetaan gcloud -komentorivillä:

docker tag [SOURCE IMAGE] [HOSTNAME]/[PROJECT-ID]/[IMAGE]

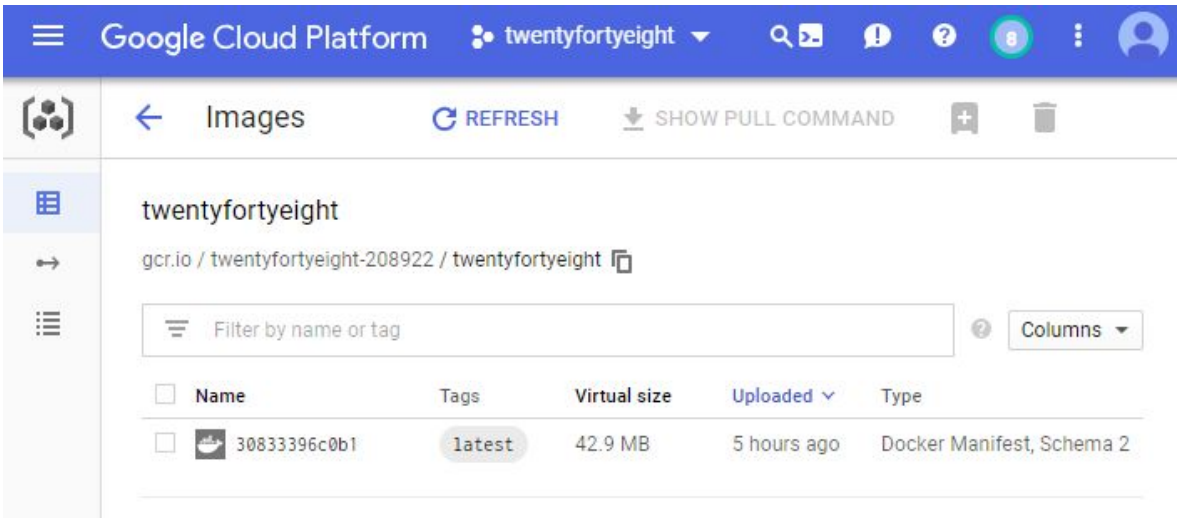
Edellä mainitun syntaksin suorittaminen tuottaa uuden tägätyn docker-kuvan, joka voidaan osoittaa oikeaan GCP projektiin komentoriviltä seuraavasti:

docker push [HOSTNAME]/[PROJECT-ID]/[IMAGE]

Kuvassa 7 näkyy kuinka docker-kuva tägätään palvelin nimellä gcr.io, joka edustaa palvelimia Yhdysvalloissa, minkä jälkeen kuva työnnettiin onnistuneesti Google Cloud konttirekisteriin. Kuvassa 8 “twentyfortyeight” -projektin Container Registry -näkymä.



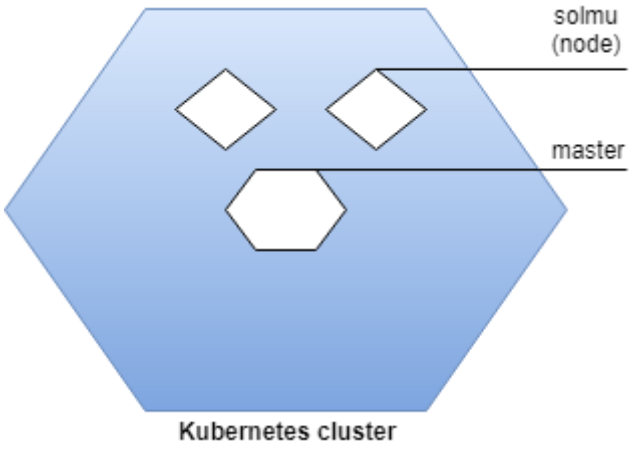
Kuva 7. Docker-kuvan lataaminen Google Container Registry gcloud -komentoriviltä



Kuva 8. Google Container Registry -näkymä

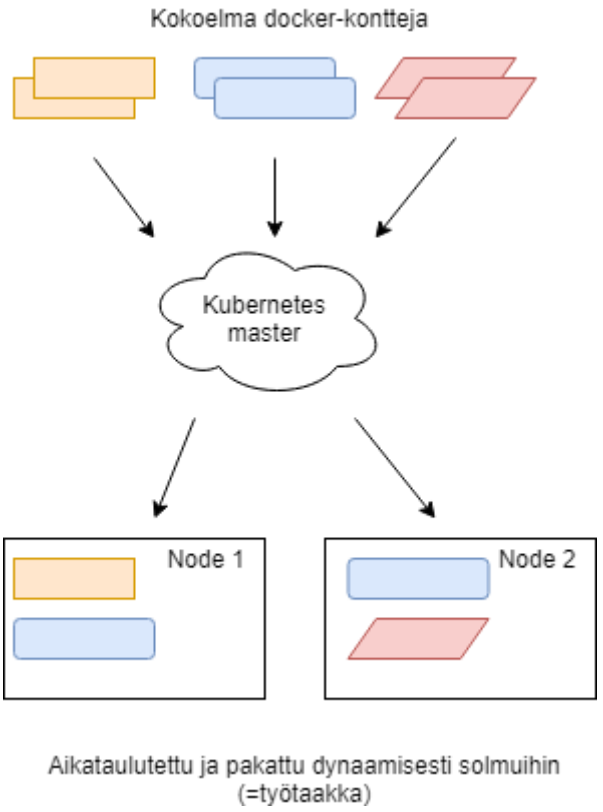
3.2 Kubernetes Engine

GCP:lla kontteja hallinnoidaan pääasiallisesti Kubernetes-moottorilla. Kubernetes tarjoaa ympäristön kontti-sovelluksien käyttöönottoon, hallintaan ja skaalaamiseen käyttäen Googlen infrastruktuuria [16]. Käyttäjät voivat muutaman klikkauksen ja määrittelyn jälkeen käynnistämään useita virtuaalikone -instansseja suorittamaan kontteja. Usean yhtenäisesti toimivien virtuaalikone -instanssien ryhmää kutsutaan klusteriksi, GCP:n tapauksessa Kubernetes **klusteriksi** (kts. kuva 9).



