jamk.fi

Big data arkkitehtuurit ja teknologiat

Miikka Haatanen

Opinnäytetyö
Joulukuu 2016

Tekniikan ja liikenteen ala
Insinööri (AMK), tietotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Haatanen, Miikka	Opinnäytetyö, AMK	7.12.2016
	Sivumäärä 52	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Big data arkkitehtuurit ja teknologiat

Tutkinto-ohjelma
Tietotekniikan koulutusohjelma

Työn ohjaaja(t)
Mika Rantonen
Antti Häkkinen

Toimeksiantaja(t)
JYVSECTEC
Marko Vatanen

Tiivistelmä

Opinnäytetyön tavoitteena oli tutustua big datan arkkitehtuuriin ja sen taustalla toimiviin teknologioihin. Lisäksi tavoitteena oli pystyttää kolmen noden Hadoop-klusteri ja testata sen toimintaa.

Big dataa kuvaa parhaiten kolmen v-kirjaimen yhdistelmä: volyymi, vauhti ja vaihtelevuus. Ne kuvastavat hyvin miten erilaista dataa nykyään tuotetaan ja kuinka paljon sitä kertyy. Tulevaisuudessa datan määrä tulee moninkertaistumaan, joten big data alkaa ilmiönä viimeistään nyt olemaan ajankohtainen.

Big datan alueella on paljon potentiaalia, jota ei ole ymmärretty, tai haluttu vielä ottaa käyttöön yrityksissä. Teoriaosuudessa tutkittiin big datan arkkitehtuurin ja teknologioiden lisäksi myös mahdollisia hyötyjä ja haittoja yritysten kannalta, jotka big dataa aikovat hyödyntää.

Käytännön osuudessa pystytettiin IBM:n BigInsights QuickStart Edition, jossa otettiin käyttöön kolmen Hadoop-klusterin palvelinympäristö. Asennusvaiheessa tärkeänä osana oli avoimen lähdekoodin Hadoop-monitorointityökalu Ambari, jonka avulla voitiin selaimen kautta asentaa ja konfiguroida klusteria.

Avainsanat (asiasanat)
Big data, Hadoop, Ambari, YARN, MapReduce

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Haatanen, Miikka	Bachelor's thesis	07.12.2016
		Language of publication: Finnish
	Number of pages: 52	Permission for web publication: x

Title of publication
Big Data architectures and technologies

Degree programme
Information Technology

Supervisor(s)
Mika Rantonen
Antti Häkkinen

Assigned by
JYVSECTEC
Marko Vatanen

Abstract

The purpose of this thesis was to study in detail the fundamentals of Big Data architectures and technologies that make it possible to work. Additionally, the objective was to implement and test a three-node Hadoop cluster.

The best way to describe Big Data is with three V letters: Volume, Velocity and Variety. Those Vs describe well how much different type of data is produced and how much of it will accumulate. In the future the amount of data will multiply, therefore, finally Big Data as a phenomenon will be timely.

There is plenty of potential in Big Data that has not yet been understood or implemented by corporations. In the theory part of this thesis Big Data architecture and technologies are studied in detail, however, the thesis also reflects on some of the benefits and drawbacks of implementing Big Data in corporation environment.

The practical part describes the implementation of IBM BigInsights QuickStart Edition with a three-node Hadoop cluster server environment. Open-source Hadoop monitoring tool Ambari had a key role in the installation process, which made it possible to install and configure the cluster via browser.

Keywords/tags (subjects)
Big Data, Hadoop, Ambari, YARN, MapReduce

Miscellaneous

Sisältö

Kuviot

Lyhenteet

NIST	National Institute of Standards and Technology
SLA	Service Level Agreement
HDFS	Hadoop Distributed File System
API	Application Programming Interface
NLP	Natural Language Processing
SQL	Structured Query Language
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
DaaS	Data as a Service

1 TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantajana toimi JYVSECTEC, joka on kyberturvallisuuden tutkimus-, kehitys- ja koulutuskeskus. Se tuottaa konsultointi-, tutkimus-, testaus- ja koulutuspalveluja, sekä kyberturvallisuusharjoituksia. (JYVSECTEC, 2015)

JYVSECTEC toimii Jyväskylän ammattikorkeakoulun kanssa yhteistyössä Dynamon kampuksella Lutakossa. Sen käyttämä RGCE-kehitysympäristö (Realistic Global Cyber Environment) mahdollistaa organisaatioiden kyberturvallisuuden kehittämisen ja harjoitusten toteuttamisen nykyaikaisissa puitteissa ja asiantuntevien henkilöiden tuella. (JYVSECTEC, 2015)

JYVSECTEC:n rakennus käynnistyi Jyväskylän ammattikorkeakoulun IT-instituutin projektina syyskuussa 2011. Sen päämääränä oli luoda yksi Suomen johtavista kyberturvallisuuden tutkimus-, kehitys- ja koulutuskeskuksista, samalla kehittäen kansallista ja kansainvälistä yhteistyöverkostoa yrityksille. Rahoittajina projektissa toimivat Keski-Suomen Liitto ja Euroopan aluekehitysrahasto. Jatkorahoituksen myötä turvallisuuskeskittymän kehitys jatkuu ainakin vuoden 2017 loppuun saakka. (JYVSECTEC, 2015)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli tutustua big data –arkkitehtuuriin ja sen taustateknologioihin. Samalla tutustuttiin eri tuotteiden toteutustapoihin ja niiden etuihin ja haittoihin. Kun tietoperustassa oli tutustuttu yleisesti big dataan ja siinä käytettäviin tekniikoihin, mahdollisti se demoamisen olemassa olevilla tuotteilla.

Tavoitteena on myös hyödyntää opinnäytetyöstä opittua tietoa työharjoittelussa, joka on jatkumoa opinnäytetyölle.

2 BIG DATA

2.1 Mitä on big data?

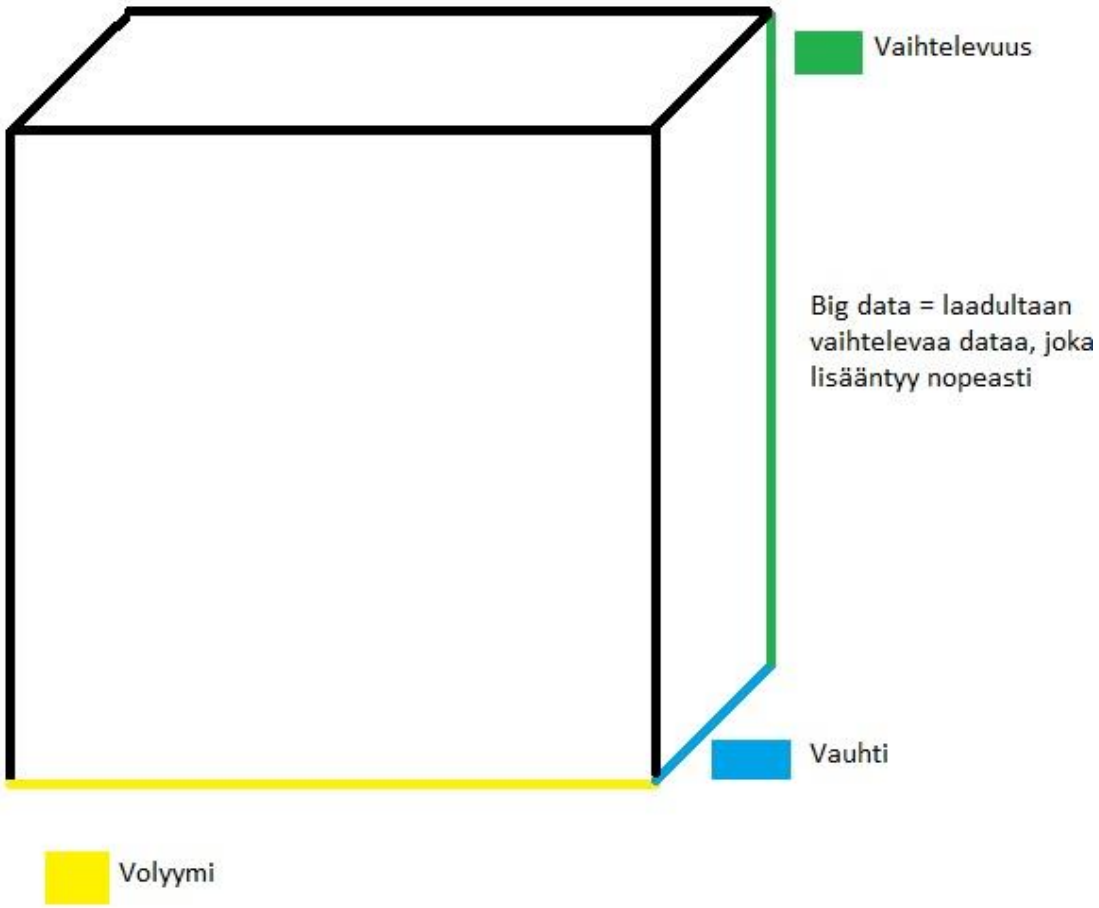
Big Datasta on puhuttu kiihtyvällä tahdilla vuodesta 2005 lähtien, mutta hypeksi asti se on noussut vuoden 2011 jälkeen. Nykyisen datan sisältö on entistä vaihtelevampaa ja tulevaisuudessa se tulee entistä vaihtelevammaksi, jota kuvaa hyvin kolmen V-kirjaimen yhdistelmä: Volume, Variety ja Velocity. Suomeksi sanat kääntyvät myös kolmelle V:lle: volyymi, vaihtelevuus ja vauhti. (Salo 2014, 26.)

Volyymilla kuvataan datan suurta määrää ja moni mieltääkin big datan käsitteenä liittyvän vain datan määrään. Big data on kuitenkin muutakin kuin suuri koko, kuten muut V-kirjaimet kertovat. Siinä mielessä datan määrä on merkittävässä roolissa, että sitä tuotetaan jatkuvasti enemmän ja tahti vain kiihtyy. Tämä ilmiö on ollut merkittävänä tekijänä big data käsitteen synnyssä. Nykyisen arvion mukaan maailmassa on noin 5 zettatavua dataa, kun vuonna 2011 vastaava luku oli 2 zettatavua. (Salo 2014, 26.)

Vaihtelevuudella tarkoitetaan datan laadun vaihtelua. Seassa on strukturoitua dataa, mutta ylivoimainen osuus datasta on strukturoimatonta. Näiden kahden välistä eroa on joskus vaikea hahmottaa, eikä hahmotusta helpota käsite semistrukturoidusta datasta. Hyvänä esimerkkinä strukturoidusta datasta on asiakkaasta kerätyt henkilö- ja ostotiedot. Strukturoimatonta dataa kuvastaa puolestaan valvontakameran videomateriaali. Semistrukturoitua dataa voi olla esimerkiksi video, jossa metatiedoissa on mainittu kuvan henkilöt, aika tai paikka. (Salo 2014, 27.)

Vauhdilla tarkoitetaan datan virtausnopeutta ja sen myötä nopean reagoinnin tarpeeseen, kuten liiketoiminnassa. Jos data käsitellään nopeasti, se voi ratkaiseva tekijä automaattisesti tehtävissä ratkaisuissa, kuten osakemarkkinoilla ja terveydenhuollossa. Koska dataa tulee niin valtavalla nopeudella, ei kannata kaikkea tallentaa erikseen, vaan parempi on käsitellä raakadatan tiedot lennosta ja suorittaa tallennus sen jälkeen, joko sellaisenaan tai koottuna pakettina. Tällöin datavirran esikäsittely nousee entistä suurempaan rooliin big dataa hyödyntäessä. (Salo 2014, 27.)

