Pasi Riikonen

Pilvipalveluratkaisun skaalautuminen

Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2018

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Riikonen Pasi

Työn nimi: Pilvipalveluratkaisun skaalautuminen

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: Pilvipalvelut, skaalautuminen, virtualisointi, avoin lähdekoodi

Työn tilaajana oli Bittium Oyj:n Kajaanin yksikkö, Bittium Wireless Oy. Bittium on luotettaviin ja turvallisiin viestintä- ja liitettävyysratkaisuihin erikoistunut yritys. Bittium kehittää myös tietoturvaratkaisuja mobiililaitteille ja kannettaville tietokoneille sekä terveydenhuollon teknologian tuotteita ja palveluja.

Työn tarkoituksena oli tutkia avoimen lähdekoodin ohjelmistoja Bittiumin kehittämää pilvipalvelua varten. Työssä tutkittiin pilvipalvelun kykyä vastata skaalautumiseen liittyviin haasteisiin. Skaalautuminen kuvastaa ohjelmistojen kykyä kasvaa tai pienentyä vaihtelevan tarpeen mukaan. Skaalautumisen avulla ohjelmistot pystyvät vastaamaan korkeaan tarpeeseen ilman, että resursseja varataan turhaan tarpeen vähentyessä. Lisäksi pilvipalvelua tutkittiin mahdollisten ongelmakohtien ja puutteiden havaitsemiseksi.

Raportissa tutkitaan avoimen lähdekoodin ohjelmistoja, joita pilvipalvelun kehitykseen on suunniteltu käytettävän. Ohjelmistojen tutkimisessa keskitytään skaalautumiseen liittyviin ominaisuuksiin, kuten tiedon hajautettuun säilytykseen ja ohjelmistojen rinnakkaissuoritukseen. Raportissa kuvaillaan myös itse pilvipalvelun rakenne ja kuinka käsiteltyjä ohjelmistoja voidaan käyttää sen toteuttamiseen.

Työssä tutkittiin erityisesti pilvipalvelun eri toimitustapoja, eli millä tavoin pilvipalvelua voidaan tarjota asiakkaille, sekä ohjelmistojen yhteensopivuutta ja erityispiirteitä. Työssä havaittiin ohjelmistojen välisiä yhteensopivuusongelmia, jotka täytyy ottaa huomioon pilvipalvelun jatkokehityksessä. Työssä tutkituista ohjelmistoista havaittiin myös skaalautumiseen liittyviä puutteita, joille löydettiin ratkaisukeinoja.

Työssä tehdyn tutkimuksen perusteella pystytään toteamaan, että pilvipalvelu kykenee nykyisessä muodossaan vastaamaan suurimpaan osaan sille asetetuista haasteista. Kaikkiin haasteisiin vastaaminen vaatii kuitenkin jatkokehitystä.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Abstract

Author: Riikonen Pasi

Title of the Publication: Scalability of a Cloud Service Solution

Degree Title: Bachelor of Engineering, Information Technology Engineering

Keywords: Cloud services, scalability, virtualization, open source

This work was ordered by the Bittium Corporation's Kajaani unit, Bittium Wireless Ltd. Bittium is a company that specializes in developing secure and reliable connectivity and communications solutions. Bittium also provides information security solutions for mobile devices and portable computers, as well as healthcare technology related products and services.

The aim of this work was to study open source software solutions for a cloud service that Bittium is developing. The work sought to find whether the cloud service was able to respond to scalability related challenges. Scalability describes the ability of a software to grow or shrink to meet changing demands. For example, a scalable piece of server software can serve large influxes of customers, while freeing up valuable computing resources at times of inactivity. The cloud service in question was also studied to detect possible issues and deficiencies in its current design.

The report studies the selected pieces of open source software with a research emphasis on scalability related features, such as distributed storing of data and parallel processing. The report describes the general architecture of the cloud service, as well as how the selected pieces of software can be used to implement it.

The work studied in more detail the different ways of delivering the cloud service and the compatibility and special features of the software. The work detected issues related to compatibility between the software, which need to be considered as the cloud service is developed further. The work also found solutions to scalability related deficiencies that were detected in the software.

In conclusion, the cloud service can respond to most of the presented challenges. Further development is however needed, before the cloud service can respond to all of them.

Sisällys

Lyhenteet ja määritelmät

AMQP	Advanced Message Queuing Protocol, avoimen standardin viestinvälitysprotokolla
DataNode	Hadoopin osa, joka säilöö ja hallitsee tietoa Hadoopin hajautetussa tiedostojärjestelmässä
HDFS	Hadoop Distributed File System, Hadoopin hajautettu tiedostojärjestelmä
IaaS	Infrastructure as a Service, infrastruktuuri palveluna
JournalNode	Hadoopin osa, jota käytetään NameNodejen synkronisointiin
NameNode	Hadoopin osa, joka hallitsee Hadoopin hajautettua tiedostojärjestelmää
NIST	National Institute of Standards and Technology, yhdysvaltalainen virasto
PaaS	Platform as a Service, sovellusalusta palveluna
ResourceManager	YARN:n osa, joka hallitsee koko Hadoop-järjestelmän resursseja
SaaS	Software as a Service, sovellus palveluna
YAML	YAML Ain’t Markup Language, datan serialisointiin keskittyvä merkintäkieli
YARN	Yet Another Resource Negotiator, resurssienhallintaan keskittyvä Hadoopin järjestelmä

1 Johdanto

Pilvipalvelut ovat nopeasti yleistyvä ja kehittyvä tietotekniikan ala. Pilvipalvelut ovat vastaus alati kasvavan käyttäjämäärän palvelemiseen kustannustehokkaasti, olivat käyttäjät missä päin maailmaa tahansa.

Työn tilaajana toimii Bittium Oyj:n Kajaanin yksikkö, Bittium Wireless Oy. Bittium on suomalainen luotettaviin ja turvallisiin viestintä- ja liitettävyysratkaisuihin erikoistunut yritys. Bittium Oyj oli aikaisemmalta nimeltään Elektrobit Oyj. Yhtiön nimi muuttui sen myydessä oikeudet Elektrobit-nimeen ja Automotive-liiketoimintansa 1.7.2015. Viestintä- ja liitettävyysratkaisujen lisäksi Bittium kehittää tietoturvaratkaisuja mobiililaitteille ja kannettaville tietokoneille sekä terveydenhuollon teknologian tuotteita ja palveluja. [1.]

Bittiumilla on tuotekehitysprojekti, jossa kehitetään pilvipalvelua avoimen lähdekoodin ohjelmistojen avulla. Pilvipalvelun tarkoituksena on analysoida ja tallentaa ihmisestä mitattua tietoa. Tämä työ keskittyy pilvipalvelun kehityksen haasteisiin, erityisesti tiedon tallentamiseen ja pilvipalvelun tietovarannon skaalautumiseen liittyviin ongelmiin. Tällä hetkellä tiedossa olevia haasteita ovat tietovarannon kasvaminen, käyttäjämäärän kasvaminen sekä tiedon saatavuus kaikissa tilanteissa.

Ongelmia tutkitaan pilvipalvelun eri toimitusmallien näkökulmasta. Työllä pyritään löytämään käytännön ratkaisuja havaittuihin ongelmiin sekä etsimään mahdollisia kompastuskiviä ja kehityskohteita. Työ toteutetaan tutkimalla tiettyjä avoimen lähdekoodin ohjelmistoja ja pohtimalla niiden soveltuvuutta Bittiumin tarpeeseen.

Seuraavassa osassa tutustutaan pilvipalveluiden teoriaan, jonka jälkeen tutkitaan avoimen lähdekoodin ohjelmistoja, joita pilvipalvelussa on suunniteltu käytettävän. Seuraavaksi tutustutaan skaalautumiseen, minkä jälkeen esitellään pilvipalvelun rakenne sekä miten ohjelmistoja käytetään siinä. Esittelyn jälkeen käsitellään pilvipalvelun skaalautumista ohjelmistojen avulla. Loppua kohden kerrataan pilvipalvelulle tiedossa olleet haasteet, ja pohditaan, kuinka se kykenee työn aikaisessa muodossaan vastaamaan niihin. Lopuksi käsitellään työn aikana havaittuja puutteita, sekä esitetään kehityskohteita ja jatkotutkimusaiheita, minkä jälkeen raportti päätetään yhteenvetoon.

2 Pilvipalveluista yleisesti

Pilvipalvelut ovat internetin kautta tarjottuja ohjelmistopalveluja. Yhdysvaltalainen NIST (National Institute of Standards and Technology) määrittelee pilvipalvelun viiden keskeisen ominaisuuden, kolmen palvelumallin sekä neljän toimitusmallin avulla [2].

2.1 Keskeiset ominaisuudet

NIST:n määrittelemät viisi ominaisuutta ovat:

1. Itsepalvelu
2. Laaja käyttökyky
3. Jaetut resurssivarannot
4. Nopea joustavuus
5. Palvelunkäytön mittaaminen

Itsepalvelulla tarkoitetaan, että asiakas voi varata tarvitessaan pilvestä resursseja, kuten talletustilaa ja prosessointitehoa ilman, että asiakkaan tarvitsee olla erikseen yhteydessä palveluntarjoajaan. Asiakkaan ei tarvitse tietää pilven infrastruktuurista mitään, sillä itse fyysiset laitteet ja niiden hallinta ovat palveluntarjoajan vastuulla. [2.]

Laajalla käyttökyvyllä tarkoitetaan, että tarjottu palvelu on helposti saatavilla yleisten mekanismien kautta monenlaisilla päätelaitteilla. Asiakas voi esimerkiksi käyttää palvelua puhelimellaan tai työkoneellaan käyttäen yksinkertaista web-käyttöliittymää. Palveluun pääsy ja sen käyttäminen on mahdollista riippumatta asiakaskohtaisista laite- ja ohjelmistoeroista. [2.]

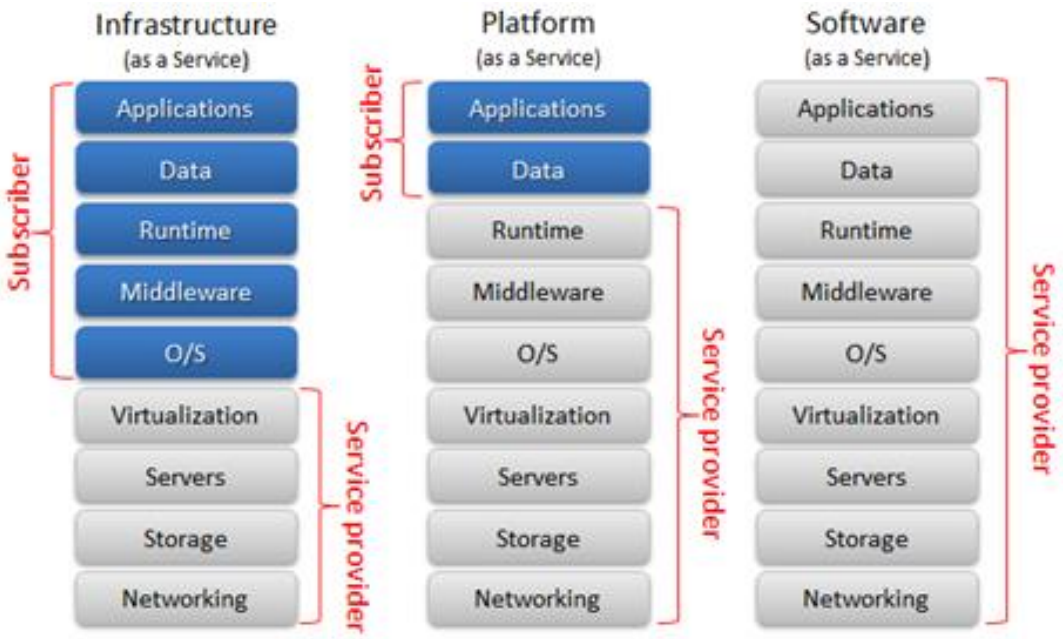
Pilvipalveluille ominaista on resurssien kerääminen yhteen varantoon. Varanto sisältää sekä fyysisisiä että virtuaalisia resursseja, joita palvelu dynaamisesti siirtää asiakkaiden käyttöön heidän tarpeensa mukaisesti. Varannon avulla asiakkaalta piilotetaan itse resurssien sijainti. Asiakas voi kuitenkin tarpeen mukaan pyytää, että resurssit tarjotaan esimerkiksi tietystä konesalista tai maanosasta. [2.]

Nopea joustavuus kuvastaa pilvipalveluiden kykyä vastata vaihtelevaan kysynnän määrään. Joustavuus perustuu pilvien kykyyn varata ja vapauttaa resursseja kysynnän mukaisesti. Asiakkaalle tämä näkyy loputtomana resurssivarantona, josta hän voi milloin tahansa varata itselleen resursseja käyttöön. [2.]

Palvelunkäytön mittaaminen on ominaisuus, jota käytetään pilven resurssienkäytön optimointiin sekä mahdollisesti asiakkaan laskuttamiseen. Palvelu mittaa palvelun tyypille merkityksellisiä asioita, kuten esimerkiksi prosessorin ja säilytystilan käyttöä. Mittaamisen avulla luodaan läpinäkyvyyttä pilven toimintaan ja tehostetaan sen suorituskykyä. [2.]

2.2 Palvelumallit

Pilvipalveluilla on kolme päämallia: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) ja SaaS (Software as a Service). Kuvassa 1 esitetään, miten vastuu, ja samalla valinnanvapaus, jakautuu palveluntarjoajan ja asiakkaan välillä eri palvelumalleissa.



Kuva 1. Vastuiden jako eri palvelumalleissa (mukailtu [3])

2.2.1 Infrastructure as a Service

IaaS-mallissa asiakas saa palveluntarjoajalta resursseja, kuten prosessointitehoa ja säilöntätilaa, joiden avulla hän pystyy suorittamaan ohjelmistojaan. Palveluntarjoaja hallinnoi itse palvelun infrastruktuuria, eikä asiakas pääse vaikuttamaan sen rakenteeseen. IaaS-mallin palvelut skaalautuvat käyttömäärän mukaisesti, ja asiakasta laskutetaan käytettyjen resurssien perusteella [4]. [2, s. 2.]

2.2.2 Platform as a Service

PaaS-mallissa asiakkaalle tarjotaan sovellusalusta, jonka päälle hän voi rakentaa palvelun tarjoamien työkalujen avulla sovelluksen tai suorittaa alustan kanssa yhteensopivia sovelluksia [2, s. 2–3]. Sovellusalustat tukevat koko sovelluskehitysprosessia rakentamisesta toimituksen jälkeiseen hallinnointiin, jättäen infrastruktuurin hallinnoinnin ja sovelluslisensseistä koituvat kulut palveluntarjoajan vastuulle [5].

2.2.3 Software as a Service

SaaS-mallissa asiakkaalle tarjotaan valmis pilvisovellus tai -sovelluksia, joita asiakas käyttää joko verkkoselaimen tai soveltuvan ohjelmiston kautta [2, s. 2]. Palveluntarjoaja vastaa sovelluksen saatavuudesta ja toimivuudesta, ja asiakasta laskutetaan esimerkiksi kuukausimaksulla tai käyttömäärän mukaan [6].

2.3 Toimitusmallit

Toimitusmallit kuvaavat, kuinka pilven käyttäjäkunta rajautuu. Toimitusmallin valinta riippuu pilven käyttötavasta. Yritykset voivat esimerkiksi pitää sisäisen tiedon ja palvelut turvassa yksityisessä pilvessä, kun taas SaaS-mallin palveluita voidaan tarjota mahdollisimman monelle julkisen pilven kautta.

2.3.1 Yksityinen pilvi

Yksityisen pilven omistaa yksi organisaatio, joka myös hallinnoi vapaasti sen resursseja [2]. Pilven fyysisen infrastruktuurin omistaa ja ylläpitää joko itse organisaatio tai ulkopuolinen tekijä ja voi omistajasta riippuen sijaita joko organisaation omissa tiloissa tai niiden ulkopuolella. Pääsy yksityiseen pilveen pyritään rajaamaan organisaation sisälle, mutta organisaatio voi halutessaan antaa pääsyn muillekin osapuolille. Pilven omistaja voi esimerkiksi antaa alihankkijalle osittaisen tai täyden pääsyn omaan yksityiseen pilveensä. [7.]

