Atte Söderlund

Kontit ja virtuaalipalvelimet pilvipalvelussa

Tietotekniikan Pro gradu -tutkielma

8. syyskuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Atte Söderlund

Yhteystiedot: atte902@gmail.com

Ohjaaja: Vesa Lappalainen, Antti-Jussi Lakanen

Työn nimi: Kontit ja virtuaalipalvelimet pilvipalvelussa

Title in English: Containers and virtual machine servers in the cloud

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 87+0

Tiivistelmä: Tässä tutkielmassa luodaan suunnittelututkimusta hyödyntäen testialusta erilaisille pilvipalveluille. Testialustan tarkoituksena on saada tietoja pilvipalvelun hinnasta ja skaalautuvuudesta. Tutkimuksessa ovat mukana Azure, AWS ja Google Cloud, ja testialustaan kuuluu chat-konttisovellus sekä jokaisen pilvipalvelulle tarvittavat resurssien luomiseen tarvittavat komennot. Testialustan avulla voidaan helposti testata pilvipavelun hinta ja mahdolliset piilokulut. Aihe on kiinnostava, sillä yritykset ovat kasvavissa määrin siirtymässä käyttämään pilvipalveluita ja konttisovelluksia. Tutkimuksen mukaan yksikään pilvipalvelu ei ole toistaan parempi kaikissa osa-alueissa, ja että jatkotutkimus testien automatisointiin olisi tarpeen, jotta kattavampia testejä olisi mahdollista suorittaa.

Avainsanat: Pilvi, Azure, AWS, Google Cloud, Ohjelmistokontti

Abstract: This thesis manages to develop test bench for different kinds of cloud providers using design research. With the test bench it is possible to get more information about the cloud provider prices and scalabilities. Research includes cloud providers Azure, AWS and Google Cloud. The test bench consists of chat container app and creation commands for every recourse needed for every cloud provider included in the research. With the test bench you can easily test cloud provider prices and if there is any hidden costs. This research is important, because companies are more and more moving to cloud and to using software containers. Research revealed that there is no one cloud provided that would be best in eve-

ry category and it would be important to develop automatic tests for more comprehensive testing.

Keywords: Cloud, Azure, AWS, Google Cloud, Container

Termiluettelo

Ohjelmistokontti: Ohjelmistokontti eli lyhyemmin kontti on virtuaalinen ympäristö, joka luodaan tarvittavista riippuvuuksista, ja jossa voidaan ajaa ohjelmaa. Kontteja ajetaan käyttöjärjestelmän päällä eli ne käyttävät isäntäjärjestelmän kerneliä. Tämän vuoksi niitä voidaan luoda ja lopettaa nopeasti tarpeen mukaan.

Docker: Eräs suosituimmista konttityökaluista. Dockeria voidaan ajaa eri käyttöjärjestelmissä ja kontin sisällä voi nykyään olla myös Windows-käyttöjärjestelmä.

Pilvipalvelu: Pilvipalveluita on monia erilaisia, mutta suosituimmat julkiset pilvipalvelut ovat nykyään pitkälti samanlaisia. Pilvipalvelut tarjoavat alustaa ohjelmille, jotta niitä voidaan ajaa ilman omia konesaleja tai palvelimia. Pilvipalvelut usein tarjoavat myös erilaisia tallennusratkaisuja.

Virtuaalikone: Virtuaalikone jäljittelee oikeaa fyysistä tietokonetta ja sen komponentteja. Virtuaalikonetta voidaan käyttää kuten oikeaa palvelinta ja virtuaalikoneita voi olla yhdellä fyysisellä koneella monia samaan aikaan käynnissä.

Skaalautuvuus: Skaalautuvuus tarkoittaa sitä, miten nopeasti resurssi skaalautuu. Skaalautumisella tarkoitetaan uusien instanssien lisäämistä ja poistamista. Pilvipalveluilla on erilaisia skaalausasetuksia, joilla määritetään, miten uusia instansseja voidaan luoda automaattisesti.

Kuormituksen tasaaja: Kuormituksen tasaajaa (engl. load balancer) tarvitaan, kun instansseja on käytössä enemmän kuin yksi. Kuormituksen tasaaja jakaa tulevan Internet-liikenteen instansseille tasaisesti.

Resurssi: Pilvipalvelun palvelu eli vaikka virtuaalikone tai PaaS-resurssi, kuten AWS:n *Beanstalk*.

Instanssi: Yksi resurssin yksikkö. Esimerkiksi yksittäinen virtuaalikone.

Kuviot

Taulukot

Sisältö

1 Johdanto

Tässä tutkimuksessa pyritään löytämään testialustaa pilvipalveluille, jonka avulla erilaisia pilvipalveluita voidaan helposti testata. Testialusta testaa pilvipalvelun hintaa ja skaalautuvuutta chat-konttisovelluksella.

Tavallisesti pilvipalvelut ilmoittavat hintoja siten, että niistä voi olla vaikea laskea todellisia kuluja. Vaikka pilvipalveluilla on yleensä myös omia laskureita, ei voida olla varmoja, tuleeko hintaan lisäksi jotain piilokuluja, jos ei ole laskurissa ottanut tiettyä asiaa huomioon.

Hinta ei kuitenkaan kerro kaikkea pilvipalvelusta. Jos ajatellaan pilvipalvelun etua normaaliin, omaan tai vuokrattavaan palvelimeen, pilvipalvelussa voidaan ottaa käyttöön ja poistaa käytöstä uusia palvelimia helposti ja nopeasti. Siten voidaan vastata ruuhkapiikkeihin paremmin. Se miten uusia instansseja luodaan, kutsutaan skaalaukseksi. Skaalaukselle voidaan asettaa erilaisia ehtoja, joiden mukaan instansseja lisätään ja poistetaan. Skaalausta tutkitaan, jotta tiedetään pilvipalvelun taso ja voidaan verrata sitä hintaan.

Aihe on kiinnostava, koska yritykset siirtyvät pilveen kasvavissa määrin (Gupta, Seetharaman ja Raj 2013). Gupta, Seetharaman ja Raj (2013) mukaan pienet yritykset haluavat pilveen, koska pilvi tarjoaa turvaa ja pienempiä kustannuksia. Pilvessä kiinnostaa myös käytön helppous ja se, että niihin päästään käsiksi monella laitteella. Cito ym. (2015) toteaa, että pilvipalveluita on tutkittu paljon palveluita tarjoavien näkökulmasta, mutta vähemmän palveluita käyttävien näkökulmaan. Tämä tutkimus keskittyy pilvipalveluita käyttävien näkökulmaan. Pilvipalvelun käyttäjä ei ole loppukäyttäjä, vaan sovelluksen omistaja, joka haluaa käyttää pilvipalvelua sovelluksen ylläpitämiseen.

Idea tähän tutkimukseen tulee Adafy Oy:ltä, jolla on itsellään vahva kokemus Azuren käytöstä, mutta ei muista pilvipalveluista. Myös Adafy on siirtymässä vahvasti käyttämään ohjelmistokontteja, joten aihe on heille tärkeä.

Tutkimuksessa luodaan testialusta vertailemalla virtuaalikoneita ja *Platform as a Service* (PaaS) -resursseja pilvipalveluissa. Aihetta on tutkittu aikaisemmin ainakin Joy (2015) tutkimuksessa, jossa testattiin konttien suorituskykyä ja skaalautuvuutta ilman pilvipalvelua.

Joyn mukaan kontit ovat tehokkaampia. Sen sijaan konttien ja virtuaalikoneiden vertailua ei ole tutkittu pilvessä.

Ohjelmistokontit vaikuttavat tulevaisuuden teknologialta, joka on syrjäyttämässä virtuaalikoneet pilvipalveluissa. Tutkimuskysymyksenä on, ovatko ohjelmistokontit parempia pilvipalveluiden virtuaalikoneilla vai PaaS-resursseilla.

Tutkimus suoritettiin kolmella tunnetulla pilvipalvelulla: Microsoft Azure, Amazon AWS ja Google Cloud. Ohjelmistokontit, tai lyhyesti kontit, ovat kasvattaneet suosiotaan ja pilvipalvelut nykyään tukevat ohjelmistokontteja suoraan PaaS-resursseilla eli konttien vieminen pilveen on tehty helpoksi.

Pilvipalveluista valittiin testiin sellaiset, jotka ovat suosittuja, julkisia, ja joilla on riittävästi ominaisuuksia tätä tutkimusta varten. Pilvipalvelussa on oltava työkalut kontin ajamiseen PaaS-resurssina. Lisäksi pilvipalveluun on saatava virtuaalikoneita. Tutkimukseen valitut pilvipalvelut tukevat näitä ja osassa voi jopa asettaa virtuaalikoneen automaattisesti käynnistämään kontin.

Pilvipalvelut kehittyvät nopeasti ja monet niistä kopioivat toistensa ominaisuudet nopeasti omakseen. Pilvipalveluiden hinnat vaihtelevat paljon keskenään. Toiset pilvipalvelut suosivat tiettyjä tekniikoita, esimerkiksi ohjelmointikieliä ja jopa hylkivät toisia. Tämän takia yritysten ja muiden pilvipalveluiden käyttäjien voi olla vaikea valita, mitä pilvipalvelua käyttää. Siksi tässä tutkimuksessa on käytetty useampaa suosittua pilvipalvelua, jolloin saadaan kattavampi tulos ja selviää, onko pilvipalveluiden välillä eroja.

Tutkimusmetodina tässä tutkimuksessa käytettiin suunnittelututkimusta, mutta vähän poikkeuksellisesti. Suunnittelututkimuksessa tuotetaan jokin lopputulos tai tuote eli artefakti. Artefaktina tässä tutkimuksessa muodostuu kokonaisuus, jolla pilvipalveluita voidaan arvioida. Tähän kokonaisuuteen kuuluvat sovellus, jolla testit suoritettiin, resurssien luominen ja itse pilvipalvelu.

Testit suoritettiin tutkimuksen aikana tähän tarkoitukseen kehitetyllä chat-sovelluksella, joka käyttää Docker-teknologiaa. Testeissä selvitettiin resurssien perushintaa ja hintaa rasituksessa, eli luodaan liikennettä palveluun. Liikennettä luomalla selvitettiin myös skaalautuvuutta.

Palvelua ruuhkautettiin, jotta saatiin se skaalautumaan ja pystyttiin mittaamaan, kuinka kauan skaalautuminen kestää.

Luvussa 2 kerrotaan pilvipalveluista, siitä miten pilvipalvelut ovat kehittyneet ja määritellään vähän, mitä pilvipalveluilla tarkoitettaan. Luvussa 3 kerrotaan ohjelmistokonteista ja virtuaalikoneista tarkemmin kirjallisuuden pohjalta. Luvussa 4 kuvataan tutkimusta varten kehitetty chat-sovellus teknisesti ja valitut pilvipalvelut dokumenttien pohjalta. Luvussa 5 kuvataan yksityiskohtaisesti resurssien luominen jokaisella pilvipalvelulla erikseen. Luvussa 6 esitellään tulokset. Lopuksi luvussa 7 tehdään yhteenveto, johtopäätökset ja pohditaan tuloksia sekä mahdollisia jatkotutkimuskohteita.

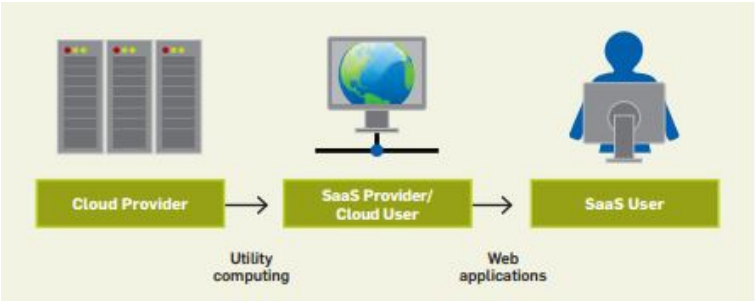
2 Pilvipalvelut

Pilvipalvelulla tarkoitetaan palvelua, joka myy resursseja pilvestä. Pilvi voidaan rakentaa erilaisilla tekniikoilla, jotta sitä on helpompi hallita, esimerkiksi fyysisiä koneita voidaan irrottaa pilvestä ja sammuttaa, vaikka huoltoa varten tai vain säästämään rahaa. Pilvi on yksinkertaistettuna joukko tietokoneita, jotka muodostavat ison palvelun, josta muut voivat ostaa osan. Tällaiset tietokonejoukot kasataan usein eri alueille ympäri maapalloa ja tätä joukkoa kutsutaan klusteriksi.

Pilvipalvelun käyttäjä voi ostaa jonkun pilvipalvelun resurssin käyttöönsä usein käyttöön perustuvalla maksulla. Resurssilla tarkoitetaan palvelua, jota pilvipalvelu tarjoaa. Tällainen voi olla esimerkiksi virtuaalipalvelin, jossa on usein erilaisia vaihtoehtoja koneen tehon suhteen. On olemassa myös resursseja, joilla voidaan pyörittää omaa sovellusta pilvessä ilman omaa palvelinta. Lisäksi on olemassa muun muassa tallennusresursseja, tietokantaresursseja ja koneoppimisresursseja.

Resurssin ohella pilvipalveluilla on paljon erilaisia omia ja lainattuja termejä, joita tässä tutkimuksessa käytetään paljon. Yleisesti kirjallisuudessa mainitaan ainakin kolmenlaisia pilvipalveluita:

- **SaaS** eli sovellus palveluna (engl. Software as a Service),
- **IaaS** eli infrastruktuuri palveluna (engl. Infrastructure as a Service) ja
- **PaaS** eli alusta palveluna (engl. Platform as a Service).



Kuvio 1. SaaS-pilvipalvelumalli (Armbrust ym. 2010)

Kuviossa 1 kuvataan SaaS-malli, jossa pilvipalvelun käyttäjä eli yritys, joka tarjoaa jotain

palvelua, julkaisee sovelluksensa pilvipalvelussapalvelussa, johon saadaan yhteys erilaisilla laitteilla, kuten matkapuhelimella tai selaimella. Loppukäyttäjälle näkyy vain sovellus ja pilvipalvelun käyttäjä hoitaa kaiken taustalla. (Dillon, Wu ja Chang 2010; Armbrust ym. 2010)

Hyvä esimerkki SaaS-palvelusta on SalesForce[1], joka tekee yrityksille muun muassa myyntianalytiikkasovelluksia. Toinen esimerkki SaaS-palvelusta on Microsoftin Office 365 eli webpohjainen Office-ohjelmisto, johon kuuluu muun muassa tekstinkäsittelysovellus Word, sähköpostisovellus Outlook ja pilvitallennuspalvelu OneDrive. Myös Googlella on samanlainen SaaS-palvelu nimellä G-suite, jossa on pitkälti samoja ominaisuuksia. (Dillon, Wu ja Chang 2010)

IaaS on käytännössä virtuaalikone pilvessä. Etuna normaaliin, fyysiseen, omistettavaan palvelimeen, on se, että uusia koneita voidaan ottaa käyttöön ja poistaa käytöstä helposti. Lisäksi koneen tehoja voidaan myös vähentää ja lisätä tarpeen mukaan. Virtuaalikoneelle voidaan esimerkiksi lisätä kovalevytilaa ja toiset pilvipalvelut tarjoavat jopa SSD-kovalevy tilaa. Lisäksi yrityksen ei tarvitse huolehtia fyysisesti laitteista ja niiden tietoturvasta. (Yang ym. 2011; Dillon, Wu ja Chang 2010)

PaaSilla voidaan ajaa koodia ilman omaa keskitettyä ympäristöä niin sanotusti hiekkalaatikossa eli eristetysti (Yang ym. 2011). PaaS on kuin toisille tarjottu SaaS eli yritys voi tarjota muille omaa sovellustaan SaaS-mallilla. PaaSin idea on siis tarjota alusta sovellukselle, johon saadaan yhteys monella eri laitteella, kuten SaaSissa. (Dillon, Wu ja Chang 2010) Tässä tutkimuksessa tutkitaan, onko PaaS halvempi kuin IaaS ohjelmistokonteilla.

Perusmallien lisäksi on muita jaotteluita. Youseff, Butrico ja Da Silva (2008) ovat jakaneet pilvipalvelut kerroksiin, jossa samassa kerroksessa ovat palvelut, jotka ovat suunnattu sovelluskehittäjille ja eri kerroksessa pilvipalvelut, jotka ovat suunnattu loppukäyttäjälle. Kerroksia on yhteensä kuusi:

- Pilvikerros,
- sovelluskerros,
- ohjelmistokerros,
- infrastruktuurikerros,

1. https://www.salesforce.com

- sovellusydinkerros ja
- komponenttikerros.

SaaS voitaisiin sijoittaa tämän luokittelun mukaan pilvikerrokseen, PaaS sovelluskerrokseen ja IaaS infrastruktuurikerrokseen.

Alemmiksi kerroksiksi jää itse pilvi eli sovellusydinkerros, joka on pilven hallintaohjelmisto, joka hallitsee klusteria. Alimmaksi jää itse komponenttikerros, josta esimerkkinä on annettu IBM:n supertietokone Kittyhawk. (Youseff, Butrico ja Da Silva 2008)

Pilvipalvelu-termi on monimutkainen määriteltävä ja määrittelyissä on pieniä eroja. Mell ja Grance (2011), jotka työskentelevät NISTin (National Institute of Standards and Technology) palveluksessa, ovat määritelleet NISTin määrityksen pilvipalvelulle julkaisussaan seuraavasti:

- **Käyttöön perustuva itsepalvelu** eli käyttäjä voi tilata laskentatehoa tai muuta palvelua automaattisesti ilman ihmiskontaktia.
- **Laaja pääsy** eli pilveen pääsee käsiksi monelta laitteelta esim. matkapuhelimelta, tabletilta tai tietokoneelta.
- **Reurssien jakaminen** eli käyttäjät saavat käyttöönsä vain osan resursseista ja jakavat loput muiden kanssa. Käyttäjä ei näe resurssien fyysistä sijaintia.
- **Nopea elastisuus** eli palvelu on skaalautuva.
- **Palvelu on oltava mitattavissa** eli palvelu itse optimoi resurssien käyttöä ja käyttäjä voi seurata ja hallita resurssien käyttöä.

Määrittelyssään Mell ja Grance (2011) eivät kertoneet, onko käyttäjä loppukäyttäjä vai muu käyttäjä, esimerkiksi yritys, jonka kautta palvelu menee loppukäyttäjälle. Dillon, Wu ja Chang (2010) tutkimuksessa käyttäjä on aina itse pilvipalvelun käyttäjä, eli vaikka yritys, joka tarjoaa SaaS-palveluna omaa sovellusta. Jotkin NISTin määrittelyn kohdista sopisivat SaaS-pilvipalveluihin, joilla käyttäjä voisi olla myös loppukäyttäjä.

Böhm ym. (2010) keräsivät eri tutkimuksissa käytettyjä pilvipalveluita kuvaavia ominaisuuksia. Tutkimus osoitti, että eniten tutkimukset kuvasivat pilvipalveluita ominaisuuksilla: palvelu, komponentit, ohjelmisto, skaalautuvuus ja Internet/verkko. Myös käyttöön perustu-

va maksu ja virtualisointi mainittiin usein.

Kuten Böhm ym. (2010) ovat tutkimuksessaan havainneet, pilvilaskenta on askel tietojenkäsittelyn historiassa. Böhm ym. (2010) itse aloittavat historian laskimista, josta seuraava askel on Turing-valmiit (engl. Turing capable) tietokoneet. Näistä siirryttiin massatuotannolla tuotettuihin keskustietokoneisiin. Siitä koneet alkoivat pienentyä minitietokoneiksi. Tietokoneiden kehitys johti myös PC:n kehittämiseen. Tietokoneiden pienentyessä kehitettiin vielä kannettavia tietokoneita ja mobiililaitteita. (Böhm ym. 2010)

Myös Internet on iso osa tietokoneiden kehitystä, joka on lähtenyt yksinkertaisesta kommunikaatioverkosta WWW:hen. Yksinkertaisesta hypertekstistä verkkopalvelut kehittyivät paremmiksi ja interaktiiviksi Javan, PHP:n ja Javascriptin avulla. Tämä mahdollisti palvelut kuten Facebook, jota pidettiin aluksi SaaS-palveluna. Verkkolaskenta (engl. grid computing) sai alkunsa jo 90-luvulla, mutta vasta 2007 sen päälle on kehitetty pilvilaskentaa. (Böhm ym. 2010)

Nykyään pilvipalveluita jaotellaan julkisuuden puolesta kolmeen kategoriaan: yksityinen, julkinen ja hybridi. (Dillon, Wu ja Chang 2010)

Pilvipalvelu voi olla yksityinen, jolloin se on usein vain yhden yrityksen käytössä. Yksityinen pilvi voidaan rakentaa yrityksen omille palvelimille ja yritys voi hallinnoida sitä itse, mutta pilvi voidaan myös ulkoistaa kolmannen osapuolen palvelimille. Etuna yksityisissä pilvissä on yksityissyys ja tietoturva. (Dillon, Wu ja Chang 2010; Fox ym. 2009)

Julkinen pilvi on muuten sama kuin yksityinen pilvi, mutta kaikille avoin ja usein maksusta. Esimerkkinä julkisista pilvipalveluista ovat AWS, Azure, Google Cloud, IBM Cloud ja SalesForce. (Dillon, Wu ja Chang 2010) Tässä tutkimuksessa käytettiin juuri julkisia pilvipalveluita.

Yksityinen pilvi ja julkinen pilvi voidaan yhdistää hybridipilveksi, jolloin jotain palveluita käytetään julkisen pilven puolella ja osaa yksityisen pilven puolella. Nämä palvelut voivat myös kommunikoida keskenään. (Dillon, Wu ja Chang 2010) Tätä tutkimusta tehdessä tapahtui merkittävä yhdistyminen pilvipalvelutarjoajien kesken. IBM osti Red Hatin, koska IBM oli kiinnostunut Red Hatin pilvipalveluista. Yhdessä nämä yritykset muodostavat suu-

rimman hybridipilvipalvelun. (“IBM ostaa itsensä mukaan tulikuumiin it-kemuihin” 2019)

Oli pilvipalvelu sitten julkinen tai yksityinen, niiden kannattavuus johtuu siitä, että pilvipalvelut voivat helposti koota yhteen palvelinkeskuksia sellaiseen paikkaa, jossa sähkö-, viilennys- ja tilakustannukset ovat matalat. (Armbrust ym. 2010) Tämä on paljon tehokkaampaa verrattuna siihen, että jokaisella on oma pienempi palvelinkeskus paikassa, joka ei ole sille sopiva.