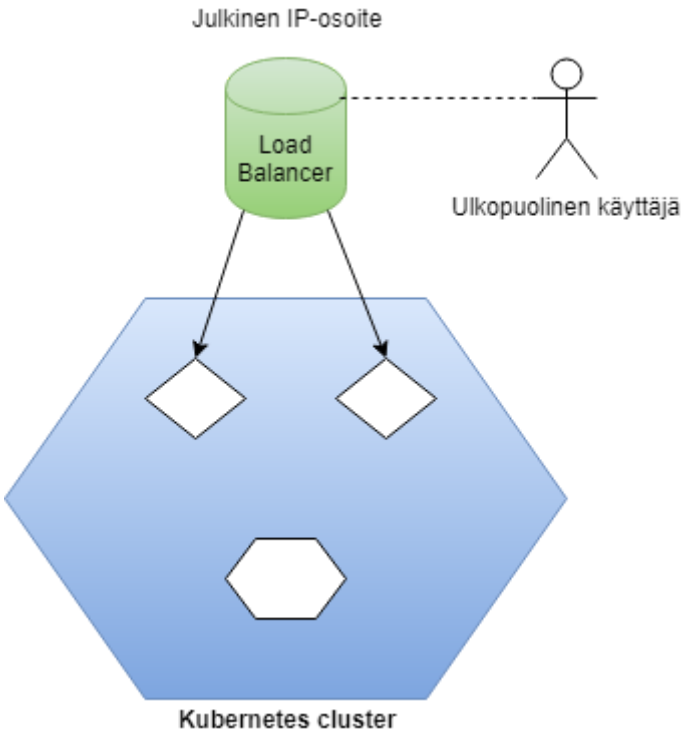
Kuva 9. Kubernetes klusteri

Kubernetes toimii kontteihin pakatuilla sovelluksilla, kuten esimerkiksi Dockerilla. Näitä aikataulutettuja ja pakattuja kontteja kutsutaan Kubernetesillä kollektiivisesti työtaakoiksi (*engl. workloads*), jotka lisätään Kubernetes klustereihin solmuina (*engl. Node*). Kuvassa 10 yleiskuva Kubernetesin toimintaperiaatteesta.



Kuva 10. Kubernetes yleiskuva konttien järjestämisestä käyttöönotettaviksi

Jotta ulkopuolinen liikenne pystyttäisiin ohjata oikeille virtuaalikoneille ja työtaakoille, täytyy GCP:n edelleenlähetys resurssit ohjata liikenne kuormitustasapainottajalle (*engl. load balancer)*. Kuormitustasapainottaja tukee runsasta liikennettä ja skaalautuvuutta, tunnistaa ja automaattisesti korjaa virheellisiä virtuaalikone -instansseja sekä ohjaa liikennettä lähimmälle virtuaalikoneille [17].

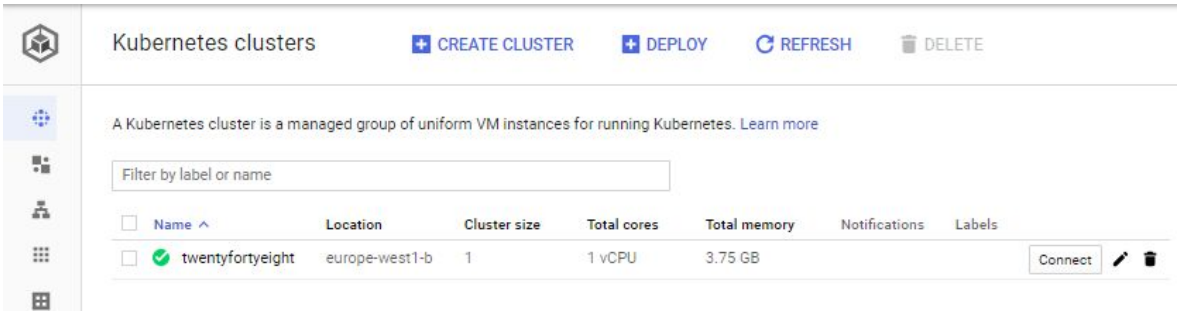


Kuva 11. Load-balancer

Työssä käytettiin Kubernetes-moottorin käyttämää avointa klusterin hallintajärjestelmää, joka tarjoaa mekanismeja muokkaamaan ja hallinnoimaan klustereita. Kyseinen hallintajärjestelmä tarjoaa muun muassa seuraavia toimintoja:

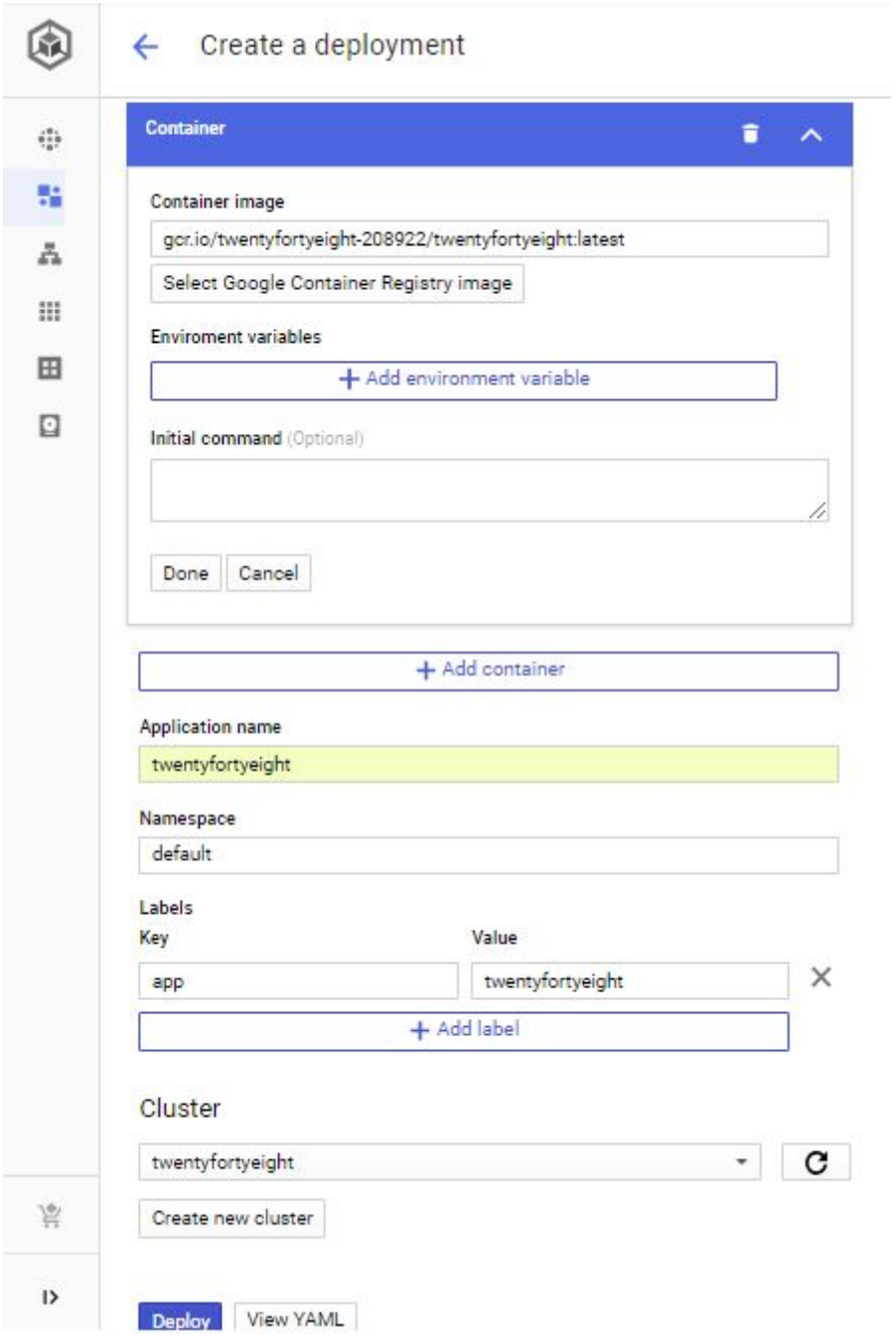
- Kuormituksen tasapainottaminen instansseille (*engl. load-balancing*):
- Automaattinen skaalaus
- Automaattinen solmun korjaus
- Lokikirjaus ja klusterin valvonta

Liitteessä 2. tarkemmin GCP:n näkymä klusterin luonnista, Kuvassa 12 GCP:n näkymä onnistuneesti luodusta klusterista nimeltä *twentyfortyeight*.

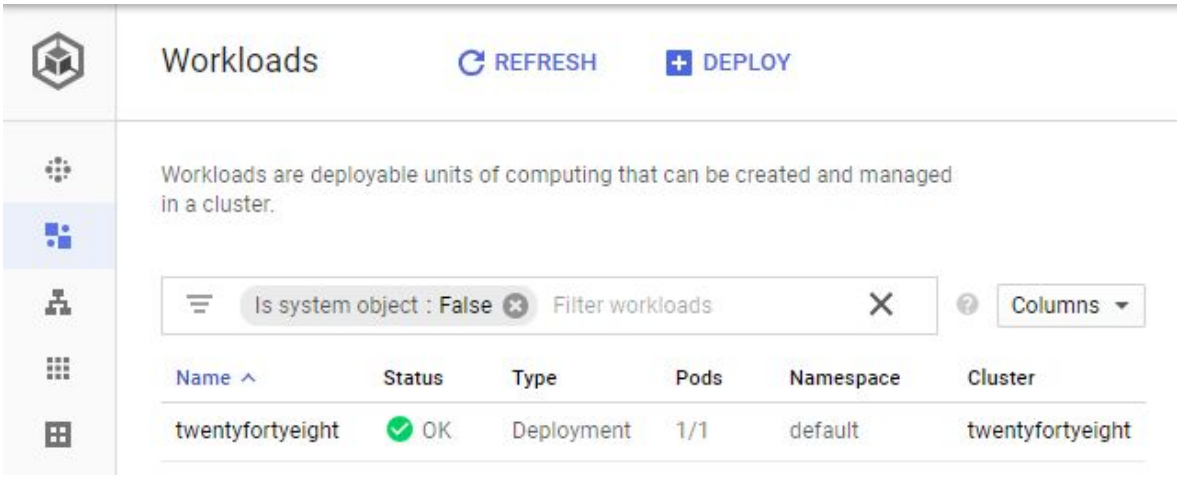


Kuva 12. GCP -näkymä onnistuneesti luodusta klusterista

Klusteriin lisättiin työtaakka aikaisemmin konttirekisteriin ladatusta *twentyfortyeight:latest* -konttikuvasta klikkaamalla deploy -painiketta (kts. kuva 12). Työtaakan (solmun) käyttöönottaminen on kokoonpano, joka määrittelee kuinka Kubernetes hallinnoi, skaalaa ja ottaa käyttöön kontin kuvan. Käyttäjä valitsee kontin-kuvan (ja mahdolliset ympäristömuuttujat), määrittelee sovelluksen nimen ja klusterin johon lisäys toteutetaan. Kuvassa 13 2048-pelille annettiin nimeksi *twentyfortyeight*, ja se lisättiin *twentyfortyeight* -klusteriin.