2.3.2 Yhteisöpilvi

Yhteisöpilvessä pilven infrastruktuuri on jaettu tietylle käyttäjäkunnalle [2]. Pilven infrastruktuurin omistajana ja ylläpitäjänä voi toimia yksi tai useampi yhteisöön kuuluva asiakas tai kolmas osapuoli, mutta sen resursseja hallitsee yksi tai useampi yhteisön jäsen. Pilvi voi sijaita omistajan tiloissa tai niiden ulkopuolella. Pilveen pääsy ja käyttöoikeus sen palveluihin on rajattu yhteisön sisälle. Yhteisön jäsenillä on jokin suhde toisiinsa, ja pilvi jaetaan jonkin yhteisen syyn, kuten yhteisen toimialan, tietoturvan tai samankaltaisten toimintatapojen vuoksi. [7.]

2.3.3 Julkinen pilvi

Julkisessa pilvimallissa infrastruktuuri on suuren yleisön avoimesti käytettävissä [2]. Pilven resursseja hallitsee palvelun tarjoaja, jonka tiloissa itse pilvi myös sijaitsee. Palveluntarjoaja ja siten pilven omistaja voi esimerkiksi olla koulutuslaitos, yritys, hallinnollinen elin taikka jokin yhdistelmä näistä. Julkisen pilvipalvelun käyttöön liittyy hyvin vähän rajoituksia, tai ei ollenkaan. Palveluun pääsyä voivat kuitenkin rajoittaa ulkoiset tekijät, kuten paikallinen laki. [7.]

2.3.4 Hybridipilvi

Hybridipilvi on kahden tai useamman toimitusmallin yhdistelmä. Pilvessä voi esimerkiksi olla omat palvelunsa julkiselle ja yksityiselle puolelle, joista kummatkin pysyvät omina kokonaisuuksinaan. Pilvet sitovat toisiinsa jokin standardisoitu tai patentoitu teknologia, mikä mahdollistaa resurssien jakamisen ja yhteistoiminnan pilvien välillä. Pilven omistaja ja hoitaja on joko asiakasorganisaatio tai kolmas osapuoli, ja pilvi sijaitsee joko omistajan tiloissa tai niiden ulkopuolella [7]. [2.]

3 Ohjelmistot

Työssä käsiteltävään pilveen on suunniteltu käytettävän ilmaisia, avoimen lähdekoodin ohjelmistoja. Ohjelmistojen halutut ominaisuudet ovat korkea saatavuus (engl. *high availability*) sekä kyky skaalautua pilviympäristössä. Korkea saatavuus tarkoittaa ohjelmistojen kykyä säilyä toiminnallisena, vaikka ne törmäisivät johonkin fyysiseen tai ohjelmistopohjaiseen häiriöön. Korkea saatavuus voidaan saavuttaa joko ohjelmistojen omien ominaisuuksien avulla tai luvussa 4.3 esiteltävien ohjelmistokonttien avulla.

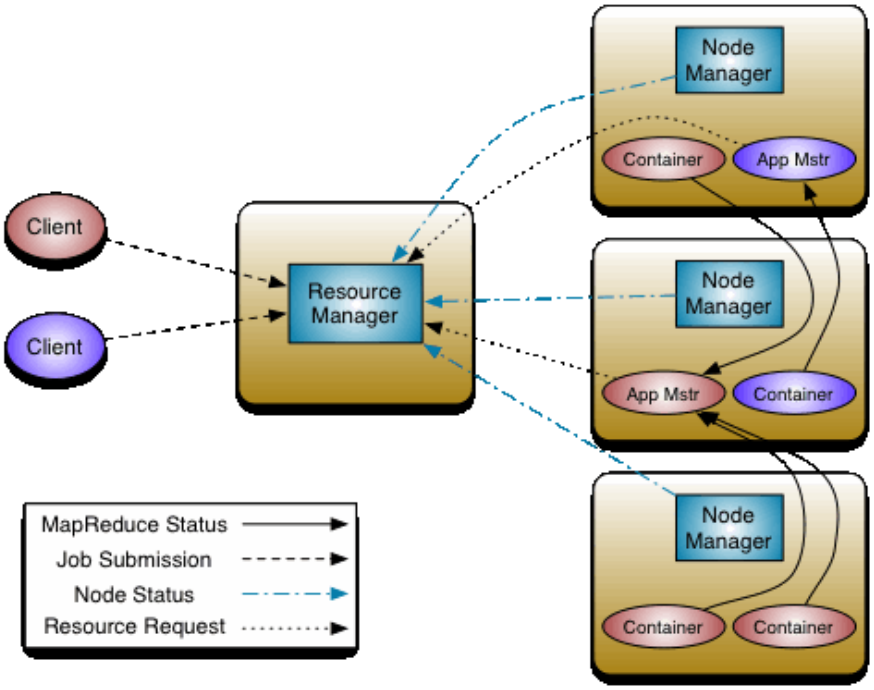
Monet tässä luvussa esitetyt ohjelmistot käyttävät tiedon säilyvyyden varmistamiseksi replikointia. Replikoinnilla tarkoitetaan tiedon kopiointia useampaan paikkaan, kuten palvelinklusterin eri solmuille (engl. *node*). Solmulla tarkoitetaan yksittäistä verkkoon liitettyä laitetta, kuten palvelinta. Kopiot eli replikat pidetään synkronisoinnin avulla ajan tasalla toistensa kanssa. Replikoinnilla varmistetaan, että tietoa ei menetetä eikä sen saatavuus kärsi, vaikka jokin järjestelmän solmu lakkaisi toimimasta. [8.]

3.1 Apache Hadoop

Apache Hadoop on ohjelmistokehys (engl. *software framework*), jonka avulla kehitetään ohjelmistoja, joissa tiedon säilöntä ja prosessointi tapahtuvat hajautetusti usean palvelimen tai konesalin välillä. Hadoopin pääkomponentit ovat resursseja ja järjestelmän töitä hallinnoiva YARN (Yet Another Resource Negotiator), hajautettu tiedostojärjestelmä HDFS (Hadoop Distributed File System) sekä hajautetun datan prosessointiin keskittyvä järjestelmä Hadoop MapReduce. Komponentit itse rakentuvat useista erikoistuneista taustaprosesseista (engl. *daemon*). [9.]

3.1.1 Hadoop YARN

YARN hallinnoi koko Hadoop-klusterin resursseja ja työtehtäviä. YARN sisältää useita prosesseja, jotka yhdessä luovat järjestelmän, joka suorittaa työtehtäviä hajautetusti skaalautuvalla tavalla. YARN:n rakenne on esitetty kuvassa 2. [10.]



Kuva 2. YARN:n rakenne [10]

YARN:n ylin toimija on järjestelmän resursseja hallinnoiva ResourceManager-prosessi. ResourceManager päättää resurssien jakamisesta järjestelmässä suoritettaville ohjelmille, pyrkien samalla optimoimaan koko klusterin resurssienkäyttöä. [11.]

Järjestelmässä on aina vain yksi aktiivinen ResourceManager, mutta järjestelmään voidaan lisätä vikasietoisuuden vuoksi useita ResourceManagereita. Toiset ResourceManagerit ovat valmiustilassa, ja jokin niistä käynnistyy joko järjestelmän ylläpitäjän toimesta, tai automaattisesti, mikäli aktiivinen ResourceManager siirtyy toimimattomaan tilaan. Ominaisuuden käyttö vaatii kuitenkin luvussa 6.2 esiteltävän palvelun käyttöönoton. ResourceManager voidaan myös konfiguroida siten, että se käynnistää itsensä uudelleen vikatilanteissa [12]. [13.]

Jokainen klusterin solmu sisältää NodeManager-prosessin. NodeManager valvoo resurssikontteja, jotka kuvastavat tiettyä määrää tietylle ohjelmistolle varattuja tietokoneresursseja, kuten muistia ja prosessointitehoa. NodeManager raportoi ResourceManagerille resurssikonttien tilan ja kuinka paljon ne käyttävät resursseja. NodeManager raportoi myös solmun yleisestä terveystilasta tietoa, joka sisältää mm. tiedon solmun jäljellä olevasta säilytystilasta. Tiedon perusteella ResourceManager voi päättää, määritetäänkö ohjelmistoille resurssikontteja kyseisestä solmusta. [10.] [14.]

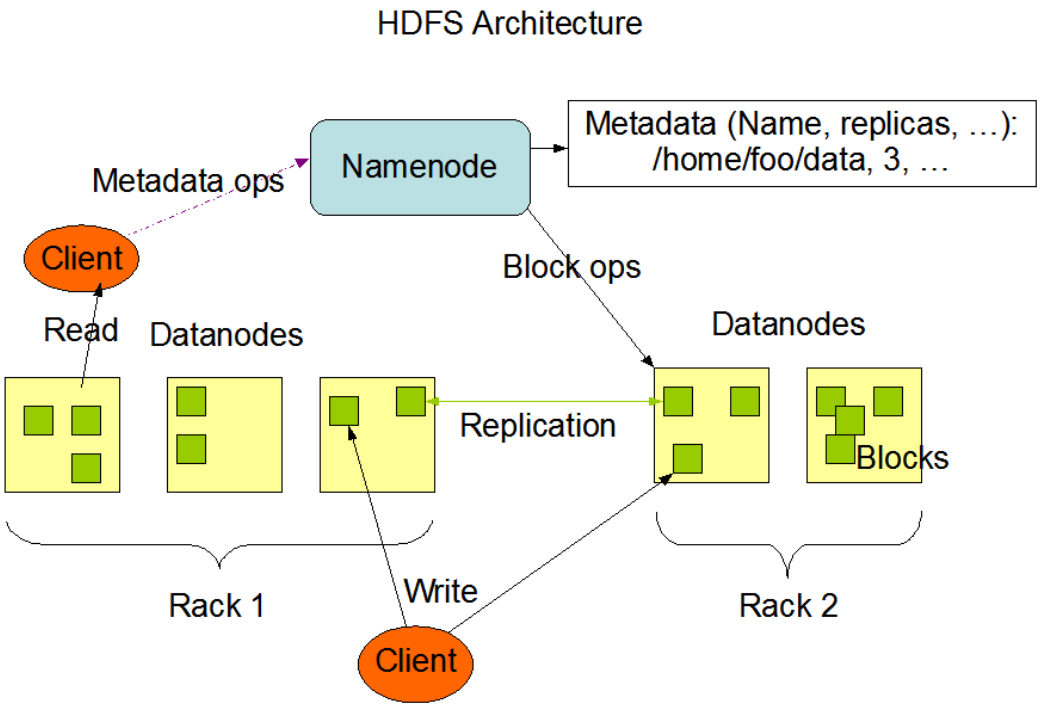
Jokaista järjestelmässä suoritettavaa ohjelmaa kohti on niitä valvova ApplicationMaster-prosessi. ApplicationMasterien vastuulla on pyytää ohjelmien suorittamista varten resurssikontteja ResourceManagerilta. ApplicationMasterit käynnistävät ohjelmat ja valvovat NodeManagerien avustuksella niiden tilaa. [10.]

3.1.2 Hadoop Distributed File System

Hadoopin hajautettu tiedostojärjestelmä on erittäin skaalautuva: yksi HDFS-instanssi voi koostua tuhansista palvelimista, joista jokainen osallistuu datan prosessointiin ja säilöntään. HDFS on ohjelmoitu Java-ohjelmointikielellä, mikä tekee siitä helposti siirrettävän erilaisten alustojen välillä. Järjestelmä on suunniteltu käytettäväksi tavallisilla, matalakustanteisilla laitteistokomponenteilla, ja Hadoopin ainoa ohjelmistovaatimus on kyky suorittaa Java-ohjelmia. Massiivisen skaalan takia laiterikot ovat yleisiä ja HDFS on suunniteltu havaitsemaan ne ja palautumaan niistä automaattisesti. [15.]

HDFS organisoi tiedostot hierarkkisesti monen muun tavanomaisen tiedostojärjestelmän tapaan. Käyttäjä tai tiedostojärjestelmää käyttävä ohjelma voi luoda ja poistaa hakemistoja ja tiedostoja, nimetä niitä uudelleen sekä siirtää tiedostoja hakemistojen välillä. HDFS tukee käyttörajoja ja pääsyoikeuksia, ja sen rakenne ei estä käyttäjiä lisäämästä tukea puuttuville ominaisuuksille. [15.]

HDFS on suunniteltu käsittelemään erittäin suurta määrää tiedostoja, joista jokainen voi olla kooltaan teratavujen kokoisia. HDFS keskittyy tiedonsiirron määrään nopeuden sijasta. Tiedonsiirron hitaus tekee siitä paremmin soveltuvan suuren tietomäärän prosessointiin kuin interaktiiviseen käyttöön. Tiedostojärjestelmän rakenne on esitetty kuvassa 3. [15.]

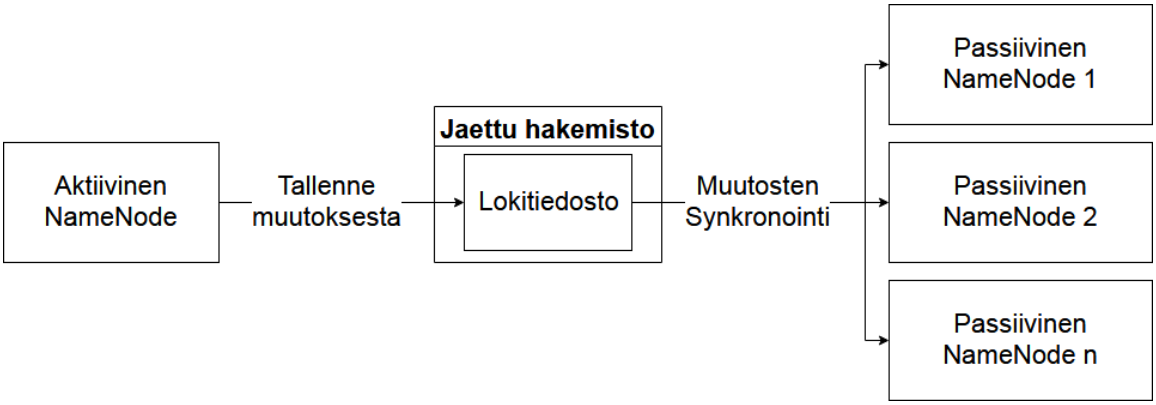


Kuva 3. Hadoopin hajautetun tiedostojärjestelmän rakenne [15]

HDFS koostuu NameNodeista ja DataNodeista. NameNode on järjestelmän osa, joka hallitsee tiedostojärjestelmän nimiavaruutta sekä käyttäjien pääsyä tiedostojärjestelmään. NameNode suorittaa myös muut tiedostojärjestelmän nimiavaruuteen liittyvät tehtävät, kuten tiedostojen avaamisen ja sulkemisen sekä tiedostojen ja hakemistojen uudelleennimeämisen. Lisäksi NameNode määrittää tiedostolohkojen sijoituksen DataNodeihin. [15.]

NameNode pitää muistissaan kaikki tiedostojärjestelmässä tapahtuneet muutokset. NameNodeja voi yhdessä klusterissa olla yksi tai useampi, mutta vain yksi niistä on aktiivinen. Toiset NameNodet ovat passiivisessa tilassa valmiina siirtymään aktiiviseen tilaan, mikäli sen hetkinen aktiivinen NameNode putoaa verkosta tai siirtyy toimimattomaan tilaan. Automaattinen siirto vaatii kuitenkin luvussa 6.2 esiteltävän palvelun käyttöönoton. [16.]

NameNodejen synkronointiin käytetään joko verkkolevyjärjestelmässä (engl. *Network File System*) olevaa NameNodejen kesken jaettua hakemistoa, tai Hadoopin Quorum Journal Manager -ominaisuutta. NameNodejen välinen synkronointi jaetun hakemiston avulla on havainnollistettu kuvassa 4. [16.]