Pilvipalvelun etu Armbrust ym. (2010) mukaan on siinä, että käyttäjän ei tarvitse maksaa etukäteen, vaan vain siitä, mitä käyttää. Tämä tarkoittaa myös sitä, että käyttöä voidaan kasvattaa helposti eli pilvi on niin sanotusti skaalautuva.

Pilvipalveluissa voidaan ajaa ohjelma tehokkaasti ja varsinkin kustannuksia säästellen, sillä Armbrust ym. (2010) mukaan on sama, ajetaanko yhtä palvelinta tuhat tuntia vai tuhatta palvelinta tunnin. Tämä on melkoinen väite, johon saamme tässä tutkimuksessa vastauksen.

Tuhat yrityksen omaa palvelinta vaatisi yritykseltä melkoisen panostuksen palvelimiin, ja jos niitä tosiaan ajettaisiin vain tunnin tai vaikka tunnin vain kerran kuussa, palvelimet ovat käyttämättöminä turha kuluerä. Pilvipalvelut laskuttavat usein käytetyistä resursseista riippuen resurssin tyypistä. Tässä tutkimuksessa tuotetaan testialusta juuri tämänkaltaisten ongelmien testaamiseen.

Pilvipalveluiden hyötyjen lisäksi niillä on myös haittoja. Yang ym. (2011) tutkivat pilvipalveluiden virtuaalikoneiden tehoja, koska artikkelin mukaan palvelut tarjoavat usein vain halutun määrän tuntemattomia prosessoreita. Tämä ei kuitenkaan kerro käyttäjälle kuin tehokkaita kyseiset prosessorit ovat. Pilvipalvelu voisi siis laskuttaa erittäin tehokkaasta prosessorista samaa hintaa kuin huonommastakin.

Vanhat prosessorit eivät ole ongelmista kuitenkaan ainoa. Muita pilvien ongelmia Dillon, Wu ja Chang (2010) tutkimuksessa listataan:

- Tietoturva,
- kulumalli,
- laskutusmalli,
- palvelutasolupaus ja

- mitä viedä pilveen.

Tietoturvauhkia pilvessä ovat ainakin datan katoaminen, kalasteluhuijaukset ja bottiverkot. Kulumallissa ongelmana on, että vaikka yritys tai muu taho saisi esimerkiksi palvelimen halvemmalla, tulee datalle suurempi hinta, koska dataa on liikuteltava enemmän. Datan hinta nousee varsinkin, jos käytössä on eri pilvipalvelutarjoajia, joiden rajapintoja joudutaan käyttämään maksusta. (Dillon, Wu ja Chang 2010)

Laskutusmallin ongelma on Dillon, Wu ja Chang (2010) mukaan ollut se, että laskutus perustuu esimerkiksi virtuaalikoneisiin, eikä komponentteihin. Palvelutasolupaus on tärkeä osa pilvipalvelua. Pilvipalveluiden on taattava, että data on turvassa ja aina saavutettavissa. Lisäksi pilvipalvelun on toimittava oletetulla varmuudella, esimerkiksi laskentateho on oltava luvatussa tehossa. Yksi ongelma on käyttäjän puolella, mitä pilveen kannattaa viedä. (Dillon, Wu ja Chang 2010)

Ongelmista eroon pääsemiseksi Durkee (2010) kuvailee tulevaisuuden pilveä, pilvi 2.0, jossa ei laskutettaisi käytetystä ajasta vaan, kuinka paljon prosessori saa aikaan eli tehokkuuden perusteella. Prosessorin tehoon perustuva laskutus on siinä mielessä tärkeää, että pilvipalvelut eivät voisi käyttää pelkästään hitaita prosessoreita tai muita komponentteja ja laskuttaa niiden käytöstä yhtä paljon kuin huippukomponenttien. Tämä auttaisi ongelmaan, josta Yang ym. (2011) tutkimuksessaan kertoivat, eli missä huonoa prosessoria laskutetaan paremman hinnalla, koska kaikki maksavat saman verran.

3 Ohjelmistokontit ja virtuaalikoneet

Ohjelmistokonttien historia alkaa *chroot*-komennosta (engl. change root), jotka esiteltiin UNIX-versiossa 7 vuonna 1979. *Chrootilla* voidaan vaihtaa juurihakemistoa. Vuonna 1988 FreeBSD jatkoi *chroottia Jaililla*. *Jaililla* voidaan jakaa käyttöjärjestelmää pienempiin osiin. Sun Solaris 10 taas toi *Zonet*, jossa virtualisoinnin avulla luotiin kontinkaltaisia alueita. Myöhemmin Linux kehitti LXC:n, jota käytettiin rajapintojen ja komentorivityökalujen kautta. (Ismail ym. 2015; Bernstein 2014)

Konttityökaluilla tarkoitan tässä tutkimuksessa teknologioita tai ohjelmia, joilla kontteja luodaan. Pahl (2015) listaa artikkelissaan seuraavia konttityökaluja:

- Docker,
- LXC,
- OpenVZ ja
- Sandboxie.

Kontit tarjoavat resurssienhallinnan ja eristyksen Linux-ympäristössä. Termi kontti perustuu laivojen kontteihin, joita käytetään tavaran säilyttämiseen ja kuljetukseen. Kontit tarjoavat geneerisen tavan eristää prosessi muusta järjestelmästä. (Dua, Raja ja Kakadia 2014) Kontti on ikään kuin virtuaalikone, mutta siinä missä virtuaalikone mallintaa tietokonetta komponenttien tasolta lähtien, kontit mallintavat vain käyttöjärjestelmää (Merkel 2014).

Jos verrataan kontteja virtuaalikoneisiin, virtuaalikoneiden ongelmana on se, että niissä on liikaa kerroksia. Ensinnäkin virtuaalikone on kopio täydestä käyttöjärjestelmästä, sen alla on oikea isäntäkoneen käyttöjärjestelmä ja alimpana on komponenttikerros. Näiden kaikkien kerrosten päällä on vasta ajettava sovellus. (Anderson 2015)

Lisäksi etuna virtuaalikoneisiin nähden on se, että LXC:hen perustuvat kontit ajetaan käyttöjärjestelmän päällä, jolloin niitä voidaan pitää päällä useita kappaleita ja niitä voi käynnistää ja sammuttaa nopeammin kuin virtuaalikoneita. (Bernstein 2014) Tämä tarkoittaa sitä, että kontit ovat vain ohjelmia käyttöjärjestelmän päällä ja niitä availlaan kuin muitakin sovelluksia. Virtuaalikoneella kestää useita minuutteja käynnistyä, koska sen on ladattava koko

käyttöjärjestelmä ja tehtävä kaikki alustukset, kuten tavallisessa käynnistyksessä tietokone tekee (Pahl 2015). Kontit itsessään vaikuttavat siis olevan nopeampia ja kevyempiä.

Koska kontteja ajetaan käyttöjärjestelmän päällä, kontit jakavat saman kernelin. LXC käyttää kernel-tason nimiavaruutta eristääkseen kontin isäntäjärjestelmästä ja täten varmistaa, että kontin pääkäyttäjä ei ole pääkäyttäjänä isäntäjärjestelmässä. (Merkel 2014)

Kontit käyttävät usein kolmea eri toimintoa hyödykseen: *chroot*, *cgroups* ja kernelin nimiavaruutta. Monet kontit käyttävät *chroot* komentoa vaihtamaan kyseisen prosessin ja sen lapsiprosessien juuripolun uuteen hakemistoon. Tätä kautta haetaan eristystä muusta järjestelmästä ja kontit eivät pääse käsiksi muiden prosessien tai isäntäjärjestelmän tiedostoihin.

Cgroups eli control groups on kernelin osajärjestelmä, jolla resursseja, kuten prosessoria, muistia ja prosesseja jaetaan. Lähes kaikki kontit käyttävät *cgroupia* resurssinhallintaan. *Cgroupilla* voidaan myös rajoittaa prosessien resurssinkäyttöä. (Dua, Raja ja Kakadia 2014)

Kernelin nimiavaruudella pyritään luomaan lisää rajoja konttien välillä eri tasoilla. Prosessi-id-nimiavaruus luo jokaiselle kontille oman prosessitunnuksen. Verkkonimiavaruus mahdollistaa sen, että jokaisella kontilla voi olla oma verkkoartefaktinsa, kuten reititystaulu, IP-taulu ja rajapintasilmukka. IPC-nimiavaruus tarjoaa rajausta erilaisissa IPC mekanismeissa, kuten viestijonoissa ja jaetuissa muistisekmenteissä. (Dua, Raja ja Kakadia 2014)

Vaikka kontit ovat tietoturvallisia, niitä ei pidetä täysin turvallisina tai ainakaan yhtä turvallisina kuin virtuaalikoneita. Nimiavaruusjaottelu on ollut Linuxissa jo vajaan kymmenen vuotta. Cgroupit ovat olleet olemassa vielä kauemmin. Kuitenkin kaikki ajetaan samassa kernelissä, joten jos se kaatuu, kaikki kontit kaatuvat. (Merkel 2014)

Kontit voivat kuitenkin ratkaista erään ongelman pilvipalveluissa. Pilvipalveluiden ongelmana on tähän asti ollut se, että kun valitsee yhden pilvipalvelutarjoajan, on lukittuna valintaan (Dillon, Wu ja Chang 2010). Kontit kuitenkin toimivat kaikissa pilvipalveluissa, joten ongelmasta tullaan pääsemään eroon.

Yleensä pilvipalvelut ovat rakennettu hypervisorin eli virtualisoinnin kautta, mutta esimerkiksi Google, IBM / Softlayer ja Joyent ovat menestyneitä pilvipalvelutarjoajia, jotka käyttävät kontteja rakenteena. Tämä tuo tehokkuutta näille pilvipalveluille, koska kontteja voi-

daan ajaa suoraan pilvipalvelun käyttöjärjestelmän päällä ilman, että välissä on virtuaalikone. (Bernstein 2014) Tutkimuksessa voidaan siis olettaa, että Google Cloud on tehokkain PaaSissa, mutta jää nähtäväksi, auttaako se skaalautumisessa.

3.1 Docker

Docker on suosituin konttityökalu ja se pohjautuu LXC:hen (Pahl 2015). Se on hieman erikoinen työkalu siksi, että sitä voidaan käyttää kuin versiohallintatyökalua (engl. repository). Dockerin rekisteriin voidaan viedä oma sovellus ja muiden sovelluksia voidaan tuoda omalle koneelle Dockerin rekisteristä. Dockerissa on palveluprosessi (engl. Dæmon), joka hallitsee ja pyörittää kontteja. Palveluprosessi on API-rajapinta, johon saadaan yhteys komentoriviltä tai työpöytäsovelluksella. (Merkel 2014) Kannattaa huomata, että Dockerin rekisteriä ei ole pakko käyttää vaan kehittäjällä voi olla lokaali palveluprosessi, jossa ajaa vain omaa konttiaan, mutta pilvipalveluiden kanssa on usein käytettävä Dockerin rekisteriä tai jotain muuta rekisteriä, josta kontti haetaan.

Docker toimii pitkälti tavallisen Linux-käyttöjärjestelmän mukaan. Perinteisesti Linux käynnistyy siten, että se ottaa käyttöönsä tiedostojärjestelmän vain lukuoikeuksilla ja tarkistaa tiedostojärjestelmän eheyden ennen kuin vaihtaa päälle kirjoitusoikeuden. Docker aloittaa samalla tavalla, mutta sen sijaan, että annettaisiin kirjoitusoikeus tiedostojärjestelmään, lisätäänkin uusi lisätiedostojärjestelmä. (Pahl 2015)

Docker voidaan ajatella kerroksina. Jos Docker-konttiin halutaan asentaa lisäksi vaikka PHP, se lisätään kerroksena konttiin, jota voidaan vain lukea. Jos PHP olisi jo asennettu, Docker huomaa sen ja ei tee mitään. Sovelluskin on yksi kerros. Jos sovellukseen tulee muutos, Docker osaa muuttaa vain sitä kerrosta, jolloin koko konttia ei tarvitse muuttaa. Docker-konttiin voidaan halutessa asentaa LAMP-pino tai vaikka Apache-palvelin. Tämä joustavuus tekee Dockerista tehokkaan työkalun. (Anderson 2015; Pahl 2015)

Docker-kontti käyttää imagea, joka rakennetaan DockerFile-tiedoston mukaan. DockerFilessä on lueteltu komentoja, joilla ympäristö asennetaan. Komennot ovat käytännössä `apt-get`-komentoja. (Bernstein 2014) Komennoista muodostetaan aikaisemmin mainittuja kerroksia, joita ei välttämättä jouduta rakentamaan uudestaan, jos jotain muutetaan. Vaikka missään ei

asiasta mainittu, tässä tutkimuksessa näytti siltä, että jos muokattu rivi sijaitsi alussa, kaikki jouduttiin tekemään uudestaan.

Imagen lisäksi Dockerissa on muitakin hyviä ominaisuuksia. Dua, Raja ja Kakadia (2014) listaavat Dockerin tärkeimmät ominaisuudet:

- **Prosessi** - Jokaiselle kontille luodaan uniikki prosessitunnus ja yksityinen IP.
- **Resurssien eristäminen** - Käyttämällä *cgrouppeja* ja nimiavaruuksia Linuxin konttikonseptista eristetään resurssit isäntäkoneen ja muiden konttien resursseista.
- **Verkon eristäminen** - Pohjautuu LXC:hen. Jokaisella kontilla on oma verkko ja kontit voivat ottaa yhteyttä isäntäkoneeseen tai muihin kontteihin vain ulkoverkon yli.
- **Tiedostojärjestelmän eristäminen** - Käyttää myös LXC:n ominaisuuksia. Kontilla ei ole pääsyä muualle kuin omalle alueelleen tiedostojärjestelmässä.
- **Konttien elinkaari** - jota hallitaan palveluprosessin ja komentotyökalun kautta.
- **Kontin tila** - Dockerissa on mahdollista tallentaa ja palauttaa kontin tiloja.

Dockerilla voidaan ajaa kuitenkin useampaa prosessia. DockerFileen määritellään komento, joka toimii kontin aloituspisteenä (engl. entrypoint). Komennossa voidaan suorittaa scripti, joka käynnistää useampia prosesseja. Tämä ei kuitenkaan ole suositeltua, sillä prosessien sammuttamisesta on huolehdittava tällöin itse. (“Run multiple services in a container” 2019)

3.2 Virtuaalikoneet

Virtuaalikoneita on ollut jo 60-luvulta asti ensin idean ja sitten toteutuksen tasolla. Virtuaalikoneiden etuna pidetään monipuolisuutta, siirrettävyyttä ja turvallisuutta (Chen ja Noble 2001). Virtuaalikoneella tarkoitetaan virtualisoitua tietokonetta.

J. E. Smith ja Ravi Nair (2005) kuvailevat artikkelissaan, että virtualisoinnin hyöty tulee siitä, että sen avulla voidaan virtualisoida erilaisia prosessoreita ja saada ohjelmia toimimaan tietokoneella, johon ne eivät ole tarkoitettu. Käytännössä Windows-käyttöjärjestelmälle kehitettyjä ohjelmia voidaan ajaa Linux-käyttöjärjestelmässä, jos virtualisoidaan Windows-käyttöjärjestelmää. Artikkelin mukaan prosessorit on suunniteltu toteuttamaan tietty rajapinta ja tiettyä rajapintaa käyttävät ohjelmat toimivat vain prosessorissa, joka kyseisen ra-

japinnan toteuttaa. Virtualisoinnilla päästään yli tästä ongelmasta, sillä voimme simuloida prosessoria, joka toteuttaa tuon ohjelman käyttämän rajapinnan.

Samanlainen virtualisointi voidaan toteuttaa kaikille tietokoneen osille, jolloin päästään komponenttien asettamista rajoituksista (J. E. Smith ja Ravi Nair 2005). Tällaisella komponenttien virtualisoinnilla on etuina turvallisuus ja liikuteltavuus. Turvallisuutta tuo se, että virtuaalikone on eriytettynä sitä ajavasta isäntäkoneesta ja isäntäkoneen ei tarvitse luottaa virtuaalikoneen käyttöjärjestelmään vaan se luottaa ainoastaan virtuaalikoneen imageen. Lisäksi etuna fyysiseen laitteeseen virtuaalikoneilla on se, että niitä voi muokata helposti, koska ne ovat tehty ohjelmoimalla. Myös koska ne ovat ohjelmoitu, voidaan myös muokata virtuaalikoneen tilaa helposti. Tiloja voidaan tallentaa, kloonata, salata, siirtää ja palauttaa. Nämä edut tekevät virtuaalikoneesta hyvän työkalun tutkimuksille. Tutkimuksien toistettavuuden takia, tiloja on usein tallennettava ja palautettava, jotta tutkimuksista saadaan toistettavia. (Chen ja Noble 2001)

Virtuaalikoneiden haittana voidaan pitää suorituskykyä, koska virtuaalikone lisää vain virtualisoidun kerroksen, joka on sovelluksen ja suorittavan komponentin välissä. Välikerrokselle toimitetut kutsut pitää muuttaa isäntäkoneelle ymmärrettävään muotoon ja suorittaa. Toisena haittana on ainakin abstraktio, joka virtuaalikoneen ja isäntäkoneen välillä vallitsee. Virtuaalikoneella ei ole tietoa isäntäkoneesta ja tämän tiedostojärjestelmästä. Lisäksi yksi virtuaalikone ei välttämättä ole tietoinen toisista virtuaalikoneista. (Chen ja Noble 2001)

Virtualisointia voidaan käyttää myös korkean tason ohjelmointikielissä, kuten Javassa tehdään. Javahan toimii siten, että Java on optimoitu virtualisointia varten ja sitä voidaan ajaa Javan omassa virtuaaliympäristössä. (J. Smith ja R. Nair 2005)

Ennen pilveä oli jo käsite verkkolaskemisesta (engl. grid computing), jolla 90-luvun puolessa välissä tarkoitettiin teknologioita, joilla kuluttaja voisi hankkia laskentatehoa tilauksesta (Foster ym. 2008). Kuulostaa siltä, mitä pilvipalvelut nykyisin tarjoavat. Foster ym. (2008) väittävät tutkimuksessaan, että pilvilaskenta on kehittynyt juuri verkkolaskemisesta juuri siksi, että niillä on niin paljon yhteistä. Heidän mukaansa pilvilaskenta ja verkkolaskenta eivät kuitenkaan ole täysin samoja asioita.

Pilvessä virtualisointia on käytettävä, koska siten pilvi voidaan sulauttaa yhteen. Pilvi koos-

tuu useasta palvelimesta, joita voidaan hallita vähentämällä tai lisäämällä fyysisiä palvelimia. Se on tärkeää, koska palvelinten keskimääräinen käyttöaste voi olla vain 5–20 %, mutta käyttöpiikit voivat olla jopa kymmenenkertaisia. Palvelimelle on jätettävä varaa piikkejä varten, joten resursseja menee hukkaan osan ajasta. (Armbrust ym. 2010; Xiao, Song ja Chen 2013) Barroso ja Hölzle (2007) tutkimuksen mukana palvelinten käyttöaste on vain 10–50 %.

Pilvipalvelun ongelma on pitää riittävä määrän fyysisiä koneita päällä, jotta kaikki virtuaalikoneet voidaan pitää päällä luvatulla teholla. Mitä vähemmän fyysisiä koneita on päällä, sitä enemmän sähköä säästyy. (Xiao, Song ja Chen 2013)

Pilven tehokkuudesta on tehty paljon tutkimusta:

- Xiao, Song ja Chen (2013) tutkivat virtuaalikoneiden dynaamista allokointia, jolla voidaan säästää helposti rahaa vähentämällä fyysisten koneiden määrää.
- Andreolini ym. (2009) tutkivat virtuaalikoneiden siirtoa palvelimelta toiselle.
- Barroso ja Hölzle (2007) tutkivat pilvipalveluiden sähkön kulutusta.
- Beloglazov ja Buyya (2012) tutkivat, miten dynaaminen allokointi kannattaa tehdä, jotta säästää sähköä.
- Beloglazov ja Buyya (2010) aikaisempi tutkimus dynaamisen allokoinnin vaikutuksista sähkökulutukseen.
- Guo ym. (2012) tutkivat aikataulutuksen optimointia.
- Myös Pandey ym. (2010) tutkivat aikataulutuksen optimointia.

Näistä tutkimuksista päätellen sähkö ja dynaaminen allokointi on tärkeä osa pilvipalvelun virtuaalikoneiden hallintaa. On selvää myös, että pienillä toimilla voidaan saada suuria säästöjä, koska pilvipalvelut ovat suuria. Moni tutkimus sähkönkulutuksesta muistuttaa myös siitä, että sähkönkulutusta tehostamalla vaikutetaan hiilidioksidipäästöihin, mikä on tänä päivänä tärkeää.

Armbrust ym. (2010) listaavat pilven hyödyt käyttäjille verrattuna tavallisiin palvelinkeskuksiin:

- Loputtoman laskentatehon illuusio,

- etukäteismaksun poistaminen,
- resurssin voi ostaa vain vähäksi aikaa käyttöön,
- taloudellinen etu, koska pilvipalveluiden keskukset ovat isompia, jolloin ne skaalautuvat edullisemmin ja
- laskenta jakautuu tasaisesti eri palvelimien päälle, koska resursseja virtualisoidaan.

Virtuaalikoneiden vienti pilvipalveluihin näyttää taloudellisesti järkevältä, koska resurssit voidaan jakaa paremmin eri palvelinten välille ja samat yritykset voivat tietämättään käyttää samaa palvelinta virtuaalikoneilleen.