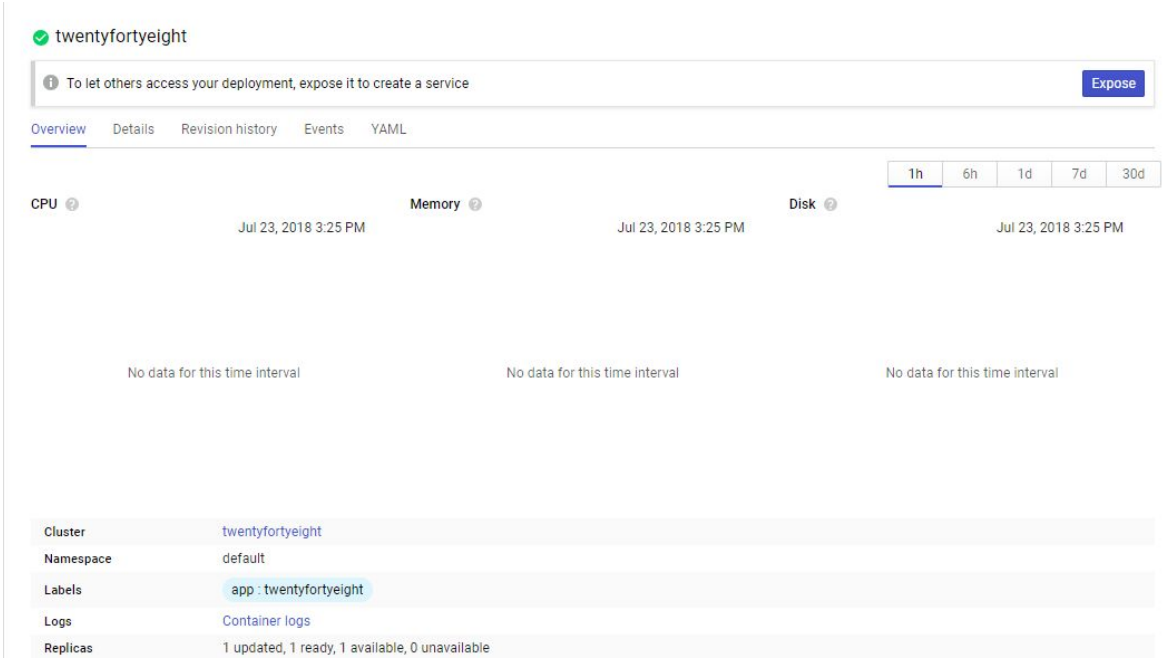


Kuva 13. Työtaakan lisääminen GCP -näkymä

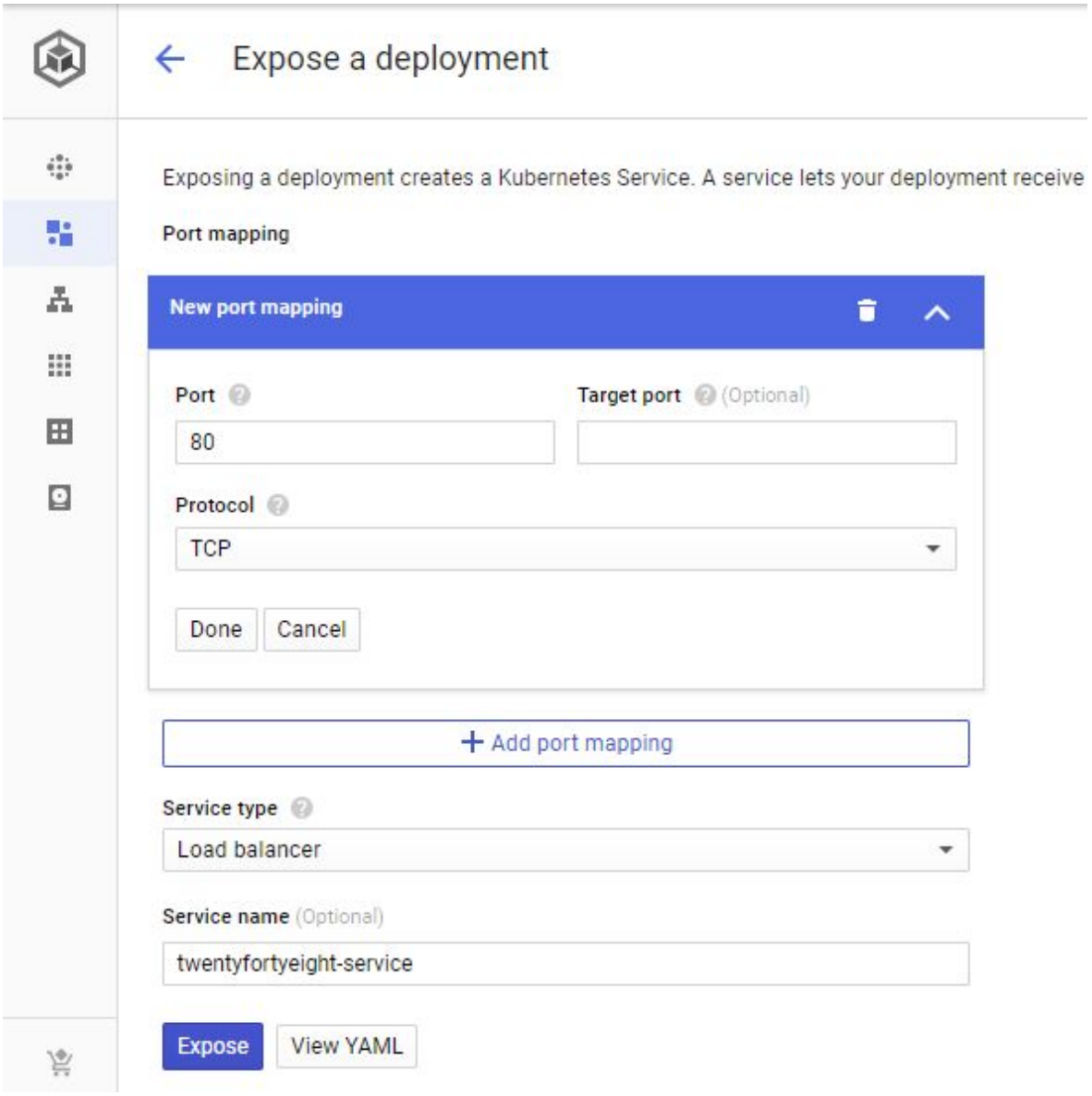


Kuva 14. GCP-näkymä onnistuneesta solmun lisäämisestä

Työtaakalle täytyi luoda load-balancer, jotta liikenne pystyttiin ohjaamaan kyseiselle solmulle. Klikkaamalla kuvan 14. työtaakan nimeä (*twentyfortyeight*) käyttäjä pääsee kuvan 15 näkymään, josta Expose -painiketta painamalla luotiin load-balancer -palvelu.