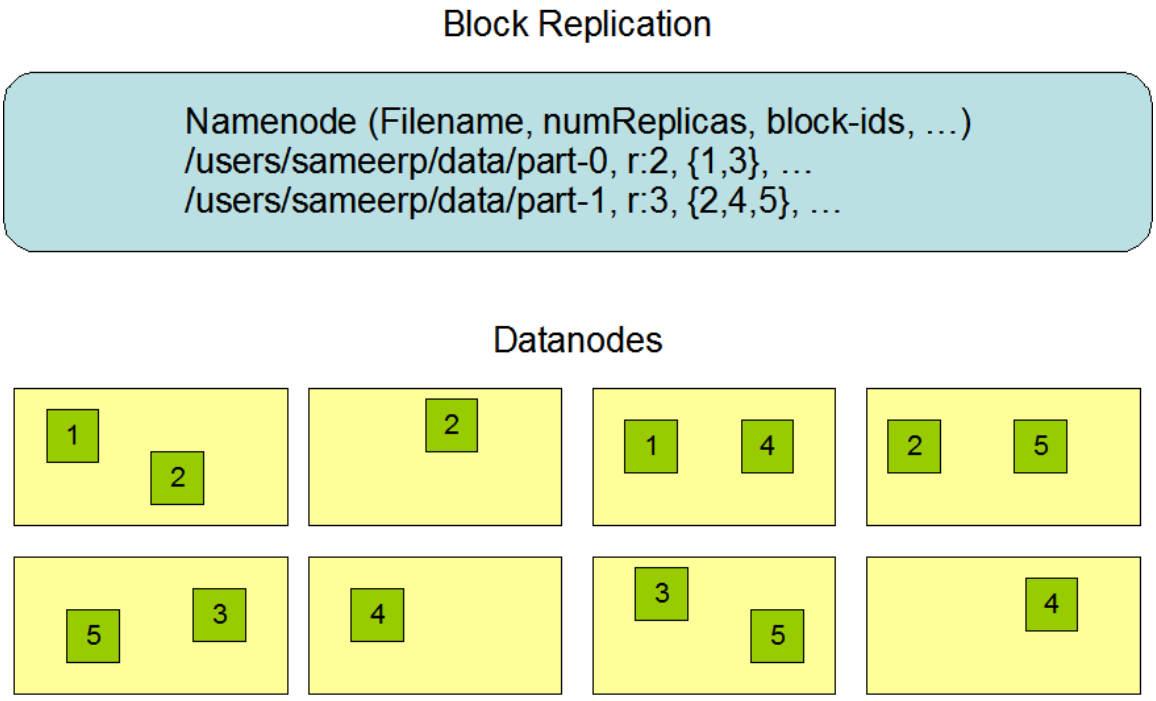
Kuva 4. NameNodejen synkronointi

Aina kun aktiivinen NameNode tekee muutoksen tiedostojärjestelmän nimiavaruuteen, kuten uudelleennimeää hakemistoja, se tekee myös tallenteen muutoksesta jaetussa hakemistossa sijaitsevaan lokitiedostoon. Passiiviset NameNodet pitävät silmällä tätä hakemistoa ja synkronoivat muutokset omiin versioihinsa koko nimiavaruudesta. Aktiivisen NameNoden häiriötilanteessa passiiviset NameNodet varmistavat, että ne ovat synkronoineet oman nimiavaruutensa jaetussa hakemistossa olevien muutosten kanssa, ennen kuin yksi niistä siirtyy aktiiviseen tilaan. [16.]

Quorum Journal Manager käyttää erikoistuneita JournalNode-prosesseja tiedon synkronoimiseen NameNodejen välillä, ja niiden toimintaperiaate perustuu enemmistöön. JournalNodeja on niitä käyttävässä järjestelmässä vähintään kolme, ja niitä lisätään siten, että järjestelmässä on aina pariton määrä JournalNodeja. Tällä varmistetaan, että JournalNodejen välille syntyy aina enemmistö. NameNode lähettää muutoksen tehdessään jokaiselle JournalNodelle tallenteen muutoksesta. Toisin kuin jaettu hakemisto, JournalNodet pystyvät kestämään häiriötilanteet, kuten yhden tai useamman JournalNodea suorittavan laitteen rikkoutumisen. Järjestelmä kykenee pysymään toiminnallisena niin kauan, kun JournalNodeja on olemassa tarpeeksi monta enemmistön syntymistä varten. [17.]

Järjestelmän ylläpitäjän vastuulla on määrittää tapa, jolla häiriöstä kärsivä NameNode poistetaan käytöstä. Passiiviset NameNodet eivät pysty siirtymään aktiiviseen tilaan, jos aktiivinen NameNode tekee muutoksia nimiavaruuteen ja siten estää synkronoidun tilan saavuttamisen. Passiivinen NameNode voidaan myös siirtää aktiiviseksi tarpeen mukaan, kuten aktiivisen NameNoden tarvitessa päivityksiä tai huoltoa. [16.]

DataNode hallitsee omien solmujensa säilytystilaa ja käsittelee tietojärjestelmän asiakkailta tulleita luku- ja kirjoituspyyntöjä. Tiedostojärjestelmään tallennetut tiedostot pilkotaan tiedostokohtaisten asetuksien perusteella lohkoiksi. Asetukset määräävät lohkojen koon viimeistä lohkoa lukuun ottamatta sekä replikointikertoimen, joka kertoo järjestelmälle, kuinka monta replikaa jokaisesta lohkosta tehdään ja säilötään. Tiedostokohtaiset asetukset on säilötty NameNodeen, jonka käskemänä DataNodet suorittavat lohkojen luontia, poistamista ja replikointia. Lohkojen replikointi ja säilöntä on havainnollistettu kuvassa 5. [15.]



Kuva 5. Lohkojen säilöntä ja replikointi [15]

NameNode pyrkii tyydyttämään lukupyynnöt DataNodesta, joka on lähimpänä lukijaa. Jokaista lohkoa varten lasketaan tarkistussumma (engl. *checksum*), jotka säilötään piilotettuna tiedostona samaan nimiavaruuteen. Summien ja lohkojen vertaamisella havaitaan datan korruptoiminen, johtui se mistä syystä tahansa. [15.]

Hadoop on tietoinen laitetelineistä (engl. *rack*), joilla sitä suoritetaan. Tietoisuuden avulla replikoita voidaan tietoisesti sijoittaa toisiin laitetelineisiin, millä vältetään kokonaisen laitetelineen toimimattomuudesta johtuvat häiriötilanteet. HDFS sisältää myös muita käytänteitä, jotka NameNode ottaa huomioon määrittäessään, mihin ja minkälaiselle massamuistilaitteelle lohkot sijoitetaan järjestelmässä. [15.]

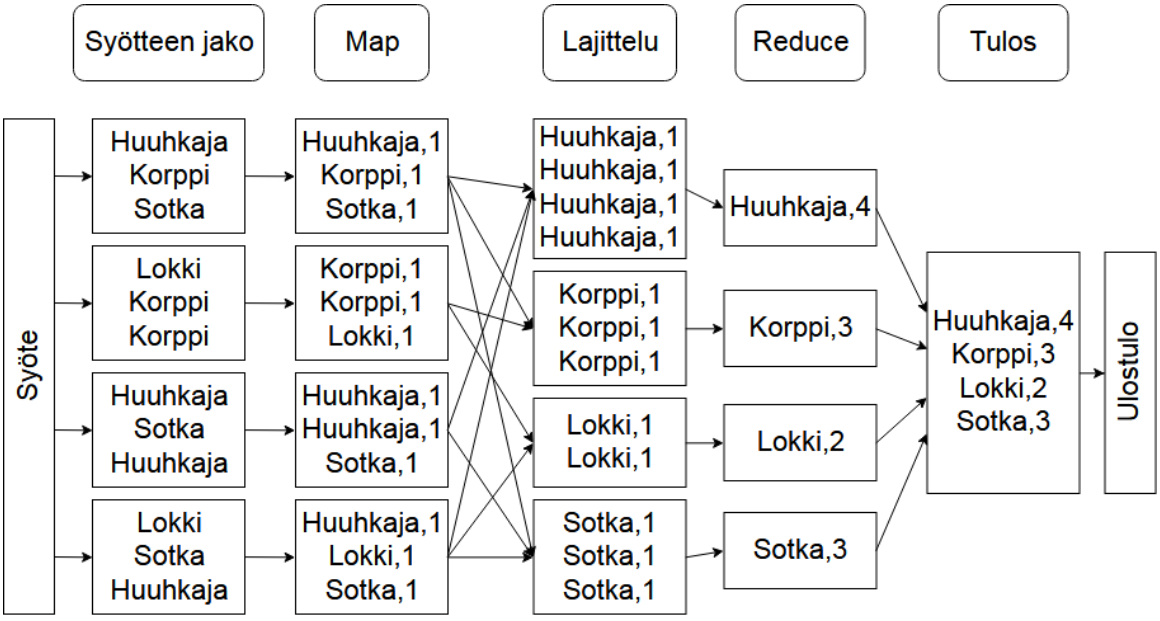
DataNodet lähettävät aika ajoin kaikille NameNodeille Heartbeat-viestejä ja Blockreport-raportteja. Heartbeat-viesti kertoo NameNodelle, että DataNode toimii oikein. NameNode tulkitsee Heartbeat-viestin puuttumisen häiriötilaksi, jolloin se merkitsee DataNoden kuolleeksi, eikä enää lähetä sille luku- tai kirjoituspyyntöjä. [15.]

3.1.3 Hadoop MapReduce

Hadoop MapReduce on ohjelmistokehys, jonka avulla HDFS:iin säilöttyä massiivista datamäärää prosessoidaan rinnakkain tietokoneklusterilla. Itse MapReduce-ohjelmointimallin lisäksi ohjelmistokehys tarjoaa kaiken tarvittavan ohjelmistojen suorittamiseen hajautetun tiedostojärjestelmän kanssa. [18.]

Datan säilönnän ollessa hajautettua on tehokkaampaa, että myös prosessointi suoritetaan hajautetusti. Tällä tarkoitetaan sitä, että dataa käyttävää ohjelmaa suoritetaan rinnakkain samoilla laitteilla, jotka säilövät dataa, sen sijaan, että kaikki data kerättäisiin yhteen ja siirrettäisiin lähemmäksi sitä käyttävää ohjelmaa. Koska tiedostot voivat olla teratavujen kokoisia, on niiden siirtäminen verkon ylitse erittäin raskasta, mikä voi vaikuttaa vahvasti palvelun yleiseen suorituskykyyn. Massiivisen datamäärän prosessoiminen kerralla olisi myös hidasta ja vaatisi erittäin tehokkaan tietokoneen, jolloin myös menetettäisiin Hadoopin tarjoama, hajauttamiseen ja tavanomaisiin laitteisiin perustuva häiriönsietokyky. [15.]

Kuvassa 6 esitetään MapReduce-prosessi. Esimerkkinä on ohjelma, jossa syötedatasta lasketaan yksittäisten sanojen ilmestymiskerrat.



Kuva 6. MapReduce-prosessi

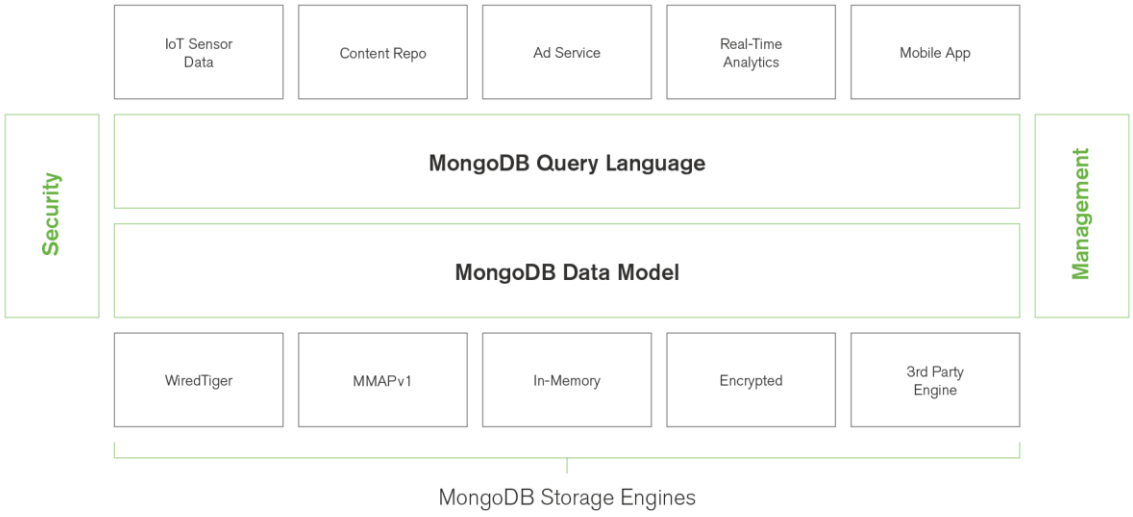
MapReduce alkaa syötedatan pilkkomisella paloihin. Paloja prosessoidaan rinnakkain usealla koneella itsenäisesti toisista paloista. Map on käyttäjän toteuttama funktio, joka tässä esimerkissä muuttaa jokaisen syötepalassa ilmestyvän sanan avain-arvo-pariksi asettaen avaimeksi sanan itse ja arvoksi 1. Map-funktioiden tuottamat välitulokset lajitellaan ja kerätään yhteen avaimen perusteella. Lajiteltua dataa käytetään syötteenä Reduce-funktiolle, joka on Map-funktion tapaan käyttäjän toteuttama. Tässä esimerkissä Reduce-funktiot yksinkertaisesti laskevat yhteen avain-arvo-parien arvot. Tämä tuottaa yksittäisiä avain-arvo-pareja, joiden arvona on alkuperäisten parien arvojen yhteenlaskettu arvo ja avaimena alkuperäinen avain. Lopuksi kaikkien Reduce-funktioiden tulokset kerätään yhteen ja tallennetaan tiedostojärjestelmään. [18.]

3.2 MongoDB

MongoDB on hajautettu dokumenttitietokanta, joka on suunniteltu mahdollisimman joustavaksi ja tehokkaaksi. MongoDB sisältää valmiiksi korkean saatavuuden mahdollistavia ominaisuuksia, kuten automaattisen skaalauksen sekä datan replikoinnin. [19.]

MongoDB:n rakenne tekee siitä erittäin mukautuvaisen erilaisten ohjelmistojen vaatimuksiin. Esimerkiksi MongoDB:n säilöntäjärjestelmän toiminnallisuutta voidaan muokata siihen liitettävien varastomoottorien (engl. *storage engine*) avulla. Järjestelmään voi olla lii-

tettynä samaan aikaan useampi varastomoottori, joista jokaisella voi olla oma erikoisuutensa, kuten järjestelmässä suoritettavien operaatioiden nopeus tai tietoturvallisuus. MongoDB:n rakenne on esitetty kuvassa 7.



Kuva 7. MongoDB:n rakenne [20]

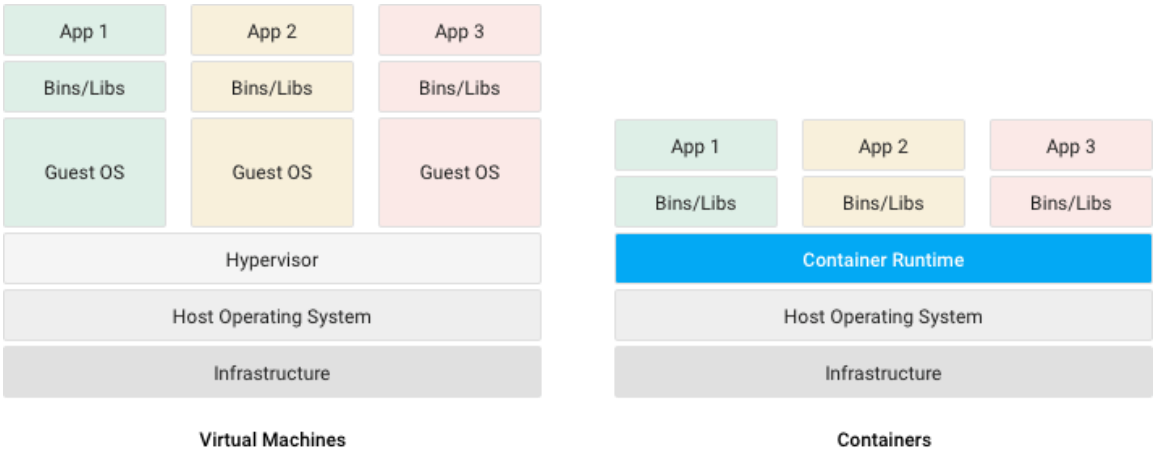
Tieto säilötään MongoDB:ssä joustavaan dokumenttimuotoon, jonka rakenne voi vaihdella dokumenttien välillä ja muuttua ajan kuluessa. Monet perinteiset relaatiotietokannat (engl. *relational database*) taas eivät ole kovin joustavia, sillä ne säilövät tietoa yleensä riveistä ja pylväistä koostuviin taulukkoihin. Joustavuuden ansiosta säilöttyä tietoa on helpompi käyttää ohjelmistojen kanssa, ja säilöttyjä dokumentteja on mm. helpompi yhdistää keskenään. [20.]