Kolmea V-kirjainta on käytetty kuvaamaan ilmiötä paineesta, joka suurista ja nopeasti kasvavista datamääristä aiheutuu. Tästä ilmiöstä on muodostunut käsite big datasta, kuten kuviosta 1 havaitaan. Mitä enemmän kolme V-ulottuvuutta täyttyvät, sitä selkeämmin dataa voidaan määritellä big dataksi. (Salo 2014, 28.)



Kuvio 1. Big datan kolmen V-kirjainta

Yleisesti data voidaan määritellä luonnosta saatavien esimerkkien mukaisesti paikallaan olevaksi dataksi, eli järveksi, tai liikkuvaksi datavirraksi, eli joeksi. Big data -käsitteeseen sisältyy molemmat ulottuvuudet. Parhaana esimerkkinä virtaavasta datasta toimii sensorit, jotka tuottavat valtavan määrän dataa ja tulevat jatkossa tuottamaan sitä entistä enemmän. Tälläistä datamäärää ei ole järkevää tallentaa sellaisenaan, mutta sitä voidaan silti hyödyntää. Esimerkkinä laitteiden sensoridatan hyödyntämisestä ovat mottorien, hissien tai vastaavien laitteiden ennakoivat huoltotoimenpiteet. Sama periaate pätee myös sairaalassa toimivien laitteiden kanssa, jolloin niissä havaittuihin poikkeuksiin pystytään reagoimaan. (Salo 2014, 29.)

2.2 Big data -ilmiön nousu

Starttina big datan kiinnostuksen kasvulle voidaan pitää arvovaltaisen McKinseyn julkaisemaa selvitystä vuodelta 2011, nimeltään *Big data: The next frontier of innovation, competition and productivity*. Raportissa mainittiin jopa 1000 mrd. € vuotuista lisäarvoa ja kustannussäästöjä big datan avulla. Raportti sai paljon julkisuutta ja sitä on siteerattu paljon. Myös google-haut ovat kasvaneet julkaisun jälkeen. (Salo 2014, 30.)

Big data –ilmiön kasvussa suuressa roolissa on digitaalisen datan valtava kasvunopeus, joka on seurausta tietotekniikan roolin kasvusta. Esimerkiksi erilaiset lokitiedot ovat jatkuvasti kasvava datan lähde. Toinen esimerkki big datasta ja digitaalisesta datasta on sosiaalinen media kuten Twitter -viestit. Nekin kuitenkin ovat datamäärältään pieniä verrattuna kuva- ja videopalveluihin, kuten Facebook ja Instagram. Koska nykyiset päätelaitteet ovat niin kehittyneitä, ei datan määrän kasvulle näy loppua.

Tärkeimpänä tekijänä ilmiön taustalla ovat sensoridata ja älykkäät koneet, joiden tuottaman datan kokoa ei ole vielä täysin ymmärretty. Verrattaessa digitaaliseen dataan, on sensoridatan vaikutus kuitenkin moninkertainen big datassa. Tietotekniikan tulisikin mahdollistaa uutta liiketoimintaa, eikä estää sitä, mikä on ollut kasvattamassa painetta reagoida big dataan.

Big data on pilvipalveluiden tapaan kattokäsitteenä havainnoille, teorioille ja ratkaisuille. Yleisesti voidaan jakaa big data kahteen asiaan:

1. Havaintoon datan määrän valtavasta kasvunopeudesta ja sen monipuolistumisesta.
2. Tuotteisiin, palveluihin ja tekniikoihin, jolla datan kolmen V:n tuottamiin haasteisiin pyritään vastaamaan.

Esimerkkinä datan kasvulle käytetään yleensä sosiaalista mediaa, kuten twiittien määrää tai youtubeen ladattuja videoita. Sensoridatan esimerkkinä toimii parhaiten lentokoneiden tuottama data. Konkreettisia lukuja on käsitelty IDC:n julkaisemassa raportissa vuonna 2011, jossa maailmassa arvioitiin luotavan 1,8 zettatavua dataa kyseisenä vuonna. Ennusteessa määrän arvioitiin kasvavan 35-kertaiseksi vuoteen

2020 mennessä. Lentokone Boeing 787 taas tuottaa dataa keskimäärin puoli teratavua lentoa kohden. (Salo 2014, 31.)

Dataa on siis todella paljon ja vihdoin on alettu ymmärtää haaste, joka on tällä hetkellä jo ajankohtainen ja tulevaisuudessa moninkertainen. Nykyiset tietojärjestelmät eivät ole optimaalisia valtavien ja moninaisten datamäärien käsittelyyn. Näin ollen osa datan sisältämästä informaatiosta jää kokonaan käyttämättä. Käyttämättömässä datassa saattaa piillä tietoa, jotka ratkaisevasti voivat muuttaa päätöksentekoa. On siis havahduttu siihen, että datassa on valtavasti käyttämätöntä arvoa ja tietoa. (Salo 2014, 31.)

2.3 Avoin data

Avoin data ja linkitetty data ovat tärkeänä osana big data –ilmiössä. On tärkeää havaita, että muiden lähteiden tuottama avoin data voi antaa yrityksille huomattavaa lisäarvoa, kun sitä linkitetään yrityksen itse tuottaman datan kanssa. Yleisesti avoin data mielletään julkishallinnon avattuihin datavarastoihin, mutta myös yritysten keskinäinen datanvaihto on tärkeää. Jo nyt datamarkkinat ovat pilvessä suurilta osin ja tulevaisuudessa todennäköisesti kokonaan. Pilven datamarkkinat onkin suurimpia liiketoiminnan mahdollisuuksia big datassa. (Salo 2014, 43.)

Ainoastaan datavarantojen avaaminen julkishallinnossa ei ole riittävää kehitystä julkisen datan näkökulmasta, vaikka se on hyvä alkutekijä. Tarvitaan myös standardoidut formaatit, käyttöehdot ja rajapinnat, sekä on saatava helposti tietoa, mitä dataa on saatavilla, jotta vältytään ylimääräiseltä työtä datan hyödyntämisessä. (Salo 2014, 43.)

Avoimessa datassa olisi tärkeää keskittyä uuden datan tuottamiseen, eikä niinkään olemassa olevien varastojen avaamiseen. Datan keräämishetkellä ei voida tietää, missä kaikessa sitä voidaan hyödyntää, joten ilman tunnistettavia esteitä kannattaisi dataa julkaista pilvessä, standardointeja unohtamatta. Jos avointa dataa on hajallaan, ei sitä voida hyödyntää kuin marginaalisesti. Sen sijaan keskitetysti hallitulla ja helposti käyttöönotettavalla datalla hyödyt voivat olla suuret. Linkitettyjen datavarastojen analysointi ja yhdistäminen onnistuu vain, jos tiedot löytyvät yhdestä

paikasta. Oikea paikka avoimelle datalle on siis julkisessa pilvessä, jossa sen yhdistäminen muihin avoimiin datavarastoihin ja datamarkkinoiden maksullisiin aineistoihin mahdollistuu. (Salo 2014, 44.)

Avoimessa datassa piilee suurta potentiaalia ajatellen uusia liiketoimintamahdollisuuksia. Kun dataa kerätään ja tuotetaan paljon ja sitä on saatavilla julkisesti, voi syntyä uusia innovaatioita, joita ei aikaisemmin ole edes kuviteltu. Siksi onkin tärkeää kerätä paljon dataa ja monessa eri muodossa, koska kaikille sama data ei ole tärkeää. Datan tallennuskustannusten jatkuva lasku edesauttaa datan keräämismahdollisuuksia. (Salo 2014, 45.)

McKinsey julkaisi vuonna 2013 raportin *Open data: Unlocking innovation and performance with liquid information.* Raportissa arvioitiin avoimen datan taloudellisen lisäarvon olevan 3000 – 5000 mrd. $:n haarukassa vuotuisesti. Luvut kuulostavat absureilta, mutta niistä saa hyvin käsityksen, kuinka valtava potentiaali avoimessa datassa piilee. (Salo 2014, 35.)

2.4 Tietoturva big datassa

2.4.1 Väärinkäytön kohteet

Raakadata ei yleensä ole arvokasta, mutta siitä prosessoidut tiedot ja analyysit puolestaan ovat. Toisaalta raakadatan päätyessä vääriin käsiin voi koitua yrityksille imagohaittoja ja liiketoimintatappioita. On siis tärkeää suojata data heti sen keräysvaiheesta aina lopullisiin analysointituloksiin asti, myös vanhentuneiden tietojen osalta. Pahimmassa tapauksessa prosessi- tai rekisteritietojen vuotaminen voi vaarantaa koko liiketoiminnan. Tietomurrot eivät yleensä kohdistukaan raakadatavarastoihin, vaan prosessoituihin tuloksiin tai algoritmeihin, joista voidaan havaita liiketoiminnan ominaisuuksia. Tärkeimpänä kiinnostuksen kohteina tietomurtajilla ovat yksilölliset tiedot, kuten maksuvälinetiedot, tilitapahtumat tai henkilön omat tiedot. (Salo 2014, 51.)

Datan arvo määrittyy analysoinnista saaduilla tuloksilla, joten uhkana onkin tulosten väärentäminen omaa etua ajatellen. Parhaana esimerkkinä toimii pörssikaupat, joissa käytössä on päätöksentekoon vaikuttavia algoritmeja. Tietomurtaja voisi halutessaan

muokata lähdetietoja, jolloin automaattinen kaupankäynti muuttuu itselleen edulliseksi. (Salo 2014, 51.)

2.4.2 Fyysinen tietoturva

Big data –ratkaisut ovat usein klusterimuotoisia tallennus- ja laskenta-arkkitehtuureiltaan, kuten suosituin, avoimen lähdekoodin ratkaisu Hadoop. Tätä ratkaisumallia ei ole alunperin ajateltu salaisen tiedon säilytykseen. Sama pätee myös NoSQL-tietokantoja käyttäviin ratkaisuihin. Fyysisesti laitteisto on samanlaista kuin muissakin tietojärjestelmissä tietoturvan kannalta, mutta komponentit, jotka hidastavat luku-, kirjoitus- ja laskentaoperaatioita. Data on yleensä suojaamattomana järjestelmän levyllä suorituskyvyn maksimoinniksi. Yksittäisten data-alkioiden erillinen määrittely tietoturvaluokituksien mukaisesti on lähes mahdotonta. (Salo 2014, 52.)

Paras ratkaisu onkin pyrkiä reaaliaikaiseen monitorointiin ja automaattiseen reagointiin tietoturvauhkia havaittaessa. Kun määritetään sallitut toimenpiteet järjestelmässä ja yhdistetään koneoppiminen, voidaan uhkat havaita automaattisesti, sekä myös estää niiden toteutuminen. Silloin ihmiselle jää tehtäväksi tarkastaa raportit ja toimenpiteiden oikeudellisuuden tarkastus. (Salo 2014, 52.)

2.4.3 Pilvipalveluiden tietoturva

Pilvipalvelut tarjoavat skaalautuvat ja lähes rajattomat mahdollisuudet big data sovelluksille, mutta luottamuksen puute on ollut hidastamassa kehitystä. Palveluntarjoajat ovat pyrkineet osoittamaan luottamuksensa, eikä heidän kannaltaan olisi edes järkevää ottaa riskejä, omien liiketoiminnan volyymin ollessa moninkertaisia asiakkaisiin nähden. (Salo 2014, 53.)

Esille on noussut tapahtumia, kuten yhdysvaltalaisten pilvipalveluiden joustaminen salaiselle palvelulle kansallisen turvallisuuden nimissä. Jokainen vastaavanlainen tapaus aiheuttavat epävarmuutta siirtyä pilvipalveluihin. Yritysten ei kuitenkaan kannata olla huolissaan, sillä vastaavat tapahtumat ovat äärimmäisen harvinaisia. Turvallisuutta voi myös kasvattaa siirtämällä toiminta yksityiseen pilveen tai itsehallittuihin laitteisiin. (Salo 2014, 53.)