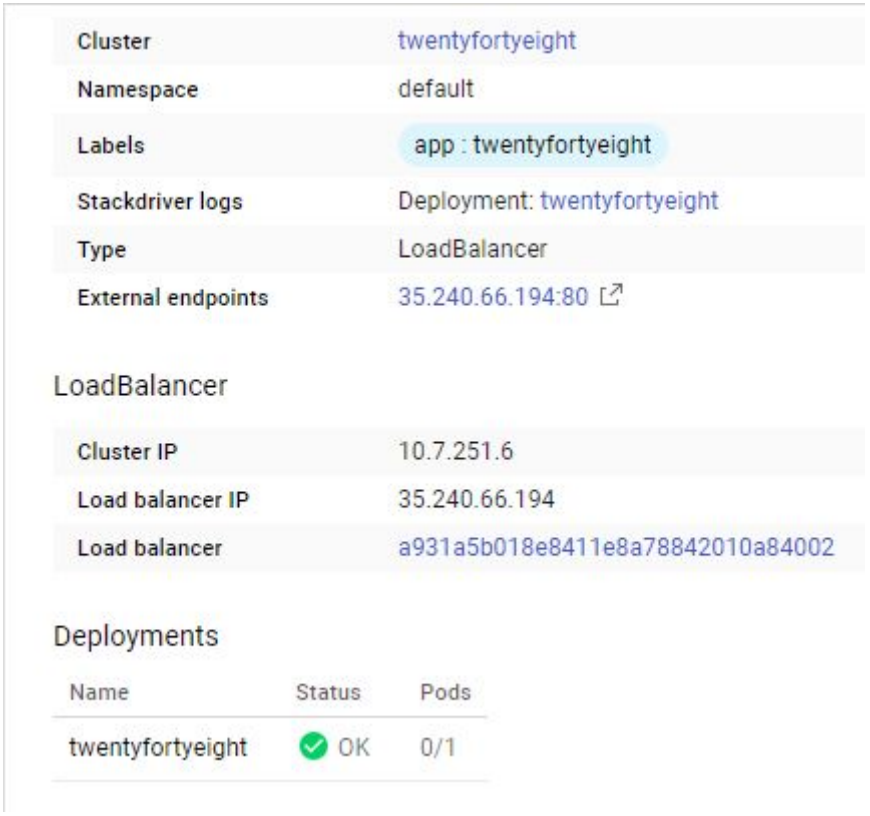


Kuva 15. GCP käyttöönottotiedot -näkymä



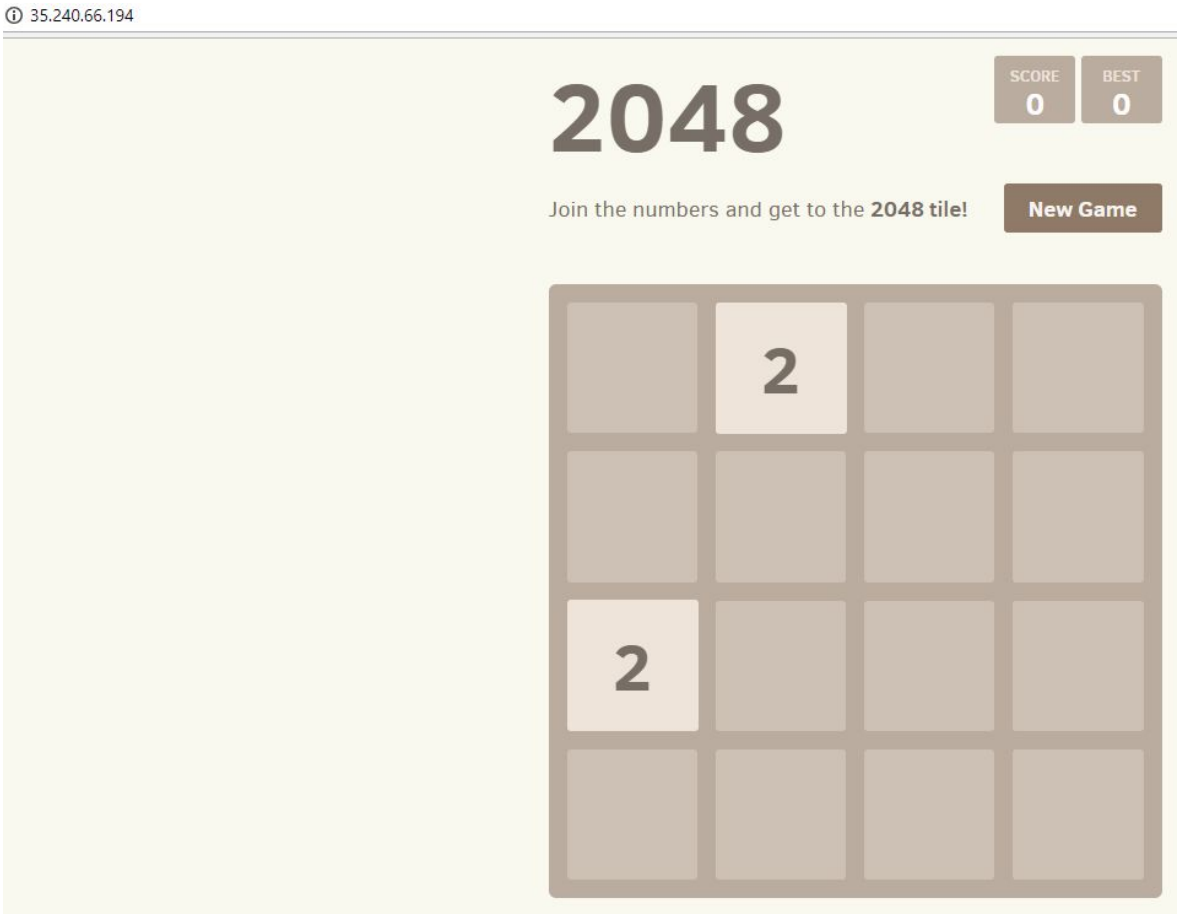
Kuva 16. Kubernetes service: Load-balancer -lisäys