3.3 Docker

Docker on ohjelmisto, jonka avulla ohjelmistoja koteloidaan ns. ohjelmistokontteihin. Ohjelmiston siirtämistä toimimaan tällaisen kontin sisällä kutsutaan tässä työssä kontitukseksi. Ohjelmistokontteja ei tule sekoittaa Hadoop YARN:n resurssikontteihin.

Ohjelmistokontti toimii abstraktiokerroksena ohjelmiston ja suoritusympäristön välillä, ja se tarjoaa virtuaalikoneen tavoin siinä suoritettavalle ohjelmalle kaiken, mitä se suoritukseensa tarvitsee. Abstraktion ansiosta samaa kontitettua ohjelmaa voidaan suorittaa millä järjestelmällä tahansa, jolla konttien ajonaikaista ympäristöä (engl. *runtime environment*) voidaan suorittaa. [21.]

Toisin kuin virtuaalikoneet, jotka virtualisoivat koko tietokoneen laitteiston, ohjelmistokontit virtualisoivat vain käyttöjärjestelmän. Tämä tekee ohjelmistokonteista paljon kevyempiä ja helpompia siirtää erilaisten järjestelmien välillä. Ohjelmistokontteja voidaan käyttää yhdessä virtuaalikoneiden kanssa, mikä tekee ohjelmistojen toimituksesta käyttäjille joustavampaa. Yhdistelmä tekee järjestelmästä turvallisemman, sillä se lisää ylimääräisen eristyskerroksen varsinaisen ohjelmiston ja laitteiston välille. Kuvassa 8 on esitetty ohjelmistokonttien ja virtuaalikoneiden rakenne. [21.]



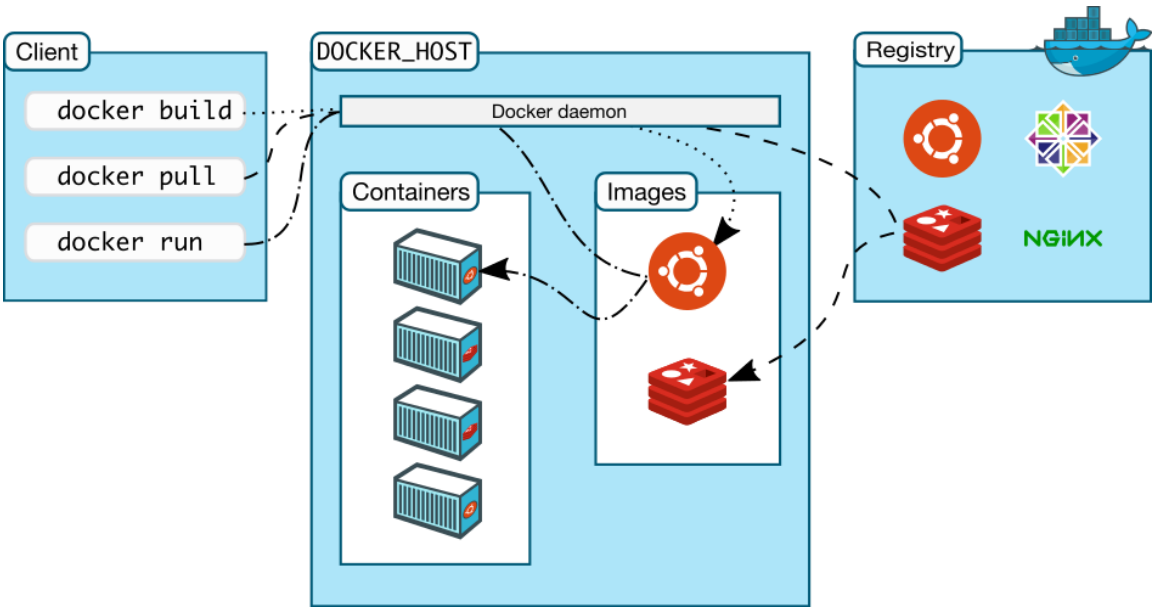
Kuva 8. Virtuaalikoneet ja ohjelmistokontit [22]

Ohjelmistokontti sisältää näköistiedoston (engl. *image*), josta sen sisällä suoritettava ohjelmisto rakentuu. Näköistiedosto sisältää mm. ohjelmiston koodin, konfigurointitiedostot sekä mahdolliset ohjelmistokirjastot, joista ohjelmisto on riippuvainen. Kontitetut ohjelmistot päivitetään rakentamalla uusi näköistiedosto päivitetyistä osista. Näköistiedosto ja ohjelmistokontti luovat yhdessä ohjelmiston, joka toimii käytännössä identtisesti kaikissa ympäristöissä. [23.]

Näköistiedostojen rakentaminen automatisoidaan Dockerin omilla tiedostoilla, joita kutsutaan Dockerfileiksi. Dockerfileen määritellään yksinkertaista syntaksia käyttäen ohjelmiston rakentamiseen vaadittavat askeleet, jotka käyttäjä muuten suorittaisi itse komentoriviä käyttäen, kuten ohjelmiston riippuvuuksien lataamisen ja konfiguroinnin. Docker lukee tiedoston ja suorittaa siihen määritellyt askeleet järjestyksessä näköistiedoston luomiseksi. [24.] [25.]

Näköistiedostojen säilytystä ja kehitystä varten käytetään näköistiedostorekistereitä (engl. *image registry*). Näköistiedostot julkaistaan tällaiseen rekisteriin, josta käyttäjät voivat ladata ne ja luoda niistä ohjelmistokontteja. Rekisterit helpottavat huomattavasti mm. kon-

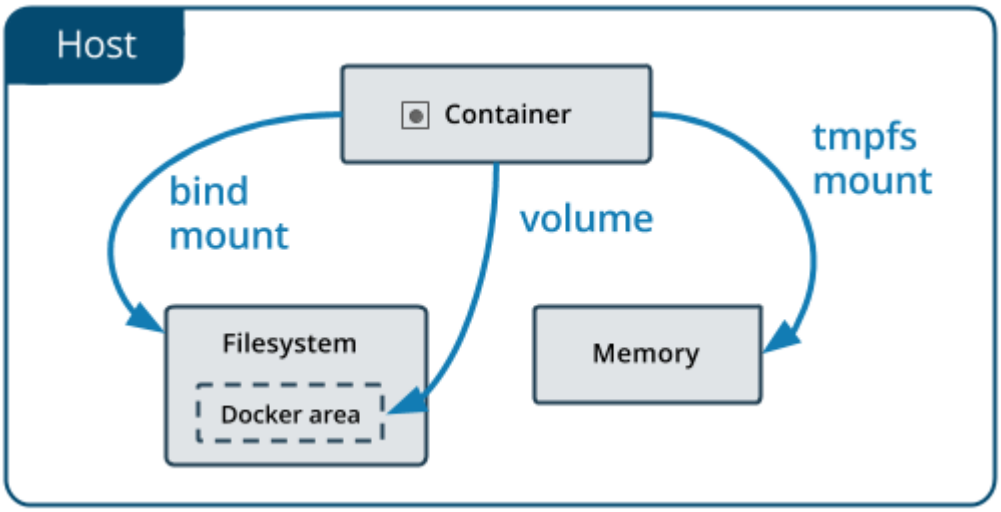
titettujen ohjelmistojen päivitystä ja jakamista. Ohjelmistopäivityksen jakaminen suoritetaan esimerkiksi päivittämällä ohjelmistokontin Dockerfile ja julkaisemalla siitä rakennettu näköistiedosto näköistiedostorekisteriin. Päivitysten jälkeen ohjelmistojen käyttäjät voivat ladata uuden näköistiedoston ja luoda sen avulla päivitettyjä kontteja. Dockerin arkkitehtuuri on esitetty kuvassa 9. [26.]



Kuva 9. Dockerin arkkitehtuuri [24]

Ohjelmistokontit toimivat parhaiten sellaisten ohjelmistojen kanssa, jotka koostuvat useasta erillisestä komponentista tai palvelusta. Yksittäiset osat voidaan kontittaa erikseen, mikä tekee ohjelmiston rakenteesta joustavamman. Myös monoliittiset ohjelmistot, joita ei voi jakaa itsenäisiksi osiksi, voidaan kontittaa, mutta niiden kohdalla menetetään muutama konttien tarjoama hyöty. Yksittäisestä kontista koostuva ohjelmisto pitää päivittää kerralla, ja se ei kykene skaalautumaan useasta osasta koostuvan palvelun lailla. [22.]

Kontitettu ohjelma on usein suositeltavaa suunnitella siten, että samassa kontissa ei säilytetä dataa, jota ohjelma itse käsittelee. Ohjelmistokontit on suunniteltu lyhytikäisiksi, ja konttien sisältämä data katoaa konttien mukana. Konttien datan säilytykselle on kolme vaihtoehtoa: kontissa itsessään eli isäntäkoneen muistissa, isäntäkoneen tiedostojärjestelmässä tai Dockerin hallitsemissa datavolyymeissä, jotka myös sijaitsevat isäntäkoneen tiedostojärjestelmässä. Säilöntävaihtoehdot on havainnollistettu kuvassa 10. [27.]



Kuva 10. Datan säilöntä konttien kanssa [27]

Kaikki vaihtoehdot näyttävät kontin sisällä olevalle ohjelmalle samalta, mutta ne eivät ole samanarvoisia, kun datan säilyvyydellä on merkitystä ja sitä täytyy esimerkiksi pystyä siirtämään samaa konttia käyttävien laitteiden välillä. Kontin sisällä tehdyt muutokset eivät tallennu sen näköistiedostoon. Kontin sisälle tallennettu tieto katoaa siis samalla, kun kontti itse syystä tai toisesta sammuu, ja sille varattu muisti vapautuu. [27.]

Isäntäkoneelle tallentaminen tapahtuu liittämällä (engl. *mount*) kontin sisäiseen tiedostojärjestelmään jokin hakemisto isäntäkoneen tiedostojärjestelmästä. Isäntäkoneen tiedostojärjestelmään tallennettu data säilyy kontin tilasta riippumatta. Datan siirtäminen toiselle samaa konttia käyttävälle laitteelle on kuitenkin hankalaa, sillä isäntäkoneen hakemistoon viitataan sen hakemistopolun avulla. Toisella laitteella täytyy siis olla identtinen hakemistorakenne ensimmäisen laitteen kanssa, jotta sen oma kontti pystyy käyttämään dataa. Tiedon tallentaminen suoraan isäntäkoneen tiedostojärjestelmään sisältää myös väärinkäytön riskin: konttiin eristetty ohjelma pystyy tekemään suoraan muutoksia isäntäkoneen tiedostojärjestelmään, mikä voi johtaa esimerkiksi isäntäkoneen käyttöjärjestelmän korruptoitumiseen. [27.]

Datavolyymit ovat erityisiä Dockerin luomia ja hallitsemia hakemistoja. Datavolyymeissä säilytetty data on konttien tapaan eristetty isäntäkoneesta ja ne sijaitsevat Dockerin hallitsemassa hakemistossa isäntäkoneen tiedostojärjestelmässä. Dockerin hallinnoinnin avulla volyymit voivat sijaita eri hakemistoissa eri laitteilla, mikä tekee niistä paljon helpompia siirtää. [27.]

Volyymit ovat Dockerin suosittelema keino datan säilyttämiseen, mutta muillakin vaihtoehdoilla on hyötynsä. Kontin sisäiseen järjestelmään voidaan tallentaa kaikki sellainen

data, minkä säilyvyydellä ei ole merkitystä, tai jota ei haluta säilyttää esimerkiksi datan arkaluonteisuuden takia. Isäntäkoneen tiedostojärjestelmästä liitetty hakemisto pitää liitettäessä sisältönsä, minkä avulla isäntäkone voi jakaa esimerkiksi konfigurointitiedostoja konttien kanssa. [27.]

Ohjelmistokontit ovat nousseet suosioon mm. niiden mahdollistaman helpon toimituksen vuoksi. Niiden hallintaan on kehitetty useita ohjelmia ja työkaluja, joista tälle työlle merkittäviä ovat konttien hallintaan keskittyvä Rancher sekä Docker Compose, jonka avulla konteista muodostetaan palvelukokonaisuuksia.

3.3.1 Docker Compose

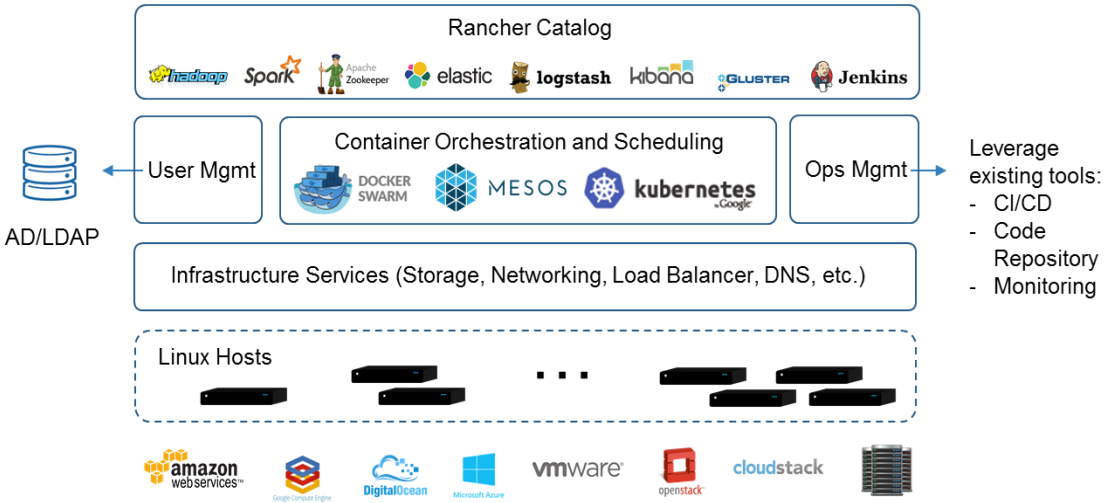
Docker Compose on työkalu, jolla erillisistä konteista määritellään tiettyä asiaa suorittava kokonaisuus. Esimerkiksi jokin tietty palvelu saattaa koostua tietokannasta, webpalvelimesta sekä jostain laskentaa suorittavasta ohjelmistosta, joista jokainen sijaitsee omassa ohjelmistokontissaan. Näiden konttien riippuvuudet ja asetukset määritellään YAML-merkintäkielellä (YAML Ain’t Markup Language) konfigurointitiedostoon. Compose lukee tiedoston, jonka avulla se pystyy luomaan ja käynnistämään kaikki palvelun muodostavat kontit kerralla. [28.]

3.3.2 Rancher

Rancher on pilvessä suoritettava infrastruktuurin ja ohjelmistokonttien hallinnointialusta, jota suoritetaan myös ohjelmistokontissa. Rancher käyttää konttien suorittamiseen joko virtuaalisia tai fyysisiä Linux-koneita, joilta se saa mm. prosessointitehoa, säilöntätilaa ja muistia. Koneet suorittavat Dockeria, ja ne liitetään osaksi Rancherin järjestelmää Rancherin agenttiohjelmistoa suorittavan ohjelmistokontin avulla. Rancher sisältää myös kontitettuja infrastruktuurin hallintaan liittyviä palveluja, jotka auttavat mm. palvelun tietoverkkojen sekä säilöntätilan hallinnassa. [29.]