4 Tutkimusalusta

Tutkimuksessa käytetään suunnittelututkimusmetodia, joka Von Alan ym. (2004) mukaan koostuu artefaktin muodostuksesta, jota evaluoidaan ja parannetaan. Tämä metodi on hyvä oikean maailman ongelmien ratkomiseen, joka auttaa luonnontieteen tutkimuksen ongelmien tutkimisessa. Tämä tutkimusmetodi sopii tämän tutkimuksen tapaiseen tutkimukseen, jossa ongelma ei ole helppo, ja jonka ratkaisemiseen tarvitaan luovuutta ja innovatiivisia ratkaisuita. (Hevner ja Chatterjee 2010)

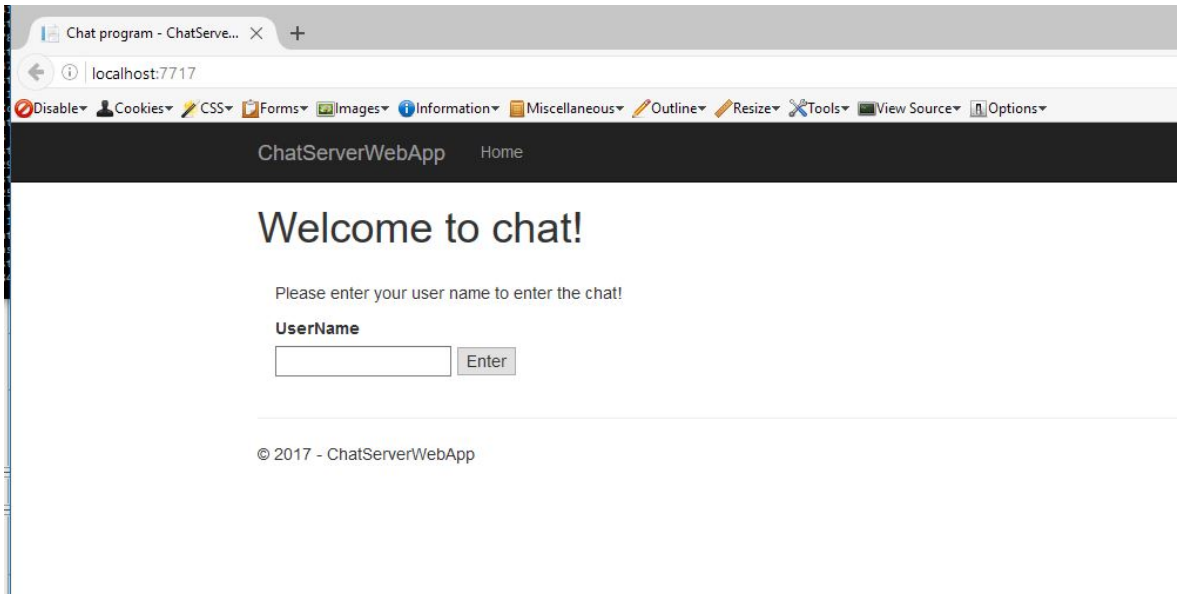
Tässä tutkimuksessa artefakti muodostuu (1) chat-sovelluksesta, jolla testejä ajetaan, (2) resursseista, joita testeissä käytetään ja (3) itse pilvipalvelusta. Artefaktia käytetään evaluoimaan pilvipalvelua, jotka tässä tutkimuksessa ovat Microsoft Azure, Google Cloud ja Amazon AWS.

Testialustaa on kehitetty tutkimuskysymyksen kautta: ovatko ohjelmistokontit parempia pilvipalveluiden virtuaalikoneilla kuin PaaS-resursseilla. Tähän pyrittiin saamaan vastaus suorittamalla hinta- ja skaalautuvuustestejä.

Tutkimusta varten kehitettiin chat-sovellus Asp.Net Core 2.1:llä, jotta sitä voidaan ajaa Dockerilla. Asp.Net on Microsoftin tuote ja sitä kehitetään Visual Studiolla, joka on myös Microsoftin tuote, joten Visual Studiossa on valmiiksi todella hyvä tuki Azurelle. Tämä ei kuitenkaan ole eduksi Azurelle, koska sovellus on Docker-kontin sisällä, jolloin pilvipalvelu pyörittää pelkästään konttia. Mikään pilvipalvelu ei tiedä, mitä kontin sisällä on. Chat-sovellus on toteutettu siten, että se ei käytä hyväkseen pilvien omia ominaisuuksia, kuten salaisuuksien säilömistä.

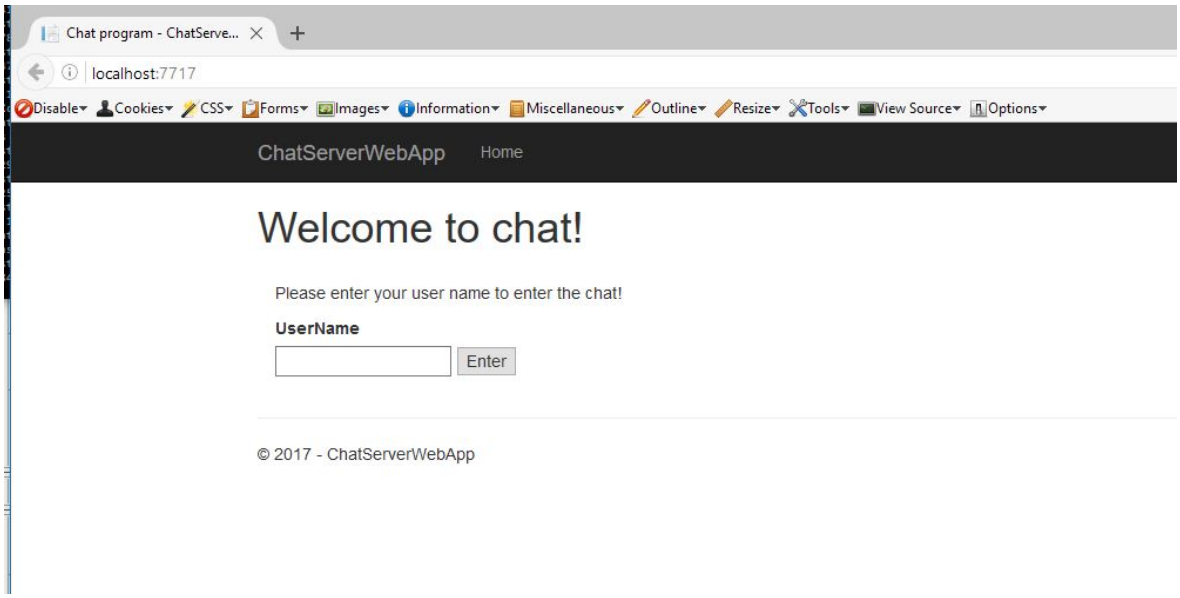
Tutkimuksessa mitattiin skaalautuvuutta eli aikaa, joka instanssien luomiseen menee konteilla ja virtuaalikoneilla. Samalla tutkittiin hintaa eri pilvipalveluiden kesken. Hinnan mittaamiseen käytettiin pilvipalveluiden omia graafeja ja mittaustietoja. Rasitustestin avulla luotiin liikennettä, jolla saadaan nostettua kuluja. Erilaisilla liikennemäärillä saatiin tutkittua sitä, miten kuormitus vaikuttaa resurssien käyttöön ja sitä kautta kuluihin.

4.1 Chat-sovelluksen käyttöliittymä



Kuvio 2. Etusivu.

Sovellus aukeaa kuvion 2 näkymään, jossa käyttäjä voi syöttää nimensä. Tämä ohjaa käyttäjän chattiin, jossa käyttäjän nimi tallennetaan sessioon. *Enter*-painike ohjaa käyttäjän chat-sivulle ja käyttäjä rekisteröidään kantaan. Käyttäjänimi annetaan osoitteessa parametrina.



Kuvio 3. Chat-ikkuna.

Kuviossa 3 on chat-näkymä, jossa on käyttäjien lähettämiä viestejä. Avaamalla toisen yhteyden *incognito*-tilassa, voi samalla koneella keskustella kahdella käyttäjällä. *SignalR* hoitaa viestien lataamisen, kun joku käyttäjä lisää uuden viestin.

4.2 Chat-sovelluksen toteutus

Ensimmäinen versio chat-sovelluksesta kirjoitettiin Asp.Net Core 2.0:lla, mutta koska sen tuki *SignalR:lle* oli vasta preview-vaiheessa ja versio 2.1 julkaistiin sopivasti, ja jossa oli tuki *SignalR:lle*, vaihdettiin sovellus käyttämään Asp.Net Core 2.1:stä. *SignalR:ää* käytetään sovelluksessa signaalien antamiseen uusista viesteistä. Sovellus käyttää tiedontallennuksessa *EntityFrameworkia*, jolla tallennetaan entiteettejä suoraan luokista kantaan. Chat-sovellusta ajetaan Docker-kontissa.

Kantana toimii PostgreSQL-kanta, joka pyörii Google Cloudin virtuaalikoneessa ja image siihen on otettu suoraan Google Cloudista. Chat-sovelluksessa on kovakoodattu IP-osoite tietokantaan, joten Google Cloudissa on varattu IP, jotta se ei vaihdu.

Git-versionhallinnasta chat-sovelluksesta on kaksi Asp.Net-projekteja yhden *solutionin* alla. Google Cloudilla pystyy ottamaan käyttöön lokipalveluita Asp.Net sovelluksen käynnistyksessä, joten Google Cloudille on oma projekti, joka on kuitenkin täysi kopio toisesta projektista. Kaikilla pilvipalveluilla jouduttiin muokkaamaan DockerFilejä, jotta sovellus toimisi.

Google Cloudin projekti on tehty Googlen omasta projektipohjasta Asp.Net-projektille, joka tulee Visual Studioon Google Cloud -lisäosan myötä. Google Cloudiin käytetään pääsääntöisesti ChatServerGoogleCloud-projektia. Projektipohjaa on muutettu, koska lisäosa on vasta kehitysvaiheessa ja se ei toiminut kovinkaan hyvin. Pohjaa muokattiin siten, että käynnistyksessä Googlen diagnostiikat käynnistetään, mutta muuten konfigurointi tehdään kuten normaalissa Asp.Net Core 2.1 projektipohjassa. Toinen ero normaaliin projektipohjaan Googlen pohjassa oli se, että se käytti npm-työkalua riippuvuuksien hallinnointiin. Työkalu otettiin pois toiminnasta, jotta chat-sovelluksen sai toimimaan Google Cloudissa.

Kontti on julkisessa Docker-rekisterissä[1], jota käytetään AWS:ssä ja Azuressa, mutta Google

1. https://cloud.docker.com/repository/docker/atte902/chatwebapp

Cloudiin oli tehtävä oma image Google Cloudin omaan rekisteriin.

4.3 Tutkimukseen valitut pilvipalvelut

Tutkimuksen pilvipalvelut valittiin Dignan (2018) artikkelin mukaan käytetyimmät tai nopeimmin kasvavimmat pilvipalvelut. Artikkeli esittelee lukuja pilvipalveluista, joista selviää, että Amazonin AWS oli käytetyin pilvipalvelu ja toisena tuli kovaa kasvuvauhtia Azure ja Azuren kanssa samaa vauhtia kasvava Google Cloud. Suosion lisäksi syy näiden palveluiden valitsemiseen oli se, että ne tukivat tarvittavia ominaisuuksia yhdenvertaisten testien tekemiseen.

4.3.1 Microsoft Azure

Azure on Microsoftin tarjoama pilvipalvelu, joka oli Dignan (2018) artikkelin mukaan toiseksi suosituin pilvipalvelu, kun mitattiin yritysten käyttöönottamia pilvipalveluita. Azuressa on paljon ominaisuuksia, joista osan saa käyttöön ilmaiseksi ja osasta joutuu maksamaan.

Azuressa pääosassa vaikuttaisi olevan *App Service*, joka on Azuren PaaS-resurssi, jolla voidaan ajaa ASP.NET-, ASP.NET Core-, Java-, Ruby-, Node.js-, PHP- tai Python-sovelluksia. Tällaisia sovelluksia *App Servicessä* kutsutaan *Web Appeiksi*. Azure osaa skaalata *Web Appeja* automaattisesti ja pilvi takaa saavutettavuuden. (“App Service documentation” 2018) Saavutettavuus on pilvipalveluiden suurin etu. Koodi vain annetaan pilveen suoritettavaksi ja käyttäjä ei tiedä, missä koodia suoritetaan, mutta pilvipalvelu osaa automaattisesti ajaa sitä vaikka useammalta palvelimelta.

Tietysti Azuresta löytyy myös virtuaalikoneet ja kuormituksen tasaajat, joilla voidaan ohjata liikennettä eri virtuaalikoneille. Azuresta yli sata erilaista palvelua muun muassa koneoppimiseen, paljon erilaisia tietokantapalveluita, konttipalveluita ja tekoälyyn liittyviä palveluita, joista moni toimii rajapinnan kautta, eli niitä voi käyttää myös Azuren ulkopuolelta.(“Azure products” 2018)

Kirjoitushetkellä Azure tarjosi käyttöön ensimmäiseksi 30 päiväksi 200 dollaria tai 170 euroa pilvipalvelun käyttöön. Tämän lisäksi vuoden ajan suosituimmat palvelut olivat ilmaisia

(“Azure pricing” 2018):

- 750h/kk Linux-virtuaalikone,
- 750h/kk Windows-virtuaalikone,
- 2x64GB levytilaa,
- 5GB *Blob Storage* -tilaa,
- 5GB tiedostotallennustilaa,
- 250GB SQL-palvelin,
- 5GB Cosmos DB -palvelin ja
- 15GB verkkokaistaa.

Kirjoitushetkellä koko ajan ilmaisia palveluita olivat (“Azure pricing” 2018):

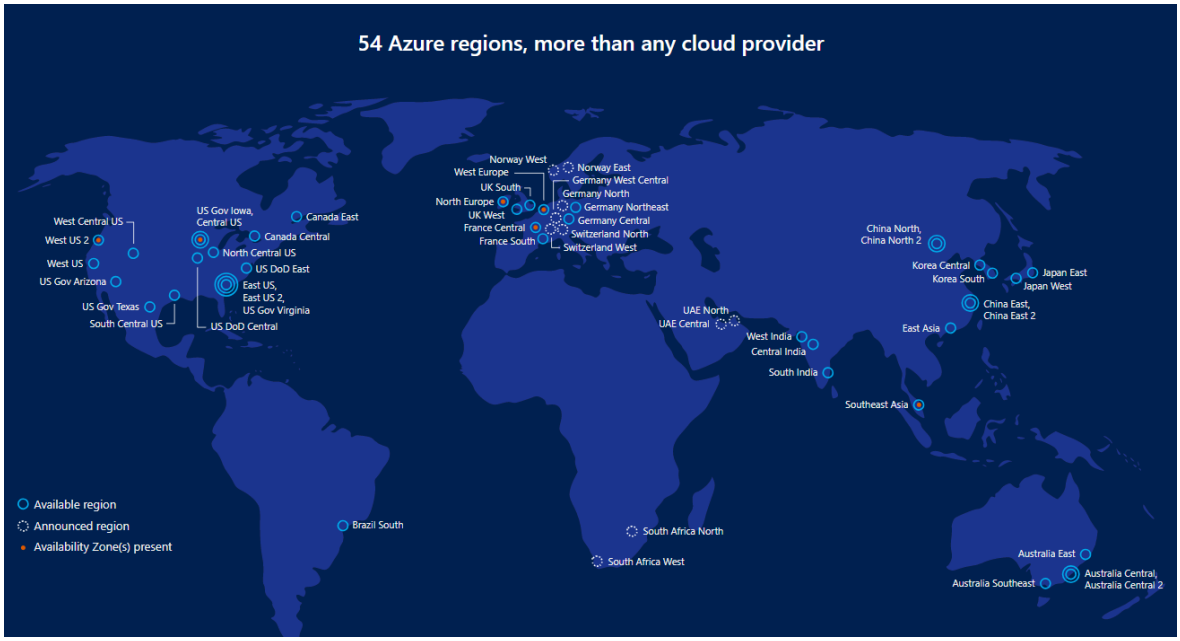
- 10 *App Serviceä*, jolla voi luoda ohjelmia mille alustalle tahansa,
- miljoona *Azure Functions* kutsua kuukaudessa,
- *Azure Kubernetes Service* ja
- *Active Directory* 500 000 objektia, joita käytetään käyttäjän tunnistamisessa.

Ilmaisilla työkaluilla pystyy hyvin pyörittämään yksinkertaisia sovelluksia vaikka testimielessä. Azurella ei kuitenkaan näyttäisi olevan *Blob Storagea* tai pientä tietokantaa ilmaisena, johon sovellus tallentaisi dataa.

Rahalla saa Azuresta paljon irti. Esimerkiksi virtuaalikoneita on tarjolla satoja erilaisia kokoonpanoja. Halvassa B-sarjassa on tarjolla yhdestä virtuaaliprosessorista kahdeksaan virtuaaliprosessoriin asti erilaisia kokoonpanoja. Hinta kasvaa 0,005 dollarista tunnissa aina 0,14 dollariin tunti. Azure mainostaa myös ND-virtuaalikonetta, jossa on 6 virtuaaliprosessoria ja 112 gigatavua muistia sekä NVIDIA Tesla P40 -näytönohjain. ND-virtuaalikone on tarkoitettu tekoälyn ja paljon laskentatehoa vaativien operaatioiden suorittamiseen, ja se maksaa yli 1 500 dollaria tunnilta. (“Azure pricing” 2018)

Virtuaalikoneita on myös mahdollista ostaa kolmeksi vuodeksi ennakkoon, jolloin saa jopa 72 % alennuksen verrattuna, että maksaa pelkästä käytöstä. *App Service* on ilmainen, mutta ilmaisella palvelulla ei saa skaalautuvuutta, sovelluksen koko on rajoitettu yhteen gigatavuun ja se ei tue omaa verkkotunnusta. *Basic*-taso *App Servicestä* tarjoaa kolme instanssia, mutta

ei autoskaalautuvuutta. *Basic*-tasolla saa myös oman verkkotunnuksen. Jotta autoskaalautuvuuden saa käyttöön, on otettava käyttöön *Standard*-taso, joka maksaa 0,10 dollaria tunnilta. (“Azure pricing” 2018)



Kuvio 4. Azuren palvelinkeskukset (“Azure homepage” 2018).

Azurella on palvelinkeskuksia joka puolella maailmaa, mutta suurin osa näyttää sijoittuvan Eurooppaan ja Pohjois-Amerikkaan, kuten kuviosta 4 nähdään.

Azure oli syyskuussa 2018 näyttävästi esillä, kun sen palvelut olivat kaatuneet maailmanlaajuisesti. “Postmortem: VSTS 4 September 2018” (2018) blogikirjoituksessa kuvataan tarkkaan, miten kaikki tapahtui. Kaikki sai alkunsa ukkosmyrskystä eteläisessä Texasissa, lähellä Azuren palvelinkeskusta. Ukkosmyrsky vioitti jäähdytysjärjestelmää ja palvelinkeskus aloitti alasajon estääkseen datan häviämisen. Valitettavinta oli, että palvelinkeskus oli niin keskeinen, että monia palveluita kaatui tämän takia, vaikka käyttäjät eivät olisikaan käyttäneet juuri kyseistä palvelinkeskusta. (“Postmortem: VSTS 4 September 2018” 2018) Pilvipalveluissa on tärkeää, että yksi solmu ei vaikuta muihin ja käyttäjien olisi mahdollista siirtyä toiseen solmuun tarpeen vaatiessa.

4.3.2 Amazon AWS

Dignan (2018) artikkelin mukaan AWS oli vielä toistaiseksi johdossa markkinaosuudessa ja johdon ennustettiin jatkuvan. Artikkelin mukaan AWS johtaa varsinkin virtuaalikoneiden määrässä.

AWS:stä löytyy satoja erilaisia palveluita liittyen tietokantaan, esineiden Internettiin, robotiikkaan, lohkoketjuihin, virtuaalitodellisuuteen ja tekoälyyn sekä muihin laskentatehoa vaativiin toimintoihin ("Cloud Products" 2018). *AWS Elastic Beanstalk* on Azuren *App Serviceä* ja Google Cloudin *App Engineä* vastaava palvelu, joka tukee Java-, .NET-, PHP-, Node.js-, Python-, Ruby-, Go- ja Docker-sovelluksia ("AWS Elastic Beanstalk" 2018). Myös tavalliset virtuaalikoneet, joita AWS:ssä kutsutaan nimellä *EC2*, ja kuormitukset tasaajat ovat käytössä ("Amazon EC2" 2018).

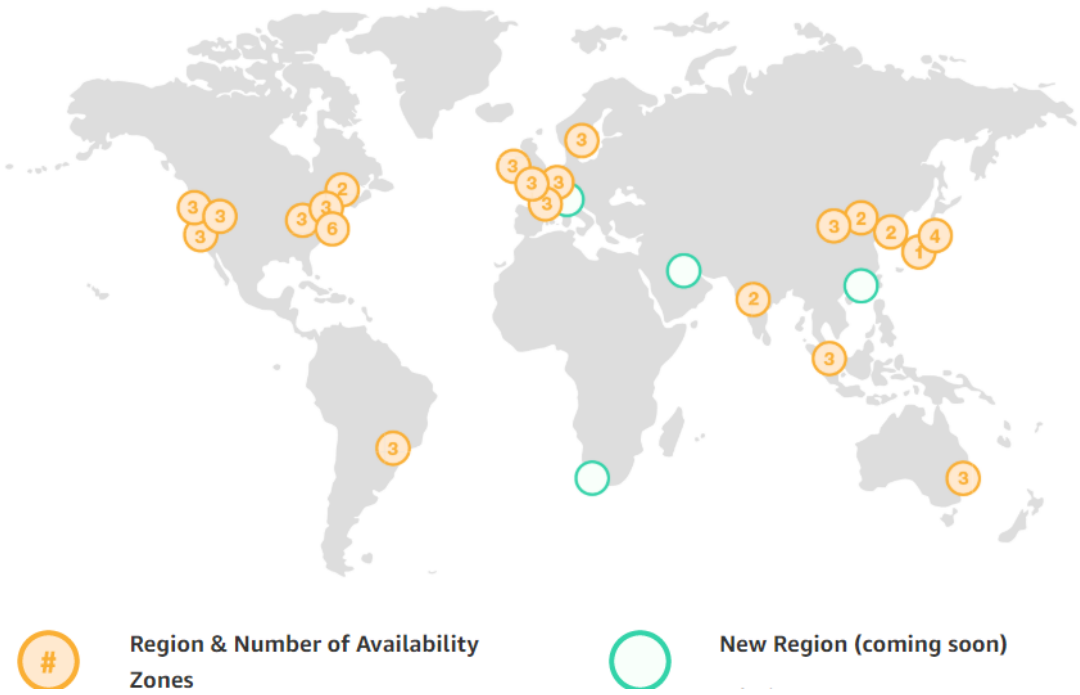
Kirjoitushetkellä AWS tarjosi muun muassa seuraavia palveluita ilmaiseksi ensimmäisen vuoden ajan ("AWS free tier" 2018):

- *Amazon EC2* -virtuaalikonepalvelu 750h kuukaudessa,
- *Amazon RDS* -relaatiotietokantapalvelu 750 tuntia kuukaudessa,
- *Amazon S3* -tallennustilaa 5 gigatavua kuukaudessa,
- *Amazon Elastic File System (EFS)* 5 gigatavua kuukaudessa ja
- *Amazon Cloud Directory* 1 gigatavu kuukaudessa.

Kirjoitushetkellä AWS:n aina ilmaiset palvelut olivat muun muassa ("AWS free tier" 2018):

- *Amazon DynamoDB* 25 gigatavua,
- miljoona *AWS Lambda* -käskyä kuukaudessa,
- *Amazon Glacier* -pitkäaikainen ja turvallinen objektivarasto 10 gigatavua ja
- *Amazon CloudWatch* monitorointiin: 10 mittaria ja 10 varoitusta.