Kuvassa 16 palvelulle määritettiin portti 80, joka mahdollistaa yleisesti HTTP -liikenteen. Lisäksi palvelun tyypiksi määritettiin Load Balancer -tyyppiseksi. Palvelun tarkemmat tiedot näkyvät kuvassa 17.



Kuva 17. Load Balancer -palvelun tarkemmat tiedot

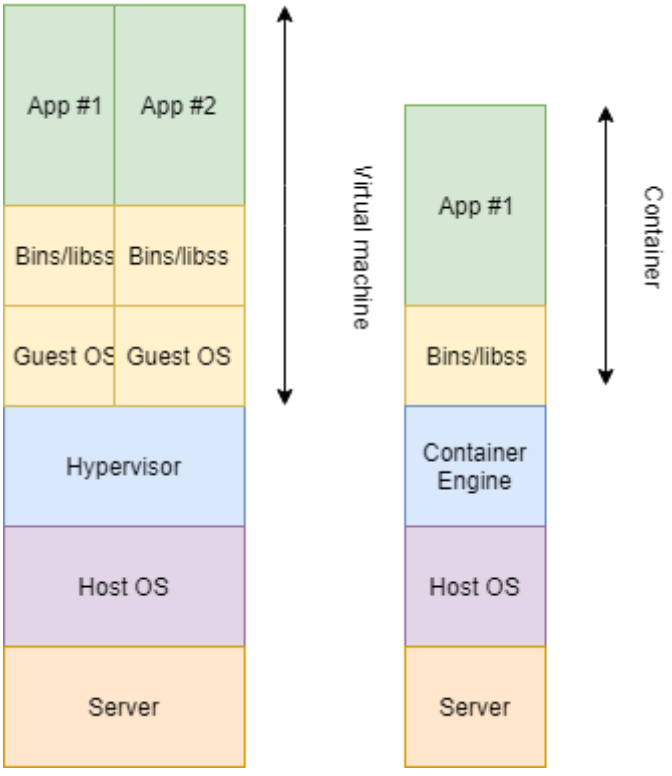
Näin ollen Load Balancer -palvelun julkinen IP-osoite on *35.240.66.194*. Kuka vain ulkopuolinen käyttäjä voi syöttää IP-osoitteen selaimeen ja päästä pelaamaan peliä.



Kuva 18. 2048-peli Internet-selaimella

4 Analyysi empiirisistä aiemmista olevista ratkaisuista

Ennen konttiratkaisuja, IT-yritykset tyypillisesti käyttivät virtuaalikoneita, joiden avulla virtualisointi toteutettiin. Virtuaalikoneet mahdollisti usean sovelluksen suorittamisen samalla systeemillä. Kehitysosastot hyötyivät tästä tekniikasta myös merkittävästi, sillä se vapautti palvelimilta tilaa uudelleenmääritykseen. Tällä lähestymistavalla oli kuitenkin omat haasteensa. Kuvassa 19 näkyy oleellinen ero virtuaalikoneiden ja konttien arkkitehtuurien välillä.



Kuva 19. Virtuaalikoneet vs Sovelluskontit -arkkitehtuurit

Virtuaalikoneilla on siis erillinen käyttöjärjestelmä, mikä lisää muistia ja varastointia. Virtuaalikoneiden raskas luonne luo monimutkaisuutta kaikkiin ohjelmistokehityksen elinkaaren vaiheisiin - kehityksestä ja testauksesta aina tuotantoon asti.

Blekingenin teknillisessä korkeakoulussa tutkittiin virtuaalikoneiden ja konttien eroja suorituskyvyn, skaalautuvuuden ja käyttäjän eristämisen suhteen [18]. Simon Vestmanin tutkimuksen tavoitteena oli vertailla näiden kahden ratkaisun etuja ja haasteita, jotta voitaisiin määrittää kumpi vaihtoehto sopii paremmin sovelluksen tarjoamiseen pilvisovellus -alustalla. Vestmanin empiirinen tutkimus perustui pääasiassa fyysisten resurssien käytön ja hallinnan analysointiin.

Tutkimuksessa selvisi, että docker-kontit varaavat muistia tarpeenmukaisesti, kun taas virtuaalikoneet varaavat muistia sen mukaan, mitä niille on määritelty luomis hetkellä. Näin ollen virtuaalikoneet vaativat merkittävästi enemmän resursseja käyttöönottoon. Suorituskyvyn suhteen molemmat ratkaisut olivat lähes tasavertaisia, missä docker-kontit suorittivat hieman nopeammin CPU-toimintoja. Virtuaalikoneet puolestaan käyttivät päämuistia nopeammin. HTTP-pyyntöjen käsittelynopeus oli virtuaalisessa ympäristössä nopeampaa kuin kontti ympäristössä, mutta suurempien tiedostojen siirtäminen verkon välityksellä oli taas nopeampaa kontti ympäristössä. Liikenteen määrän kasvaessa, virtuaalikoneet suorittivat tehokkaammin HTTP-pyyntöjä, mutta vaativat kuitenkin enemmän resursseja [18].