Usean kontin hallintaa, joka sisältää mm. konttien käynnistämisen, sammuttamisen ja päivityksen, kutsutaan konttien orkestroinniksi. Rancher käyttää sen omaa Cattle-nimistä orkestroijaa kontitettujen infrastruktuuripalvelujen sekä käyttäjän omien ohjelmistokonttien orkestrointiin. Rancher tukee myös kolmansien osapuolien orkestrointiratkaisuja, joita

myös hallinnoidaan Cattlen avulla. Rancherin yleinen rakenne ja ominaisuudet ovat esitetty kuvassa 11. [29.]



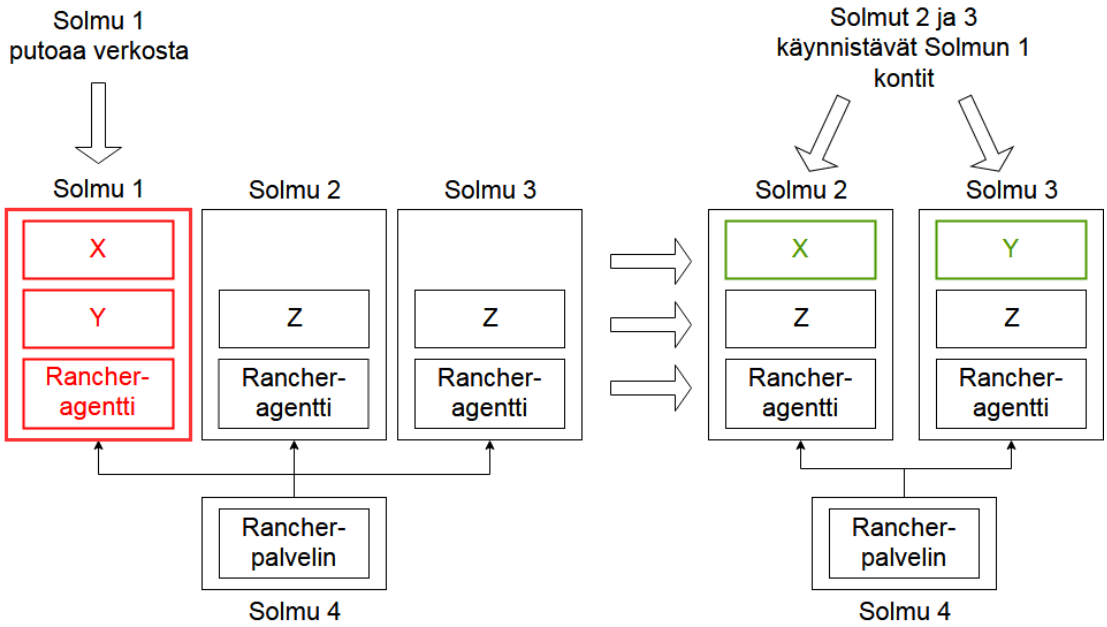
Kuva 11. Yleiskuva Rancherista [29]

Rancher sisältää myös katalogin, joka on verrattavissa Dockerin näköistiedostorekisteriin. Katalogi sisältää mallipohjia (engl. *template*), joiden avulla luodaan ja käynnistetään useista ohjelmistokonteista koostuvia palveluja. Näköistiedostorekisterien tapaan käyttäjä voi luoda omia katalogejaan. [30.]

Palvelujen käynnistämiseen Rancher käyttää Rancher Compose -työkalua, joka on useaa isäntäkonetta tukeva versio Docker Composesta. Rancher Compose käynnistää palvelun kontit useammalla Dockeria suorittavalla palvelimella riippuen käyttäjän määrittämistä säännöistä sekä palvelinten kuormitustilasta. [31.]

Järjestelmään voidaan tarvittaessa lisätä useampi Rancherin pääpalvelinta suorittava ohjelmistokontti korkean saatavuuden tilan saavuttamiseksi. Järjestelmään täytyy lisätä ainakin kolme pääpalvelinta suorittavaa konttia, jotka tallentavat tietonsa ulkoiseen tietokantaan. Tietokannan tulisi myös olla korkean saatavuuden varmistamiseksi hajautettu ja replikointikykyinen. Lisäksi tarvitaan palvelu, joka tasoittaa webliikenteen eri palvelinten välillä. [32.]

Rancher pystyy käynnistämään häiriöiden vuoksi sammuneet kontit automaattisesti uudestaan. Kokonaisen solmun pudotessa järjestelmästä esimerkiksi laitevian vuoksi Rancher pystyy luomaan ja käynnistämään palvelimella olleet kontit toisilla palvelimilla automaattisesti. Ohjelmistokonttien uudelleenkäynnistys on esitetty kuvassa 12. [33.]

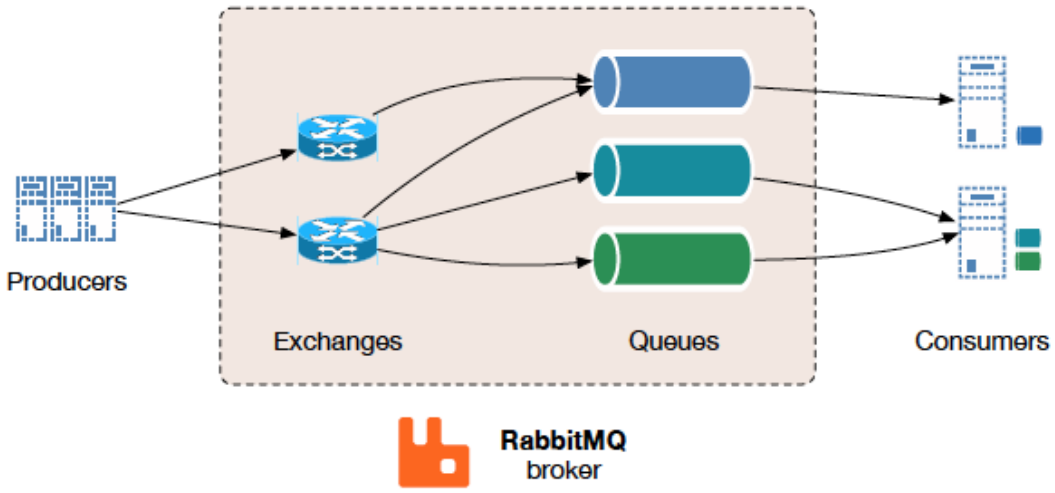


Kuva 12. Konttien uudelleenkäynnistys vikatilanteessa

3.4 RabbitMQ

RabbitMQ on viestinvälityspalvelin. RabbitMQ käyttää mm. AMQP-viestinvälitysprotokollaa (Advanced Message Queuing Protocol) viestien välittämiseen, mikä mahdollistaa kommunikaation eri ohjelmointikielillä toteutettujen ohjelmistojen välillä. Kommunikaatio RabbitMQ:n kautta voi tapahtua joko synkronisesti tai asynkronisesti tarpeen mukaan. RabbitMQ sisältää myös muita ominaisuuksia, jotka auttavat mm. tietoturvallisuuden ja viestien reitityksen kanssa.

RabbitMQ on mahdollista toimittaa hajautetusti ja sille on saatavilla toiminnallisuutta lisääviä liitännäisiä. RabbitMQ sisältää korkean saatavuuden mahdollistavia ominaisuuksia, kuten viestijonojen (engl. *message queue*) replikoinnin. RabbitMQ:n viestinvälitysprosessin rakenne on havainnollistettu kuvassa 13. [34.]



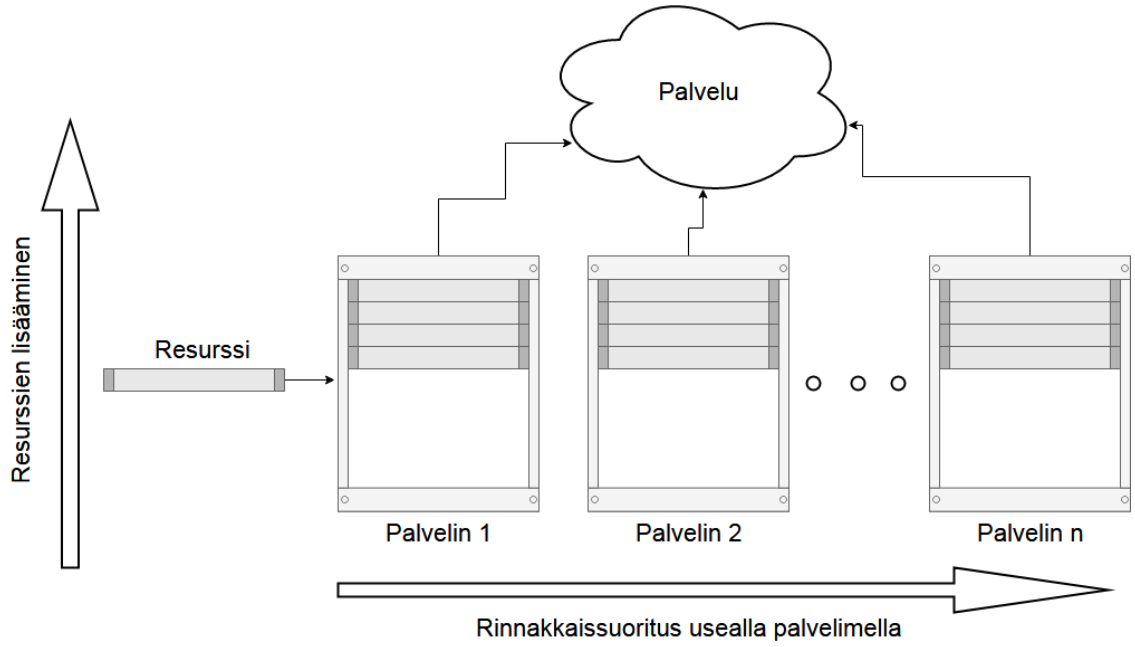
Kuva 13. RabbitMQ:n viestinvälitysprosessi [34]

Viestien lähettäjät lähettävät viestinsä RabbitMQ:n vaihteille (engl. *exchange*), joiden ansiosta lähettäjien ei tarvitse itse tietää mitään viestien reitityksestä. Vaihteet lähettävät viestit eteenpäin tietyille viestijonoille, joita jokainen kyseisiä viestejä haluava vastaanottaja kuuntelee. RabbitMQ tukee mekanismeja, joiden avulla varmistetaan, että viesti on mennyt sen haluavalle vastaanottajalle perille. Järjestelmään voidaan esimerkiksi määrittää, että vastaanottaja lähettää RabbitMQ:lle ilmoituksen aina, kun se vastaanottaa viestin. Mikäli RabbitMQ ei saa vastaanottoilmoitusta, se lähettää viestin kyseiselle vastaanottajalle uudelleen. [34.] [35.]

4 Pilvipalveluiden skaalaaminen

Pilvipalveluiden skaalaamisella tarkoitetaan tässä työssä sekä palveluiden joustavuutta, eli kuinka palvelut skaalautuvat kuormituksen mukaan, että esimerkiksi sitä, kuinka palvelut muokkautuvat eri asiakkaiden erityisvaatimusten mukaisesti. Palveluiden skaalaamisella pyritään optimoimaan palveluiden resurssienkäyttöä siten, että korkean saatavuuden tila säilyy nopeasti vaihtelevasta käyttömäärästä riippumatta. Samalla käyttämättömien resurssien määrä pyritään pitämään minimissä.

Pilvipalvelut skaalautuvat sekä ylöspäin että sivulle. Skaalautumista on havainnollistettu kuvassa 14.



Kuva 14. Skaalaus eri suuntiin

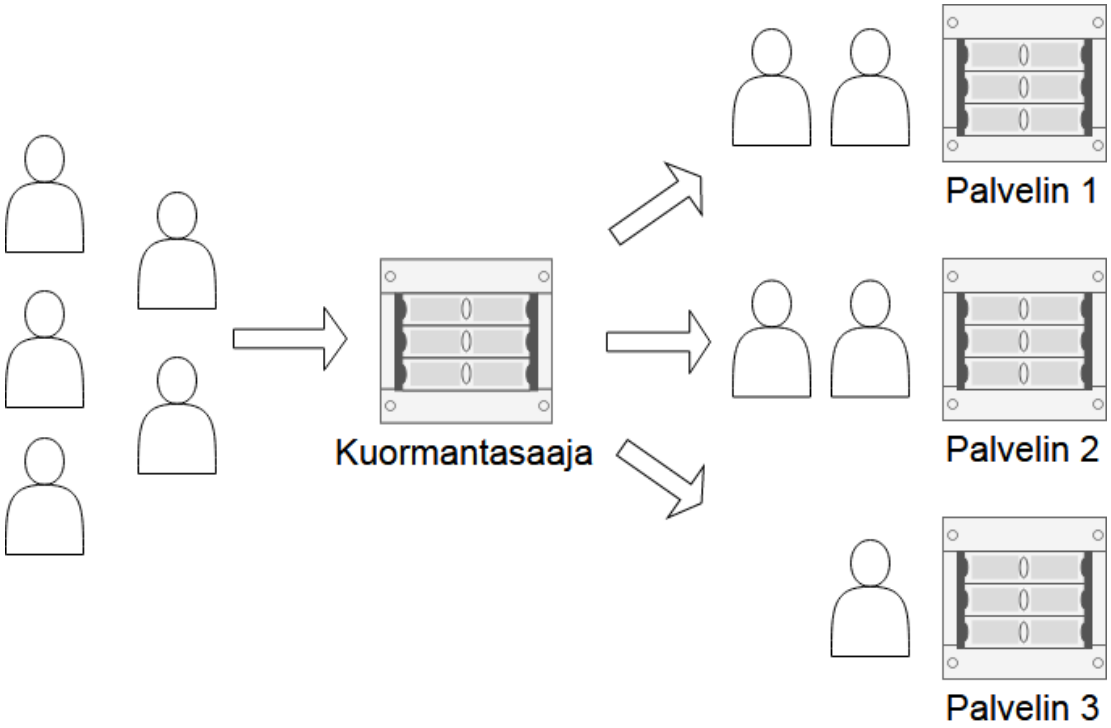
Ylöspäin skaalaamisella tarkoitetaan esimerkiksi palvelimen toimintakyvyn tehostamista päivittämällä sen laitteistoa. Yksittäinen palvelin voi kuitenkin skaalautua ylöspäin vain laitteiston sallimissa rajoissa. Koko palvelun ajaminen yhdellä laitteella tekee siitä myös haavoittuvaisen käyttökatkoksille, joita voi aiheutua mm. laiterikoista ja sähkökatkoksista. Yksittäisen palvelimen skaalaaminen alaspäin on myös haasteellista, sillä fyysisten resurssien vähentäminen vie aikaa ja voi aiheuttaa laiterikkoja. Voimakkaan palvelimen käyttäminen vähällä käytöllä olevan pilvipalvelun ylläpitämiseen ei myöskään ole kannattavaa.

Sivuittaissuuntainen skaalautuminen tarkoittaa palvelun suorittamista rinnakkain useammalla laitteella. Laitteiston sivuittaissuuntainen skaalautuminen voi jatkua käytännössä loputtomasti, mikäli sen päällä suoritettava palvelu kykenee skaalautumaan sen mukana. Sivuittaissuuntaista skaalaamista kutsutaan myös ulos- ja sisäänpäin skaalaamiseksi, jossa ulospäin skaalatessa rinnakkain suorittavia laitteita lisätään ja sisäänpäin skaalatessa vähennetään. Sivulle skaalaaminen on kuitenkin verrannollisesti paljon haasteellisempaa, sillä palvelu on suunniteltava käytännössä alusta saakka tukemaan suoritusta useammalla laitteella.

Kolmas tapa skaalata palveluita on lisätä palveluinstanssien määrää. Usean instanssin avulla pilvipalvelu koostuu verrannollisesti pienistä, dynaamisesti käynnistettävistä ja sammutettavista palveluista yhden suuremman palvelun sijaan. Pienempiä palveluita voidaan käynnistää ja sammuttaa tarpeen mukaan, mikä yksinkertaistaa skaalausprosessia. Useammalla instanssilla on kuitenkin omat haasteensa, kuten yksittäisten instanssien koon määrittäminen, kuinka nopeasti uudet instanssit käynnistyvät, miten dataa säilytetään ja niin edelleen.