2.4.4 Yksityisyydensuoja

Big data on herättänyt paljon kritiikkiä yksityisyydensuojan näkökulmasta, sillä dataliikenteen solmukohdassa toimijoilla on mahdollisuus tuottaa varsin tarkkoja analyyseja yksittäisistä asiakkaista. Profilointiin riittää jo pelkästään päätelaitteen tunnistaminen ja sen suorittamat toimenpiteet, eli palvelun ei tarvitse olla edes asiakkaalle näkyvä. Kun usealla käyttökerralla profiilista muodostuu uniikki, on se mahdollista linkittää henkilön identifioivaan tietoon ja sitä kautta yksilön identiteettiin. (Salo 2014, 54.)

Palvelun siirtyessä kuluttajan käyttäytymisen seuraamiseen reaalimaailmassa, nousee analyysien mahdollisuudet merkittävästi. Kaupat voisivat profiloida asiakkaitaan ostosten ja tapojen perusteella, jolloin voi tehdä erilaisia johtopäätöksiä. Verkossa sosiaalisen median kautta puolestaan kuluttajia voidaan profiloida erilaisten tahojen toimesta ja tarkoitusperiä voi olla monia. (Salo 2014, 54.)

Kuluttajan kannalta häneen liittyvästä datasta on tärkeää selvitä sen hyödyntämisen keinot ja tarkoitukset. Nykymaailmassa kuluttaja olettaa saavansa tiedon, mitä he saavat vastineeksi tietojensa luovutusta vastaan. Historiassa tiedot on onnistuneesti pidetty salassa ja niitä hyödyntävät menetelmät ovat olleet liikesalaisuuksia. Palveluiden käyttöehdot ovat nykyisellään monesti niin pitkiä, ettei niitä kovin moni edes vaivaudu lukemaan, vaikka samalla muodostuukin juridinen sopimus. Käyttäjän kannalta onneksi säädökset vaativat yrityksiltä selkokielistä kuvausta ehtojen pääkohdista. Yrityksille onkin ohjeistettu, ettei asiakasta kannata yllättää tarkkuudella, jolla se asiakkaansa tuntee. Kun datankeruun tavoitteet kerrotaan selkeästi, on mahdollista varoittaa asiakasta ja kertoa samalla prosessin hyödyistä asiakkaalle. (Salo 2014, 55.)

Dataliiketoimintaan liittyviä lakeja voi olla vaikeaa tulkita, varsinkin kun osa niistä on ajoilta ennen internetiä. Kun asiakkaita on vielä globaalisti, eivät resurssit riitä joka maan lainsäädäntöön tutustumiseen ja arviointiin. Yrityksen kannalta asiakkaan profiilitietoja ei saakaan usein hyödyntää muuten kuin osassa liiketoimintaa, kuten markkinoinnissa. Profiilin käyttöä palveluiden hinnoitteluun ja rajaukseen ei ole suotavaa käyttää. (Salo 2014, 55.)

Kehitys on menossa siihen suuntaan, että kuluttaja saa itse päättää itseään koskevasta datasta. Tämä on seurausta lainsäädännöstä ja kuluttajapalveluiden käytännöistä. Hiljalleen siirrytään aikaan, jossa asiakas saa omat asiakastietonsa palvelusta omasta pyynnöstään. Tulevaisuudessa asiakkaan data pitää todennäköisesti olla vakiomuotoista, jolloin palveluntarjoajan vaihdossa voidaan tietoja siirtää ongelmitta. (Salo 2014, 55.)

EU:n tietoturvadirektiivi on aiheuttanut paljon keskustelua, sillä siinä on määritelty kuluttajalle oikeus kieltää oman datansa hyödyntämistä. On myös suunniteltu mahdollisuutta omien tietojen poistoon tietokannoista, ellei sille ole lainsäädännöllistä estettä. Tekninen toteuttaminen on osoittautunut kuitenkin haasteelliseksi ja kovaa vastustusta on kuulunut esimerkiksi hakukoneyrityksistä. Määräykset eivät ole tällä hetkellä kovin selkeitä, mutta suuntauksena on päätäntävallan siirtäminen kuluttajalle. (Salo 2014, 55.)

Lopullinen vastuu on aina käyttäjällä itsellään, vaikka lainsäädännön suuntaukset ovatkin vaikuttamassa tilanteeseen. Julkiseen verkkoon päätyvä data voi olla mahdotonta poistaa kokonaan. Verkkoidentiteetit vahvistuvat ja yksilöllistyvät entisestään yrityste fuusioituessa ja integroituessa. Palveluiden keskinäinen kilpailu nostaa identiteettien arvoa ja niitä käytetäänkin rajaa hipovalla tavalla, samalla koittaen saada kuluttajan hyväksynnän. Nämä toimenpiteet ovatkin omiaan muokkaamaan verkkopalveluiden moraalikoodistoa. (Salo 2014, 56.)

3 PILVIPALVELUT

3.1 Yleiskuvaus pilvipalveluista

Pilvipalveluita voidaan yleisesti kuvata NIST:n, eli yhdysvaltalaisen mittaustekniikoita, standardeja ja tekniikkaa kehittävän viraston määrittelemällä tavalla: *Cloud computing on toimintamalli, joka mahdollistaa pääsyn vapaasti konfiguroitaviin ja skaalautuviin tietotekniikkaresursseihin, jotka voidaan ottaa käyttöön tai poistaa käytöstä helposti ja nopeasti.* Tietotekniikkaresursseilla tarkoitetaan laskentatehoa,

tallennustilaa, sovellusalustoja ja sovelluksia, joihin asiakkaalla on pääsy verkossa. NIST on nimennyt viisi ominaispiirrettä pilvipalveluille:

1. itsepalvelullisuus
2. laiteriippumaton pääsy palveluihin
3. yhteiskäyttöiset resurssit
4. nopea joustavuus
5. tarkka käytön mittaus

Itsepalvelullisuus mahdollistaa resurssien käyttöönoton tai käytöstä poistamisen ilman yhteydenottoa palveluntarjoajaan. Käyttäjä voi myös määrittää, mitä resursseja tarvitaan, sekä miten ja milloin niitä käytetään. Vaihtoehtona on myös on-demand palvelu, jossa resurssit ovat saatavilla tarvittaessa, mutta eivät aiheuta kuluja, jos niille ei ole tarvetta. (Salo 2014, 93.)

Päätelaiteriippumattomuudella tarkoitetaan sitä, että käyttö on mahdollista työasemilla ja läppäreillä, mutta myös mobiililaitteilla. Silloin optimaalisesti resurssien hyödyntäminen onnistuu laitteesta riippumatta verkon avulla, sillä palvelut mukautuvat päätelaitteessa. (Salo 2014, 93.)

Resurssien yhteiskäytössä asiakkaalla ei ole tietoa palveluiden toteutustavoista tai sen sijainnista. Usea asiakas käyttää samaa ohjelmisto- tai laitekapasiteettia, näin kasvattaen palveluntarjoajan resurssikäyttöastetta. Sillä voidaan tehostaa ylläpitoa ja edullistaa hintoja. Haasteita tämä tuo asiakkaiden eristämisessä toisistaan ja vahingollisen toiminnan rajaamiseen siten, että yksi käyttäjä ei voi häiritä muita käyttäjiä. (Salo 2014, 94.)

Nopealla joustavuudella tarkoitetaan palveluiden skaalautumista, niin ylös- kuin alaspäinkin. Asiakkaan näkökulmasta kapasiteettia riittää lähes rajattomasti, mikä nopeuttaa sovellusten käyttöönottoa ja kehitystä. Muiden resurssien, kuten laskenta-, tallennus- ja tietoliikennekapasiteetin lisäys on lähes välitöntä. (Salo 2014, 94.)

Pilvipalvelu on mahdollista toteuttaa myös yrityksen sisällä. Silloin kyseessä on yksityinen pilvi, eli private cloud. Pilvipalveluita kuvataan kahdella tavalla:

1. pilvi tietotekniikka-arkkitehtuurina – itsepalvelullinen ja tehokas resurssien yhteiskäyttö, jonka käyttöä mitataan, samalla kasvattaen skaalautuvuutta ja joustavuutta tehokkuuden lisäksi.
2. pilvi palveluarkkitehtuurina – hyödynnetään palveluntarjoajan resursseja, jossa laskutus on käyttöön perustuvaa.

NIST on esittänyt yhteensä neljä vaihtoehtoa pilvipalveluille: yksityinen, yhteisöllinen, julkinen ja hybridipilvi. (Salo 2014, 94.)

Yksityinen pilvi on organisaation omistuksessa ja vain sen käytössä. Hallinto voi olla kolmannen osapuolen vastuulla ja laitteisto voi sijaita silti muissa kuin yrityksen tiloissa. Yhteisöllinen pilvi eroaa yksityisestä siten, että se on usean organisaation käytössä ja omistuksessa. (Salo 2014, 95.)

Julkisen pilven palvelut ovat asiakkaiden saatavilla maksua vastaan. Toimitus, hallinnointi, laitteisto, ohjelmisto ja palvelut ovat palveluntarjoajan vastuulla. Hybridipilvi on kaikkien edellä mainittujen yhdistelmä, jossa osa pilvestä on julkista ja osa yksityistä. (Salo 2014, 95.)

3.2 Pilvipalvelukategoriat

3.2.1 IaaS

IaaS, eli Infrastructure as a Service tarkoittaa infrastruktuuria palveluna. Sillä haetaan ratkaisua perinteisiin itsehallittujen palvelinkeskusten ongelmiin. Omistetun infrastruktuurin käyttöasteet eivät yleisesti yllä 80%:n yläpuolelle ja infrastruktuurin ylläpitoon kuluu IT-investoinneista jopa 60-80%. Palveluntarjoajat tarjoavatkin asiakkaan käyttöön omia resurssejaan palveluna, eli IaaS:nä. Alustan on oltava joustava, skaalautuva, itsepalveluna käytettävä ja ilman minimiostovelvoitetta, jotta sitä voidaan kutsua aidoksi pilvipalveluksi. Omistamisen ja pitkäkestoisen sitoutumisen sijaan otetaan resursseja käyttöön joustavasti tarpeen mukaan. Aidoissa pilvipalveluissa sopimukset voidaan mitata jopa minuuteissa. Lisäksi osto tapahtuu verkossa luottokortilla, eikä tarvita ihmiskontaktia. (Salo 2014, 97.)

Tavanomaiseen ulkoistamiseen nähden erona on joustavuus, resurssien yhteiskäyttö, itsepalvelullisuus, automaatio ja käyttöön perustuva laskutus. Resurssit ovat yleensä

virtualisoituja ja skaalautuvuus sekä ylläpito on automatisoitu suurilta osin. Palvelun käyttöä monitoroidaan tarkasti, jolloin laskutus perustuu tarkasti käytön mukaan, eikä etukäteissopimuksiin. Parhaassa tapauksessa asiakas voi ottaa palvelut käyttöön ja jatkaa käyttöä ilman mitään vuorovaikutusta palveluntarjoajaan. (Salo 2014, 97.)

IaaS tarjoaa käyttäjilleen suurimmat vapaudet ja kontrollin, suurimpana rajoitteena on se, ettei fyysisiä laitteita pääse näkemään. Palveluntarjoaja vastaakin resurssien toimivuudesta ja turvallisuudesta, samalla pitäen huolen riippumattomuuksista asiakkaiden kesken yhteiskäytetyillä alustoilla. Asiakkaan vastuulle jää omien ratkaisuiden tai sovellusten toimivuus, kuten päivitykset ja tietoturva sekä kuormantasaus. (Salo 2014, 97.)