AWS:ssä näyttää olevan paljon erilaisia tallennuspalveluita. Virtuaalikoneet ovat vain ensimmäisen vuoden ilmaisia. Lisäksi *Elastic BeanStalk* ei ole lainkaan ilmainen ("AWS free tier" 2018).



Kuvio 5. AWS:n palvelinkeskukset (“AWS Global Infrastructure” 2018).

Kuviosta 5 nähdään, että AWS-palvelinkeskukset sijoittuvat melko tasaisesti ympäri maailmaa ja uusia on tulossa myös Afrikkaan ja Lähi-itään.

AWS:ssä saa alennusta, jos varaa käyttöaikaa ennakkoon verrattuna siihen, että maksaa vain käytöstä. AWS:n mukaan näin voi, säästä jopa 75 %. Käyttäjä voi myös valita oman palvelimen pilvestä, joka ei ole muiden käytössä. (“Amazon EC2 pricing” 2018) Käyttäjällä on myös valinnanvaraa, minkälaisen virtuaalikoneen haluaa. AWS tarjoaa paljon erilaisia kokoonpanoja, joissa virtuaalisten prosessorien määrä, muistin määrä ja kovalevytila sekä -laatu vaihtelevat. Esimerkiksi laskentatehoa kaipaava voisi valita kokoonpanon, jossa on NVIDIA Tesla näytönohjain, 768 gigatavua muistia ja kaksi 900 gigatavun NVMe SSD -kovalevyä. (“Amazon EC2 Instance Types” 2018)

Saavutettavuus on yksi pilvipalveluiden tärkeimmistä ominaisuuksista. Pilvipalvelulle on tärkeää, että sillä on luotettava maine. Vuonna 2017 AWS oli näkyvästi esillä työntekijän näppäilyvirheen takia, joka kaatoi monia suuria nettisivustoja. Työntekijän piti ajaa komento, jolla poistetaan pieni määrä palvelimia, mutta kirjoitusvirheen takia komento poisti suu-

ren määrän palvelimia, joista yksi oli tärkeä osa erästä aliverkkoa, ja sen takia monet palvelut eivät toimineet. (“Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region” 2018)

4.3.3 Google Cloud

Google Cloud kasvattaa markkinaosuuttaan kovaa tahtia. Kasvun myös ennustetaan jatkuvan. AWS:ää, Azurea ja Google Cloudia voidaan pitää markkinoiden kolmena pääpelurina. (Dignan 2018)

Google Cloudissa on kymmeniä erilaisia palveluita, joista tärkeimpinä: virtuaalikoneet, *App Engine*, kuormituksen tasaaja, *Cloud functions*, *Kubernetes Engine* ja konttipalveluita (“Products and services” 2018). Googlen vastine Azuren *App Servicelle* ja AWS:n *Elastic Beanstalkille* on *App Engine*. *App Engine* tukee Node.js-, Java-, Ruby-, C#-, Go-, Python-, ja PHP-sovelluksia. *App Enginellä* voi ajaa kontteja, jotka skaalautuvat automaattisesti siten, että Google Cloud hoitaa liikenteen jakamisen ja tietoturvan. (“GOOGLE APP ENGINE” 2018)

Google Cloud tarjoaa vuoden mittaisen ilmaisen kokeilujakson, jonka aikana palvelussa on 300 dollaria käytössä (“Google Cloud Platform Free Tier” 2018). Lisäksi palvelu ei ala velottamaan senkään jälkeen ilman lupaa (“GCP Free Tier” 2018).

Kirjoitushetkellä Google Cloudin tarjoamat ilmaiset palvelut olivat muun muassa (“Google Cloud Platform Free Tier” 2018):

- *App Engine* jopa 28 instanssilla ja viiden gigatavun sovelluksella,
- skaalautuvat NoSQL-tietokanta,
- skaalautuva *f1-micro*-instanssin virtuaalikone 30 gigatavun kovalevyllä,
- *Kubernetes Engine*,
- Google Stackdriver monitorointipalvelu ja
- serverless-funktiot.

Google Cloud tarjoaa todella hyvät palvelut ilmaiseksi. Ilmaisilla työkaluilla pystyy hyvin testaamaan erilaisia sovelluksia.

Kuvio 6. Google Cloudin palvelinkeskukset (“Cloud locations” 2018).

Google Cloudilla näyttää kuvion 6 perusteella olevan kattava palvelinkeskuskattavuus, ja jopa Suomessa on keskus.

Google Cloud näyttäisi myös olevan luotettava. Muutama vuosi sitten Google Cloudilla oli muutama suuri käyttökatko, mutta vuoden 2018 aikana *App Engine* ja virtuaalikoneiden häiriöt ovat olleet pieniä, eivätkä ole aiheuttaneet juurikaan käyttökatkoja. Google Cloud lupaa virtuaalikoneiden käyttäjilleen 99,95 % saavutettavuuden, joka tarkoittaa vuodessa enintään neljän tunnin ja 23 minuutin käyttökatkoa. (“How reliable is Google Cloud?” 2018; “Google Cloud Status Dashboard” 2018)

5 Testialusta

Tässä luvussa kuvataan tutkimuksen tuloksena syntyneet artefaktit. Artefaktit ovat lopputulos, johon on päästy tutkimalla pilvipalveluita yrityksen ja erehdyksen kautta. Jokaisella pilvipalvelulla lopputulokseen on päästy hieman eri keinoin, mutta yleisesti kaikkien kohdalla on lähdetty vakioasetuksista, joita on paranneltu yrityksen ja erehdyksen kautta käyttämällä hyväksi dokumentaatiota.

Jokaista artefaktin versiota ei ole testattu yhtä kattavasti kuin viimeistä, joka on esiteltynä tässä luvussa. Olisi ollut parempi, jos testikierrokset olisi pystynyt ajamaan ja validoimaan ennen lopullista artefaktia, mutta se ei ollut tämän tutkimuksen aikataulun puitteissa mahdollista. Jokaista versiota testattiin vain kerran tai pari, jotta nähtiin, toimiko tehdyt muutokset vai ei.

Hinnan tutkimisessa on vaikea löytää halvinta mahdollista, koska hinnat voivat muuttua jopa tutkimuksen aikana. Tutkimuksessa saatiin kuitenkin hyvä kuva siitä, miten edullisesti tai kalliisti yksinkertaisen konttisovellus saadaan pilvipalveluun. Kaikilla valituilla pilvipalveluilla on hinnasto, jota vertailemalla voidaan helposti valita halvin alue, jota käytetään. Alueen lisäksi ei hintaan voidaan vaikuttaa, mutta yleensä resurssin tehon valinta voi olla ratkaisevaa. Tutkimuksessa vertaillaan myös hintaa omaan palvelimeen. Vertailusta on hyötyä yrityksille, jotka miettivät pilveen siirtoa, mikä tuo lisäarvoa artefaktille.

Koska tutkimuksessa on luotu testijärjestelmä pilvipalveluille, pystytään tutkimuksen avulla vastaamaan pilvipalveluiden ongelmiin: piilokulut, tehon ja hinnan suhde ja vastaavatko hinnat laskureita.

Skaalautuvuuden asetuksissa ei kovinkaan suurta eroa vakioasetuksiin tullut, koska prosessorin kuormaan perustuvan skaalauksen asetuksia ei välttämättä pysty muokkaamaan ollenkaan. Prosessoriin perustuva skaalaus on kuitenkin tutkimuksen kannalta paras, koska kaikki valitut pilvipalvelut tukevat sitä, jolloin tuloksia voidaan vertailla.

5.1 Testeissä käytettävät resurssit

Chat-sovellukselle suoritetaan testit seuraavilla resursseilla eri pilvipalveluissa.

- PaaS-sovelluksena:
  - Azuressa *Container Instanceilla* ja *Web Appilla*,
  - Google Cloudissa *App Enginellä* ja
  - AWS:ssä *Elastic Beanstalkilla*.
- Virtuaalikone:
  - Ubuntu:
    - skaalauksella ja
    - ilman skaalausta.
  - Core OS:
    - skaalauksella ja
    - ilman skaalausta.

Azuressa voi ajaa *App Servicessä* konttia, mutta Azureen on myös vähän aikaa sitten tullut uusi resurssi *Container Instance*, joka on lähes sama asia, mutta ei vaadi kallista *App Service*-tilausta. *Container Instance* ei kuitenkaan tue skaalautumista, joten vain hintaa voidaan tutkia sen kanssa. Tarkoituksena on tarjota kontinajoa palveluna eli PaaSina. Tätä verrataan Googlen *App Engineen* ja AWS:än *Elastic Beanstalkiin*, mutta skaalautumisen kanssa.

Ei ole mielekästä ajaa Azuressa normaalia *Asp.Net App Service*-sovellusta, koska koko palvelu on suunniteltu ja optimoitu .NET-koodin ajamiseen. *Beanstalk* tukee kyllä MSBuildia, mutta tutkimuksessa ei lähdetty tutkimaan, miten se on toteutettu. MSBuildilla käännettään .NET-koodia ajettavaan muotoon. Googlella ei pysty ajamaan .NET-ohjelmaa suoraan vaan se muutetaan aina kontiksi. AWS:llä kaikki toimii virtuaalikoneiden avulla.

Eri pilvipalveluiden välillä pyritään käyttämään samankaltaisia resursseja, jotta tuloksia voidaan vertailla. Myös skaalautuvuusasetukset valitaan samoiksi. Asetetaan prosessorin käytön keskiarvo 70 %:iin, jolloin luodaan uusi instanssi ja vastaavasti alle 30 %:ssa poistetaan instanssi, jos tämä on mahdollista. Kaikille asetetaan yksi instanssi ja maksimi-instanssimäärä neljään instanssiin.

Jotta kaikki testit kohdistuvat vain kyseisiin palveluihin kuten virtuaalikone tai PaaS-palvelu, käytetään jokaisessa samaa tietokantaa.
Tietokantana käytetään PostgreSQL, joka on pystytetty Google Cloudiin Google Cloudin marketin imagesta vakioasetuksilla.

5.2 Hinnan tutkiminen

Hintoja vertaillaan eri pilvipalveluiden kesken. Hintaa merkitään euroissa ja se ilmoitetaan muodossa euroa päivää kohden verottomana. Jokaisesta resurssista on taulukoitava perushinta eli hinta, jonka resurssi vie ilman, että sitä käytetään eli siihen ei saa kohdistua liikennettä. Tämä hinta ei saa olla resurssin luomispäivältä vaan resurssin luomisen ja testin suorittamisen välillä on oltava vähintään yksi kokonainen vuorokausi.

Perushinnan lisäksi selvitetään hintaa kuormituksen alla. Kuormitusta generoidaan *SuperBenchmarker*-sovelluksella.[1] Sovelluksella voidaan kuormittaa sovellusta tietyllä määrällä latauksia tai tietyn aikaa. Kuormitus tehdään chat-näkymään, jossa näkyy muutama chat-viesti, jotta kannasta saadaan haettua dataa.

Rasitustesti suoritetaan komennolla `sb -u "url" -c 200 -N 7200 -B -W 5`, joka generoi kaksi tuntia pyyntöjä 200 säikeellä. Rasitustestistä kirjataan pyyntöjen määrä, jotta saadaan laskettua hinta pyynnölle, joka on todella pieni, mutta auttaa vertailemaan eri pilvipalveluita.

Seuraavissa alaluvuissa on pilvipalvelu kerrallaan esitetty komennot, miten resurssit on pystytetty sekä kerrottu, miten hintaa seurataan. Jokaisella pilvipalvelulla kulujen kategorisointi tapahtuu hieman eri tavalla. Jokainen resurssi tarvitsee jotenkin kategorisoida omakseen, jotta kulut voidaan eritellä oikein. Kun kulut saadaan kategorisoitua, voidaan ladata pilvipalvelusta kulut siten, että ne kattavat päivän, jona resurssi on luotu, vähintään yhden välipäivän, josta saadaan perushinta, rasitustestipäivän ja resurssin poistamispäivän.

1. https://github.com/aliostad/SuperBenchmarker

5.2.1 Azuren hintatestit

Azuressa voi seurata *cost analysis* -palvelun kautta eri resurssien kuluja. Kulujen kategorisointi tapahtuu resurssiryhmien kautta. Resurssiryhmiä on helppo ja nopea luoda. Resurssilla on myös pakko olla oma resurssiryhmänsä. Erottelemalla resurssit omiin ryhmiinsä, nähdään helposti myös mahdolliset piilokulut.

PaaS

PaaS-testejä varten taas luodaan muutama erilainen resurssi. *Container Instance* on hyvä resurssi, jos halutaan helposti pystyttää yksi skaalautumaton kontti. *Container Instancet* tukee maksimissaan neljää ydintä WestEurope-sijainnissa. Vakiona käytössä on vain yksi ydin ja alustavien testien mukaan hinta nousee suorassa suhteessa ydinten määrään. Instanssi voidaan luoda komennolla:

```
az container create --resource-group AtenTestiPaas
--name chat-paas-container2 --image atte902/chatwebapp:testi
--dns-name-label chat-testi --ports 80 51069
--environment-variables 'ASPNETCORE_ENVIRONMENT'='Development'
--cpu 4
```

Komennossa on määritelty resurssin nimi, kontin image, DNS-nimi, portit ja ympäristömuuttuja. DNS-nimi voi olla oikeastaan ihan mitä vain, kunhan se ei ole käytössä. Portit on määritelty chat-sovelluksen käyttämään porttiin 51069 ja viimeisenä on määritelty ydinten määrä.

Skaalautuvaksi PaaS-resurssiksi tarvitaan *Web App for Containers*-resurssia, joka luo *App Service*-toteutuksen kontille. Tämä resurssi luodaan komennolla:

```
az webapp create --name chat-container-web-app --plan AtenServicePlan
--resource-group AtenTestiPaas
--deployment-container-image-name atte902/chatwebapp:testi
```

Komennossa luetellaan resurssiryhmä, nimi, *Service Plan* ja kontin image. Määritelty *Service Plan* AtenServicePlan pitää myös luoda. Tutkimuksessa se on luotu käsin ja se on S1-tasoinen eli halvin standard-taso. Standard-tasoa tarvitaan, jotta *Web App* tukee skaalautuvuutta leveyssuuntaisesti. Korkeussuuntainen skaalaus tapahtuu tasoa korottamalla. Jokai-

sella tasolla on oma Azure Compute Units (ACU) -arvonsa, joka määrittää laskentatehon. S1- ja B1-tasolla arvo on 100 ACU.

Service Planille on käytössä skaalautuvuuden säännöt:

- Instanssien vähimmäismäärä 1,
- instanssien enimmäismäärä 4,
- ylösskaalauksen raja-arvo 70 % prosessorin käytössä viiden minuutin ajalta,
- ylösskaalaus 1 virtuaalikone kerrallaan,
- alasskaalauksen raja-arvo 25 % prosessorin käytössä viiden minuutin ajalta ja
- alasskaalaus 1 virtuaalikone kerrallaan.

Cli-komentoon ei saa suoraan sovellusasetuksia mukaan, joten vielä on muutettava yhtä asetusta, jotta kontti luodaan oikeaan porttiin:

```
az webapp config appsettings set --resource-group AtenTestiPaas
--name chat-container-web-app
--settings WEBSITES_PORT=80
```

Virtuaalikoneet

Skaalautuvuuden testaamista varten on luotava *Scale Set* -resurssi Azureen. Tämä luodaan komennolla:

```
az vmss create --name chatscaleset3
--resource-group AtenTesti3
--image Canonical:UbuntuServer:18.04-LTS:latest
--vm-sku Standard_B1s
--authentication-type password
--admin-username atte --admin-password AsD123456789
--instance-count 1 --vnet-name chat-vnet
--load-balancer chat-lb
--public-ip-address chat-scale-ip
```

Komento luo *Virtual Machine Scale Setin (vmss)*, jossa on määritelty nimet kuormituksen tasaajalle, IP:lle, virtuaaliverkolle ja määritetty admin-käyttäjä. Komento luo yhden instanssin *Scale Setin*, jossa on:

- Käyttöjärjestelmä Linux Ubuntu Server 18.04 LTS,
- West Europe -sijainti, joka tulee AtenTesti3 resurssiryhmästä,
- Standard B1S, jossa on 1vcpu, 1 gigatavua muistia ja maksimi IOPS 400 ja
- käytetään Azuren hallinnoimia kovalevyjä, jotta ei tarvitse huolehtia niistä itse.

B1S tukee premium-kovalevyjä eli SSD-kovalevyjä, mutta silti Azuren mukaan sen ennakoidut kustannukset ovat pienemmät kuin normaalilla kovalevyllä eli HDD-kovalevyllä. B1S:ssä on halvimmat ennakoidut kustannukset, joten se on valittu testiin.

Seuraavaksi ajetaan komento, jolla lisätään lisäosa, joka käynnistää kontin jokaiselle uudelle instanssille:

```
az vmss extension set --publisher Microsoft.Azure.Extensions
--version 2.0 --name CustomScript
--resource-group AtenTesti2 --vmss-name chatscaleset2
--settings '{"script":"IyEvYmluL3NoCmlmIFsgYHN1ZG8gc3
lzdGVtY3RsIGlzLWFjdGl2ZSBkb2NrZXJgICE9ICJhY3RpdmUiIF0K
dGhlbgoJYXB0IHVwZGF0ZQoJYXB0IGluc3RhbGwgLXkgZG9ja2VyLm
lvCmZpCmlmIFsgJChkb2NrZXIgaW5zcGVjdCAtZiAne3suU3RhdGUu
UnVubmluZ319JyBjaGF0KSA9ICJmYWxzZSIgXQp0aGVuCglkb2NrZX
Igc3RhcnQgY2hhdAplbGlmIFsgJChkb2NrZXIgaW5zcGVjdCAtZiAn
e3suU3RhdGUuUnVubmluZ319JyBjaGF0KSA9ICJ0cnVlIiBdCnRoZW4
KCWVjaG8gIkFscmVhZHkgcnVubmluZyIKZWxzZQoJZG9ja2VyIHJ1
biAtZCAtLW5hbWUgY2hhdCAtZSBBU1BORVRDT1JFX0VOVklST05NRU
5UPSdEZXZlbG9wbWVudCgLXAgNTEwNjk6ODAgLS1yZXN0YXJ0IGFsd
2F5cyBhdHRlOTAyL2NoYXR3ZWJhcHA6dGVzdGkgCmZp"}'
```

Skripti on base64-koodattu, koska tiedostomuotoinen skripti ei toiminut. Vika oli varmaan konversiossa, joka muuttaa \r\n-rivinvaihtomerkit Linuxin \n-rivinvaihtomerkeiksi. Skripti on:

```
#!/bin/sh
if [ `sudo systemctl is-active docker` != "active" ]
then
    apt update
    apt install -y docker.io
fi
```

```
if [ $(docker inspect -f '{{.State.Running}}' chat) = "false" ]
then
    docker start chat
elif [ $(docker inspect -f '{{.State.Running}}' chat) = "true" ]
then
    echo "Already running"
else
    docker run -d --name chat -e ASPNETCORE_ENVIRONMENT='Development'
    -p 51069:80 --restart always atte902/chatwebapp:testi
fi
```

Skripti asentaa ensin Dockerin, jos se ei ole jo asennettu. Sen jälkeen tarkastetaan, onko chat-kontti jo olemassa, joka käynnistetään tai luodaan.

Seuraavaksi luodaan *Healt Probe*, jota tarvitaan kuormituksen tasaajassa:

```
az network lb probe create --name chat-lb-probe --lb-name chat-lb
--port 51069 --protocol tcp --resource-group AtenTesti3
```

Healt Probe luodaan porttiin 51069, joka oletusasetuksilla tarkastelee porttia 15 sekunnin välein. Kuormituksen tasaajan jälkeen voidaan luoda sille sääntö:

```
az network lb rule create --resource-group AtenTesti3
--name chat-bl-Rule --lb-name chat-lb
--backend-pool-name chat-lbBEPool  --backend-port 51069
--frontend-ip-name loadBalancerFrontEnd
--frontend-port 80 --protocol tcp --probe-name chat-lb-probe
```

Komennon jälkeen chat-sovellukseen saadaan yhteys portista 80, joka ohjautuu palvelimen porttiin 51069.

Autoskaalautuvuus luodaan komennolla:

```
az monitor autoscale create --resource-group AtenTesti3
--resource chatscaleset3
--resource-type Microsoft.Compute/virtualMachineScaleSets
--name chatautoscale
--min-count 1   --max-count 4   --count 1
```

Komento lisää vain automaattiskaalauksen, joka ei vielä toimi, koska sille ei ole ehtoja. Ehdot lisätään komennoilla:

```
az monitor autoscale rule create --resource-group AtenTesti3
--autoscale-name chatautoscale
--condition "Percentage CPU > 70 max 5m" --scale out 1

az monitor autoscale rule create --resource-group AtenTesti3
--autoscale-name chatautoscale
--condition "Percentage CPU < 30 avg 5m" --scale in 1
```

Komennot lisäävät samanlaiset skaalausasetukset kuin PaaS-resurssilla:

- Instanssien vähimmäismäärä 1,
- instanssien enimmäismäärä 4,
- ylösskaalauksen raja-arvo 70 % prosessorin käytössä viiden minuutin ajalta,
- ylösskaalaus 1 virtuaalikone kerrallaan,
- alasskaalauksen raja-arvo 25 % prosessorin käytössä viiden minuutin ajalta ja
- alasskaalaus 1 virtuaalikone kerrallaan.

Skaalaus ei voi olla nopeaa, koska prosessorin käyttöä seurataan viiden minuutin ajalta. Viisi minuuttia on pienin arvo, jonka tuohon saa laitettua. *Auto Scale Rulelle* saa myös määritettyä cooldown-attribuutin, joka on oltava vähintään minuutti. Tämä on väli, joka odotetaan skaalauksien välissä. Vakiona tämä väli on 5 minuuttia ja se on alustavien testien perusteella hyvä, koska skaalaus kestää niin kauan, että edellinen skaalaus ei ole ehtinyt kunnolla vaikuttamaan prosessorikuormaan.

Lisäksi alustavissa testeissä kävi ilmi, että jos *overprovision*-asetus on päällä, joka vakiona on, ja cooldown on liian alhainen, skaalaus tapahtuu odotettua aikaisemmin. Overprovisionin ollessa päällä Azure lähtee skaalaamaan kahdella instanssilla, joista toinen pudotetaan pois, kun ensimmäinen valmistuu. Jos cooldown antaa myöden ja aikaisempi skaalaus ei ole valmis, otetaan molemmat instanssit käyttöön.