Skaalautumiskyky on välttämätön ominaisuus pilvipalveluille, mutta sen toteuttamiseen liittyy paljon haasteita. Jokaisen palvelun komponentin on kyettävä skaalautumaan yhtä tehokkaasti, jotta palveluun ei muodostu pullonkauloja. Toinen haaste on resurssien loppuminen kesken. Palveluiden pitää pystyä reagoimaan resurssien loppumiseen etukäteen, ja niiden täytyy kyetä ottamaan lisätyt resurssit käyttöön automaattisesti.

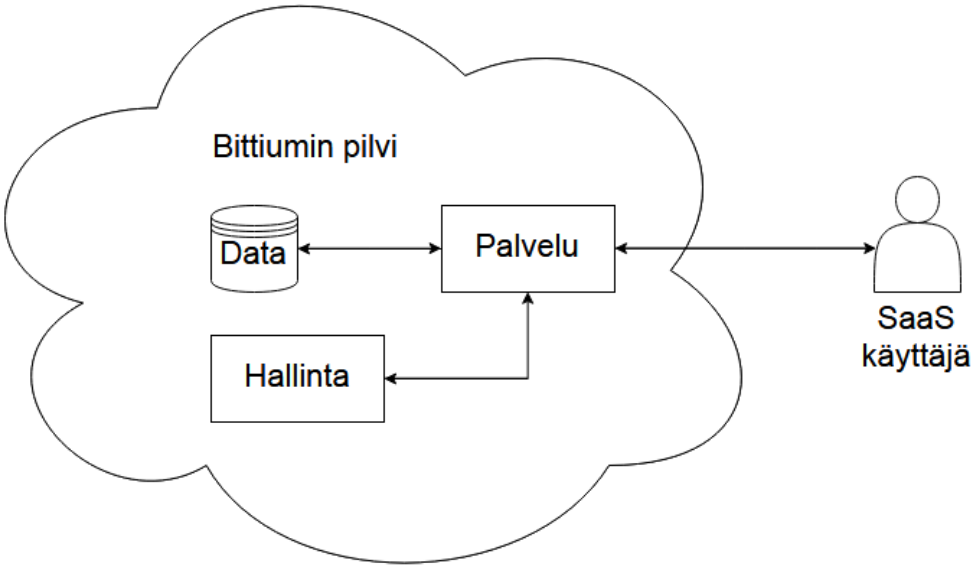
Kuormantasaus (engl. *load balancing*) on myös osa skaalautuvuutta. Kuormantasauksella tarkoitetaan palveluun kohdistuvan käytön ohjaamista infrastruktuurin osien kuormituksen perusteella. Esimerkiksi jos järjestelmä huomaa jonkin palvelimen olevan suuren kuormituksen alaisena, ei kyseiselle palvelimelle ole kannattavaa ohjata lisää käyttäjiä. Tämä pätee, vaikka palvelin olisi esimerkiksi sijaintinsa perusteella paras vaihtoehto. Tällaisessa tilanteessa järjestelmän täytyy kyetä ohjaamaan liikenne automaattisesti matalammalla kuormituksella olevalle palvelimelle. Kuvassa 15 on esitetty yksinkertainen kuormantasausprosessi, jossa tasaaja pyrkii jakamaan käyttäjät tasaisesti palvelimille. [36.]



Kuva 15. Kuormantasaus

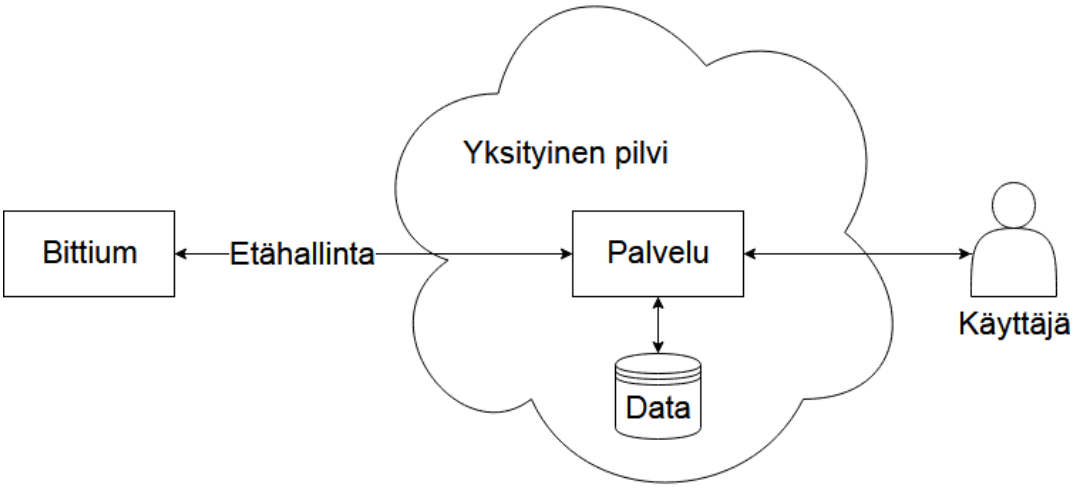
5 Bittiumin pilvipalvelun toteutus

Bittiumin pilvipalvelulle ollaan suunnittelemassa tällä hetkellä kolmea erilaista toimitustapaa. Ensimmäisessä toimitustavassa asiakkaalle tarjotaan puhdas SaaS-mallinen palvelu, jonka infrastruktuurista vastaa Bittium. Toisessa toimitustavassa asiakkaalle toimitetaan palvelun komponentit, joiden avulla hän voi pystyttää palvelun omiin tiloihinsa tai ulkopuolisen tekijän tarjoamalle pilvialustalle. Kolmannessa toimitustavassa asiakkaalle toimitetaan sekä palvelun komponentit että tarvittavat fyysiset resurssit, kuten palvelimet.



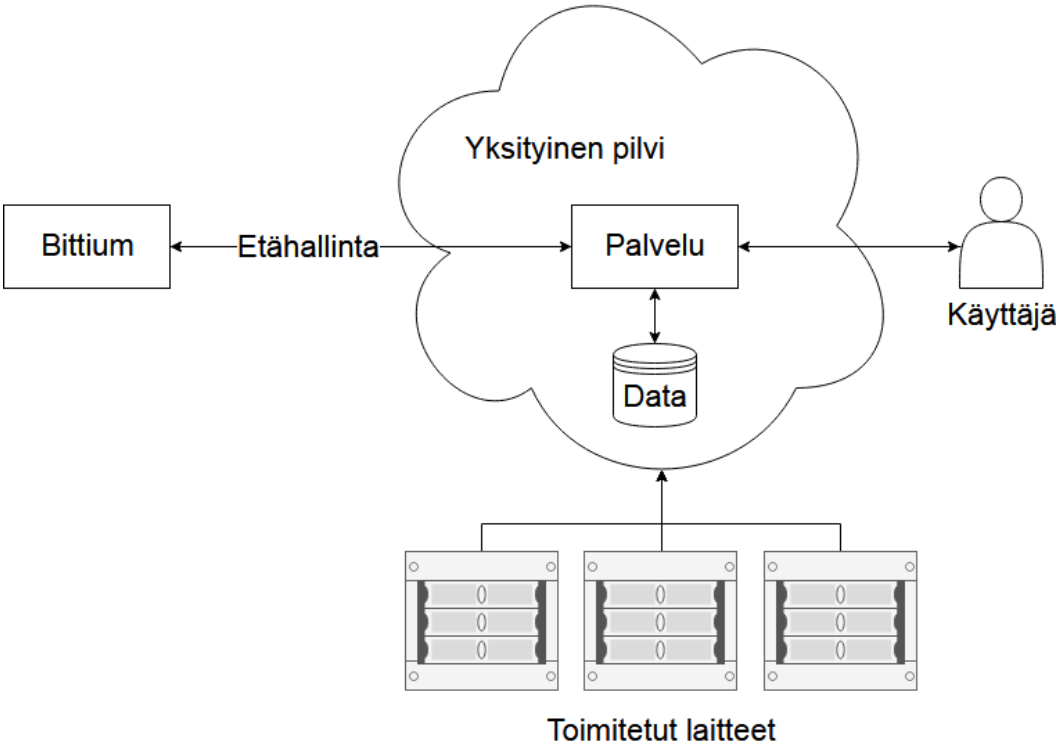
Kuva 16. SaaS-toimitus

SaaS-toimitustapa mahdollistaa toimituksen asiakkaille, joilla ei ole omaa infrastruktuuria, tarvittavaa teknistä osaamista tai henkilöstöresursseja palvelun asentamiseen ja ylläpitoon. Palvelu sijaitsee Bittiumin pilvessä, jonka ansiosta Bittium voi suorittaa tarvittavat päivitys- ja konfigurointityöt suoraan pilven sisältä. Palvelu lähettää ohjelmistojen tuottamat lokitiedostot suoraan palvelun ulkopuolelle paikkaan, jossa ne ovat ylläpidon saatavilla. Asiakkaiden palveluun tallentama tieto jää joka tapauksessa asiakkaalle yksityiseksi.



Kuva 17. Ohjelmistokomponenttien toimitus

Palvelun ohjelmistokomponenttien toimitus sopii varsinkin asiakkaille, joilla on jo valmiiksi tarvittava infrastruktuuri palvelun käyttöönottamiseksi. Konfigurointityöt ja ohjelmistojen päivitykset suoritetaan etähallintayhteyttä käyttäen pilven sisältä. Palvelu lähettää ohjelmistojen toimintaan liittyvät lokitiedostot etähallintayhteyttä käyttäen takaisin Bittiumille. Järjestelmään tallennettu tieto jää kuitenkin SaaS-mallin tapaan asiakkaalle yksityiseksi.



Kuva 18. Ohjelmistokomponenttien ja laitteiden toimitus

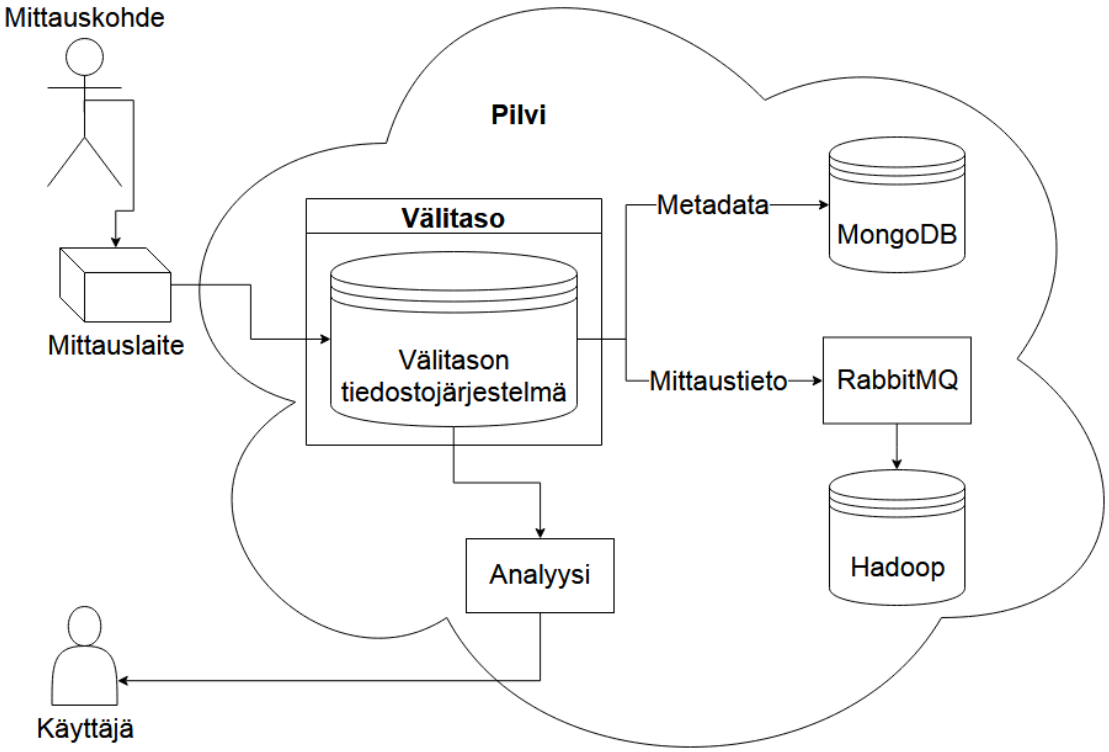
Laitetoimitus sopii mahdollisiin erikoistilanteisiin ja esimerkiksi asiakkaille, jotka haluavat hallinnoida infrastruktuuriaan itse, mutta joiden infrastruktuuri ei syystä tai toisesta ole riittävä palvelulle. Erikoistilanteita ovat esimerkiksi tilanteet, joissa asiakas haluaa pitää palvelun erillään omista palvelimistaan. Konfigurointi- ja päivitystyöt suoritetaan etähallintayhteydellä, kuten pelkän ohjelmiston toimituksen kanssa. Järjestelmään tallennettu tieto jää muiden mallien tapaan asiakkaalle yksityiseksi.

5.1 Palvelun rakenne

Suurin osa palvelun komponenteista on suunniteltu kontitettavan Dockerin avulla, mikä helpottaa toimitusta eri infrastruktuurin omaaville asiakkaille ja mahdollistaa yhtenäisen hallinnan konttiorkestraation avulla. Konttien orkestrointiin käytetään tässä työssä Rancheria, joka huolehtii konttien välisestä kuormantasauksesta sekä hallitsee konttien elinkaarta.

Palveluun tarvitaan välitaso (engl. *cache*) mittauslaitteen ja pitkäaikaissäilönnän välille. Välitason avulla asiakas pystyy analysoimaan mitattua tietoa kontitetun analysointiohjelmiston avulla samaan aikaan, kun laite sitä tuottaa. Välitaso itse on myös Docker-kontti, jonka isännän tiedostojärjestelmään mittaustieto väliaikaisesti tallennetaan. Mittauslaitteen lähettäessä tiedon mittauksen loppumisesta välitaso poistaa paikallisesti tallennetun tiedon.

Laite lähettää välitasolle tiedostoja, jotka sisältävät määrätyn mittaisen ajanjakson tiedon. Nämä tiedostot lähetetään viesteinä RabbitMQ-palvelimelle, joka sijaitsee välitason ulkopuolella, jonka jälkeen RabbitMQ-palvelin välittää viestit eteenpäin Hadoopille. Välitason rakenne ja rooli palvelussa on havainnollistettu kuvassa 19.



Kuva 19. Välitason rakenne

Välitaso lähettää myös metadataa siitä erillisenä olevalle MongoDB-tietokannalle. Metadata sisältää mm. tietoa mittauskohteesta, mittauksen alkamis- ja loppumisajan, mittaustunnisteen sekä mittaustiedon sijainnin Hadoopin tiedostojärjestelmässä. Metadataa käytetään hyödyksi monella tapaa Hadoopin toiminnan tehostamiseksi. Mittauksen alkamis- ja loppumisajan perusteella voidaan esimerkiksi päätellä, että onko koko mittaus tallennettu Hadoopin tiedostojärjestelmään. Jos havaitaan, että tallennetussa mittauksessa on puutteita, voidaan mittauslaitetta pyytää lähettämään puuttuvat osat.

Dockerin oma pilvipalvelu, Docker Hub, sisältää viralliset näköistiedostosäilöt (engl. *image repository*) sekä MongoDB:n että RabbitMQ:n näköistiedostoille. Yleisesti saatavilla olevat ja aktiivisesti ylläpidetyt näköistiedostot nopeuttavat palvelun kehitystä huomattavasti, sillä suurimmat ohjelmistojen kontitukseen liittyvät tekniset haasteet on todennäköisesti jo niissä selätetty. Välitaso ja analyysiohjelmisto ovat myös konteissa suoritettavia ohjelmistoja.

Hadoopin eri komponentit ovat myös kontitettavissa, mutta tämän työn vaatimuksia täyttävää valmista näköistiedostoa ei ole löytynyt. Hadoopin kontitukseen liittyy myös paljon

haasteita, jotka tällä hetkellä ylittävät kontituksen tarjoamat hyödyt, kuten palvelun yhtenäisen hallinnan konttiorkestraation kautta. Seuraavissa luvuissa käsitellään, mitä haasteita Hadoopin kontitukseen liittyy, mitä eri toimitusmalleissa on otettava huomioon, sekä miten koko järjestelmää hallinnoidaan ja miten se skaalautuu.