3.2.2 PaaS

PaaS, eli Platform as a Service on sovellusalusta palveluna tarjoaa alustan ja rajapinnan sovellusten kehitykselle, testaukselle ja ylläpidolle. Infrastruktuurista ei tarvitse huolehtia ja toiminnallisuudet ovat usein saatavilla moduuleina tai rajapintoina, jolloin kehitystyö on mahdollisimman suoraviivaista, kustannustehokasta ja skaalautuvaa. Huolenaiheita tässä ratkaisussa on pelko yhteen palveluntarjoajaan lukittumisessa, tietoturva ja kehitystyön osaamis- ja ylläpitövaatimukset. Näistä aiheellisin on lukittuminen, joka voi aiheutua standardoimattoman PaaS –ratkaisun käytöstä. Datan siirto uudelle palveluntarjoajalle ei ole ongelma, mutta alustakohtaiset riippuvuudet, kuten lisäarvopalvelut ja rajapinnat voivat olla ongelmallisia. (Salo 2014, 98.)

Suurena etuna tässä ratkaisussa on sovelluskehitys kustannustehokkaassa, nopeassa, tietoturvallisessa ja kapasiteettirajoittamattomassa ympäristössä. Kustannustehokkuus mahdollistaan uusien yritysten saapumisen markkinoille pienemmälläkin panostuksella. Lisäksi kehittäjät voivat keskittyä pelkästään koodamiseen, kun palveluntarjoaja on vastuussa toiminnasta, ylläpidosta, skaalautuvuus ja päivittäminen. (Salo 2014, 98.)

3.2.3 SaaS

SaaS, eli Software as a Service tarkoittaa sovelluksia palveluna, eli yritys voi ostaa valmiiksi asennetut sovellukset käyttöönsä tarvittaessa. Päivitykset ja ylläpidon

hoitaa palveluntarjoaja. Lisenssimaksujen sijaan yritys maksaa joko kone- tai aikaperusteista maksua. Tässä ratkaisussa ohjelmisto- ja laitekustannuksia, sekä vapauttaa henkilöstöresursseja ylläpidosta. Palveluntarjoajan käyttöaste saadan korkeaksi, kun usea asiakas yhteiskäyttää samoja resursseja, kuitenkin jokaiselle oman käyttäjäkokemuksen ja ylläpidon tarjoten. (Salo 2014, 98.)

3.2.4 DaaS

DaaS, eli Data as a Service tarkoittaa dataa palveluna, joka on samankaltainen kuin SaaS. DaaS on itsenäinen palvelu muuhun alustaan nähden, joka mahdollistaa pilveen yhdistämisen ja datan käsittelyn. Usea datapalvelu on spesifioitu tiettyä käyttöä varten, kuten Googlen tarjoama palvelu, joka voi prosessoida kyselyn viiden teratavun datasta 15:ssä sekunnissa. Normaalissa datakeskusympäristössä aikaa kuluu kymmenkertainen määrä tälläisessä kyselyssä. (Hurwitz ym. 2013, Chapter 6)

3.3 Pilvipalvelun edut

Vaikka usein kuuleekin puhuttavan kustannussäästöistä pilvipalveluiden yhteydessä, on pilvipalveluiden merkittävin elementti uudet prosessit. Kustannuksia tärkeämpää onkin pilvipalvelun tuottama kilpailukyky ja sen mahdollistamat toimintatavat, joita aiemmin ei ole pidetty mahdollisina. (Salo 2014, 102.)

Lisäksi pilvipalvelut ovat skaalautuvia, joustavia ja päätelaiteriippumattomia, sekä kustannuksiltaan läpinäkyviä. Lisäarvopalveluilla voidaan saada myös merkittävää hyötyä. (Salo 2014, 103.)

3.4 Pilvipalvelun riskit

Tyypillisiä riskialueita pilvipalveluissa voidaan kuvata seuraavasti:

1. datahuolet: pysyvyys, tietosuoja, yksityisyys
2. käyttäjähallinta: turvallinen yhteys, tilitietojen suoja, tilin kaappausriski
3. suorituskyky: saavutettavuus, luotettavuus, suorituskykyheittely, ennakoitavuus
4. hallinta: läpinäkyvyys, mitattavuus, kontrolli

5. sopimusehdot: SLA, poikkeustilanteet, muutokset, vastuu, lukittautuminen
6. tekninen toteutus: tuen pituus, muutokset, dokumentointi
7. palveluntarjoaja: henkilöstö, tilat, läpinäkyvyys, tiedotus, palautuminen vikatilasta
8. säännöt: lainsäädäntö, standardit, toimialan vaatimukset, asiakkaan vaatimukset

Ylläolevasta listasta voidaan havaita, miksi pilvipalveluiden käyttöönotto on hidastellut joissain yrityksissä. (Salo 2014, 104.)

Yrityksen data ei itsessään ole arvokasta, ennen kuin sitä jalostetaan ja käyttöönotetaan. Idea on sama kuin big datassa. Pilvipalveluissa on aina kysymys datan liikuttelusta, tallennuksesta tai käsittelyssä jossain muodossa, joka kasvattaa tietomurron riskejä. Myös tiedustelupalvelut saattavat olla kiinnostuneita suurista datankeruumahdollisuuksista, kuten USA:n 2013 kohussa. Pilvipalveluita käyttävät asiakkaat haluavatkin, ettei tiedot ole ulkopuoliselle taholle saatavilla, tai palveluntarjoajan itsensä luettavissa. Myös lainsäädäntö vaikuttaa datan säilytykseen. (Salo 2014, 104.)

Palveluntarjoajien harmiksi on ollut tapauksia, joissa on vahingossa, tai tarkoituksellisesti vaarannettu asiakkaiden dataa. Kaiken lisäksi uutisointi on ollut näkyvää vahinkojen tai rikosten sattuessa. Perinteisessä palvelukeskuksissa virheiden sattuessa ei ulospäin tietoa leviä, joka on haitallista pilvipalvelun näkökulmasta. Toisaalta se kannustaa pilvipalveluntarjoajia entistä säntillisempään toimintaan. (Salo 2014, 105.)

Yrityksen kannalta onkin tärkeä luokitella data sen perusteella, onko sen tallentaminen pilveen riskien arvoista. Suurin osa datasta on varmasti tallennettavissa pilveen ilman suurempia tietoturvahuolia. (Salo 2014, 105.)

Pysyvyyden suhteen ei kannata olla kovin huolissaan, sillä dataa on hävinnyt hetkellisesti tai käyttökatkoja ilmennyt, mutta lähes aina data on saatu palautettua. Varmuuden vuoksi voisi datan hajauttaa useammalle palveluntarjoajalle, jolloin riskit ovat lähellä nollaa. (Salo 2014, 105.)

Tietojärjestelmien heikoin lenkki on ihminen, myös pilvipalvelun kohdalla. Huolimattomuus ja tietämättömyys on ongelmallista, tahallisesta pahanteosta puhumattakaan. Näissä tapauksissa voi olla vaikea valmistautua tilanteeseen etukäteen. Hakkerien metodeina onkin käytetty ihmisiä hyväksi, kuten sähköpostin liitetiedostoilla, joita ihmiset avaavat miettimättä enempää. Toinen ongelma on salasanojen heikkous, joillakin ne on jopa paperille kirjoitettuna näkyvissä. (Salo 2014, 108.)

Pilvipalvelut ovat loputtoman laskentatehonsa ja tallennuskapasiteetin ansiosta houkuttelevia vaihtoehtoja, mutta toimiakseen ne vaativat luotettavan verkkoyhteyden ja tasaisen suorituskyvyn. Palveluntarjoajan onkin syytä tarjota laadukas palvelu, vaikka ympäristömuuttujia esiintyisikin. Sopimuksissa mainitaan nykyään lähes aina palvelutasosopimuksessa, ei SLA:ssa tasot, jotka eivät saa alittua. Riskittömyyteen ja virheettömyyteen ei koskaan päästä, mutta se ei saisi olla syy palvelun käyttämättömyydelle. (Salo 2014, 109.)

Ongelmallisimpia tilanteita ovat suorituskykyongelmat, joita on vaikea mitata tai paikallistaa. Ongelma voi olla käyttäjän laitteessa, tietoliikenneyhteyksissä, pilvipalvelun järjestelmissä, tai pilvipalvelussa itsessään. Jos palvelu ei kokonaan ole poikki, ongelman paikallistaminen voi olla mahdotonta. (Salo 2014, 109.)

Luottamuksen kannalta merkittävä seikka on hallinnan puute, kun pääsyä fyysisesti laitteille ei ole, eikä sijainti välttämättä tiedossa. Palvelun käyttäjällä on käytössään laitteet ja rajapinnat, jotka palveluntarjoaja antaa tai tukee. Luottamusta parantaakseen palveluntarjoajat käyttävät sertifikaatteja, laatustandardeja, viestintää, maineenhallintaa ja sopimuksia, joilla pyritään parantamaan julkista kuvaa. Sopimukset ovat jatkuvasti kehittyneet ja niissä on määritelty yhä tarkemmin palvelutaso- ja sisältölupaukset. Käyttöehtojen lukemattomuudesta voi kuitenkin aiheutua ongelmia. Liiketoiminnan kannalta keskeisimpien toimintojen osalta sopimukset on syytä tehdä ja ymmärtää täsmällisesti. (Salo 2014, 110.)

Vaikka pilvipalvelun tarjoajat ovatkin avoimia tarjoamistaan palveluista ja rajapinnoista, ei aivan kaikkea haluta kertoa tietoturvan ja kilpailun takia. Pienikin epätieto voi kuitenkin haitata luottamusta. Suorituskyvyn ongelmista on hyvitettävä

sopimusten mukaisesti, mutta niiden ennakointi on vaikeaa, jos asiakas ei tiedä palveluntarjoajan toimenpiteistä. (Salo 2014, 111.)

Julkisen pilvipalvelun kohdalla tulee kysymykseen lainsäädäntö, toimialakohtaiset suositukset ja toimintatavat. Esimerkiksi henkilötietojen säilytystä säädellään tarkasti ja moni yritys säilyttääkin niitä omilla palvelimillaan. (Salo 2014, 111.)

3.5 Pilvipalvelut ja big data

Big data ja pilvipalvelut ovat käyttäytyneet trendeinä samalla tavalla ja kummankin määrittely ei ole täsmällistä. Big data käsitteen alle on luokiteltu monenlaisia tekniikoita, tuotteita ja palveluita, joissa yhteinen tekijä on data. Osa on vanhoja tallennus- ja analytiikkaratkaisuita uudelleennimettyinä, kun toinen osa on täysin uutta. Moni palveluntarjoaja on siirtynyt big dataan tai uudelleenkategorioinut palveluitaan. Aiemmin myyty pilvipalvelu on muuttunut big dataksi pilvipalveluna, eli analytiikkaa tai tallennusratkaisua pilvessä. Tallennusratkaisusta esimerkkinä voidaan mainita Amazon Web Services S3 ja Google Cloud Storage. Analytiikasta esimerkkinä voidaan mainita Microsoftin Azure HDInsight ja Googlen BigQuery. (Salo 2014, 162.)

Pilvipalveluratkaisun etuna on ettei laite- tai ohjelmistoinvestointeja tule, sopimukset ovat käyttöön perustuvia ja kapasiteettia voi tarpeen mukaan muuttaa. Hinnat ovat erittäin kilpailukykyisiä perinteisiin omistamiseen ja ostamiseen nähden. Hyödyt ja riskit ovat yleisesti samat kuin pilvipalveluissa yleensäkin. (Salo 2014, 162.)