Skaalautumattomat instanssit luodaan jättämällä skaalauksen komennot pois ja Core OS -instanssit vaihtamalla vain imagea eli ensimmäisen komennon attribuuttia muuttamalla.

5.2.2 Google Cloudin hintatestit

Google Cloudissa on laskutusosio, josta voidaan hakea kuluja erilaisilla hakuehdoilla. Eri resurssit on luotava omiin projekteihinsa, jotta kulut voidaan kategorisoida oikein. Tämä myös helpottaa tulosten tulkitsemista, mutta projekteja voi olla vain rajoitettu määrä laskutuksessa. Projekteja on myös todella hidas luoda, koska projektit eivät pysty käyttämään kovinkaan paljon muiden projektien resursseja, kuten pohjia, joilla resursseja luodaan. Projektien ansiosta voidaan kuitenkin tarkkailla piilokuluja.

PaaS

Google Cloudissa on pakko käyttää Google Cloudin omaa konttirekisteriä viedessään sovellusta *App Engineen*. Ensin luodaan image ChatServerGoogleCloud-projektista komennolla:

```
docker build -t gcr.io/chatpaas/chatwebapp:testi .
```

Komento toimii vain ChatServerGoogleCloud-projektin kansiossa. Komento luo imagen projektille chatpaas, joka seuraavaksi viedään Google Cloudin rekisteriin komennolla:

```
docker push  gcr.io/chatpaas/chatwebapp:testi
```

Projektilla on kuusi eri yaml-tiedostoa eli kaikille resursseille omansa, koska siinä määritellään ydinten määrä ja skaalautuvuus. Resurssin luominen onnistuu komennolla:

```
gcloud app deploy app1cpuNoScaling.yaml
--image-url gcr.io/chatpaas/chatwebapp:testi
```

Yaml-tiedostot pitää tehdä erikseen kaikille resursseille, mutta samaa imagea voi käyttää. Yamlin sisältö on kaikissa:

```
runtime: custom
env: flex
liveness_check:
  check_interval_sec: 15
  timeout_sec: 5
  failure_threshold: 2
  success_threshold: 2
```

```
  env_variables:
  ASPNETCORE_ENVIRONMENT: "Development"
```

Ytimien ja muistin määrä on määritelty yaml-tiedostossa jokaiselle resurssille erikseen:

```
resources:
  cpu: 4
  memory_gb: 4
  disk_size_gb: 10
```

Yhdellä ytimellä riittää gigatavu muistia, kahdella ytimellä riittää 1,5 gigatavua muistia, mutta Google Cloud vaatii 0,9 gigatavua muistia ydintä kohden neljällä ytimellä, joten neljälle ytimelle määritetään pyöreä 4 gigatavua.

Skaalautuvuus on määritelty yaml-tiedostossa:

```
automatic_scaling:
  min_num_instances: 1
  max_num_instances: 4
  cool_down_period_sec: 60
  cpu_utilization:
    target_utilization: 0.7
```

Määrityksessä asetetaan resurssille vähintään yksi instanssi ja enimmillään neljä. Jäähdytysaika on minuutissa, joka on pienin mahdollinen arvo. Jäähdytysaika on aika, joka odotetaan ennen kuin instanssi otetaan käyttöön eli annetaan aikaa alustaa itsensä. Skaalaus yrittää pitää prosessorin käytön 70 %:ssa, mutta dokumentaatiossa ei kerrota, miten se pyrkii pitämään sen.

Ilman skaalausta olevilla on yaml-tiedostossa:

```
manual_scaling:
  instances: 1
```

Tämä täytyy määrittää, koska Google Cloud vakiona nostaa kaksi instanssia.

Virtuaalikoneet

Skaalautuvat virtuaalikoneet saadaan aikaan seuraavilla komennoilla.

```
gcloud compute firewall-rules create chat-port --allow tcp:8080 \
  --description "Allow incoming traffic on TCP port 8080" \
  --direction INGRESS
```

Komento luo palomuurille säännön päästää portin 8080 TCP-liikenne läpi. Tämän jälkeen luodaan pohja, jonka mukaan uudet instanssit luodaan:

```
gcloud compute --project=chatvm instance-templates
create instance-template-chat-core-os
--machine-type=custom-1-1024
--network=projects/chatvm/global/networks/default
--network-tier=STANDARD
--metadata=gce-container-declaration=spec:$'\n'\ \
containers:$'\n'\ \ \ \ -\
name:\ instance-template-chat-core-os$'\n'\ \ \ \ \ \
image:\ \'gcr.io/chatpaas/chatwebapp:testi\'$'\n'\ \ \ \ \ \
env:$'\n'\ \ \ \ \ \ \ \ -\
name:\ ASPNETCORE_ENVIRONMENT$'\n'\ \ \ \ \ \ \ \ \ \
value:\ Development$'\n'\ \ \ \ \ \
stdin:\ false$'\n'\ \ \ \ \ \
tty:\ false$'\n'\ \
restartPolicy:\ Always$'\n'$'\n'\#\
This\
container\ declaration\ format\ is\ not\ public\ API\
and\ may\ change\ without\ notice.\
Please$'\n'\#\ use\ gcloud\ command-line\ tool\
or\ Google\ Cloud\ Console\ to\ run\
Containers\ on\ Google\ Compute\
Engine.,google-logging-enabled=true
--maintenance-policy=MIGRATE
--service-account=559147344938-compute@developer.gserviceaccount.com
--scopes=https://www.googleapis.com/auth/devstorage.read_only,
https://www.googleapis.com/auth/logging.write,
https://www.googleapis.com/auth/monitoring.write,
https://www.googleapis.com/auth/servicecontrol,
https://www.googleapis.com/auth/service.management.readonly,
https://www.googleapis.com/auth/trace.append --tags=chat-port
--image=cos-stable-73-11647-112-0
```

```
--image-project=cos-cloud --boot-disk-size=10GB
--boot-disk-type=pd-standard
--boot-disk-device-name=instance-template-chat-core-os
--labels=container-vm=cos-stable-73-11647-112-0
```

Komento on pitkä, koska instanssi määritellään käynnistämään kontti suoraan. Komennossa määritellään resurssin nimi, projekti, vakioverkot, käyttöjärjestelmäimage, kovalevy- ja palomuurisäännöt.

`--machine-type=custom-1-1024` määrittää, että kyseessä on yhden ytimen prosessori ja gigatavu muistia. Loput komennosta määrittää konttia eli asettaa kontti-imagen osoitteen, käynnistymään aina automaattisesti ja ympäristömuuttujan.

Seuraavana luodaan *Healt Check* eli ajoittain palvelun toimivuutta tarkasteleva palvelu. Arvot ovat vakioehdotukset eli 10 sekunnin välein tarkistus, vasta kolmas epäonnistuminen aiheuttaa toimenpiteitä ja tarkistus porttiin 8080, jota chat-sovellus käyttää Google Cloudissa. Komentona:

```
gcloud compute --project "chatvm" health-checks
create tcp "port8080-healtcheck"
--timeout "5" --check-interval "10" --unhealthy-threshold "3"
--healthy-threshold "2" --port "8080"
```

Tämän jälkeen voidaan luoda instanssiryhmä komennolla:

```
gcloud beta compute --project=chatvm
instance-groups managed create instance-group-scaling-ubuntu
--base-instance-name=instance-group-scaling-ubuntu
--template=instance-template-chat-core-os --size=1
--zone=europe-west1-b --health-check=port8080-healtcheck
--initial-delay=300
```

Instanssiryhmälle annetaan perustietoja kuten nimi, projekti, sijainti ja *Healt Checkin* ja instanssipohjan nimi. Lisäksi voidaan määrittää, että oletuksena on vain yksi instanssi käytössä attribuutilla `--size=1` ja käynnistysaika, jonka aikana ei *Healt Checkiä* suoriteta, attribuutilla `--initial-delay`, jonka vakioarvo on 300 sekuntia.

Lopuksi vielä määritellään skaalaus komennolla:

```
gcloud compute --project "chatvm" instance-groups managed
set-autoscaling "instance-group-scaling-ubuntu"
--zone "europe-west1-b" --cool-down-period "30"
--max-num-replicas "4" --min-num-replicas "1"
--target-cpu-utilization "0.7"
```

Tässä asetetaan skaalaus prosessorin käytön mukaan ja prosessorin käytön raja-arvoksi määritellään 70 % sekä minimi ja maksimi instanssien määrät. Käytössä on samat asetukset kuin PaaS-resurssilla.

Kuormituksen tasaajan luominen komennolla osoittautui vaikeaksi, joten se on tehty web-käyttöliittymän kautta. Käytetään HTTP kuormituksen tasaajaa, johon pitää luoda *backend service* ja *frontend configuration*. Lisätään nämä seuraavilla asetuksilla:

- Backend service:
  - Valitaan *Instance Group*,
  - portti 8080,
  - prosessorin maksimi kuormaksi valitaan vakio 80 %,
  - kapasiteetti 100,
  - valitaan luotu healt check porttiin 8080 ja
  - *Connection draining timeout* vakio 300.
- Frontend configuration:
  - Protokollaksi HTTP,
  - palvelun tasoksi premium, koska hinnassa ei mainita olevan eroa,
  - IP:nä käytetään vaihtuvaa. IP:n voi varata maksutta, jos sitä käyttää, muuten se maksaa.
  - IPv4 ja
  - portti 8080.

Ubuntu-resursseille pohja luodaan käyttäen hyväksi jo luotuja resursseja:

```
gcloud compute --project=chatvm instance-templates
create instance-template-ubuntu --machine-type=custom-1-1024
```

```
--network=projects/chatvm/global/networks/default
--network-tier=STANDARD
--metadata startup-script='#!/bin/sh
if [ `sudo systemctl is-active docker` != "active" ]
then
    apt update
    apt install -y docker.io
fi
if [ $(docker inspect -f '{{.State.Running}}' chat) = "false" ]
then
    docker start chat
elif [ $(docker inspect -f '{{.State.Running}}' chat) = "true" ]
then
    echo "Already running"
else
    docker run -d --name chat -e ASPNETCORE_ENVIRONMENT='Development'
       -p 51069:80 --restart always atte902/chatwebapp:testi
fi'
--maintenance-policy=MIGRATE
--service-account=559147344938-compute@developer.gserviceaccount.com
--scopes=https://www.googleapis.com/auth/devstorage.read_only,
https://www.googleapis.com/auth/logging.write,
https://www.googleapis.com/auth/monitoring.write,
https://www.googleapis.com/auth/servicecontrol,
https://www.googleapis.com/auth/service.management.readonly,
https://www.googleapis.com/auth/trace.append
--tags=http-server --image=ubuntu-1804-bionic-v20190320
--image-project=ubuntu-os-cloud --boot-disk-size=10GB
--boot-disk-type=pd-standard
--boot-disk-device-name=instance-template-ubuntu
```

Asetukset ovat muuten samat, mutta konttia ei voida käynnistää suoraan, joten sitä varten on määritettävä skripti, joka ajetaan instanssien käynnistyksessä. Käytetään samaa skriptiä kuin Azuren virtuaalikoneiden kanssa, joka asentaa Dockerin ja käynnistää chat-kontin.

Skaalautumattomat versiot tehdään samoilla komennoilla, mutta jätetään *gcloud compute instance-groups managed set-autoscaling* -komento pois, eikä luoda kuormituksen tasaajaa.

5.2.3 AWS hintatestit

AWS:ssä laskutusta voidaan kategorisoida tageilla. Tagit ovat erikseen aktivoitava laskutuksen erittelyä varten. Jotkut tagit ovat päivitettävä käsin resursseille, mutta skaalauksessa luotavien instanssien tagit saadaan lisättyä automaattisesti. Hyvä puoli tageissa on se, että ne toimivat avain-arvo-pareilla. Kaikille resursseille voidaan siis määrittää tagi yhdellä avaimella, jolle voidaan asettaa eri arvoja.

Virtuaalikoneille valitaan instanssityypiksi t3.nano ja t2.micro. T3.nano on AWS:n sivuilla[2] ilmoitettu halvimmaksi vaihtoehdoksi kirjoitushetkellä. Halvimmillaan se oli Pohjois-Euroopan alueella. Instanssityypissä on kaksi ydintä, mutta vain puoli gigaa muistia. Tämä eroaa Google Cloudista ja Azuresta, joissa on molemmissa 1 ydin ja 1 gigatavu muistia. T2.micro sen sijaan on 1-ytiminen ja 1 gigatavun resurssi.

T2.microa ei ollut saatavilla Pohjois-Euroopan alueella, joten on käytettävä Länsi-Euroopan aluetta. T2.micro on käytössä vain Core OS- ja PaaS-testeissä, koska testeissä kävi ilmi, että Core OS ja *Elastic Beankstalkin* mukana tulevat sovellukset käyttävät niin paljon muistia, että t3.nano-resurssin puoli gigatavua ei riittänyt rasitustestien ajamiseen. Docker kaatui, kun muistin käyttö ylitti noin 20 %. Perushinnat kuitenkin on mitattu Core OS:n osalta molemmissa resursseissa.

PaaS

PaaS-resurssin luodaan helpoiten komentoriviltä, navigoimalla chat-sovelluksen projektikansioon, jossa on Docker-tiedosto. Helpoimmalla pääsee, jos asettaa Dockerin portiksi 80 Docker-tiedostoon.

Resurssin luominen onnistuu suorittamalla projektikansiossa komennot:

```
eb init -p docker chat-1 -r eu-west-1
eb create chat-1 -r eu-west-1 -k atte-key -i t2.micro
--elb-type classic --tags Key=Project,Value=PaaS
```

Komento asettaa instanssityypiksi t2.micro ja kuormituksen tasaajan tyypiksi *classic*. Tällöin resurssi on lähes sama kuin virtuaalikoneiden testissä. Ainoa ero on se, että resurssilla on

2. https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/

rajoittamattomuusasetus päällä (engl. T2/T3 Unlimited), joka mahdollistaa sen, että resurssi voi tarvittaessa ottaa käyttöönsä enemmän tehoa.

Virtuaalikoneet

Ensin luodaan tietoturvaryhmä (engl. security group) komennoilla:

```
aws ec2 create-security-group --group-name ChatSecurityGroup \
--description "Security Group for EC2 instances
to allow port 22 and 80"
aws ec2 authorize-security-group-ingress \
--group-name ChatSecurityGroup \
--protocol tcp --port 22 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress \
--group-name ChatSecurityGroup \
--protocol tcp --port 80 --cidr "0.0.0.0/0, ::/0"
aws ec2 describe-security-groups --group-names ChatSecurityGroup
```

Ensimmäinen komento luo itse tietoturvaryhmän. Toinen ja kolmas komento avaa portit 22 ja 80, jotta instansseihin saa HTTP- ja SSH-yhteyden. Viimeinen komento kertoo tietoturva ryhmän tunnuksen, jota käytetään kuormituksen tasaajan luomisessa.

Seuraavaksi luodaan pohja (engl. launch template), jolla uudet instanssit luodaan. *Launch template* on uusi resurssi AWS:ssä. Aikaisemmin on tehty konfigurointipohjia (engl. Launch Configuratioita). Konfiguraatiopohjilla ei voi asettaa rajoittamattomuusasetusta, joten uudenlaista pohjaa on käytettävä, joka on luotava JSON-tiedostosta tai syöttämällä JSON suoraan komentoriville. Käytössä oleva pohja on kuitenkin sen verran pitkä, että se on tallennettu erilliseen tiedostoon, joka annetaan komennolle.

Pohja luodaan komennolla:

```
aws ec2 create-launch-template --launch-template-name chat-template \
--launch-template-data file://template.json
```

Template.json sisältö Ubuntu-instanssille on seuraava:

```
{
"Monitoring": {
```

```
    "Enabled": true
},
"CapacityReservationSpecification": {
    "CapacityReservationPreference": "open"
},
"ImageId": "ami-5e9c1520",
"BlockDeviceMappings": [
    {
        "DeviceName": "/dev/sda1",
        "Ebs": {
            "Encrypted": false,
            "DeleteOnTermination": true,
            "VolumeType": "gp2",
            "VolumeSize": 10,
            "SnapshotId": "snap-0903e7da38397555c"
        }
    }
],
"KeyName": "atte-key",
"HibernationOptions": {
    "Configured": false
},
"EbsOptimized": false,
"Placement": {
    "Tenancy": "default",
    "GroupName": "",
    "AvailabilityZone": "eu-north-1a"
},
"InstanceType": "t3.nano",
"CreditSpecification": {
    "CpuCredits" : "standard"
},
"UserData": "IyEvYmluL3NoCmlmIFsgYHN1ZG8
gc3lzdGVtY3RsIGlzLWFjdGl2ZSBkb2NrZXJgICE9
ICJhY3RpdmUiIF0KdGhlbgoJYXB0IHVwZGF0ZQo
JYXB0IGluc3RhbGwgLXkgZG9ja2VyLmlvCmZpCmlm
IFsgJChkb2NrZXIgaW5zcGVjdCAtZiAne3suU3Rhd
```

```
GUuUnVubmluZ319JyBjaGF0KSA9ICJmYWxzZSIgXQ
p0aGVuCglkb2NrZXIgc3RhcnQgY2hhdAplbGlmIFs
gJChkb2NrZXIgaW5zcGVjdCAtZiAne3suU3RhdGUu
UnVubmluZ319JyBjaGF0KSA9ICJ0cnVlIiBdCnRoZ
W4KCWVjaG8gIkFscmVhZHkgcnVubmluZyIKZWxzZQ
oJZG9ja2VyIHJ1biAtZCAtLW5hbWUgY2hhdCAtZS
BBU1BORVRDT1JFX0VOVklST05NRU5UPSdEZXZlbG9
wbWVudCgLXAgODA6ODAgLS1yZXN0YXJ0IGFsd2F5
cyBhdHRlOTAyL2NoYXR3ZWJhcHA6dGVzdGkgCmZp",
"NetworkInterfaces": [
    {
        "DeviceIndex": 0,
        "Description": "",
        "SubnetId": "subnet-23c8314a",
        "DeleteOnTermination": true,
        "Groups": [
            "sg-04c28b13484f03fcb"
        ]
    }
]
}
```

Core OS:lle tarvitaan vähän erilainen pohja, jossa on muutettu imagen tunnus arvoon ami-00e03f7974618119a, joka on Core OS -image, instanssi t2.micro, snapshot-tunnus ja tietoturvaryhmän tunnus, jotka ovat nyt eri arvoja, koska alue on toinen.

Lisäksi AWS:ssä ei ole samanlaista suoraa Docker-tukea kuin Azuressa ja Google Cloudissa, vaan instanssin luomiseen määritellään skripti. Ubuntulla skripti on sama kuin muissa pilvipalveluissa, mutta Core OS:llä on jo Docker valmiiksi asennettu, joten skriptistä voidaan ottaa Dockerin asennus pois.

```
#!/bin/sh
if [ `sudo systemctl is-active docker` != "active" ]
then
    apt update
    apt install -y docker.io
fi
```

```
if [ $(docker inspect -f '{{.State.Running}}' chat) = "false" ]
then
    docker start chat
elif [ $(docker inspect -f '{{.State.Running}}' chat) = "true" ]
then
    echo "Already running"
else
    docker run -d --name chat -e ASPNETCORE_ENVIRONMENT='Development'
    -p 51069:80 --restart always atte902/chatwebapp:testi
fi
```

Skaalautuvilla resursseilla luodaan tässä vaiheessa kuormituksen tasaaja. Skaalautumattomat tarvitsevat enää vain instanssin luomisen pohjasta.

Luodaan kuormituksen tasaaja komennolla:

```
aws elb create-load-balancer --load-balancer-name chat-lb \
--listeners "Protocol=HTTP,LoadBalancerPort=80,InstanceProtocol=HTTP,
InstancePort=80" \
--availability-zones eu-north-1a --security-groups sg-04c28b13484f03fcb
```

Kuormituksen tasaajia on erilaisia, kuten Google Cloudissa, mutta käytetään Googlen Cloudin kanssa saman tapaista kuormituksen tasaajaa eli HTTP-kuormituksen tasaajaa. Chat-sovellus käyttää porttia 80, jolloin HTTP-kuormituksen tasaaja on muutenkin järkevä valinta. Kuormituksen tasaajalle ei voinut asettaa tietoturvaryhmää nimellä, vaan käytetään aikaisemmin haettua tunnusta.

Skaalautumattomilla resursseilla seuraava komento on ainoa tarvittava ilman kuormituksen tasaajan asettamista, eli `--load-balancer-names chat-lb`-attribuuttia, joka asetetaan skaalautuville resursseille.

```
aws autoscaling create-auto-scaling-group \
--auto-scaling-group-name chat-scale-group \
--launch-template '{"LaunchTemplateName": "chat-template"}'\
--min-size 1 --max-size 4 \--load-balancer-names chat-lb \
--default-cooldown 300 --availability-zones eu-north-1a
```

Komento luo aikaisemmin luodusta pohjasta uuden skaalautuvan ryhmän, jolla voi olla enim-

millään neljä instanssia ja vähimmillään yksi. Vakio jäähdytysaika (engl. default cooldown) asetetaan 300 sekuntiin, joka määrittää, kuinka kauan odotetaan ennen kuin uusi skaalaustoimenpide voi alkaa.

Skaalautuville resursseille määritetään vielä skaalausohjeet seuraavalla komennolla:

```
aws autoscaling put-scaling-policy --policy-name chat-scaling \
--auto-scaling-group-name chat-scale-group \
--policy-type TargetTrackingScaling  --estimated-instance-warmup 60 \
--target-tracking-configuration '{"TargetValue": 70.0,
"PredefinedMetricSpecification":
{"PredefinedMetricType": "ASGAverageCPUUtilization"}}'
```

Komennolla asetetaan tavoiteltavaksi prosessorin keskimääräiseksi kuormaksi 70 %, kuten muillakin pilvipalveluilla. Arvioidaan, että instansseilla menee 60 sekuntia käynnistyä, jonka aikana niille ei ohjata liikennettä.

Lopuksi liitetään kuormituksen tasaaja skaalausryhmään:

```
aws autoscaling attach-load-balancers --load-balancer-names chat-lb \
--auto-scaling-group-name chat-scale-group
```

Vaikka skaalausryhmää luodessa kerrottiin kuormituksen tasaaja, tämä komento oli ajettava, jotta instanssit liittyivät kuormituksen tasaajaan.