5.2 Hadoopin kontituksen haasteet

Hadoopin kontituksen haasteellisuus johtuu pääasiallisesti sen omasta rakenteesta. Esimerkiksi Hadoopin häiriösietoisuus perustuu eri komponenttien sijoittamiseen eri laitteille: järjestelmän aktiivinen ja passiivinen NameNode, sekä niiden synkronisoinnista huolehtivat JournalNodet, sijoitetaan tietoisesti palvelinklusterin eri solmuille. Tällä varmistetaan, että yksittäiset laitehäiriöt eivät koskaan pysäytä koko järjestelmää.

Ohjelmistokontit toimivat abstraktiokerroksena ohjelmiston ja laitteiston välillä, jonka takia konttiorkestroijan on huolehdittava, että esimerkiksi HDFS:n NameNodeja ei koskaan suoriteta samalla solmulla. Asiat monimutkaistuvat entisestään varsinkin sellaisissa konesaliympäristöissä, joissa palvelimet ovat aina virtuaalikoneita, sillä ne lisäävät myös abstraktiokerroksen. Kontit myös hankaloittavat laitteistotietoon perustuvien ominaisuuksien hyödyntämistä, kuten Hadoopin laitetelinetietoisuuteen perustuvaa replikointia.

Hadoopiin kuuluu paljon erikoistuneita taustaprosesseja. Suurinta osaa näistä prosesseista täytyy suorittaa omissa ohjelmistokonteissaan, sillä monet niistä erikoistuvat yksittäisen tehtävän suorittamiseen. Osa prosesseista, kuten YARN:n ResourceManager ja HDFS:n NameNode, ovat niin tärkeitä järjestelmän toiminnalle, että niitä suoritetaan yleensä yksinomaan niitä varten varatuilla laitteilla. Pääasiallisesti tämä johtuu siitä, että molempia on järjestelmässä aktiivisena kullakin hetkellä vain yksi, minkä takia niiden toimintaa ei ole suotavaa häiritä muilla prosesseilla. Varatut laitteet myös vähentävät resurssien loppumisen riskiä.

Myös muita kontitettuja Hadoopin prosesseja voi olla hankalaa suorittaa toisten ohjelmistojen ohjelmistokonttien kanssa. Esimerkiksi DataNodet lukevat ja tallentavat tietoa erittäin paljon, mikä voi estää muita kontteja käyttämästä solmun massamuistia. Massiivisten tiedostojen replikoiminen verkon ylitse vaatii myös erittäin paljon verkkokaistaa, mikä voi heikentää muiden konttien suorituskykyä, tai jopa estää niiden suorituksen. DataNodet käyttävät myös solmujensa prosessointitehoa, kun järjestelmässä suoritetaan esimerkiksi MapReduce-prosessia.

Hadoopia varten riittää yksi näköistiedosto, joka asentaa konttiin sekä Hadoopin että sen riippuvuudet. Järjestelmään vaaditaan kuitenkin jokin keino, jolla konfiguroidaan, mitä Hadoopin taustaprosessia tietty kontti suorittaa. Konttiorkestroijan täytyy myös pystyä erottamaan eri prosesseja suorittavat kontit toisistaan, jotta se voi ottaa huomioon prosessien erityispiirteet, kuten vaihtelevat resurssivaatimukset. Erityispiirteet täytyy ottaa huomioon myös säilöntätapojen valinnan ja verkkoyhteyksien kanssa, sillä esimerkiksi NameNodejen täytyy tietää JournalNodejen ja DataNodejen verkko-osoitteet, mutta JournalNodejen ei tarvitse tietää DataNodeista mitään.

5.3 Palvelun hallinta ja skaalautuminen

Palvelun kaikkien komponenttien paitsi Hadoopin kontitus suoraviivaistaa palvelun hallintaa ja skaalautumista, mikä myös rajaa järjestelmän solmujen tyypit kahteen: ohjelmistokontteja Rancherin avulla suorittaviin solmuihin sekä Hadoopia ja sen komponentteja suorittaviin solmuihin. Hallinta joudutaan suorittamaan erikseen solmutyypeille, sillä kontteja hallitaan Rancherilla, ja Hadoopia hallitaan sen omien käyttöliittymien kautta.

Ohjelmistokontit ja Hadoop kykenevät molemmat skaalautumaan sivuittain. Hadoopiin voidaan esimerkiksi liittää uusia sen prosesseja suorittavia solmuja ja Rancheriin voidaan liittää uusia Dockerin sisältäviä solmuja, joissa se voi suorittaa palvelun ohjelmistokontteja. Kontitettuja ohjelmistoja voidaan myös sammuttaa tai käynnistää lisää vastaamaan vaihtelevaan tarpeeseen.

Hadoopin kyky vastata tarpeeseen riippuu sen prosesseja suorittavien solmujen määrästä, minkä takia Hadoop ei kykene skaalautumaan nopeasti sisään tai ulos. Hadoop-järjestelmän koko on siis suunniteltava vastaamaan odotettua käyttömäärää, jotta resursseja menee mahdollisimman vähän hukkaan.

Palvelu kykenee skaalautumaan samalla tavalla kaikissa toimitusmalleissa, kunhan sen saatavilla on tarpeeksi resursseja. Korkean saatavuuden tila saavutetaan myös siihen kykenevillä ohjelmistoilla toimitusmallista riippumatta. Tämä pätee niin kauan, kun järjestelmässä on tarvittava määrä solmuja ja resursseja ominaisuuksien käyttämiseksi.

5.4 Palvelun toimitus

Kuten luvussa 5.3 mainittiin, palvelun solmutyypit rajautuvat kahteen: Dockeria suorittaviin solmuihin, jotka kytketään Rancheriin, sekä Hadoopia ja sen taustaprosesseja suorittaviin solmuihin. Jako on kannattavaa tehdä, sillä vaikka Hadoop-solmuilla on mahdollista suorittaa Docker-kontteja, voi Hadoopin resurssienkäyttö haitata tai heikentää niiden toimintaa ja päinvastoin. Esimerkiksi YARN ja Rancher saattavat vaatia solmun resursseja MapReduce-tehtävää tai ohjelmistokonttia varten samaan aikaan, mikä voi johtaa arvaamattomiin lopputuloksiin.

Eri toimitusmalleissa on otettava huomioon kohdeinfrastruktuuri. Hadoop on suunniteltu suoritettavaksi konesaleissa, joissa jokainen laite sisältää suunnilleen samankaltaisen laitteiston ja joissa ohjelmisto asennetaan suoraan laitteille ilman virtualisointia. Tämä tarkoittaa, että Hadoop tekee tiettyjä olettamuksia sen prosesseja suorittavasta infrastruktuurista, mitä Hadoop käyttää mm. replikoiden sijoittamiseen. [37.]

Hadoopia voidaan suorittaa virtuaalikoneilla, mutta sitä varten täytyy tehdä ylimääräistä työtä, jotta Hadoop toimisi oikein. Jos virtuaalikoneessa suoritettavalle DataNodelle annetaan esimerkiksi kolme virtuaalista kovalevyä, joista jokainen sijaitsee samalla fyysisellä kovalevyllä, voidaan kaikki levyille tallennettu tieto menettää, mikäli fyysinen kovalevy vioittuu. Hadoopin virtuaalikoneilla suorittaminen sisältää myös muita mm. tietoturvaan liittyviä ongelmia. [37.]

Seuraavissa aliluvuissa käydään läpi, mitä eri toimitusmalleissa täytyy ottaa huomioon. Luvuissa myös ehdotetaan toimituskeinoja, jotka olettavat, että Hadoopin virtuaalikoneilla suorittamiseen liittyvät haasteet on ratkaistu.

5.4.1 SaaS-toimitus

SaaS-toimituksessa Bittiumilla on täysi tieto infrastruktuurin rakenteesta ja vapaat kädet sen hallinnointiin. Helpoin ja varmin tapa pystyttää Hadoop-järjestelmä on asentaa se laitteistoiltaan samankaltaisille fyysisille palvelimille, mikä vastaa Hadoopin olettamuksia. Tämä on myös parhaiten dokumentoitu tapa Hadoopin asentamiseen ja käyttämiseen. Dockeria suorittavat solmut voivat olla joko fyysisiä palvelimia tai virtuaalikoneita. Fyysiset

koneet ovat virtuaalikoneita tehokkaampia, mutta virtuaalikoneet mahdollistavat infrastruktuurin helpomman hallinnan ja nopeamman skaalauksen tilanteissa, joissa järjestelmään täytyy lisätä uusi solmu.

5.4.2 Ohjelmistokomponenttien toimitus

Palvelun ohjelmistokontit voidaan toimittaa esimerkiksi Bittiumin konesalissa sijaitsevan Rancher-katalogin avulla, joka sisältää palvelun mallipohjan. Ohjelmistokonttien suhteen asiakkaalta vaadittaisiin siis vain toimivaa Rancher-palvelinta, jolla hän voi käyttää mallipohjaa palvelun konttien luomiseen ja käynnistämiseen. Hadoopia varten asiakkaille toimitettaisiin ohjeet, kuinka Hadoop-klusteri tulisi asentaa ja konfiguroida palvelun käyttöönottamiseksi.

5.4.3 Laitetoimitus

Laitetoimituksessa kaikki ohjelmistot on asennettu laitteille valmiiksi, jolloin asiakkaan täytyy vain asentaa laitteet omiin tiloihinsa palvelun käyttöönottamiseksi. Ohjelmistoja asennettaessa on varmistettava, että korkean saatavuuden mahdollistava rakenne saavutetaan. Docker-solmuilla tämä tarkoittaa sitä, että solmuja on tarpeeksi, jotta kontitetut ohjelmat voivat replikoida dataansa järkevästi. Rancher vaatii myös vähintään kolme Rancher-palvelinta korkean saatavuuden varmistamiseksi, joista jokaisen tulisi sijaita omalla fyysisellä laitteellaan, jotta järjestelmä olisi vikasietoinen laiterikkoja vastaan.

Hadoop vaatii puolestaan vähintään kaksi NameNodea ja kaksi ResourceManageria omilla fyysisillä laitteillaan vikasietoisuuden ja korkean saatavuuden varmistamiseksi. NameNodejen synkronointiin käytettäviä JournalNode-prosesseja voidaan tarvittaessa suorittaa niiden keveyden takia muita Hadoopin prosesseja suorittavilla laitteilla. Lisäksi palveluun tarvitaan tarpeeksi DataNode-prosesseja suorittavia laitteita, jotta järjestelmässä olisi tarpeeksi säilytystilaa ja replikoinnista olisi hyötyä.

Tässä toimitustavassa on myös mietittävä, että asennetaanko Hadoop virtuaalikoneina vai suoraan laitteille. Suora asennus laitteille olisi yksinkertaisin vaihtoehto, mutta toimituksessa on otettava huomioon palvelun skaalautuminen toimitettujen laitteiden ulkopuolelle. Asiakas voi esimerkiksi haluta laajentaa järjestelmää omilla solmuillaan, jotka saat-

tavat olla joko fyysisiä tai virtuaalisia palvelimia. Hadoopin suorittaminen tällaisella infrastruktuurilla, joka koostuu vaihtelevan kokoisista fyysisistä tai virtuaalisista solmuista, voi johtaa odottamattomiin ongelmiin.

6 Johtopäätökset ja pohdinta

Palveluun valittujen teknologioiden ominaisuuksien perusteella voidaan päätellä, että palvelu kykenee oikein konfiguroituna ja tarvittavilla resursseilla varustettuna vastaamaan suurimmalta osin esitettyihin haasteisiin. Seuraavissa luvuissa käydään läpi, miten palvelu vastaa eri haasteisiin sekä esitetään kehityskohteita ja jatkotutkimusaiheita.

6.1 Haasteet

Palvelulle esitettiin kolme skaalautuvuuteen liittyvää haastetta: tietovarannon kasvaminen, käyttäjämäärän kasvaminen ja tiedon saatavuus kaikissa tilanteissa. Kaikki haasteet liittyvät siis tavalla tai toisella resurssienkäytön lisääntymiseen ja optimoimiseen.

6.1.1 Tietovarannon kasvaminen

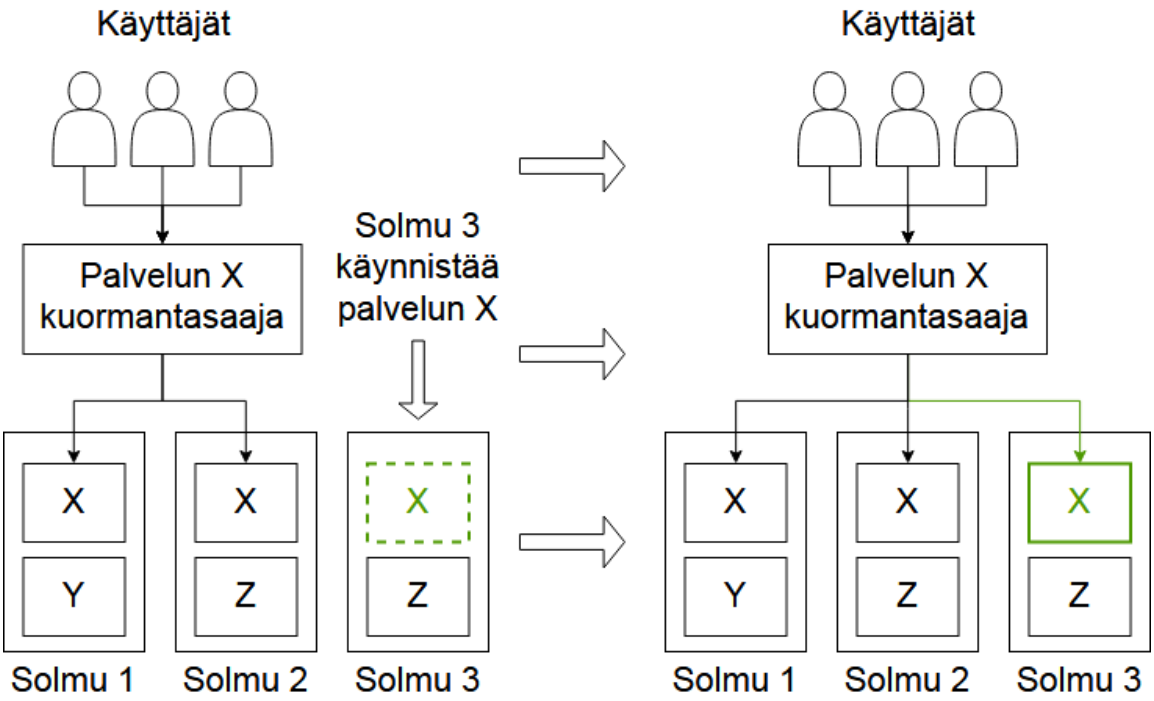
Järjestelmän pääasialliset tietoa säilyttävät komponentit ovat Hadoopin HDFS ja MongoDB. Molemmat ohjelmistot säilyttävät dataa hajautetusti usealla laitteella, mikä nostaa niille käytettävissä olevaa säilöntätilaa huomattavasti. Säilöntäkapasiteetin loppuessa järjestelmään täytyy kuitenkin lisätä uusia säilöntätilaa tarjoavia solmuja. MongoDB:tä voidaan lisäksi skaalata lisäämällä sitä suorittava ohjelmistokontti Docker-solmuille, joissa sitä ei vielä suoriteta.

Järjestelmän solmuja voidaan myös skaalata ylöspäin säilöntätilaa kasvattavilla komponenteilla. Ylöspäin skaalatessa on kuitenkin otettava huomioon Hadoopin olettamukset infrastruktuurista, jossa laitteet ovat suunnilleen yhtä tehokkaita.

6.1.2 Käyttäjämäärän kasvaminen

Suurin osa palveluun tulevasta käyttäjäliikenteestä kohdistuu johonkin ohjelmistokontissa suoritettavaan palveluun. Rancherin avulla kuormaa voidaan tasata lisäämällä järjestelmään uusi ohjelmistokontti, joka myös suorittaa suurella kuormituksella olevaa palvelua. Yksi Rancherin tarjoama infrastruktuuripalvelu on kuormantasaaja, jonka avulla käyttäjiä

voidaan ohjata järjestelmään lisätylle palveluinstanssille. Uuden ohjelmistokontin lisääminen kuorman tasaamista varten on esitetty kuvassa 20.