Analytiikkapuolen joustavuus mahdollistaa nopean analysoinnin ilman laite- tai ohjelmistoinvestointeja. Hyvänä esimerkkinä on New York Timesin konversiot artikkeliarkistosta nettiystävälliseen muotoon, jonka budjetti oli kymmenissä tuhansissa dollareissa. Konversio suoritettiin Amazon Web Services Elastic MapReducen Hadoop-klustereilla, jolloin operaatioon kului alle 24 tuntia ja hintakin oli alle tuhat dollaria. (Salo 2014, 162.)

Big datalla menestyminen vaatii suuren datamäärän keräämisen, datan yhdistämisen muiden organisaatioiden datan kanssa ja kokonaisuutta analysoimalla hyödyllisen lisäarvon tuottamisen. Yritysten tulee laskuttaa tästä lisäarvosta ja julkisella sektorilla pitää pystyä osoittamaan tuotettu lisäarvo rahoittajille. (Salo 2014, 162.)

Datan tuotannossa, tallennuksessa, jalostamisessa ja hallinnoinnissa on paljon uutta potentiaalista liiketoimintaa. Dataketjua voidaan kuvata seuraavasti: **data – informaatio – tieto – tietämys**. Näistä jokaiselle osa-alueelle löytyy edustusta yrityksissä. Suurin osa on kuitenkin keskittynyt tiedon ja tietämyksen akselille. Pelkän datan ympärille rakennettua yritystä on vähemmistö. Todennäköisesti datan tarjonnan kasvaessa markkinat avautuvat uusille mahdollisuuksille ja yrityksiä tulee lisää tälle alueelle. Myös ennustavaan, optimoivaan ja prosessiautomaation perustuvien toimijoiden tarve kasvaa. Kun dataan vaikuttavat tekijät tiedetään etukäteen, saadaan päätöksentekoon etua analytiikan avulla. Tästä saadaan oiva kilpailukyvyn mittari: kuka osaa hyödyntää datan tehokkaasti, monipuolisesti ja kauaskantoisesti. (Salo 2014, 163.)

4 ARKKITEHTUURI

4.1 Vaatimukset

Ennen arkkitehtuurin pystytystä on tärkeää huomioida, mihin käyttötarkoitukseen big dataa käytetään. Käyttötarkoituksia voi olla tallentaminen, analysointi, raportointi tai ohjelmistot. Data tulee ensin kerätä, organisoida ja integroida. Näiden tehtävien jälkeen voidaan siirtyä datan analysointiin, joka tehdään käyttötarkoituksen mukaisesti. Analysointitulosten perusteella big dataa hallitaan halutulla tavalla. Esimerkkinä tälläisestä toiminnasta voidaan käyttää verkkokauppoja, jotka analysoivat asiakkaan ostoksia ja ehdottavat sopivia tuotteita tai alennuksia tämän ostokäyttäytymisen perusteella. Datan vaiheet voidaan jakaa sykliin: **Kerää – organisoi – integroi – analysoi – reagoi**. (Hurwitz ym. 2013, Chapter 4)

Jos dataa yhdistetään useista datalähteistä, on oltava tarkkana niiden luotettavuudesta ja hyödyllisyydestä. Lisäksi tulee huomioida oman datan tietoturva jakamalla tärkeä tieto ja julkaisukelpoinen erikseen. (Hurwitz ym. 2013, Chapter 4)

Suorituskykyä miettiessä on jälleen tärkeää tietää minkälaista tietoa analysoidaan, jolloin tiedetään laitteiston vaatimukset. Osa datasta on analysoitava reaaliaikaisesti

ja osa taas on tallennettava pysyvästi. Redundattisuutta tarvitaan puolestaan viiveiden minimoimiseksi ja saavutettavuuden takaamiseksi. Näitä suorituskyvyn vaatimuksia voidaan tarkastella seuraavilla kysymyksillä:

1. datan määrä nyt ja tulevaisuudessa
2. reaaliaikaisen tai lähes reaaliaikaisen datankäsittelyn tarve
3. riskit, alakohtaiset turvallisuuskysymykset, maan säännökset
4. nopeuden tärkeys
5. datan tarkkuuden määritys.

(Hurwitz ym. 2013, Chapter 4)

4.2 Arkkitehtuurin komponentit

4.2.1 Fyysisen infrastruktuurin redundanttisuus

Big data infrastruktuuri voidaan jakaa eri tasoihin, eli big data pinoon, kuten kuviosta 2 havaitaan. Alimmalla tasolla on fyysinen infrastruktuuri, kuten palvelimet ja verkkolaitteet. Big dataa pystyttäessä tulee ottaa jokainen taso huomioon, jotta varmistutaan toimintakyvystä. Tärkeitä asioita ovat ainakin suorituskyky, saavutettavuus, skaalautuvuus, joustavuus ja kustannukset. Koska big data on nopeaa vauhdiltaan, volyymiltaan ja vaihtelevaa, on fyysinen infrastruktuuri kaiken perusta. Saatavuus pitää olla jatkuvaa ja palvelimilta vaaditaan joustoa ja redundanttisuutta, kuten myös verkkolaitteilta. Palvelimilla pitää olla kapasiteettia mukautua datan muutoksiin ja tehontarpeeseen. Kaikki voi kuitenkin kaatua vialliseen laitteistoon, eli tarvitaan redundanttisia laitteita saavutettavuuden takaamiseksi. Silloin voidaan poistaa yhden viallisen laitteen vaikutus koko järjestelmään. Samoin on tärkeää olla useampi verkkoyhteys ulkomaailmaan, ettei yhden yhdeyden katkeaminen aiheuta ongelmia. Näitä ongelmia pohtiessa ymmärtää helposti pilvipalveluiden merkityksen, koska palveluntarjoajat vastaavat laitteiston toimivuudesta. (Hurwitz ym. 2013, Chapter 4)

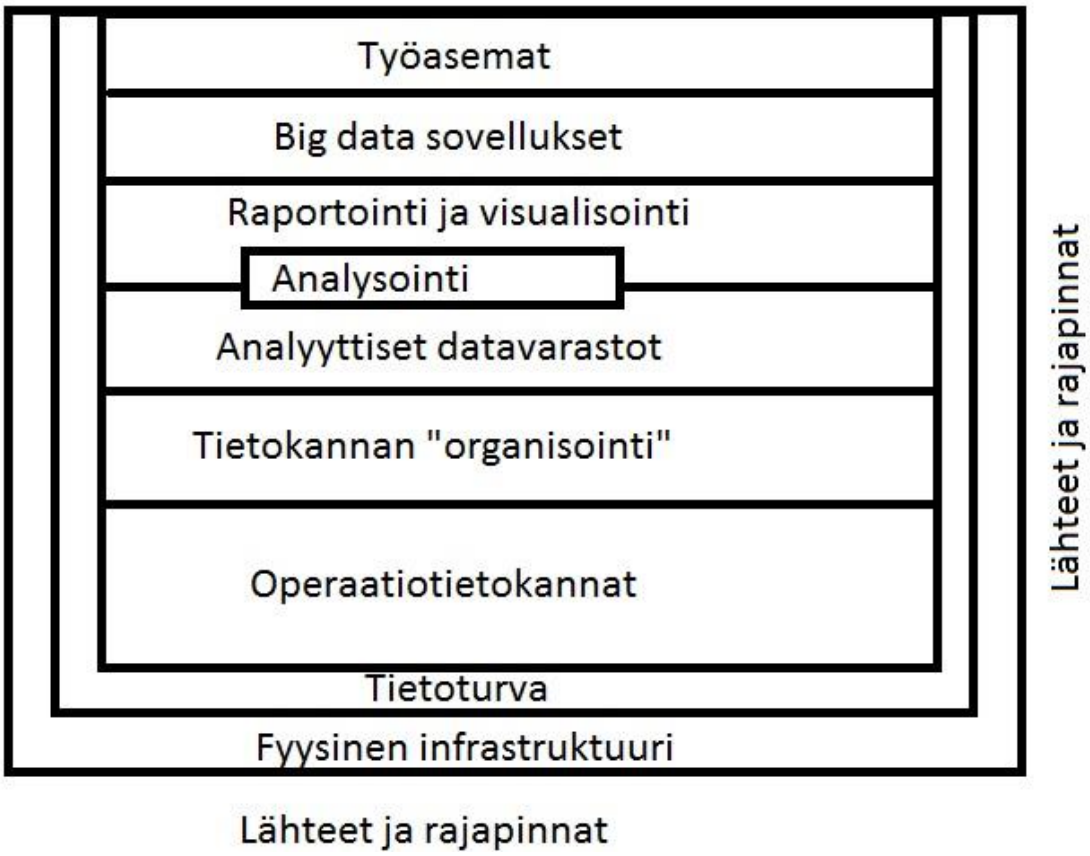
4.2.2 Turvainfrastruktuuri

Seuraavalla tasolla on tietoturva, jossa vaatimukset ovat samanlaiset kuin tavanomaisissa datakeskuksissa. Tärkeimpiä huomioonotettavia asioita ovat käyttäjän pääsy dataan, sovellusten pääsy dataan, datan salaus ja uhkien tunnistus.

Raakadataan tai analysoituun dataan saa päästä vain määrätyt henkilöt, joilla on yritystoiminnankannalta tarve tutkia sitä. Sovellusten pääsy dataan on teknisesti yksinkertaista, koska API:t suojaavat väärältä käytöltä. Datan salaus aiheuttaa runsaasti kuormaa, joten vain yritystoiminnan kannalta kriittinen tieto kannattaa salata erityisen hyvin ja se tulee pitää yrityksen omassa tietojärjestelmässä, eikä julkisessa pilvessä. (Hurwitz ym. 2013, Chapter 4)

4.2.3 Lähteet ja rajapinnat

Rajapinnat tarjoavat molempiin suuntiin pääsyn datapinossa. Rajapintoja suunnitellessa on tärkeää ottaa huomioon yhtenäinen struktuuri ja jaettavuus ulkopuolisille, kuten yrityskumppaneille. Big data tuo omat haasteensa API:en käyttöönottoon, sillä data voi olla strukturoimatonta ja ulkopuolisen lähteen tuottamaa. Uuden NLP eli Natural Language Processing –tekniikan avulla voidaan luoda kyselyitä luonnollisella kielellä, eikä SQL-tyyppisesti muodollisella kielellä. Siksi NLP on noussut suosituksi vaihtoehdoksi big datan ja sovellusten väliin. (Hurwitz ym. 2013, Chapter 4)



Kuvio 2. Big data pino ja tasot

4.2.4 Toiminnalliset tietokannat

Big datan toiminnan kannalta tietokantajärjestelmät ovat elintärkeitä, joten niiltä vaaditaan nopeutta, skaalautuvuutta ja vakautta. Joissain tilanteissa tarvitaan useampia tietokantapalvelimia, kuten tilanteissa, joissa osa datasta on relaatiotietokannassa ja osa ei. Riippuu big datan käyttötarkoituksesta mitä tietokantamallia käytetään ja millä kielellä kyselyt suoritetaan. Esimerkkinä voidaan mainita relaatiotietokannasta PostgreSQL ja Oracle, kolumnisista tietokannoista Hbase ja dokumenttipohjainen MongoDB. (Hurwitz ym. 2013, Chapter 4)

4.2.5 Datapalveluiden ja työkalujen organisointi

Tässä pinon tasolla kerätään, validoidaan ja luodaan useista big datan elementeistä sisällöltään järkeviä kokoelmia. Big datan valtavuuden takia on kehitetty tekniikoita,

jossa tiedon käsittely tapahtuu tehokkaasti. Hyvin suosittu vaihtoehto on MapReduce, joita suurin osa datan organisointipalveluista käyttää.