Virtuaalikoneet tarjoavat eristetyn tilan, missä instanssit ovat erillään isäntäkoneen käyttöjärjestelmästä, mikä tarkoittaa että isäntäkone ei ole tietoinen instanssien sisällä tapahtuvista operaatioista, ja toisinpäin. Kontit puolestaan ovat eristetty vain toistensa suhteen, ja isäntäkone on tietoinen mitä konttien sisällä tapahtuu [18]. Tämä tekee konteista alttiimpia hyökkäyksille.

Taulukossa 1 on vertailut virtuaalikoneiden ja konttien vahvuuksia ja heikkouksia.

vs	*Virtuaalikoneet*	*Kontit*
Vahvuudet	● Turvallinen (täysin eristetty isäntäkoneen käyttöjärjestelmästä) ● Nopeampi HTTP-pyyntöjen käsittelynopeus liikenteen kasvaessa	● Tehokkuus ● Siirrettävyys ● Nopeus ● Skaalautuvuus ● Modulaarisuus ● Suurien tiedostojen siirtäminen verkon välityksellä nopeampaa
Heikkoudet	● Ylimääräinen resurssien käyttö ● Siirrettävyys ● Hidas instanssien käynnistys	● Turvallisuus

Taulukko 1. Virtuaalikoneet vs Kontit [18], [4]

5 POHDINTA JA TULEVAISUUS

Konttiteknologioiden käyttöönotto osoittautui tulevaisuuden tekniikaksi, ja IT-alan yritykset ovat alkaneetkin jo siirtämään pikkuhiljaa toimintaansa pilveen. Konttitekniikan etuna on juuri se, että sillä säästetään merkittävästi aikaa ja rahaa verrattuna aiemmin käytettyihin virtuaalikoneisiin. Kuitenkin sanotaan että kontit eivät ole tulleet korvaamaan täysin virtuaalikoneita, sillä konteilla on omat heikkoutensa, kuten turvallisuus. Suuret yritykset tyypillisesti toimivat hyvin säännellyissä ympäristöissä, joissa joudutaan välttämättömästi arvostamaan tietoturvaa, ja näin ollen tietty käyttötapaus pyrkii ennemminkin määrittelemään täytyykö virtuaalikoneita käyttää konttien yhteydessä vai ei.

Nykypäivänä kehittäjät ovat entistä enemmän paineen alla tuottamaan laadukkaampia ohjelmistoja alhaisemmilla kustannuksilla yhä nopeammin, joten konttien ja jatkuvan toimituksen periaatteet ovat olleet lähes pakko sulauttaa ohjelmistokehityksen elinkaaren eri vaiheisiin. Konttitekniikka on mahdollistanut kustannusten optimointia sekä jatkuvan toimittamisen asiakkaiden nopeasti muuttuvien tarpeiden tyydyttämiseksi. Lisäksi jatkuva toimittaminen on helpottanut tuotteiden räätälöinnin ja kohdentamisen tietyille käyttäjäryhmille.

Yksi keskeisimmistä konttien mahdollisuuksista on artefaktien siirrettävyys. Kun sovellus on pakattu konttiin, sen lähettäminen esimerkiksi laadunhallinnan käsittelyyn on vaivatonta. Laadunhallinnan henkilöstön ei tarvitse miettiä minkälaisia päivityksiä tai asennuksia suoritusympäristö vaatii, laitteiston tulee ainoastaan tukea docker-ympäristöä suoriutuakseen. Näin ollen turhaa aikaa ei kulu ylimääräisiin toimenpiteisiin, ja ohjelmistokehittäjien tuottavuus paranee. Lisäksi konttitekniikan tuoma modulaarisuus tekee monimutkaisesta sovelluksesta hallittavan.

Virtuaalikoneet ja kontit ajavat samaa asiaa hieman erilailla. Niillä on eroja eri attribuuttien suhteen, joten niiden käyttö riippuu tilanteesta. Virtuaalikoneiden käyttö sopii paremmin

tilanteisiin, joissa turvallisuus tai verkkopalvelimen suorituskyky on prioriteetti. Lisäksi tilanteet, jotka vaativat raskasta kuormitusta pienillä määrillä instansseja, voidaan ratkaistaan paremmin virtuaalikoneilla.

Kandidaatintyön tehdyn konttikuljetuksen toteutus oli hyvin yksinkertainen. Toteutuksen aikana heräsi kysymyksiä, siitä miten kuljetus määriteltäisiin jos sovellus olisi monimutkaisempi. Sovelluksilla tyypillisesti voi olla eri komponentteja (esim. käyttöliittymä, tietokanta, controller), jotka sijoitetaan omina palveluina kontteihin, jolloin näiden välinen verkostoituminen tulee toteuttaa. Jatkokehitystä työstä olisi tutustua miten konttien välinen verkostoituminen toimii.

6 YHTEENVETO

Työssä tutkittiin mihin konttiratkaisut perustuu, ja mitä mahdollisuuksia ja haasteita ne tuo mukanaan. Teorian kautta tarkasteltiin mitä jatkuva toimittaminen on ja miten dockeria hyödynnetään toimittamisessa. Konttiteknologiat perustuvat aiemmin käytettyyn virtualisointi tekniikkaan, joten virtuaalikoneiden toimintaa analysoitiin ja verrattiin konttiratkaisun vahvuuksiin ja heikkouksiin. Kontit lupaavat kolmea tärkeää mahdollisuutta: tehokkuutta, modulaarisuutta ja siirrettävyyttä. Kääntöpuolena virtuaalikoneet tarjoavat vakaampaa tietoturvaa, joten virtuaalikoneiden käyttöä ei voida hylätä täysin. Tietyissä tilanteissa pyritäänkin ennemminkin määrittelemään täytyykö virtuaalikoneita käyttää konttien yhteydessä vai ei.