Kuva 20. Palveluinstanssin lisääminen

Hadoopiin kohdistuva kuorma, kuten luku- ja kirjoituspyynnöt, jakautuvat järjestelmän rakenteen vuoksi luonnollisesti DataNodejen välille. Järjestelmän NameNodeja täytyy skaalata ylöspäin, mikäli aktiviinen NameNode osoittautuu suuren kuorman alla liian vakavaksi pullonkaulaksi. Hadoopia ei kuitenkaan ole suunniteltu interaktiiviseen käyttöön, eikä sen oleteta kykenevän siihen, minkä takia sen hitaus ei ole niin merkittävää koko palvelun toiminnalle.

6.1.3 Korkea saatavuus

Kaikki järjestelmän tietoa säilyttävät palvelut sisältävät replikointikyvyn, jolla pystytään varmistumaan siitä, että tiedon saatavuus ei kärsi, vaikka jokin tietoa säilyttävä solmu putoaisi järjestelmästä. Rancherin avulla pystytään suorittamaan mm. kontitettujen palveluiden uudelleenkäynnistys häiriötilanteissa, kuormantasaus palveluiden ja käyttäjien vä-

lillä sekä palveluinstanssien lisääminen ja poistaminen tarvittaessa. Näiden ominaisuuksien avulla varmistetaan, että kontitetut palvelut ovat aina käyttäjien saatavilla, eikä resursseja varata käyttöön turhaan.

Hadoop on kehitetty vikasietoiseksi ja oikein konfiguroituna se kykenee pysymään toiminnallisena vakavissakin häiriötilanteissa. HDFS:n NameNode ja YARN:n ResourceManager vaativat kuitenkin seuraavassa luvussa esiteltävän Apache ZooKeeper-palvelun käyttöönoton, jotta kaikki niiden vikasietoisuuteen liittyvät ominaisuudet voidaan ottaa käyttöön. Ilman kyseistä palvelua passiivisen NameNoden aktivoiminen täytyy suorittaa käsin järjestelmänvalvojan toimesta, eikä ResourceManagereita voi olla järjestelmässä kuin yksi. [17.]

6.2 Kehityskohteet

Palvelun kaikkien ominaisuuksien hyödyntäminen vaatii Hadoopin ulkopuolisen palvelun, Apache ZooKeeperin käyttöönoton. ZooKeeper on keskitetty hajautettujen ohjelmistojen koordinointipalvelu, joka auttaa mm. ohjelmistojen synkronoinnissa ja hallinnassa. ZooKeeper itse on myös usealla laitteella suoritettava hajautettu, korkeasti saatava palvelu. ZooKeeper käyttää HDFS:n JournalNodejen tapaan enemmistöä toiminnan virheettömyyden varmistamiseksi. Enemmistövaatimuksen takia järjestelmässä täytyy siis olla kolme tai kolmea suurempi pariton määrä ZooKeeper-palvelimia. [38.]

HDFS:n aktiivisen ja passiivisen NameNoden automaattinen vaihto on rakennettu ZooKeeperin varaan. ZooKeeper pitää silmällä aktiivisen NameNoden terveystilaa ja tarjoaa mekanismin, jonka avulla vikatilaan ajautunut aktiivinen NameNode korvataan automaattisesti jollain järjestelmän passiivisella NameNodella. ZooKeeper myös huolehtii, että järjestelmässä on aina aktiivisena vain yksi NameNode. [17.]

Ilman ZooKeeperiä YARN:n ResourceManager ei ole korkeasti saatava ja sen vioittuminen estäisi YARN:n käyttämisen. ZooKeeperin avulla järjestelmässä voi olla useampi ResourceManager, joista jokin käynnistetään vikatilanteessa kuten NameNodejen kanssa. ZooKeeper huolehtii myös ResourceManagereiden kanssa, että vain yksi niistä on aktiivinen. [13.]

6.3 Jatkotutkimusaiheet

Hadoopin toimitukseen ja hallintaan voisi käyttää Apache Ambari -projektia, joka on Hadoop-klustereiden luontiin, hallintaan ja valvontaan kehitetty työkalu. Ambari koostuu palvelimesta ja hallinnoitaville koneille asennettavasta agenttiohjelmistosta. Ambari sisältää mm. ohjatun asennustoiminnon Hadoopin palveluiden asentamiseksi mille määrälle laitteita tahansa. Hadoop-klusterin hallinta, kuten eri palveluiden käynnistys ja pysäytys, voidaan suorittaa keskitetysti Ambarin kautta. Ambari tarjoaa myös käyttöliittymän Hadoop-klusterin tilan ja terveyden valvontaan. [39.]

Ambarin avulla valmiista Hadoop-klusterista voidaan luoda pohjapiirros, jonka perusteella on mahdollista luoda uusi samanlainen klusteri [40]. Tätä ominaisuutta voisi käyttää esimerkiksi esikonfiguroidun Hadoop-klusterin toimitukseen asiakkaille. Ambarin muut hallinnointiominaisuudet myös vähentäisivät huomattavasti Hadoop-klusterin ylläpitämiseen liittyvää taakkaa.

Ambari-palvelin ei kuitenkaan ole itse korkeasti saatava. Korkean saatavuuden tila on mahdollista saavuttaa, mutta sitä varten on tehtävä ylimääräistä työtä. Palvelimia voi esimerkiksi asentaa useamman, joista vain yksi on NameNodejen ja ResourceManagereiden tapaan aktiivinen. Palvelinten automaattista vaihtoa ja synkronisointia varten täytyy kehittää jokin mekanismi itse, tai käyttää jotain kolmannen osapuolen tarjoamaa ratkaisua. [41.]

Hadoop on mahdollista toimittaa esimerkiksi yhtenä tai useampana virtuaalikonemallina (engl. *virtual machine template*). Malli voisi esimerkiksi sisältää valmiiksi asennetun Hadoopin ja konfigurointiskriptejä, joilla valitaan, mitä Hadoopin prosessia tai prosesseja kyseinen solmu suorittaa.

Hadoop voidaan myös toimittaa ohjelmistokonttien avulla, mutta sen kontitus on haastavaa ja vaatii paljon suunnittelua [42]. Kontituksen ja varsinkin konttiorkestraation tarjoamat hyödyt voivat kuitenkin lopulta olla vaivan arvoisia.

Yksi mahdollinen Hadoop MapReduce -sovellus pilvipalvelulle on koneoppiminen. Järjestelmästä voisi esimerkiksi hakea tietyn tiedon perusteella kaikki mittaukset, joissa halutut parametrit esiintyvät. Tekoäly voidaan tämän datan avulla opettaa huomaamaan samat parametrit uusista mittauksista.

7 Yhteenveto

Työn tavoitteena oli tutkia pilvipalveluiden skaalautuvuuteen liittyviä haasteita Bittiumin tuotekehitysprojektia varten. Työllä haluttiin paljastaa mahdolliset pilvipalvelun kehitykseen liittyvät ongelmat sekä löytää mahdollisia kehityskohteita. Työ alkoi pilvipalveluiden esittelyllä, jossa kuvailtiin niiden yleiset ominaisuudet sekä toteutustavat. Seuraavaksi tutkittiin ohjelmistoja, joita palvelussa oli suunniteltu käytettävän. Ohjelmistojen kanssa kiinnitettiin erityistä huomiota niihin ominaisuuksiin, jotka liittyivät jollain tapaa esitettyihin haasteisiin.

Seuraavaksi havainnollistettiin mitä skaalautuminen oikeastaan on. Skaalautuminen käsiteltiin eri skaalautumistapojen kautta, jolloin myös esiteltiin eri tapojen hyödyt ja haasteet. Työ jatkui pilvipalvelun rakenteen syvällisemmällä kuvaamisella, jolloin myös havainnollistettiin, mitä rooleja aiemmin esitetyillä ohjelmistoilla palvelussa on. Erityistä huomiota kiinnitettiin ohjelmistokonttien käyttöön järjestelmässä.

Palvelun rakennetta kuvailtaessa huomattiin, että Hadoopin suoritukseen ohjelmistokonteissa liittyi huomattavasti haasteita, jonka takia Hadoopin kontituksesta päätettiin luopua. Tämä johti kahden solmutyypin malliin, jossa ohjelmistokontteja ja Hadoopia suoritettiin erillään. Jakautunut rakenne otettiin jatkossa huomioon, kun palvelun hallintaa ja skaalautumista käsiteltiin yhdessä sen toimitustapojen kanssa.

Seuraavaksi pohdittiin, kuinka ohjelmistot kykenevät vastaamaan työn alussa esitettyihin haasteisiin. Ohjelmistojen ominaisuuksien perusteella pystyttiin päättelemään, että ainoa komponenttien skaalautumista rajoittava tekijä on komponentteja suorittavien solmujen määrä. Lisäksi tehtiin johtopäätös, että kaikki palvelun komponentit eivät nykyisellä rakenteella ja ohjelmistoilla kykene saavuttamaan korkean saatavuuden tilaa.

Työn lopuksi esiteltiin ohjelmistoja ja ehdotuksia, joiden avulla pilvipalvelua voisi kehittää eteenpäin. Esitetyt ohjelmistot tarjosivat mm. ominaisuuksia, jotka auttaisivat palveluun sisällytettynä sen hallinnassa, sekä tekisivät lopuistakin komponenteista korkeasti saatavia.

Lähteet

1 Tietoa yhtiöstä – Bittium. Saatavilla: https://www.bittium.com/bittium_lyhyesti/tietoa_ja_taloudellisia_lukuja/tietoa_yhtiosta Viitattu 30/3/2018, 2018.

2 The NIST Definition of Cloud Computing – National Institute of Standards and Technology. Saatavilla: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf Viitattu 22/1/2018, 2018.

3 Cloud Computing Primer for IT Pros – Microsoft TechNet. Saatavilla: https://blogs.technet.microsoft.com/yungchou/2010/12/17/cloud-computing-for-it-pros-26-what-is-cloud/ Viitattu 29/1/2018, 2018

4 What is IaaS? – Microsoft Azure. Saatavilla: https://azure.microsoft.com/en-in/overview/what-is-iaas/ Viitattu 24/1/2018, 2018.

5 What is cloud computing? – IBM. Saatavilla: https://www.ibm.com/cloud/learn/what-is-cloud-computing Viitattu 24/1/2018, 2018.

6 What is SaaS? – Microsoft Azure. Saatavilla: https://azure.microsoft.com/en-in/overview/what-is-saas/ Viitattu 24/1/2018, 2018.

7 ISO/IEC 17788:2014 – International Organization for Standardization. Saatavilla: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html Viitattu 25/1/2018, 2018.

8 Data replication – IBM. Saatavilla: https://www.ibm.com/analytics/data-replication Viitattu 16/3/2018, 2018.

9 What is Apache Hadoop? – Apache Hadoop. Saatavilla: http://hadoop.apache.org/ Viitattu 31/1/2018, 2018.

10 Apache Hadoop YARN – Apache Hadoop. Saatavilla: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html Viitattu 14/3/2018, 2018.

11 Apache Hadoop YARN - Concepts and Applications – Hortonworks. Saatavilla: https://hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/ Viitattu 15/3/2018, 2018.

12 ResourceManager Restart – Apache Hadoop. Saatavilla: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceManagerRestart.html Viitattu 26/3/2018, 2018.

13 ResourceManager High Availability – Apache Hadoop. Saatavilla: https://hadoop.apache.org/docs/current2/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html Viitattu 15/3/2018, 2018.

14 NodeManager – Apache Hadoop. Saatavilla: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/NodeManager.html Viitattu 22/3/2018, 2018.

15 HDFS Architecture – Apache Hadoop. Saatavilla: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html Viitattu 31/1/2018, 2018 .

16 HDFS High Availability With NFS – Apache Hadoop. Saatavilla: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html Viitattu 18/2/2018, 2018.

17 HDFS High Availability With QJM – Apache Hadoop. Saatavilla: https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html Viitattu 21/3/2018, 2018.

18 MapReduce Tutorial – Apache Hadoop. Saatavilla: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html Viitattu 21/2/2018, 2018.

19 Introduction to MongoDB – MongoDB. Saatavilla: https://docs.mongodb.com/manual/introduction/ Viitattu 16/3/2018, 2018.

20 MongoDB Architecture – MongoDB. Saatavilla: https://www.mongodb.com/mongodb-architecture Viitattu 15/3/2018, 2018.

21 What is a Container – Docker. Saatavilla: https://www.docker.com/what-container Viitattu 26/2/2018, 2018.

22 Containers at Google – Google Cloud. Saatavilla: https://cloud.google.com/containers/ Viitattu 27/2/2018, 2018.

23 Get Started, Part 1: Orientation and setup – Docker. Saatavilla: https://docs.docker.com/get-started/ Viitattu 1/3/2018, 2018.

24 Docker overview – Docker. Saatavilla: https://docs.docker.com/engine/docker-overview/ Viitattu 19/3/2018, 2018.

25 Dockerfile reference – Docker. Saatavilla: https://docs.docker.com/engine/reference/builder/ Viitattu 19/3/2018, 2018.

26 About Docker Cloud – Docker. Saatavilla: https://docs.docker.com/docker-cloud/ Viitattu 1/3/2018, 2018.

27 Manage data in Docker – Docker. Saatavilla: https://docs.docker.com/storage/ Viitattu 1/3/2018, 2018.

28 Overview of Docker Compose – Docker. Saatavilla: https://docs.docker.com/compose/overview/ Viitattu 1/3/2018, 2018.

29 Overview of Rancher – Rancher. Saatavilla: https://rancher.com/docs/rancher/latest/en/ Viitattu 13/3/2018, 2018.

30 Catalog – Rancher. Saatavilla: http://rancher.com/docs/rancher/v1.6/en/catalog/ Viitattu 13/3/2018, 2018.

31 Rancher Compose – Rancher. Saatavilla: http://rancher.com/docs/rancher/v1.6/en/cattle/rancher-compose/ Viitattu 13/3/2018, 2018.

32 Installing Rancher Server (High Availability) – Rancher. Saatavilla: http://rancher.com/docs/rancher/v1.0/en/installing-rancher/installing-server/multi-nodes/ Viitattu 24/3/2018, 2018.

33 Concepts – Rancher Docs. Saatavilla: http://rancher.com/docs/rancher/v1.0/en/concepts/ Viitattu 24/3/2018, 2018.

34 Understanding When to use RabbitMQ or Apache Kafka – Pivotal. Saatavilla: https://content.pivotal.io/rabbitmq/understanding-when-to-use-rabbitmq-or-apache-kafka Viitattu 17/3/2018, 2018.

35 Consumer Acknowledgements and Publisher Confirms – RabbitMQ. Saatavilla: http://www.rabbitmq.com/confirms.html Viitattu 17/3/2018, 2018.

36 What is Load Balancing? – Nginx. Saatavilla: https://www.nginx.com/resources/glossary/load-balancing/ Viitattu 18/3/2018, 2018.

37 Virtual Hadoop – Hadoop Wiki. Saatavilla: https://wiki.apache.org/hadoop/Virtual%20Hadoop Viitattu 25/3/2018, 2018.

38 ZooKeeper Overview – Apache. Saatavilla: https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription Viitattu 26/3/2018, 2018.

39 Introduction – Apache Ambari. Saatavilla: https://ambari.apache.org/ Viitattu 27/3/2018, 2018.

40 Blueprints – Apache. Saatavilla: https://cwiki.apache.org/confluence/display/AMBARI/Blueprints Viitattu 27/3/2018, 2018.

41 How to setup High Availability for Ambari server? – Hortonworks Community Connection. Saatavilla: https://community.hortonworks.com/questions/402/how-to-setup-high-availability-for-ambari-server.html Viitattu 28/3/2018, 2018.

42 A Production Ops Guide to Deploying Hadoop in Docker Containers – Portworx. Saatavilla: https://docs.portworx.com/applications/hadoop-docker.html Viitattu 28/3/2018, 2018.