Datan organisointipalvelut ovat työkaluja ja teknologioita, joilla kerätään ja valmistellaan tietoa jatkoanalysointiin. Siksi työkaluilta vaaditaan integroituvuutta, tulkkausta, normalisointia ja skaalautuvuutta. Tällä tasolla teknologioihin kuuluvat:

- Hajautettu tiedostojärjestelmä, joka käsittelee datavirrat ja tarjoaa skaalautuvuutta ja tallennustilaa
- sarjoituspalvelut pysyvää datan tallennusta ja etäproseduurikutsuja varten
- koordinointipalvelut hajautettuihin sovelluksiin
- datan purun, muokkaamisen ja lataamisen työkalut eri datamuotojen siirtämiseksi Hadoopiin
- työnkulun palvelut ajastukseen ja synkronointiin eri tasoilla

(Hurwitz ym. 2013, Chapter 4)

4.2.6 Analyyttiset tietovarastot

Datavarastoissa on paljon normalisoitua tietoa eri lähteistä, jotka on koottu yrityksen analysointikäyttöön. Tieto voi olla tallennettu mihin tahansa tallennusarkkitehtuurin muotoon. Big datan kohdalla tarvitaan useita eri tietokantoja, joissa osassa käsitellään reaaliaikaista dataa ja osaan se tallennetaan jatkokäyttöä varten. Koska data on peräisin useasta eri lähteestä, big data tuottaa seuraavia eroja normaaliin datankäsittelyyn nähden: paljon erimuotoista dataa, jolloin vaaditaan sen muokkausta ennen käyttämistä. (Hurwitz ym. 2013, Chapter 4)

Perinteiset datavarastot tarjoavat yritysten johdolle strategista apua, mutta big datan myötä myös muut jäsenet yrityksessä hyötyvät niistä. Esimerkiksi asiakaspalvelija voi nähdä reaaliajassa tapahtumia tai myyntipuolella voidaan reagoida dataan. (Hurwitz ym. 2013, Chapter 4)

4.2.7 Analytiikka

Big data voidaan käsitellä nykyisillä tekniikoilla ja työkaluilla, mutta toimiakseen erilaisten datamuotojen kanssa reaaliajassa se vaatii infrastruktuurin, josta aiemmin mainittiin. Analysointityökalujen tulee myös toimia useissa toteutuksissa. Big datan myötä on tullut kolme uutta luokkaa, jotka helpottavat datan ymmärtämistä: raportointi ja ”kojelaudat”, visualisointi ja edistynyt analytiikka. (Hurwitz ym. 2013, Chapter 4)

Raportointi ja kojelaudat tuottavat informaation lukijaystävälliseen muotoon, ja samalla mahdollistaa uusien tietokantojen analysoinnin, kuten NoSQL:n.

Visualisointi on tärkeää, koska data on yleensä interaktiivista ja dynaamista. Animaatio on hyvä lisä ja mahdollista datan käsittelyn uudella tavalla, kuten mind mappien muodossa. (Hurwitz ym. 2013, Chapter 4)

Edistynyt analytiikka prosessoi tietovarastojen tietoa ihmiskäyttöön. Edistyneen tekniikan avulla voidaan saavuttaa ennennäkemättömiä trendejä yrityksiin. (Hurwitz ym. 2013, Chapter 4)

4.2.8 Sovellukset

Big datan tutkimiseen ja jakamiseen on tarjolla kolmannen osapuolen sovelluksia ja tällä tasolla onkin suurimmat innovaation ja luovuuden mahdollisuudet. Horisontaaliset sovellukset ovat tarkoitettu usealla alalle, kun vertikaaliset sovellukset ovat alakohtaisiin ongelmanratkaisuun. Tärkeimpiä kategorioita tällä hetkellä ovat logisovellukset, mainossovellukset ja markkinointisovellukset. Kehityksessä ovat myös lääketieteen, teollisuuden ja liikenteen alojen sovelluksia.

Sovelluksilta vaaditaan koko pinon alueella toimivuutta, mutta myös struktuuria, standardeita ja määritellyt API:t. (Hurwitz ym. 2013, Chapter 4)

4.3 Hadoop

4.3.1 Perustietoa

Hadoop on avoimen lähdekoodin ohjelmistoprojekti, joka pohjautuu Googlen tiedostojärjestelmään ja hajautettuun datan analysointiin. Näiden pohjalta syntyi Hadoop Distributed File System, eli HDFS ja analytiikkaosuuteen käytetään MapReducea. (Salo 2014, 72.)

Hadoop on keskeinen tekniikka big datassa ja sen ympärille on kehittynyt paljon liiketoimintaa, joista huomattavimpia ovat Cloudera, Pivotal, HortonWorks ja MapR. Hadoop-jakeluita käyttää myös suuryritykset, kuten Microsoft ja Intel. (Salo 2014, 72.)

Hadoop luo palvelinklustereita ja yhdistää ne loogiseksi kokonaisuudeksi. Palvelimet voivat olla fyysisiä tai virtuaalisia. Klusterit ovat edullisia, skaalautuvia ja toimivuudeltaan varmoja. Lisenssimaksuja ei tarvitse maksaa, koska Hadoop ja käytettävät käyttöjärjestelmät ovat avoimeen lähdekoodiin perustuvia. Toimintavarmuus perustuu hajautukseen, jonka myötä vikasietoisuus paranee. Yksikään yksittäinen palvelinongelma ei aiheuta koko järjestelmän toimimattomuutta. Hadoop ei ole pienen datamäärän ratkaisu, joten skaalautuvuus on tärkeää ja koneiden määrää voidaan helposti kasvattaa klusterissa. (Salo 2014, 73.)

4.3.2 Hadoop-ekosysteemi

On olemassa neljä tunnistettavaa tapaa ottaa Hadoop käyttöön. Projekti otetaan joko käyttöön sellaisenaan tai sisarprojekteineen, otetaan käyttöön Hadoop-jakelu, tai otetaan Hadoop käyttöön osana tuotteistettua kokonaisratkaisua. Näistä nykyisin harvemmin käytetään kahta ensimmäistä mallia ja epätodennäköistä on myös konfiguroinnin tekeminen itse alusta asti. Yleensä Hadoop otetaankin käyttöön jakeluna tai osana tuotteistettua kokonanisratkaisua. (Salo 2014, 82.)

Hadoopin tärkein etu on suuren datamäärän tallennus- ja analysointimahdollisuudet. Hadoopin edut kasvavat datamäärän kasvun myötä, vaikkei sekään pysty aivan

loputtomiin datamääriin. Jopa petatavun kokoisia massoja on olemassa. (Salo 2014, 83.)

Yhteen namenodeen rajoitetun käytön aiheuttama vikaantumispiste on poistettu kakkosversiosta alkaen, jolloin tämä riippuvuus ei enää aiheuta resursseille pullonkaulaa. Kakkosversion myötä käyttömahdollisuudet ovat lisääntynyt suurien datamäärien lisäksi iteratiivisen ja interaktiivisen datan analysointiin. (Salo 2014, 83.)

4.3.3 Hadoop-arkkitehtuuri

Hadoop ympäristö koostuu master, data ja worker –solmuista. Jokaisessa solmussa on useita ohjelmistokomponentteja, jotka ovat nimenomaisen solmun kannalta oleellisia. Useasta masterin käyttö on tärkeää, jolloin vältytään yhden pisteen vikaantumisen aiheuttamasta käyttökatkosta. Masterin tärkeimpiin elementteihin kuuluu:

- JobTracker jakaa tehtävät eri solmuille ja keskustelee asiakasohjelmistojen kanssa
- TaskTracker ottaa käskyjä vastaan JobTrackerista, kuten Map, Reduce ja Shuffle
- NameNode pitää kirjaa hakemistopuista ja kaikista tiedostoista HDFS:ssä. Asiakasohjelmistot ottavat yhteyden NameNodeen tietoja käsitelläkseen. (Schneider 2012, 27)

Datanoden tehtävä on tallentaa ja replikoida dataa klustereiden välillä, sekä kommunikoida asiakasohjelmistoille, kun ne ensiksi ovat kysyneet NameNodelta osoitetta datasolmuun. (Schneider 2012, 27)

Workersolmut ovat prosessoinnin ja analysoinnin moottoreita, joita on klustereissa satoja kappaleita. Jokaiseen workeriin kuuluu DataNode ja TaskTracker. (Schneider 2012, 27)

Hadoopissa on kolme perustasoa, joilla on looginen hierarkia. Kokonaisuus näistä kolmesta tasosta muodostaa täydellisen MapReduce –toteutuksen. (Schneider 2012, 27)

Ensimmäisenä tasona toimii ohjelmistotaso, joka mahdollistaa hajautetun laskennan ja keskustelun ohjelmistojen välillä. (Schneider 2012, 27)

Toinen taso on MapReduce työkuorman hallinta – taso, eli JobTracker on moottori, joka vastaa kaikkien aspektien koordinoinnista Hadoop –ympäristössä. Tämä taso on kaikista tärkein luotettavuuden ja suorituskyvyn kannalta. (Schneider 2012, 28)

Datatasolla on vastuu datan tallennuksesta, ja usein kyseessä on hajautettu tietojärjestelmä, kuten HDFS. Taso voi sisältää myös kolmannen osapuolen ratkaisuita. HDFS:n kehittäjät ovat suunnitelleet järjestelmän siten, että:

- tiedostot tallennetaan blokkeina, vakiona 128Mt
- replikointi parantaa luotettavuutta, kun jokainen blokki on vähintään kahdessa eri datasolmussa
- yksittäinen masterin namenode koordinoi pääsyn ja metadatan, mikä aiheuttaa keskitetyn ja yksinkertaisen hallinnan
- suurien datamäärien takia välimuistiin ei tallenneta (Schneider 2012, 29)

4.4 MapReduce

4.4.1 Perustietoa

MapReduce on nimensä mukaisesti kahden eri prosessointivaiheen yhdistelmä, jossa periaatteena on avain/arvo -parit. Map osa sisältää avaimen, joka kertoo minkälaisessa muodossa tieto on. Avaimen esimerkkejä ovat nimet, rahamäärät tai hakutermit. Arvo-osio sisältää itse datan, joka liittyy avaimeen. Avain/arvo – pareista saadaan edellisillä esimerkeillä:

1. nimi/Matti
2. rahamäärä/31
3. hakutermi/mikä on big dataa?