5.3 Skaalautuvuustestit

Skaalaustestit tehdään Core OS:lle ja Ubuntulle. Käyttöjärjestelmien välillä voi olla eroja, koska Core OS on luotu konttien ajamista varten ja siinä on valmiiksi asennettuna Docker. Testejä tehdään kolme kierrosta, joissa kuormitusta ajetaan niin kauan, että kaikki neljä instanssia on käytössä, jonka jälkeen kuormitus lopetetaan ja seurataan alasskaalausta. Testeissä käytetään samoja resursseja, joita hintatesteissä on luotu, mutta skaalausasetuksiin voidaan tehdä tarvittaessa muutoksia parhaiden asetuksien löytämiseksi.

Skaalaustesteissä tarkastellaan, kuinka nopeasti skaalaus alkaa ja kuinka nopeasti instanssi on nostettu. Vastaavasti alasskaalauksessa tarkastellaan kuinka nopeasti alasskaalaus alkaa

ja kuinka kauan siinä kestää kokonaisuudessaan. Koska skaalauksen kesto voidaan jakaa kahteen osaan, skaalauksen aloitukseen kuluva aika ja skaalauksen kesto, on mittareiksi valittu molemmat. Ylösskaalaus mittaa rasitukseen vastaamista ja alasskaalaus mittaa, kuinka tehokkaasti pilvipalvelu toimii. Kulut nousevat, jos instansseja on päällä turhana, joten täydellinen pilvipalvelu reagoisi nopeasti kuormituksen eri asteisiin.

Skaalautuvuuden testaamiseen käytetään myös *SuperBenchmarker*-sovellusta. Komento `sb -u "url" -c 200 -N 7200 -B -W 5` on riittävä skaalauksen testaamiseen. Komennon suoritus voidaan lopettaa, kun kaikki neljä instanssia ovat käytössä.

5.3.1 Azuren skaalautuvuustestit

Azuressa *Scale Setin* skaalautumista voidaan tarkkailla *Activity Logista*, minne kirjautuu erilaisia toimenpiteitä lähes kaikilla resursseilla. Tutkimuksen aikana *Acitivity Log* kuitenkin päivittyi, vaikka tietoa päivityksestä ei löytynyt. Aikaisemmin uuden instanssin luominen, hyväksytty tilapäivitys ja käynnistysaika näkyivät lokilla, jolloin saatiin sekunnin tarkkuudella skaalautumiseen käytetty aika.

Myöhemmin tuo muuttui niin, että lokille kirjattiin vain aloitus- ja valmistumisaika, jotka kirjattiin lähes samaan aikaan. Käytetään *Activity logia* skaalauksen aloituksen määrittämiseen ja alasskaalauksessa, koska se on ainut tieto, joka voidaan saada alasskaalauksessa. Ylösskaalauksessa kirjataan lokilta aloitusaika ja käydään SSH-yhteyden kautta tarkastamassa dockerin aloitusaika, josta voidaan laskea skaalauksen kesto erotuksella. Dockerin aloitusaika on otettu komennolla:

```
docker inspect --format='{{.State.StartedAt}}' <CONTAINERID>
```

Skaalausasetukset testeissä:

- 5 minuutin aikaikkuna, jossa prosessorikuorma:
  - keskiarvo yli 70 % -> lisätään yksi instanssi
  - keskiarvo alle 30 % -> vähennetään yksi instanssi
- Skaalauksen jälkeen uutta skaalausta ei voida aloittaa 5 minuuttiin

Azuren PaaS-resurssin eli *Web Appin* testaaminen *SuperBenchmarkerin* kanssa osoittautui mahdottomaksi, koska siinä on liikaa laskentatehoa. Prosessorin käyttö nousi vain 8 %:iin parhaimmillaan. Onneksi Azuressa on myös oma rasitustesti, jota on tässä tilanteessa pakko käyttää. Rasitustestille määritellään käyttäjälataus (engl. user load), joka määrittää rasituksen tasoa. Tämän lisäksi määritetään vain testin kesto ja sijainti. Valitaan sijainniksi sama kuin testattavalla resurssilla eli Länsi-Eurooppa ja kestoksi 30 minuuttia, jotta skaalaus neljään instanssiin ehtivät käynnistymään.

Aloitusaika saadaan lokilta, jonne Azure kirjoittaa tapahtuman *Autoscale scale up initiated*, kun skaalaus aloitetaan. Tämän jälkeen tulee tapahtuma *Autoscale scale up completed* samaan tapaan kuin *Scale Seteillä*, mutta se ei ole vielä täysin käynnissä siinä vaiheessa, eikä ota liikennettä vastaan. PaaS-resurssissa ei voida ottaa yhteyttä virtuaalikoneeseen ja pyytää Dockerin käynnistysajankohtaa. Onneksi *Web App* kirjaa lokille Docker-lokia, josta voidaan löytää jokaiselle instanssille kontin käynnistysaika. Jokaisesta käynnistyksestä tulee lokille ilmoitus:

```
2019-05-26 06:50:40.155 INFO - Container skaalaustesti_0 for site
skaalaustesti initialized successfully and is ready to serve requests.
```

Näiden tapahtumien aikaleiman erotuksesta saadaan skaalauksen kesto.

5.3.2 Google Cloudin skaalautuvuustestit

Skaalausasetuksia ei pysty Google Cloudissa muokkaamaan kunnolla, kun käytössä on prosessorin kuormitukseen perustuva mittari. Skaalaus on asetettu prosessorin käytön 70 % raja-arvoon. Ainoa toinen asetus, jota voi muokata on luomiskomennossa oleva –cool-down-period-attribuutti, joka määrittää vain, kuinka kauan uusi instanssi on oltava käynnissä, jotta sitä ei yritetä korjata automaattisesti. Tämä on asetettu 30 sekuntiin, mutta testeissä asetuksella ei vaikuttanut olevan mitään vaikutusta.

Google Cloudissa lokit kirjoitettaan *Stackdriveriin*. Lokia suodatetaan web-käyttöliittymän kautta näppärästi ja lokilta voidaan suodattaa kaikki turhat viestit pois. Lokille tulee kuormitustestin seurauksena paljon viestejä, joten on tärkeä saada ne suodatettua pois. Samat lokiviestit tulevat PaaS-resursseilla ja virtuaalikoneresursseilla samalla tavalla.

Loki on äärimmäisen tarkka. Kun metriikka ilmoittaa, että skaalaus voisi tapahtua, tulee lokille ilmoitus *compute.instanceGroupManagers.resizeAdvanced*, jonka jälkeen tulee ilmoitus GCE_API_CALL, jossa on joko compute.instances.delete- tai compute.instances.insert-alatyyppi. Lopuksi tulee samalle alatyypille GCE_OPERATION_DONE-ilmoitus. Näitä tulkitaan siten, että resizeAdvanced on skaalauspäätös, GCE_API_CALL on instanssin luomisen aloittamisesta ja GCE_OPERATION_DONE on skaalauksen valmistuminen.

5.3.3 AWS skaalautuvuustestit

AWS:n skaalausasetuksia oli kokeiltava paljon, jotta parhaimmat asetukset löytyivät.

AWS:n kuormituksen tasaajan oletus asetukset asettavat aktiivisuustarkastuksen (engl. healt check) siten, että

- Kutsun aikakatkaisu 5 s,
- kutsujen väli 30 s,
- epäaktiivisen raja 2 ja
- aktiivisen raja 10.

Jos kuormituksen tasaaja toteaa instanssin epäaktiiviseksi, se ei lähetä sille yhtään liikennettä. Koska aktiivisuuden toteamiseen tarvitaan 10 onnistunutta kutsua ja kutsujen väli on 30 sekuntia, menee aktiivisuuden toteamiseen 5 minuuttia.

Asetetaan arvot järkevämmäksi:

- Kutsun aikakatkaisu 5 s,
- kutsujen väli 10 s,
- epäaktiivisen raja 2 ja
- aktiivisen raja 3.

Tutkimuksessa koitettiin erilaisia asetuksia, mutta näillä asetuksilla liikenne alkaa paremmin ohjautumaan lähes heti, kun kontti saadaan pystyyn.

Skaalausasetuksista voi valita joko tiettyyn raja-arvoon perustuvan asetuksen (engl. Target tracking scaling policy), jolla voidaan asettaa prosessorin keskikuormitus 70 %:iin kuten

Google Cloudissa. Valittavissa on kuitenkin myös yksinkertainen asetus, jolla voidaan määrittää omat rajat ja aikavälit kuten Azuressa.

Testeissä yksinkertaiset rajat olivat hitaampia skaalautumaan ylöspäin, mutta tulivat nopeammin alaspäin. Syynä voi olla se, että prosessorikuormaa ei mitata tarpeeksi tarkalla mittarilla tai se, että keskiarvo ei ehdi nousta tarpeeksi nopeasti. Käytetään siksi tässä tutkimuksessa tuota tiettyyn raja-arvoon perustuvaa skaalausta ja asetetaan keskimääräiseksi prosessorikuormaksi 70 % ja arvioiduksi instanssin käynnistymisajaksi minuutti. Nämä asetukset saa kaikille resursseille samoiksi.

Elastic Beanstalk kirjoittaa lokille erilaisia tapahtumia, joita voi seurata oman *Beanstalk*-sovelluksen näkymässä. Lisäksi AWS kirjoittaa lokille kaikki tapahtumat *Cloud Trailiin*, josta ne voi ladata tiedostona. *Beanstalkin* tapahtumat eroavat *Cloud Trailin* tapahtumista hieman. *Beanstalkin* tapahtumissa on esimerkiksi tapahtuma *Removed instance from your environment* ja *Added instance to your environment.* Tutkimuksissa *Beanstalkin* lokitapahtuma näyttää tulevan siinä vaiheessa, kun instanssi on käynnistetty, mutta ei vielä ole ajanut aloitusskriptejä.

Instanssilla on käynnistysaika, jolloin sitä on lähdetty käynnistämään. Se on noin puolitoista minuuttia *Beanstalkin* tapahtumaa aikaisemmin. Vielä ennen tätä skaalausryhmän aktiviteetista näkyy, että skaalaus on aloitettu. Aktiviteettiloki vaikuttaa ensimmäiseltä indikaattorilta skaalauksesta, joten käytetään sitä aikaa instanssin luomisen aloittamiseen. Kätevästi sama lokin on kaikilla resurssityypeillä.

Dockerin käynnistämisestä ei tule mitään tapahtumaa, joten käytetään samaa metodia kuin Azuren kanssa, eli otetaan SSH-yhteys palvelimeen ja suoritetaan komento:

```
Docker inspect --format='{{.Created}}' <CONTAINERID>
```

Komennossa haetaan .State.StartedAt sijaan luomisaika. StartedAt-aika eli käynnistysaika voi olla väärä, jos kontti on kaatunut ja se on käynnistetty uudelleen, kuten alustavissa testeissä usein kävi.

Skaalauksen kesto on siis instanssin käynnistysajan ja Dockerin käynnistysajan erotus.

6 Tulokset

Tässä luvussa esitellään tulokset luvussa 5 määritellyille testeille. Ensin on hintatestien tulokset pilvipalvelun mukaan jaoteltuina. Hintatestien tulosten jälkeen on skaalaustulokset pilvipalvelun mukaan jaoteltuina. Viimeisenä lasketaan oman palvelimen juoksevia kuluja vertailua varten.

6.1 Hintatulokset

Tässä luvussa esitellään hintatulokset kaikista pilvipalveluista: Azure, Google Cloud ja AWS. Samalla analysoidaan testejä ja kerrotaan mahdollisista ongelmista.

6.1.1 Azure

Taulukko 1: PaaS-resurssien hinnat verottomana.

Resurssi	Hinta (EUR/pv)
App Service S1	1,92
App Service B1	1,07
Container Instance 1 ydin	1,10
Container Instance 2 ydintä	2,05
Container Instance 4 ydintä	3,93

Taulukko 2: Container instanssien hinnat verottomana.

Resurssi	Prosessori (EUR/pv)	Muisti (EUR/pv)
Container Instance 1 ydin	0,94	0,16
Container Instance 2 ydintä	1,89	0,16
Container Instance 4 ydintä	3,77	0,16

Taulukon 1 mukaan *Web App for Container* osoittautui muuten ilmaiseksi, mutta *App Service plan*, jota tarvitaan *Web Appin* luomiseen, oli maksullinen ja sen hinta oli koko ajan sama eli 1,92 euroa päivässä S1:llä ja 1,07 euroa B1:llä riippumatta liikenteestä. S1 taso tarjoaa skaalautuvuuden ja B1 ei.

Container Instancet sen sijaan olivat kalliita muihin verrattuna, mikä nähdään taulukosta 2. Instansseilla laskutus on tuntiperusteinen ja laskua kertyi koko ajan. Hinta määräytyi prosessorin ja muistin käytön mukaan sekunneissa. Muistia oli kaikissa 1,5 gigatavua, joten kaikkien muistin hinta oli sama. Prosessorin käyttö oli lähes suorassa suhteessa ytimien määrään.

Helmikuussa ensimmäisiä testejä tehdessä yhden ytimen instanssi maksoi 1,64 euroa päivässä ja neljän ytimen 4,92 euroa päivässä. Maaliskuun aikana kuitenkin hinnat laskivat ja huhtikuun testeissä hinnat olivat 1,10 euroa päivässä yhdeltä ytimeltä ja 3,93 euroa päivässä neljältä ytimeltä. *Container Instance* olivat vielä preview-vaiheessa, joten siihen voi tulla muutoksia.

Rasitustesteillä ei ollut vaikutusta kuluihin. Kaikki PaaS-resurssit olivat kiinteähintaisia, joten niillä on selvä etu, jos laskentaa tarvitaan enemmän.

Taulukko 3: *Scale Setin* perushinnat verottomana.

Resurssi	Virtuaalikone (EUR/pv)	Kovalevy (EUR/pv)	Kuormituksen tasaaja (EUR/pv)
Skaalautuva Ubuntu 18.04	0,24	0,17	0
Skaalautumaton Ubuntu 18.04	0,24	0,17	0
Skaalautuva Core OS	0,24	0,17	0
Skaalautumaton Core OS	0,24	0,17	0

Scale Setien eli virtuaalikoneiden perushinnat kaikilla resursseilla olivat taulukon 3 mukaan samat. Kuormituksen tasaaja ei maksanut mitään, ilman liikennettä. Kovalevy oli premium SSD, joka on Azuren hallinnoima. IP oli basic dynaaminen, joten se oli ilmainen.

Kahden tunnin rasituksessa Azuren *Scale Setin* kuormituksen tasaaja ei toiminut oikein. Skaalauksen valmistuttua, uudelle instanssille ei aina ohjautunut yhtään liikennettä. Joskus uusi instanssi sai liikennettä suoraan, mutta joskus ainoastaan *SuperBenchmarkerin* uudelleenkäynnistäminen auttoi. Uudelleenkäynnistystä ei kuitenkaan voinut tehdä, koska ajo oli asetettu kahteen tuntiin ja se oli ajettava loppuun, kuten muillakin testeillä.

Taulukko 4: *Scale Setin* rasitustestien hinnat verottomana.

Resurssi	Virtuaalikone (EUR/pv)	Kovalevy (EUR/pv)	Data (EUR/pv)	Pyyntöjen määrä
Skaalautuva Ubuntu 18.04	0,29	0,19	0,48	1 934 447
Skaalautumaton Ubuntu 18.04	0,24	0,16	0,39	1 335 404

Resurssi	Virtuaalikone (EUR/pv)	Kovalevy (EUR/pv)	Data (EUR/pv)	Pyyntöjen määrä
Skaalautuva Core OS	0,28	0,16	0,46	1 904 386
Skaalautumaton Core OS	0,24	0,16	0,32	1 094 077

Taulukon 4 mukaan rasituksessa virtuaalikoneet ja kovalevyt maksoivat saman verran kuin ilman rasitusta, jos uusia instansseja ei luotu. Skaalautuva Core OS laskutti kovalevystä saman 0,16 euroa, vaikka skaalauksen olisi pitänyt luoda uusi kovalevy uusille instansseille ja sitä kautta nostaa hintaa. Mistään ei kuitenkaan selviä, miksei hinta noussut.

Koska liikenne ei jakautunut instanssien kesken, skaalaus nosti ensin muutaman instanssin, mutta poisti ne hetken päästä. Tämän takia tuloksissa on hieman eroja. Skaalautuvat kuitenkin pääsivät pyyntöjen määrässä lähelle toisiaan, kun taas skaalautumattomat eivät.

6.1.2 Google Cloud

Google Cloudin *App Engine* laskutti koko ajan prosessorin käytöstä. Laskutus kertaantui ydinten määrällä, ja koska käytössä oli 2-ytiminen ja 4-ytiminen *App Engine*-toteutus, kuluja kertyi paljon verrattuna muihin resursseihin. Eli näiden kanssa on sama ongelma kuin Azuren *Container Instancejen*. Laskutus voi johtua kontista, koska Docker suorittaa jotain vähän väliä käyttäen alle 1 %:n prosessoriajasta. On vaikea sanoa, johtuuko Dockerin suoritus siitä, että Asp.Net-palvelin suorittaa jotain, koska sekin prosessi käytti prosessoria. Yhteensä tämä vähäinen käyttö laskutettiin samalla kaavalla kuin 100-%:nen prosessorin käyttö.

Taulukko 5: *App Enginen* perushinnat verottomana.

Resurssi	Prosessori (EUR/pv)	Muisti (EUR/pv)
Skaalautuva 1-ytiminen	1,24	0,25
Skaalautumaton 1-ytiminen	1,24	0,25
Skaalautuva 2-ytiminen	2,43	0,33
Skaalautumaton 2-ytiminen	2,43	0,33
Skaalautuva 4-ytiminen	4,86	0,74
Skaalautumaton 4-ytiminen	4,86	0,74

Taulukossa 5 on listattu kaikki *App Enginen* perushinnat. 1-ytimisellä on 1 gigatavu muistia, 2-ytimisellä 1,5 gigatavua ja 4-ytimisellä 4 gigatavua, josta johtuvat erot muistin perushinnassa. Google Cloudissa muistia on pakko lisätä ytimiä kohden. Hinnat olivat yhtenäiset Googlen ilmoittaman hinnaston mukaan.

Taulukko 6: *App Enginen* rasitustestien hinnat verottomana.

Resurssi	Prosessori (EUR/pv)	Muisti (EUR/pv)	Kuormituksen tasaaja + data (EUR/pv)	Pyyntöjen määrä
Skaalautuva 1-ytiminen	1,59	0,34	0,96	2 013 299
Skaalautumaton 1-ytiminen	1,11	0,23	0,32	1 068 590

Hinnan takia testit pystyttiin tekemään vain 1-ytimiselle. Kummallista taulukon 6 tuloksissa oli se, että skaalautumattoman prosessorin ja muistin hinta laskivat. Jostain syystä Google laskutti vain 77 884 sekunnista, kun vuorokaudessa on 86 400 sekuntia. Tutkimuksessa ei löytynyt syytä, mistä tämä johtuu. Skaalautuvalla oletetusti kulut nousivat, koska instansseja oli käytössä enemmän. Samalla dataa voitiin käsitellä enemmän, joten kuormituksen tasaajan kulut nousivat.

Taulukko 7: Virtuaalikoneiden perushinnat verottomana.

Resurssi	Prosessori (EUR/pv)	Muisti (EUR/pv)	Kuormituksen tasaaja
Skaalautuva Core OS	0,72	0,10	0,53
Skaalautumaton Core OS	0,71	0,09	-
Pelkkä Core OS -instanssi	0,71	0,09	-
Skaalautuva Ubuntu	0,71	0,09	0,53
Skaalautumaton Ubuntu	0,71	0,09	-
Pelkkä Ubuntu-instanssi	0,71	0,09	-

Taulukossa 7 on listattu Google Cloudin virtuaalikoneiden perushinnat. Perushinnat olivat kaikilla virtuaalikoneinstansseilla samat, mutta skaalautuvalla Core OS-resurssilla laskutus oli jostain syystä yli 24 tuntia, joten sitä laskutettiin sentti enemmän. Tälle ei löytynyt mitään syytä, mutta laskutuksessa käytettiin samaa hintaa kaikille ja laskutus perustui muilla 24 tuntiin.

Taulukko 8: Virtuaalikoneiden rasitushinnat verottomana.

Resurssi	Prosessori (EUR/pv)	Muisti (EUR/pv)	Kuormituksen tasaaja (EUR/pv)	Data (EUR/pv)	Pyyntöjen määrä
Skaalautuva Core OS	0,89	0,12	0,53	0,45	569 466
Skaalautumaton Core OS	0,71	0,09	-	0,03	57 330
Pelkkä Core OS -instanssi	0,71	0,09	-	0,08	342 136
Skaalautuva Ubuntu	0,90	0,12	0,53	1,29	2 388 532
Skaalautumaton Ubuntu	0,71	0,09	-	0,30	1 303 461
Pelkkä Ubuntu-instanssi	0,71	0,09	-	0,29	1 280 076

Kuten taulukosta 8 nähdään, kahden tunnin rasitus ei tehnyt tiukkaa skaalautuvalle resurssille. Google Cloudin skaalautuvuus oli todella nopeaa ja toimivaa. Liikenne jakautui tasaisesti instanssien välillä. Skaalautumattomat olivat vaikeuksissa rasituksen kanssa ainakin Core OS:llä. Ilmeisesti teho eivät riittäneet kontin pyörittämiseen. Ero voi myös johtua ajankohdasta, koska testejä ei ajettu samana päivänä. Kolmas syy voi olla *SuperBenchmarker*, jonka toiminnasta ei voi olla täysin varma, koska se on kolmannen osapuolen sovellus.

6.1.3 AWS

PaaS käytti samoja resursseja kuin virtuaalikoneet. Tästä syystä tulokset eivät eroa taulukoiden 9 ja 11 välillä muuten kuin kovalevyjen osalta, koska niitä oli PaaS-resurssilla kaksi kappaletta.

Taulukko 9: *Elastic Beanstalk* perushinnat verottomana.

t3.nano (EUR/pv)	Kuormituksen tasaaja (EUR/pv)	Kovalevy (EUR/pv)
0,13	0,64	0,07

Taulukko 10: *Elastic Beanstalk* rasitushinnat verottomana.

CPU credit (EUR/pv)	t3.nano (EUR/pv)	kovalevy (EUR/pv)	Kuormituksen tasaaja (EUR/pv)	Data (EUR/pv)	Pyyntöjen määrä
0,01	0,17	0,09	0,64	0,84	3 135 113

Taulukon 10 datan hinnassa on mukana käsittelyhinta ja itse datan siirtohinta. Datan käsittely maksoi vain 0,05 euroa ja siirto loput 0,79 euroa.

Resurssissa oli päällä asetus, joka mahdollistaa ylimääräisen tehon lisäyksen tarvittaessa. Pyyntöjen määrä on kasvanut todella isoksi juuri sen takia, että tehoja on saatu enemmän käyttöön. CPU credit -sarake näyttää paljonko tästä aiheutui kuluja. Näyttää siltä, että tuo asetus on hyvä pitää päällä, koska se kasvatti kuluja vain sentillä.