Työssä toteutettiin 2048 -pelin konttikuljetus, joka on hyvin yksinkertainen prosessi sovelluksen pakkaamisesta konttiin ja sen lataamisesta pilvisovellus -alustalle. Kuljetus toteutettiin Docker Toolbox -ympäristöllä, Google SDK -työkaluilla ja se nostettiin Google Cloud Platform -pilvipalveluun. Sovelluksen konttikuljetuksen tarkoituksena oli demonstroida miten helposti sovellukset voidaan ottaa käyttöön kyseistä tekniikkaa hyödyntäen. Pääpiirteittäin ensin määritellään Dockerfile -tiedosto, josta rakennetaan Docker Image. Docker-kuva (Docker Image) nimetään GCP:n konttirekisterin mukaiseksi ja työnnetään sinne. Tämän jälkeen GCP:lla luodaan Kubernetes klusteri, johon luodaan työtaakka lisäämällä työnnetystä docker-kuvasta kontti. Lopuksi luodaan load-balancer -palvelu sallimaan ulkopuolinen liikenne yhdistämään konttiin.

Lopuksi työssä pohditaan mikä on seuraava askel konttisovellusten kehityksessä. Työn toteutus oli yhden kontin sovellus, mutta käytännössä sovellukset tyypillisesti rakennetaan monesta eri kontista, jotka kommunikoivat keskenään.

LÄHTEET

1. Martin, J., Kandasamy, A. and Chandrasekaran, K. (2018). Exploring the support for high performance applications in the container runtime environment. [online] Hcis-journal.springeropen.com. Available at: https://hcis-journal.springeropen.com/track/pdf/10.1186/s13673-017-0124-3 [Accessed 24 Jul. 2018].
2. VMWare. (2018). Virtualization Technology & Virtual Machine Software: What is Virtualization?. [online] Available at: https://www.vmware.com/fi/solutions/virtualization.html [Accessed 25 Jul. 2018].
3. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and linux containers. In: 2015 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, New York, pp 171–172
4. Kasireddy, P. (2018). A Beginner-Friendly Introduction to Containers, VMs and Docker. [online] freeCodeCamp. Available at: https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b [Accessed 16 Jul. 2018].
5. Cito, J., Schermann, G., Wittern, J., Leitner, P., Zumberi, S. and Gall, H. (2018). An Empirical Analysis of the Docker Container Ecosystem on GitHub. [online] Peerj.com. Available at: https://peerj.com/preprints/2905.pdf [Accessed 25 Jul. 2018].
6. Azure.microsoft.com. (2018). What is PaaS? Platform as a Service | Microsoft Azure. [online] Available at: https://azure.microsoft.com/en-us/overview/what-is-paas/ [Accessed 25 Jul. 2018].
7. Wähner, K. (2018). The Container Landscape: Docker Alternatives, Orchestration, and Implications for Microservices. [online] InfoQ. Available at: https://www.infoq.com/articles/container-landscape-2016 [Accessed 16 Jul. 2018].

8. Reeder, T. (2018). Why and How to Use Docker for Development – Travis on Docker – Medium. [online] Medium. Available at: https://medium.com/travis-on-docker/why-and-how-to-use-docker-for-development-a156c1de3b24 [Accessed 16 Jul. 2018].
9. Amazon Web Services, Inc. (2018). What is Continuous Delivery? – Amazon Web Services. [online] Available at: https://aws.amazon.com/devops/continuous-delivery/ [Accessed 25 Jul. 2018].
10. Google Cloud. (2018). Continuous Delivery Pipelines with Spinnaker and Kubernetes Engine | Solutions | Google Cloud. [online] Available at: https://cloud.google.com/solutions/continuous-delivery-spinnaker-kubernetes-engine [Accessed 31 Jul. 2018].
11. Docker Documentation. (2018). About images, containers, and storage drivers. [online] Available at: https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/ [Accessed 18 Jul. 2018].
12. Tezer, O. (2018). Docker Explained: Using Dockerfiles to Automate Building of Images | DigitalOcean. [online] Digitalocean.com. Available at: https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images [Accessed 18 Jul. 2018].
13. Google Cloud. (2018). Cloud SDK | Google Cloud. [online] Available at: https://cloud.google.com/sdk/ [Accessed 18 Jul. 2018].
14. Khan, K. (2018). How does the Google Cloud SDK differ from the Google Cloud Client Libraries?. [online] Quora. Available at: https://www.quora.com/How-does-the-Google-Cloud-SDK-differ-from-the-Google-Cloud-Client-Libraries [Accessed 18 Jul. 2018].
15. Putrevu, A. (2018). Getting started with Google Cloud SDK – Mindorks – Medium. [online] Medium. Available at: https://medium.com/mindorks/getting-started-with-google-cloud-sdk-40e806c07460 [Accessed 18 Jul. 2018].

16. Google Cloud. (2018). Kubernetes Engine Overview | Kubernetes Engine | Google Cloud. [online] Available at: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview [Accessed 22 Jul. 2018].
17. Google Cloud. (2018). Load Balancing and Scaling | Compute Engine Documentation | Google Cloud. [online] Available at: https://cloud.google.com/compute/docs/load-balancing-and-autoscaling [Accessed 22 Jul. 2018].
18. Vestman, S. (2018). Cloud application platform - Virtualization vs Containerization - A comparison between application containers and virtual machines. [online] Bth.diva-portal.org. Available at: http://bth.diva-portal.org/smash/get/diva2:1112069/FULLTEXT01.pdf [Accessed 31 Jul. 2018].

LIITE 1. Dockerfile: 2048-peli

```
#Peruskuva verkkopalvelimelle
FROM nginx:latest

#Kopioi tiedostot ja hakemistot sovellukselle

COPY index.html /usr/share/nginx/html
COPY favicon.ico /usr/share/nginx/html
COPY Rakefile /usr/share/nginx/html
COPY style/ /usr/share/nginx/html/style/
COPY meta/ /usr/share/nginx/html/meta/
COPY js/ /usr/share/nginx/html/js/

#Osoita kontti oletusportille 80

EXPOSE 80
```