Map –vaihe tallentaa datalähteen tiedot map() funktioon avain/arvo pareina. Sen jälkeen funktio luo vähintään yhden väliaikaisen arvon ja sille sopivan ulostulon

avaimen. Yleensä suuri datamäärä pilkotaan useaksi yhden gigatavun tiedostoiksi, joista jokainen jaetaan eri solmulle. (Schneider 2012, 18)

Reduce osio yhdistää tietyn ulostuloavaimen arvot listaksi. Sen jälkeen *reduce()* funktio yhdistää väliaikaiset arvot yhteen tai useampaan lopulliseen arvoon samalle avaimelle. (Schneider 2012, 18)

MapReducen kanssa kannattaa ottaa huomioon asioita, kuten:

- komponenttien hajoaminen
- data tallennetaan pieniin määriin suuria tiedostoja
- datatiedostot ovat kertakirjoitteisia
- yhteen tiedostoon voi kerralla olla yhteydessä useampi säie
- pysyvä ja suorituskykyinen ympäristö

  (Schneider 2012, 18)

4.4.2 Datankäsittely

MapReducessa clientin lähettämän työkäskyn hoitaa *job* –niminen työyksikkö, joka sisältää syöttödatan, MapReduce ohjelman ja konfiguraatiotiedot. Hadoop puolestaan jakaa tehtävät kahteen taskiin, eli *mapiin* ja *reduceen*. Taskit aikataulutetaan ja suoritetaan klusterin eri nodeissa käyttämällä YARNia, joka on Hadoopin resurssinhallintajärjestelmä. Jos taskit epäonnistuvat, ne aikataulutetaan automaattisesti uudelleen toiselle nodelle. (White 2015, 52)

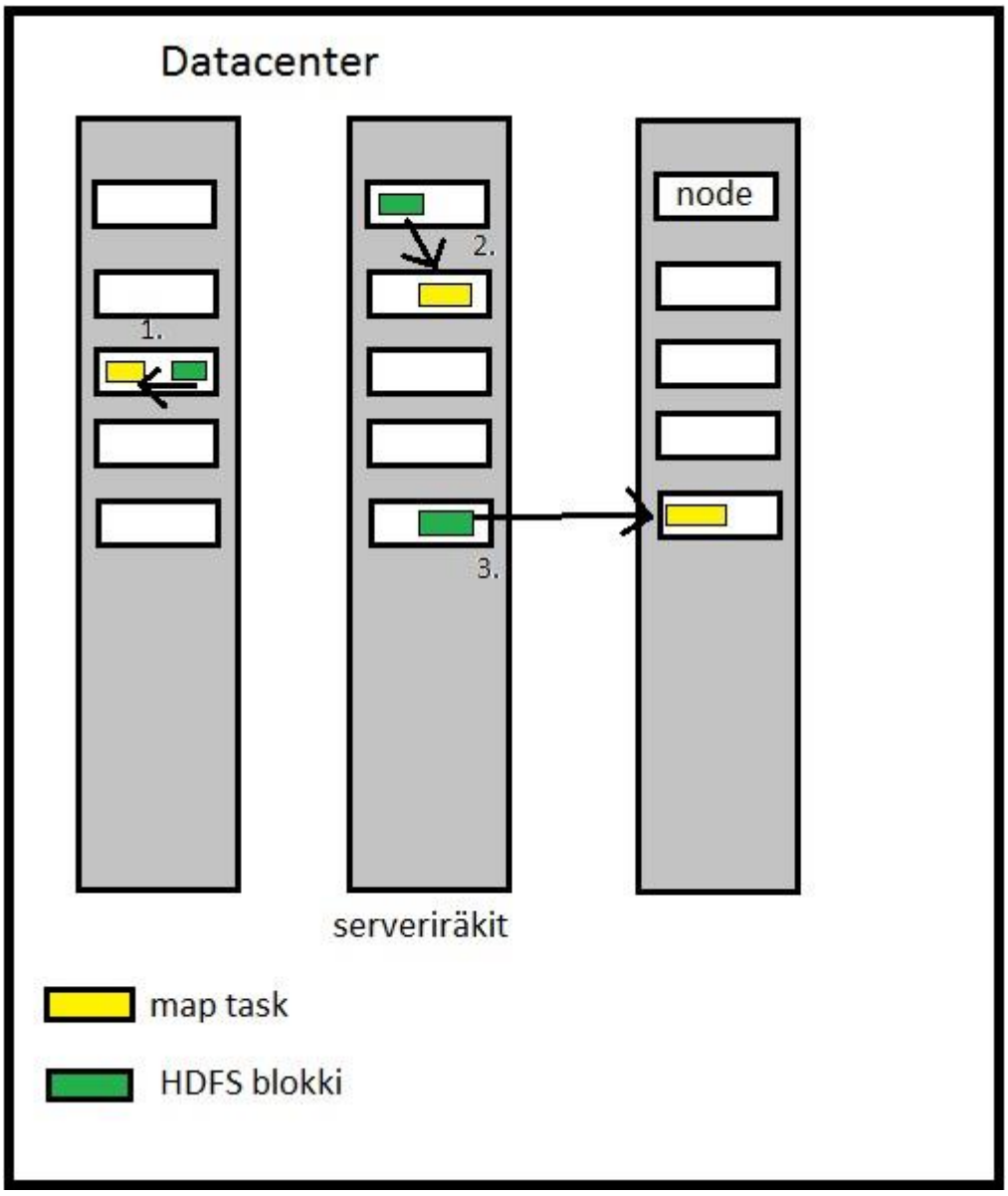
Hadoop jakaa MapReducen jobiin tulevan syöttödatan tietyn, ennalta määritellyn kokoisiksi osiksi, eli input spliteiksi. Jokaiselle splitille Hadoop luo oman map taskin, jossa suoritetaan map-funktiota. (White 2015, 52)

Usean pienikokoisen splitin käsittely on nopeaa ottaen huomioon koko syöttödatan koon. Samalla kuormantasaus paranee, jolloin nopeille koneille voidaan antaa enemmän tehtäviä kuin hitaammille. Toisaalta liian pieniä splittejä ei kannata käyttää, koska työn kokonaisajasta voi suuri osa kulua splittien hallintaan, eikä itse laskemiseen. Vakiona käytössä on HDFS:n blokin mukainen 128Mt, joka on hyvä koko, mutta sitä voidaan muuttaa klusterikohtaisesti. (White 2015, 52)

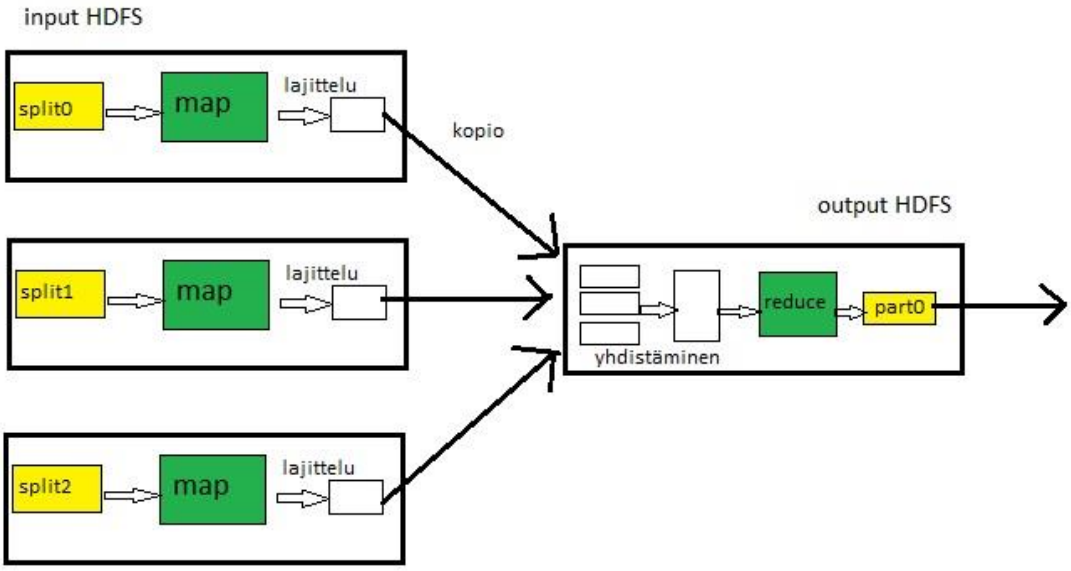
Hadoop pyrkii suorittamaan map taskit samassa nodessa, jossa syöttödata on HDFS:ssä klusterin kaistanleveyttä säästääkseen. Tätä kutsutaan datan paikallisuuden optimoinniksi. Joskus kuitenkin kaikki HDFS:n blokit ja niiden kopiot ovat suorittamassa muita tehtäviä, jolloin input spliteille etsitään vapaa map slotti samasta palveliniräkistä. Joskus joudutaan käyttämään räkin ulkopuolista nodea, jolloin verkkoliikennettä tulee mukaan. Kuviossa 3 näytetään vaihtoehtoja. (White 2015, 52)

Map taskien tulostetta ei tallenneta HDFS:ään, vaan paikalliselle levylle. Mapin tulosteen prosessoi välittömästi reducen taski, joiden yhdistelmästä syntyy lopullinen tuloste. Silloin map taskin tulosteet voidaan poistaa, jolloin HDFS:n replikoinnit sille olisivat turhaa resurssinkäyttöä. Edes epäonnistunut map task ei aiheuta ongelmaa, sillä Hadoop osaa automaattisesti siirtää suorituksen toiselle nodelle. (White 2015, 52)

Reduce taskit eivät hyödy paikallisesta datasta, koska niiden syötteenä toimii usean map taskin tulokset, jotka lopulta siirretään verkon yli nodelle, jossa reduce task toimii. Reducen tulosteet tallennetaan HDFS:ään ja replikoidaan luotettavuuden takaamiseksi. Replikoinnissa ensimmäinen HDFS -blokin kopio on paikallisella nodella ja muut kopiot räkin ulkopuolisessa paikassa. Kuviossa 4 nähdään datan virtaus MapReducessa. (White 2015, 53)

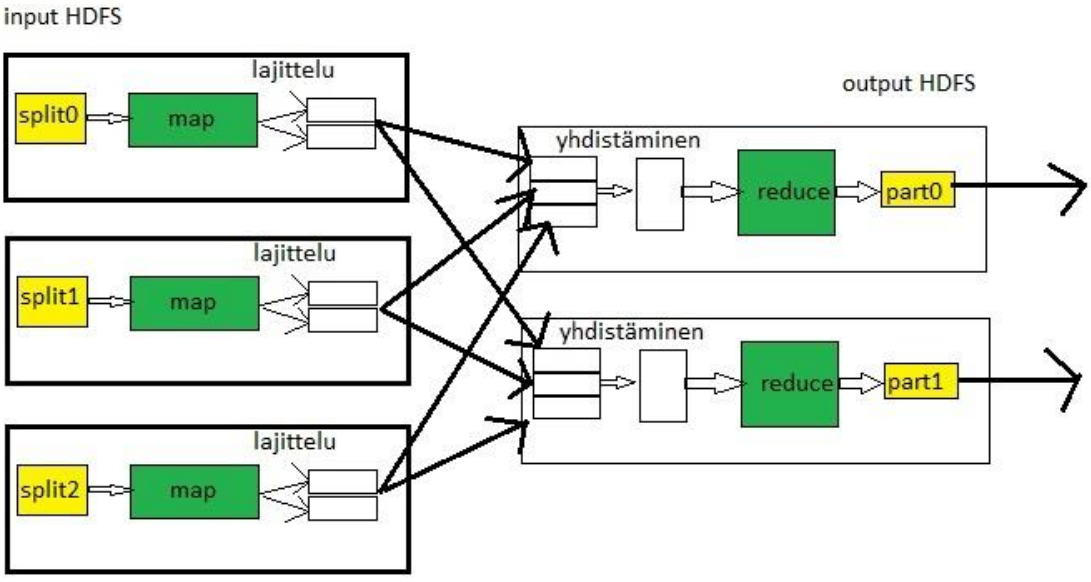


Kuvio 3. Datalokaali (1), räkin sisäinen (2) ja räkin ulkopuolinen (3) map task



Kuvio 4. Yksittäisen reduce taskin datavirtaus

Usean reducen tapauksessa jokainen map taski luo tulosteen erikseen jokaiselle reduce taskille. Kuvassa 4 nähdään usean map ja reduce taskin datavirtaus. Samalla nähdään sekoitus, jolla tarkoitetaan usean map taskin syöttödataa reducelle. Sekoitus on isossa mittakaavassa merkittävän suuri tekijä suoritusajan kannalta, joten se kannattaa säätää optimaaliseksi. Hadoopin mahdollistaa yhdistelmäfunktion suorittamisen mapin tulosteeseen, jossa yhdistelmäfunktio optimoi ja luo lopullisen syötteen reducelle. Tällä voidaan minimoida datan määrää, jolloin klusterissa jää enemmän kaistaa käyttöön. (White 2015, 55)



Kuvio 5. Usean reduce taskin datavirtaus

Mahdollista on myös se, ettei reduce taskeja tarvita ollenkaan. Silloin sekoittamista ei tarvita vaan map taskista saadaan suoraan tuloste HDFS:lle ja ainut noden ulkopuolinen dataliikenne aiheutuu HDFS:ään kirjoittaessa. (White 2015, 55)