Virtuaalikoneiden testeissä T3.nano resurssit olivat eu-north-1-alueelta ja t2.micro resurssit olivat eu-west-1-alueelta.

Taulukko 11: Virtuaalikoneiden perushinnat verottomana.

Resurssi	t3.nano (EUR/pv)	t2.micro (EUR/pv)	Kuormituksen tasaaja (EUR/pv)	Kovalevy (EUR/pv)
Skaalautuva Core OS	0,13		0,67	0,04
Skaalautumaton Core OS	0,13	0,30	-	0,04
Pelkkä Core OS -instanssi	0,13	0,30	-	0,04
Skaalautuva Ubuntu	0,13	-	0,64	0,03
Skaalautumaton Ubuntu	0,13	-	-	0,03
Pelkkä Ubuntu-instanssi	0,13	-	-	0,03

T3.nano oli kaikissa testeissä perushinnaltaan saman hintainen, kuten taulukosta 11 nähdään. Skaalautumaton Core OS pyöristi hinnan 0,17 euroon, vaikka instanssi maksoi 0,13 euroa ja kovalevy 0,03 euroa. Tarkempi hinta oli noin 0,1652 euroa, kun muilla hinta jäi hieman alle 0,165 euron, jolloin se pyöristyi 0,16 euroon.

T2.micro resurssien alueella kovalevy oli sentin kalliimpi eli 0,04 euroa. Oikeasti ero oli enemmän 0,002 euron luokkaa, mutta koska hinta pyöri kolmen ja puolen sentin tuntumassa, tuo riitti pyöristämään hinnaksi 0,04 euroa. Myös kuormituksen tasaajan hinta oli kolme sentin suurempi eli 0,67 euroa.

Taulukko 12: Virtuaalikoneiden rasitushinnat verottomana.

Resurssi	t3.nano (mukana kovalevy) (EUR/pv)	Kuormituksen tasaaja (EUR/pv)	Data (EUR/pv)	Pyyntöjen määrä
Skaalautuva Core OS	0,38	0,67	0,63	1 734 111
Skaalautumaton Core OS	0,34	-	0,38	1 165 545
Pelkkä Core OS -instanssi	0,34	-	0,38	1 168 913
Skaalautuva Ubuntu	0,16	0,64	0,64	1 896 134
Skaalautumaton Ubuntu	0,16	-	0,61	1 879 437
Pelkkä Ubuntu-instanssi	0,16	-	0,23	716 107

Rasitushinnoissa datan hinta pitää sisällään kaksi eri kulutustyyppiä: datan siirto ja datan käsittely. Kuten taulukosta 12 nähdään, skaalautuvan Ubuntun kohdalla datan käsittely maksoi vain 0,04 euroa ja siirto maksoi 0,60 euroa. Käsittelymaksu näyttää olevan vain skaalautuvilla resursseilla, joten sen on liityttävä kuormituksen tasaajaan. Ihmeellistä tuloksissa oli se, että skaalautuvalla Ubuntulla uudet instanssit olivat ilmaisia, sillä kulut ovat samat kuin skaalautumattomalla. Skaalautuvalla Core OS:llä kulut kuitenkin kasvoivat.

6.2 Skaalautuvuustulokset

Tässä luvussa esitellään skaalautuvuustulokset ja analysoidaan testejä hieman.

6.2.1 Azure

Taulukko 13: Azuren skaalaustulosten keskiarvo kolmesta testi kierroksesta.

Resurssi	Skaalauksen aloitus	Skaalausaika	Alasskaalauksen aloitus	Alasskaalauksen kesto
Core OS	500,38 s	218,47 s	403,83 s	593,68 s
Ubuntu	278,40 s	221,34 s	231,78 s	720,08 s
Web App	647,83 s	77,34 s	3593 s	1279,33 s

Taulukossa 13 aloitusaika on aikaväliä ensimmäisen skaalausoperaation aloittamisen ja kuormituksen alkamisen välissä. Skaalausaika on keskimääräinen aika, joka yhden instanssin skaalauksessa kului. Alasskaalauksen aloitus mittaa kuormituksen lopettamisen ja alasskaalausoperaation aloittamisen välistä aikaa. Alasskaalauksen kesto taas on aikaväli ensimmäisen alasskaalauksen aloittamisesta viimeisen alasskaalauksen lopettamiseen.

Azuressa voidaan määrittää skaalaussäännölle aikaikkuna, jota verrataan ehtoon. Tutkimuksessa käytettiin viiden minuutin ikkunaa, jossa tarkkailtiin, onko prosessorin käyttö keskimäärin yli 70 %. Aikaikkunaa saa pienemmäksikin, jopa minuuttiin, mutta koska skaalausaika on aina yli 200 s, on tuo viisi minuuttia hyvä, jotta keskiarvo ehtii edes vähän laskea, eikä tarvitse skaalata tarpeettomasti.

Vaikka aikaikkuna oli viisi minuuttia, jokaisen skaalauksen välissä oli lähes kuusi minuuttia. Kun aikaikkunan asetti arvoon kaksi minuuttia, skaalausväli oli aina lähes kolme minuuttia. Tämän takia teoriassa skaalauksen voisi aloittaa minuutin sisällä toisesta, mutta käytännössä vain kahden minuutin, koska Azuressa tuon aikaikkunan alaraja on minuutti.

Taulukosta 13 voidaan päätellä, että *Web App* oli selvästi hitain reagoimaan skaalauksiin,

mutta itse skaalauksen toteuttaminen kävi todella nopeasti verrattuna virtuaalikoneisiin.

Virtuaalikoneilla skaalauksen aloittaminen kuormituksen alkamisesta vaihteli paljon. Nopeimmin reagointi tapahtui alle kolmessa minuutissa ja hitaimmillaan yli seitsemässä minuutissa. Skaalaustoimenpiteet noudattivat aikaikkunaa ja minuutti siihen päälle.

6.2.2 Google Cloud

Taulukko 14: Google Cloudin skaalaustestien keskiarvo sekunneissa kolmesta testi kierroksesta.

Resurssi	Skaalauksen aloitus	Skaalausaika	Alasskaalauksen aloitus	Alasskaalauksen kesto
Core OS	83,67 s	29,07 s	641,64 s	547,75 s
Ubuntu	165,17 s	36,78 s	495,78 s	614,31 s
App Engine	83,67 s	41,22 s	83,67 s	727,15 s

Taulukosta 14 nähdään, että skaalausaika Core OS:llä oli keskiarvoltaan parempi, mutta siihen vaikuttaa varmasti se, että yhdellä testikierroksella kaikki instanssin skaalautuivat viidessä sekunnissa. Pisin skaalausaika oli Core OS:llä 54 sekuntia, Ubuntulla 55 sekuntia ja *App Enginellä* 54 sekuntia. Useimmissa tapauksissa ensimmäisen instanssin käynnistäminen kesti pisimpään.

Google Cloudin *instance groupit* aloittivat skaalauksen todella nopeasti ja skaalauksessa nousi kaikki kolme instanssia lähes yhtä aikaa. Alasskaalaus taas kesti todella kauan. Usein meni yli 20 minuuttia kuormituksen lopettamisesta, ennen kuin kaikki instanssit olivat poistettu. Alasskaalauksessa Ubuntulla oli yhdellä testikierroksella aloitusaika vajaa kuusi minuuttia rasituksen lopettamisesta, mikä laskee keskiarvoa Ubuntulle. Kaksi muuta kierrosta oli tuloksiltaan samoilla ajoilla Core OS:n kanssa.

Taulukosta 14 nähdään myös, että App Enginellä tulokset eivät eronneet virtuaalikoneiden tuloksista ylösskaalauksen osalta, mutta alasskaalauksen kesto oli huomattavasti lyhyempi

kuin virtuaalikoneilla. Alasskaalaus alkoi tosin jopa hitaammin kuin virtuaalikoneilla, mikä vähän tasoittaa tulosta.

6.2.3 AWS

Taulukko 15: AWS:n skaalaustestien keskiarvo sekunneissa kolmesta testi kierroksesta.

Resurssi	Skaalauksen aloitus	Skaalausaika	Alasskaalauksen aloitus	Alasskaalauksen kesto
Core OS	245,54 s	68,22 s	957,88 s	594 s
Ubuntu	285,67 s	79,46 s	1052,19 s	396 s
Elastic Beanstalk	408 s	214,324 s	1120,30 s	252,33 s

Taulukosta 15 voidaan päätellä, että AWS:n PaaS-resurssi *Elastic Beankstalk* oli kankea skaalautumaan. Kaikilla resursseilla skaalauksen aloitus kesti kauan. Skaalaus oli virtuaalikoneilla nopeaa. PaaS oli vain nopeampi poistamaan instansseja käytöstä. Tulokset olivat hyvin tasaisia kierrosten välillä.

6.3 Vertailukohde omasta fyysisestä palvelimesta

Tutkimuksessa verrattiin pilvipalvelun hintoja omaan fyysiseen palvelimeen. Palvelin on teoreettinen, eikä tutkimuksessa hankittu kyseistä palvelinta. Tässä luvussa kuvataan, miten palvelin on valittu ja miten laskenta tapahtuu.

Ongelmana palvelimen valinnassa oli se, että palvelintarpeita on erilaisia. Joku voi tarvita paljon laskentatehoa, mutta yhteyden ei tarvitse olla nopea. Toinen voi tarvita nopean yhteyden lisäksi nopean tai suuren tallennustilan. Lisäksi voi olla paljon erilaisia tarpeita, joita ei voida tutkimuksessa ottaa huomioon.

Olisi mahdollista vuokrata palvelin myös palvelinsalista, jolloin palvelinta ei tarvitse ylläpitää. Lisäksi yhteydet ovat paremmin hoidettu kuin esimerkiksi omasta toimistosta. Kaikki

palvelut nostavat kuitenkin kustannuksia. Tutkimuksessa lähetettiin hintapyyntö eräälle palvelimia vuokraavalle toimijalle saamatta vastausta. Palvelinsalia ei siis voida ottaa mukaan vertailuun.

Vertailukohdepalvelimeksi valittiin *Verkkokauppa.comista* löytyvä palvelin[1]. Palvelimessa on uusin Intelin prosessori ja muistia on 16 gigatavua. Palvelin on tornipalvelin, joka on helppo sovittaa minne vain. Kyseisessä palvelimessa ei tule kovalevyjä, joten palvelimeen valittiin SSD-kovalevy, jolloin kirjoittamis- ja lukuoperaatiot eivät ole pullonkaulana.

Palvelimen hinta oli 2600 euroa, jonka päälle varataan 600 euroa kovalevylle, eli yhteensä 3200 euroa. Oletetaan, että näyttö, hiiri ja näppäimistö ovat omasta takaa sekä jokin paikka palvelimelle eli ei oteta vuokraa huomioon.

Internetyhteys on palvelimelle äärimmäisen tärkeä. Kahdennettu yhteys olisi paras, mutta hinta on myös vähintään kaksinkertainen. Kahdennetulla yhteydellä tarkoitetaan sitä, että käytössä olisi kahden eri liittymäntarjoajan yhteydet. Palvelinsaleissa voi olla tällainen ratkaisu. Jos toinen yhteys katkeaa, voidaan vaihtaa lennosta toiseen. Tutkimuksessa lähetettiin tarjouspyyntö eräälle yritysliittymiä tarjoavalle Internet-operaattorille, mutta vastausta ei kuulunut. Yritysliittymät ovat huomattavasti kalliimpia kuin tavallisille kuluttajille suunnatut liittymät.

Liittymäntarjoajat hinnoittelevat liittymiään osoitteen mukaan. Tutkimuksessa käytettiin hinnan saamiseen sellaista osoitetta, jossa ei ole taloyhtiön Internet-liittymää, joka tekisi liittymästä huomattavasti halvemman asukkaille. Nopeimman liittymän tarjosi Elisa nopeudella 250 Mbit/s, joka maksaa 49,90 euroa/kk. Telia tarjosi vain 100 Mbit/s liittymää, joka maksaa 32,90 euroa/kk. Elisalla 100 Mbit/s liittymä on 39,90 euroa/kk. Tutkimuksen laskuissa päätettiin käyttää nopeampaa liittymää. Molemmilla operaattoreilla on avausmaksuja, mutta ei oteta niitä huomioon.

Juokseviin kuluihin laskettiin Internet-liittymän lisäksi myös sähkön hinta. Ilmastoinnista koituvia lisäkuluja tai muita mahdollisia sähkölaskua nostavia kuluja ei huomioitu. Palvelimen virtalähde oli 750 wattia, mutta käytännössä palvelin ei käytä niin paljon. Lisäksi komponentit käyttävät enemmän sähköä rasituksessa. Block ym. (2015) tutkimuksessa on tukittu

1. https://www.verkkokauppa.com/fi/product/24196/jjcmr/Lenovo-ThinkSystem-ST550-tornipalvelin

palvelinten virrankulutusta. Tutkimuksen mukaan pelkästään prosessorikuormasta ei voida päätellä virrankulutusta. Myös itse prosessi vaikuttaa virrankulutukseen. Tutkimuksessa tehtiin geneerisiä testejä, joissa oli mukana salauksen purkua ja pakkaamista.

Tässä tutkimuksessa käytettiin arvioon perustuen 30 % tuosta 750 watista eli 225 wattia. 225 watin tuntikulutuksella, saatiin päivän kulutukseksi 5,4 kilowattia. Kuukaudessa se tekee noin 162 kilowattia. `www.sahkonhinta.fi`-sivuilla voidaan koostaa vuoden 2019 koko Suomen sähkön keskimääräinen hinta määräaikaisena, joka oli kirjoitushetkellä koko vuodelle hieman päälle 0,17 euroa kilowattitunnilta. Tällöin tutkimukseen saatiin sähkön juoksevaksi kuluksi 27,54 euroa kuukaudessa.

Juoksevat kulut tähän tutkimukseen:

- Internetliittymä 49,90 euroa/kk ja
- sähkö 27,54 euroa/kk.
- Yhteensä 77,44 euroa/kk tai noin 2,58 euroa/pv (jaettu 30).

Palvelimeen investoitiin 3200 euroa, joka voidaan olettaa kestävän käytössä 6 vuotta. Jos oletetaan, että vuodessa on 365 päivää, joka kerrotaan kuudella, saadaan 2190 päivää. Palvelimen hinnaksi saatiin siis noin 1,46 euroa/pv, jolloin juoksevien kulujen kanssa oman palvelimen hinnaksi saatiin noin 4,04 euroa/pv.

7 Yhteenveto ja pohdinta

Tässä luvussa esitellään tuloksien yhteenveto ja tutkimuksesta tehdyt johtopäätökset sekä lopuksi pohdintaan.

7.1 Hintatulokset

Kun lasketaan yhteen kokonaiskustannukset PaaS-resurssien pyörittämisestä nähdään helposti, mikä oli edullisin resurssi.

Taulukko 16: Yhteenveto PaaS-resurssien perushinnoista verottomana.

Resurssi	Azure (EUR/pv)	Google Cloud (EUR/pv)	AWS (EUR/pv)
1-ytiminen kontti-PaaS	1,10	1,49	-
2-ytiminen kontti-PaaS	2,05	2,76	-
4-ytiminen kontti-PaaS	3,93	5,6	-
Skaalautumaton PaaS	1,07	1,49	0,20
Skaalautuva PaaS	1,92	1,49	0,84
Skaalautuva virtuaalikone	0,41	1,33	0,80
Skaalautumaton virtuaalikone	0,41	0,80	0,16

Taulukosta 16 nähdään, että Azurella *App Service planin* hinta oli vakio, joten jos resurssin käyttö ei tuottanut muita kuluja kuten datan käytöstä. *App Servicen* samalla hinnalla saa

ajettua useampaa kuin yhtä *App Serviceä*. *Container Instancet* olivat tässä tutkimuksessa lähes turhilta, koska ne laskuttivat koko ajan. Niitä voisi käyttää lähinnä sellaisiin käyttötarkoituksiin, joissa niitä ajetaan enintään muutaman kerran päivässä. Testissä kuitenkin oli chat-sovellus, joka on palvelin ja tuottaa prosessorille vähän kuormaa koko ajan, jota laskutettiin.

Google Cloudissa ja AWS:ssä PaaS-resurssi oli skaalautuva ja hinta kertaantui instanssien määrällä. Jos siis ajetaan kahta 2-ytimistä instanssia, hinta on kaksinkertainen verrattuna yhteen instanssiin samalla aikajaksolla. AWS ja Google Cloud ovat muutenkin melko samanlaisia. AWS oli ainoa, jolla PaaS hoidettiin täysin samoja instansseja käyttäen kuin virtuaalikoneilla. Tämä tarkoitti sitä, että niitä pystyi muokkaamaan yksitellen haluamakseen.

Oma palvelin maksoi 4,04 euroa/pv, jolloin vain neljä ytiminen Google Cloudin resurssi oli kalliimpi vaihtoehto. Omassa palvelimessa tosin oli 8 ydintä ja 16 gigatavua keskusmuistia. Oman palvelimen hinta oli kilpailukykyinen, jos asiaa tarkasteltiin ydintä kohden, jolloin hinta oli noin 0,51 euroa/pv.

Azuressa kuormituksen tasaaja ei maksanut yhtään, kuten Google Cloudissa ja AWS:ssä. Siksi Azure oli halvin skaalautuvissa resursseissa, mutta skaalautumattomana sen hinta oli paljon suurempi kuin AWS:n. Google Cloud osoittautui kaikista kalleimmaksi kaikilla resursseilla paitsi skaalautuvalla PaaS-resurssilla, vaikka Google Cloudia mainostettiin halvempana vaihtoehtona. Skaalautuva PaaS-resurssi oli saman hintainen kuin skaalautumaton, koska Google Cloudissa kuormituksen tasaaja ei maksanut, kun sitä ei käyttänyt.

Rasituksessa tulokset poikkesivat pilvipalvelun sisälläkin toisistaan. Testejä olisi pitänyt saada ajettua enemmän ja niitä varten olisi pitänyt kehittää varmempi liikennettä generoiva sovellus. *SuperBenchmarker* ei tuottanut kaikille testeille tasaista pyyntöjen määrää. Lasketaan kuitenkin rasituksen aiheuttamat kulut yhteen, eli perushintojen päälle tulleet kulut, ja jaetaan pyyntöjen määrällä, jolloin saadaan pyynnölle keskimääräinen hinta.

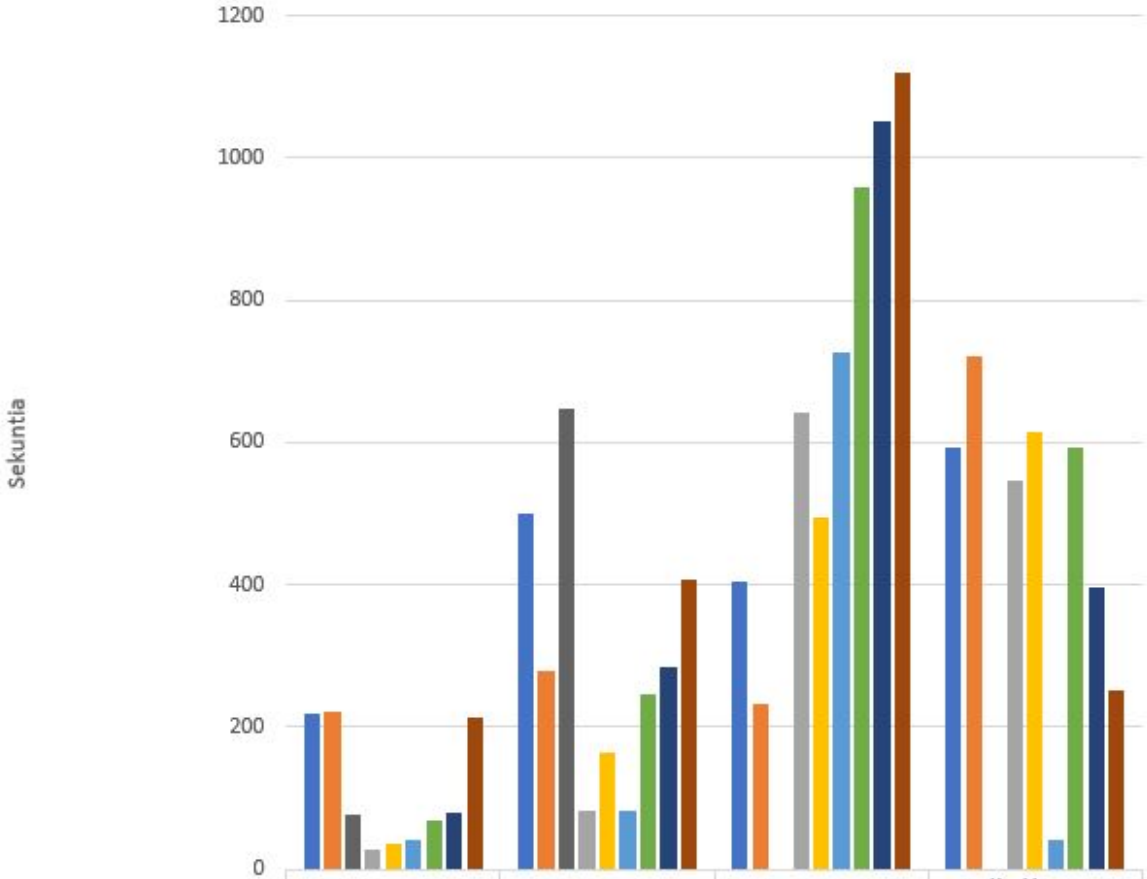
Keskimääräinen hinta miljoonaa pyyntöä kohden oli:

- Azurella:
  - VM: 0,263 euroa.

  - PaaS: 0 euroa.
- Google Cloud:
  - VM: 0,41 euroa.
  - PaaS: 0,389 euroa.
- AWS:
  - VM: 0,335 euroa.
  - PaaS: 0,268 euroa.

Azure oli näistä kolmesta pilvipalvelusta selvästi edullisin, Google Cloud kaikista kallein ja AWS näiden välissä. Azurella on etuna PaaS, johon voidaan samaan hintaan ottaa käyttöön muitakin sovelluksia ja lisärasitus ei tuota kuluja.

7.2 Skaalaustulokset



	Keskiarvo kaikista kierroksista	Keskiarvo aloitusajoista	Keskiarvo alasskaalausajoissa	Keskiarvo alasskaalauksen kestoissa
Azure VM Core OS	218,4723889	500,3846667	403,831	593,6793333
Azure VM Ubuntu	221,3372222	278,3996667	231,7763333	720,0846667
Azure PaaS	77,33844444	647,8316667	0	0
Google Cloud vm Core OS	29,07322222	83,665	641,6433333	547,7536667
Google Cloud vm Ubuntu	36,78077778	165,172	495,7846667	614,3086667
Google Cloud PaaS	41,22222222	83,66666667	727,1506667	42,23566667
AWS vm Core OS	68,21655556	245,5353333	957,8813333	594
AWS vm Ubuntu	79,45644444	285,6666667	1052,191	396
AWS PaaS	214,324	408	1120,304667	252,3333333

Kuvio 7. Yhteenveto skaalaustestien keskiarvoista sekunneissa.

Kuviossa 7 yhteenvedossa ensimmäisessä sarakkeessa on aika, joka yhdellä instanssilla kului skaalaukseen. Tässä Google Cloud oli selvästi nopein. Googlen voitto voi johtua siitä, että Googlen pilvi on luotu konttiteknologian avulla, kun muiden pilvet ovat hypervisorilla.

AWS ei kuitenkaan ollut paljon jäljessä Google cloudia virtuaalikoneiden kanssa. *Beanstalk* kuitenkin asensi AWS:n PaaS-resurssille liikaa hallinta- ja seurantasovelluksia, missä meni liian kauan. Azuren PaaS-resurssi oli AWS:n kanssa tasoissa, mutta virtuaalikoneet hävisivät reippaasti.

Kuvion 7 toisessa sarakkeessa on aloitusajat, eli kuormituksen aloituksesta skaalausoperaation alkuun kulunut aika, jossa Google Cloudin vei myös voiton. Tämä tekee Google Cloudista houkuttelevan vaihtoehdon, jos halutaan nopeasti rasitukseen reagoivaa pilvipalvelua. AWS:n ja Azuren PaaS-resurssit olivat hitaita reagoimaan, mutta virtuaalikoneiden puolella Azurellakin Ubuntu on pärjännyt AWS:n kanssa samalla tasolla.

Kuvion 7 kolmannessa sarakkeessa on alasskaalauksien ajat eli alasskaalauksen aloittamisesta viimeisen instanssin poistamisen valmistumiseen. Tässä sarakkeessa tulokset heittelivät. Azuren PaaS-resurssin aikoja ei ole kuvaajassa, koska tulokset olivat yli tunnin ja se huonontaisi kuvaajan luettavuutta. Google Cloudin ja AWS:n tulokset ovat aika tasaiset, mutta Google Cloud on vähän nopeampi. Azure vei tässä kategoriassa voiton virtuaalikoneillaan. Alasskaalauksen aika mittarina on mielenkiintoinen vain, jos halutaan säästää mahdollisimman paljon resursseja ja kuormitus on hyvin aaltoilevaa.

Alasskaalauksen kesto eli yhden instanssin poistamiseen käytetty aika on kuvion 7 viimeisessä sarakkeessa. Tässä Google Cloudin PaaS-resurssi vei selvän voiton ja toisena oli AWS:n PaaS-resurssi. Muilla resursseilla kesto oli vajaassa kuudessa minuutissa. Tutkimuksessa ei kuitenkaan selvitetty, laskuttaako pilvipalvelu tuosta kuudesta minuutista. Se voi olla merkittävä kulu, jos käytössä on kalliimpia resursseja.

Näyttää siltä, että yksikään ei ollut selvästi muita parempi kaikissa osa-alueissa vaan loistivat vain yhdessä tai kahdessa kategoriassa. Google Cloud oli paras ylösskaalauksessa, mutta Azurea heikompi alasskaalauksessa. Azure oli muutenkin todella tasainen suorittaja.

7.3 Johtopäätökset

Tutkimuksen tulokset eivät olleet johdonmukaisia, sillä tuloksissa oli huomattavaa vaihtelua saman resurssin osalta. Tämä osoittaa, että on olemassa vähintään yksi muuttuja, jota

ei ole osattu ottaa huomioon. Azuren osalta PaaS oli parempi, mutta Google Cloudin ja AWS:n osalta tulokset vaihtelivat. Kontteja oli paljon helpompi viedä pilvipalveluiden PaaS-resursseiksi kuin säätää virtuaalikoneiden kanssa. Kaikissa testissä olleilla pilvipalveluilla kontin pystytys PaaS-resurssille onnistui antamalla linkki konttirekisteriin.

Tutkimus tuotti testausalustan pilvipalveluille, jolla voidaan jatkossakin testata eri pilvipalveluita, mutta ennen kaikkea testissä olleita pilvipalveluita. Pilvipalvelut kehittyvät koko ajan, jopa tutkimuksen aikana, joten testialustasta on apua esimerkiksi yrityksille, jotka haluavat vertailla pilvipalveluita.

Lisäksi tutkimuksessa pystyttiin vastaamaan muihin kysymyksiin kuten, vastaavatko hinnat listattuja hintoja. Hinnat olivat lähellä annettuja hintoja, muttei kaikissa tapauksissa tarkalleen. Osittain erot olivat pyöristyksestä johtuvia, koska hinta oli usein oikeasti usean desimaalin tarkkuudella ilmoitettu. Kysymykseen piilokuluista ei voi antaa suoraa vastausta, koska piilokuluja huomattiin tutkimuksessa, koska pilvipalveluissa aiheutui kuluja, joita oli vaikea ennustaa, mutta joiden hinnat kyllä oli ilmoitettu. Esimerkiksi AWS:ssä datansiirrosta aiheutui kuluja, mitä ei testejä suunnitellessa tiedetty.

Armbrust ym. (2010) väittivät, että on sama, ajetaanko yhtä palvelinta tuhat tuntia vai tuhatta palvelinta tunnin. Tutkimuksen tulosten perusteella tämä voi pitää paikkansa joissain tilanteissa, mutta usein ei. Esimerkiksi kuormituksen tasaaja maksaa huomattavasti tuhannelta tunnilta, mutta alle euron tunnilta. Yhtä kuormituksen tasaajaa voidaan käyttää useampaan instanssiin, joten väite ei voi pitää paikkansa. Jos jokaisella instanssilla olisi oma kuormituksen tasaaja, kulut olisivat samat, mutta asetelmassa ei ole mieltä, koska kuormituksen tasaajan tehtävä on jakaa kuormitus resurssien kesken.

Vaikka käytössä ei olisi kuormituksen tasaajaa, jokin muu sovellus on pidettävä instansseja kasassa ja jaettava työt, josta seuraa aina lisäkuluja. Ilman työtä jakavaa sovellusta jokainen instanssi tekisi vain omaa suoritustaan ja tuhannesta instanssista ei olisi mitään hyötyä.

Toisaalta kaikissa kolmessa pilvipalvelussa laskutettiin kaikista resursseista ajan mukaan samalla tavalla jokaisen instanssin kohdalla, joten väite pitää osittain myös paikkansa. Kaikissa pilvipalveluissa oli kuitenkin ongelmia laskutuksen kanssa ja resursseja laskutettiin liian vähän tai liian paljon. Erot tulivat esiin skaalauksissa. Tutkimuksen perusteella, jos instanssit

ovat vain vähän aikaa käytössä, laskutus ei välttämättä noteeraa niitä ollenkaan. Toisaalta Googlen Cloud laskutti yli vuorokaudesta.

Durkee (2010) haaveili pilvi 2.0:sta, jossa laskutus olisi läpinäkyvää. Vaikka tässä tutkimuksessa laskutuksessa oli selittämättömiä epäjohdonmukaisuuksia, laskutus oli täysin läpinäkyvää. Laskussa luki, miten lasku on muodostettu eli käytetty aika ja hinta. Hinta määrittyy resurssin mukaan ja ne on listattu pilvipalveluiden sivuilla.

Tutkimukseen oli aluksi tarkoitus ottaa mukaan chat-sovelluksen lisäksi oikea, käytössä oleva sovellus TIM. TIM on Jyväskylän yliopiston kehittämä opetusalustasovellus, jolla tämäkin tutkimus on kirjoitettu. TIM on toteutettu käyttäen useampaa konttia ja Docker Compose -konttienhallintasovellusta.

Tutkimuksessa TIM saatiin asennettua vain Google Cloudin virtuaalipalvelimelle. Virtuaalipalvelimelle asennus oli yhtä helppoa kuin mille tahansa muulle palvelimelle tai tietokoneelle. Docker Composen käyttäminen oli hankalaa ja sitä kautta ilman ohjeita TIMin asennus oli vaikeaa. Ohjeiden avulla asennus onnistui muutamassa minuutissa.

AWS:llä olisi ollut suora tuki Docker Composelle, mutta muilla pilvipalveluilla ei. Muilla pilvipalveluilla olisi pitänyt tehdä paljon muutoksia, joita ei tutkimuksen aikataulu sallinut. Vaikka resurssien luomiseen tarvittavia komentoja on testialustassa paljon, on chat-sovelluksen asennus helpompaa kuin oikean sovelluksen asennus pilvipalveluun.

7.4 Pohdinta

Tässä tutkimuksessa toteutettiin chat-sovellus, vietiin se kolmeen eri pilvipalvelun virtuaalikoneisiin ja PaaS-resursseihin. Chat-sovelluksen toteuttaminen ja konttiteknologiaan perehtyminen vei paljon aikaa, puhumattakaan eri pilvipalveluihin perehtyminen. Testien suunnitteleminen oli haastavaa, koska testien pitää olla vertailukelpoisia.

Tässä tutkimuksessa ei ollut riittävästi aikaa, jotta testejä olisi voinut tehdä tarpeeksi, koska testien suorittaminen vei todella paljon aikaa. Kun resurssi luotiin, oli odotettava kaksi päivää, jotta pystyi ajamaan rasitustestit, jotta saadaan perushinta välipäivältä. Tämän jälkeen piti vielä odottaa rasitustestipäivä kokonaan ja resurssin pystyi poistamaan vasta rasitustes-

tien jälkeisenä päivänä. Tämän lisäksi mikään pilvipalvelu ei oikein tarjoa reaaliaikaista laskutusta, joten laskutusta piti odottaa jopa päiviä.

Tämä tutkimus kuitenkin loi hyvän pohjan jatkotutkimukselle ja antaa hyvän kuvan siitä, miten pilvipalvelut toimivat nykyisin. Tutkimus myös loi testauspohjan, jota voidaan käyttää testaamaan pilvipalveluita.

Jatkotutkimuskohteita on monia:

- *Container Instanceista* olisi selvitettävä, miten chat-sovellusta voisi kehittää, jotta se ei laskuttaisi jatkuvasti.
- Mitkä ovat parhaimmat skaalausasetukset tietyssä pilvipalvelussa.
- Miten erilaiset sovellukset kuluttavat resursseja tietyssä pilvipalvelussa.
- Kattavampia skaalaustestejä.
- Mitkä resurssit ovat hinta-laatusuhteeltaan parhaimpia.
- Laskutuksen poikkeamat testeissä.

Container Instancet ja *App Engine* laskuttivat jatkuvasti. Olisi hyvä selvittää, johtuuko se pelkästään siitä, miten chat-sovellus on kehitetty vai tapahtuuko sama myös kaikilla Asp.Net Core-sovelluksilla tai konttisovelluksilla. Jos olisi mahdollista luoda palvelinsovellus näillä resursseilla, olisi kulut voineet olla paljon pienempiä.

Skaalausasetuksissa käytettiin pelkästään prosessorikuormaa, koska kaikki pilvipalvelut tukivat sitä. Keskittyminen tiettyyn pilvipalveluun ja sen optimoiminen olisi mielekästä. Myös se, miten erilaiset resurssit tietyssä, yhdessä pilvipalvelussa, saataisiin optimoitua olisi hyvä jatkotutkimusaihe.

Skaalaustestejä tehtiin tässä tutkimuksessa kahdelle eri käyttöjärjestelmälle, kolmessa eri pilvipalvelussa, kolme kierrosta. Testit olivat työläitä, koska testejä varten ei ollut mielekästä tehdä skriptejä tai muuta automaatiota, koska niiden tekemiseen olisi mennyt kolmella eri pilvialustalla paljon enemmän aikaa kuin nyt käsin tehtyjen testien tekemiseen meni. Skaalausta pitäisi testata eri vuorokaudenaikoina, eri viikon päivinä ja eri alueilla, mikä on helpompaa automaattisilla testeillä.

Azuren *Scale Seteille* tuli uusi attribuutti `--eviction-policy`, jolla saa alasskaalau-

tumisen tapahtumaan *deallocationilla* eli instanssia ei poistettaisi kokonaan. On luultavasti nopeampi käynnistää instanssi uudelleen kuin luoda se tyhjästä. Koska testit oli jo suoritettu, ei tätä ominaisuutta ehditty tutkia.

Lisäksi tutkimuksen loppupuolella tuli idea, että olisiko nopeampi skaalaus mahdollista, jos ottaisi valmiista kovalevystä kopion, jonka uudet instanssit ottaisivat käyttöön. Silloin Docker ja chat-kontti olisivat jo asennettuna valmiiksi.

Lähteet

“Amazon EC2”. 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/ec2/.

“Amazon EC2 Instance Types”. 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/ec2/instance-types/.

“Amazon EC2 pricing”. 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/ec2/pricing/.

Anderson, Charles. 2015. “Docker [Software engineering]”. *IEEE Software* 32, numero 3 (toukokuu): 102–c3. ISSN: 0740-7459. doi:10.1109/MS.2015.62.

Andreolini, Mauro, Sara Casolari, Michele Colajanni ja Michele Messori. 2009. “Dynamic load management of virtual machines in cloud architectures”. Teoksessa *International Conference on Cloud Computing,* 201–214. Springer.

“App Service documentation”. 2018. Viitattu 15. joulukuuta. https://docs.microsoft.com/en-us/azure/app-service/app-service-web-overview.

Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica ym. 2010. “A view of cloud computing”. *Communications of the ACM* 53 (4): 50–58.

“AWS Elastic Beanstalk”. 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/elasticbeanstalk/.

“AWS free tier”. 2018. Viitattu 15. joulukuuta. https://aws.amazon.com/free/?awsf.Free.

“AWS Global Infrastructure”. 2018. Viitattu 15. joulukuuta. https://aws.amazon.com/about-aws/global-infrastructure/.

“Azure homepage”. 2018. Viitattu 8. joulukuuta. https://azure.microsoft.com/en-us/.

“Azure pricing”. 2018. Viitattu 7. lokakuuta. https://azure.microsoft.com/en-us/pricing/.

"Azure products". 2018. Viitattu 15. joulukuuta. https://azure.microsoft.com/en-us/services/.

Barroso, L. A., ja U. Hölzle. 2007. "The Case for Energy-Proportional Computing". *Computer* 40, numero 12 (joulukuu): 33–37. ISSN: 0018-9162. doi:10.1109/MC.2007.443.

Beloglazov, Anton, ja Rajkumar Buyya. 2010. "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers". Teoksessa *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing,* 577–578. Toukokuu. doi:10.1109/CCGRID.2010.45.

———. 2012. "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers". *Concurrency and Computation: Practice and Experience* 24 (13): 1397–1420. doi:10.1002/cpe.1867.

Bernstein, D. 2014. "Containers and Cloud: From LXC to Docker to Kubernetes". *IEEE Cloud Computing* 1, numero 3 (syyskuu): 81–84. ISSN: 2325-6095. doi:10.1109/MCC.2014.51.

Block, Hansfried, John Beckett, Klaus-Dieter Lange, Jeremy A Arnold, Samuel Kounev ym. 2015. "Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads". Teoksessa *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering,* 223–234. ACM.

Böhm, Markus, Stefanie Leimeister, CHRISTOPH Riedl ja Helmut Krcmar. 2010. "Cloud computing and computing evolution". *Technische Universität München (TUM), Germany.*

Chen, P. M., ja B. D. Noble. 2001. "When virtual is better than real [operating system relocation to virtual machines]". Teoksessa *Proceedings Eighth Workshop on Hot Topics in Operating Systems,* 133–138. Toukokuu. doi:10.1109/HOTOS.2001.990073.

Cito, Jürgen, Philipp Leitner, Thomas Fritz ja Harald C. Gall. 2015. “The Making of Cloud Applications: An Empirical Study on Software Development for the Cloud”. Teoksessa *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,* 393–403. ESEC/FSE 2015. Bergamo, Italy: ACM. ISBN: 978-1-4503-3675-8. doi:10.1145/2786805.2786826.

“Cloud locations”. 2018. Viitattu 16. joulukuuta. https://cloud.google.com/about/locations/.

“Cloud Products”. 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/products/.

Dignan, Larry. 2018. “Top cloud providers 2018: How AWS, Microsoft, Google Cloud Platform, IBM Cloud, Oracle, Alibaba stack up”. Viitattu 8. joulukuuta. https://www.zdnet.com/article/cloud-providers-ranking-2018-how-aws-microsoft-google-cloud-platform-ibm-cloud-oracle-alibaba-stack/.

Dillon, Tharam, Chen Wu ja Elizabeth Chang. 2010. “Cloud Computing: Issues and Challenges”. Teoksessa *2010 24th IEEE International Conference on Advanced Information Networking and Applications,* 27–33. Huhtikuu. doi:10.1109/AINA.2010.187.

Dua, R., A. R. Raja ja D. Kakadia. 2014. “Virtualization vs Containerization to Support PaaS”. Teoksessa *2014 IEEE International Conference on Cloud Engineering,* 610–614. doi:10.1109/IC2E.2014.41.

Durkee, Dave. 2010. “Why Cloud Computing Will Never Be Free”. *Commun. ACM* (New York, NY, USA) 53, numero 5 (toukokuu): 62–69. ISSN: 0001-0782. doi:10.1145/1735223.1735242.

Foster, Ian, Yong Zhao, Ioan Raicu ja Shiyong Lu. 2008. “Cloud computing and grid computing 360-degree compared”.

Fox, Armando, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin ja Ion Stoica. 2009. “Above the clouds: A berkeley view of cloud computing”. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (13): 2009.

“GCP Free Tier”. 2018. Viitattu 16. joulukuuta. https://cloud.google.com/free/docs/gcp-free-tier.

“GOOGLE APP ENGINE”. 2018. Viitattu 16. joulukuuta. https://cloud.google.com/appengine/.

“Google Cloud Platform Free Tier”. 2018. Viitattu 16. joulukuuta. https://cloud.google.com/free/.

“Google Cloud Status Dashboard”. 2018. Viitattu 16. joulukuuta. https://status.cloud.google.com/summary.

Guo, Lizheng, Shuguang Zhao, Shigen Shen ja Changyuan Jiang. 2012. “Task scheduling optimization in cloud computing based on heuristic algorithm”. *Journal of networks* 7 (3): 547.

Gupta, Prashant, A. Seetharaman ja John Rudolph Raj. 2013. “The usage and adoption of cloud computing by small and medium businesses”. *International Journal of Information Management* 33 (5): 861–874. ISSN: 0268-4012. doi:https://doi.org/10.1016/j.ijinfomgt.2013.07.001.

Hevner, Alan, ja Samir Chatterjee. 2010. “Design science research in information systems”. Teoksessa *Design research in information systems,* 9–22. Springer.

“How reliable is Google Cloud?” 2018. Viitattu 16. joulukuuta. https://medium.com/google-cloud/how-reliable-is-google-cloud-4d219a4f7e56.

“IBM ostaa itsensä mukaan tulikuumiin it-kemuihin”. 2019. Viitattu 14. kesäkuuta. https://www.tivi.fi/uutiset/ibm-ostaa-itsensa-mukaan-tulikuumiin-it-kemuihin/cea22073-9dd6-381c-b414-5c86d4d5193e.

Ismail, Bukhary Ikhwan, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke ja Ong Hong Hoe. 2015. “Evaluation of Docker as Edge computing platform”. Teoksessa *2015 IEEE Conference on Open Systems (ICOS),* 130–135. Elokuu. doi:10.1109/ICOS.2015.7377291.

Joy, A. M. 2015. “Performance comparison between Linux containers and virtual machines”. Teoksessa *2015 International Conference on Advances in Computer Engineering and Applications,* 342–346. Maaliskuu. doi:10.1109/ICACEA.2015.7164727.

Mell, Peter M., ja Timothy Grance. 2011. *SP 800-145. The NIST Definition of Cloud Computing.* Tekninen raportti. Gaithersburg, MD, United States.

Merkel, Dirk. 2014. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. *Linux J.* (Houston, TX) 2014, numero 239 (maaliskuu). ISSN: 1075-3583. http://dl.acm.org/citation.cfm?id=2600239.2600241.

Pahl, C. 2015. “Containerization and the PaaS Cloud”. *IEEE Cloud Computing* 2, numero 3 (huhtikuu): 24–31. ISSN: 2325-6095. doi:10.1109/MCC.2015.51.

Pandey, Suraj, Linlin Wu, Siddeswara Mayura Guru ja Rajkumar Buyya. 2010. “A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments”. Teoksessa *2010 24th IEEE International Conference on Advanced Information Networking and Applications,* 400–407. Huhtikuu. doi:10.1109/AINA.2010.31.

“Postmortem: VSTS 4 September 2018”. 2018. Viitattu 10. syyskuuta 2018. https://blogs.msdn.microsoft.com/vsoservice/?p=17485.

“Products and services”. 2018. Viitattu 16. joulukuuta. https://cloud.google.com/products/.

“Run multiple services in a container”. 2019. Viitattu 9. elokuuta. https://docs.docker.com/config/containers/multi-service_container/.

Smith, J. E., ja Ravi Nair. 2005. “The architecture of virtual machines”. *Computer* 38, numero 5 (toukokuu): 32–38. ISSN: 0018-9162. doi:10.1109/MC.2005.173.

Smith, J., ja R. Nair. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes.* The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science. ISBN: 9780080525402. https://books.google.fi/books?id=JPhQw41vD2MC.

"Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region". 2018. Viitattu 16. joulukuuta. https://aws.amazon.com/message/41926/.

Von Alan, R Hevner, Salvatore T March, Jinsoo Park ja Sudha Ram. 2004. "Design science in information systems research". *MIS quarterly* 28 (1): 75–105.

Xiao, Zhen, Weijia Song ja Qi Chen. 2013. "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment". *IEEE Transactions on Parallel and Distributed Systems* 24, numero 6 (kesäkuu): 1107–1117. ISSN: 1045-9219. doi:10.1109/TPDS.2012.283.

Yang, Xiaowei, Ang Li, S. Kandula ja Ming Zhang. 2011. "Comparing Public-Cloud Providers". *IEEE Internet Computing* 15 (maaliskuu): 50–53. ISSN: 1089-7801. doi:10.1109/MIC.2011.36.

Youseff, Lamia, Maria Butrico ja Dilma Da Silva. 2008. "Toward a unified ontology of cloud computing". Teoksessa *2008 Grid Computing Environments Workshop,* 1–10. IEEE.