4.5 HDFS

HDFS eli Hadoop Distributed File System on tehty suurien datamäärien tallennukseen, joka on jaettu klustereihin ja siinä on streamattu dataan pääsy. Streamatulla datanpääsyllä tarkoitetaan sitä, että data kirjoitetaan kertaalleen, mutta luetaan useasti. Kun analyyseja suoritetaan, syntyy lisää aineistoa, jota puolestaan seuraava analysoija tarvitsee. Niinpä onkin tärkeää taata koko aineiston luettavuus nopeasti, eikä vain alkuperäisen datan. HDFS ei vaadi kalliita laitteita toimiakseen, vaan klusterit parantavat suorituskykyä ja luotettavuutta. (White 2015, 64)

HDFS käyttää tiedostojärjestelmässä blokkeja, jotka ovat vakiona kooltaan 128Mt. Blokeista on hyötyä klusterissa, sillä tiedostoblokit voivat sijaita millä tahansa klusterin levyllä. Myös hallinta parantuu, sillä blokit ovat ennaltamäärätyn kokoisia, jolloin levytila on helppo arvioida. Myöskään metadata, kuten lukuoikeudet eivät sisälly blokkiin, vaan se käsitellään muualla. Blokit ovat myös hyviä replikoinnissa ja usein replikointi suoritetaan vähintään kolmelle fyysisesti erillään olevalle laitteelle. Jos yksi laite vikaantuu, kopioidaan blokki jälleen uudelle laittelle virhetilanteiden välttämiseksi. (White 2015, 66)

Klusterit toimivat master-worker –periaatteella, jossa namenode on master ja datanodet workereita. Master vastaa tiedostojärjestelmän nimiavaruudesta, sekä ylläpitää tiedostopuut ja niiden metadatan. Tiedot tallennetaan paikallisesti imagena ja muutoslokina. Namenode tietää datablokkien sijainnit, vaikkei sitä pysyvästi tallennakaan, vaan tieto generoituu datanoden käynnistyksen yhteydessä. Datanodet vastaavat blokkien tallentamisesta ja hakemisesta, joko namenoden tai clientin käskystä. Lisäksi ne raportoivat namenodelle listan blokeista, joita tallennetaan kyseisellä hetkellä. (White 2015, 67)

Ilman namenodea tiedostojärjestelmä on käyttökelvoton, sillä ainoastaan namenode osaa uudelleenluoda blokin tiedot, jotka se saa datanodelta. Onkin tärkeää pitää

huoli vikasietoisuudesta. Tämän voi toteuttaa joko kopioimalla tietojärjestelmän metadatan kyseisen tilan, tai käyttämällä toista namenodea. Toinen namenode ei käyttäydy ensimmäisen tavalla, vaan se vastaa nimiavaruuden imagen ja muutoslokien yhdistämisestä liian ison lokitiedoston välttämiseksi. Toinen namenode on yleensä erillisellä koneella, koska se vaatii paljon prosessori- ja muistikapasiteettia. Vaikka toisella namenodella on tallennettuna nimiavaruuden image, niin vikatilanteessa on datan menettäminen todennäköistä, sillä image ei ole täysin reaaliaikainen ensimmäisen namenoden kanssa. (White 2015, 67)

Datanodeen on tarvittaessa mahdollista tallentaa tiedosto välimuistiin, levyltä lukemisen sijaan. Työn aikatauluttajat, kuten MapReduce osaa hyödyntää välimuistiin tallennuksesta aiheutunutta suorituskyvyn parannusta ajamalla tehtävät datanodella, jossa blokki on varastoitu välimuistiin. (White 2015, 68)

Namenode pitää muistissa kaikki tiedostot ja blokit tiedostojärjestelmässä, jolloin muisti voi koitua skaalautuvuuden esteeksi suurissa klustereissa. HDFS federaatio ratkaisee ongelman 2. versiosta alkaen, sillä se voi lisätä klusteriin useampia namenodeja, joista jokainen saa osan tiedostojärjestelmästä itselleen. Federaatiossa jokainen namenode hallitsee metadatasta koostuvaa nimiavaruuden loogista levyä, sekä blokkipoolia, jossa on tieto tiedostojen kaikista blokeista. Blokkipoolin tiedot ei ole ositettuna, jolloin datanodet tallentavat tietoa useasta poolista ja osoittavat useaan namenodeen. Loogiset levykuvat toimivat kuitenkin itsenäisesti, eivätkä keskustele keskenään, eikä yhden levykuvan vikatilanne vaikuta muihin. (White 2015, 68)

Ennen 2. versiota ongelmana HDFS:ssä on namenoden vikatilanteen aiheuttama klusterin toimimattomuus, sillä toisen namenoden tallentaman palautuspisteen käyttöönotto ei tapahdu välittömästi. Vikatilanteessa ylläpitäjä joutuisi manuaalisesti konfiguroimaan toisen namenoden käyttöönoton, joka koostuu kolmesta osasta: Levykuvan lataamisesta, muutoslokin läpikäynnistä ja safe moden poistamiseksi sen täytyy saada datanodeilta tarpeeksi blokkiraportteja. Suurissa klustereissa operaatioon saattaa kulua kymmeniä minuutteja. (White 2015, 68)

Hadoopin 2. versiossa ongelma on korjattu HDFS HA:lla eli korkealla saavutettavuudella. Tässä versiossa namenodeilla on aktiivisia vara-namenodeja

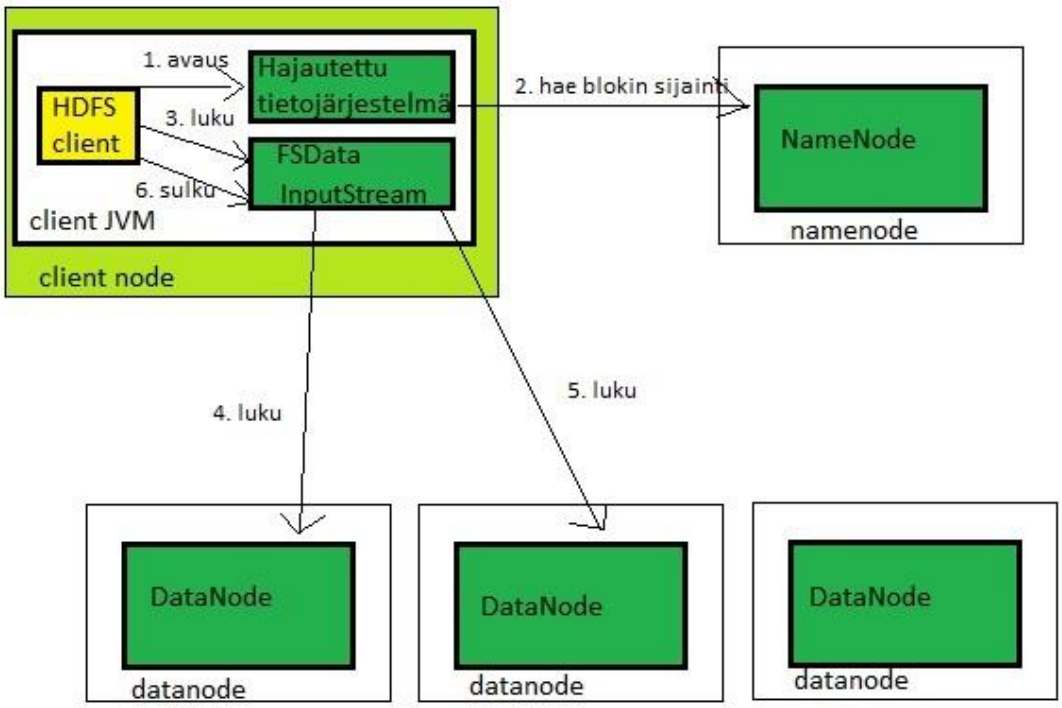
konfiguroituna, jolloin vikatilanteesta voidaan palautua ilman suuria viiveitä. Arkkitehtuurissa on otettava huomioon kuitenkin seikkoja, kuten:

- namenodet vaativat jaetun muutoslokin, joka on jatkuvasti saatavilla
- datanoden pitää lähettää raportit kaikille namenodeille, sillä mappaukset ovat namenoden muistissa, eikä levyllä
- clientien pitää suoriutua huomaamatta namenoden vaihdosta
- toisen namenoden tiedot tulee myös tallentaa vara-namenodelle, palautuspisteiden takaamiseksi

Aktiivisen namenoden vaihdosta vastaa failover controller –systeemi. Kontrollereita on useita, mutta vakiona käytössä on ZooKeeper. Sen tehtävä on varmistaa, että aktiivisena on ainoastaan yksi namenode. Tiedon namenodeilta se saa prosessilta, joka on päällä jatkuvasti niissä, eikä prosessi vaadi paljoa tehoa. (White 2015, 69)

4.6 Datavirtaus

HDFS:n, namenoden ja datanoden välinen datavirtaus koostuu kuudesta kohdasta, joita kuvataan kuviossa 6.



Kuvio 6. HDFS clientin datan luku

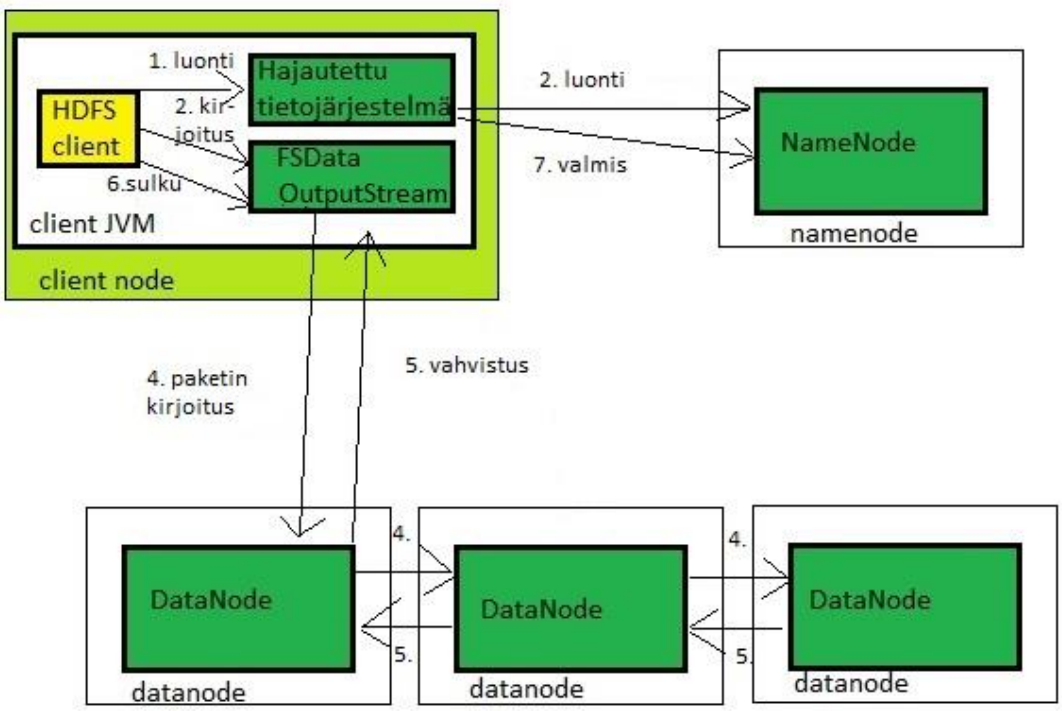
Ykkösvaiheessa clientti avaa tiedoston käyttämällä *open()* kutsua tietojärjestelmän objektille, jolle HDFS on instanssina. Toisessa vaiheessa tietojärjestelmä kutsuu namenodea etäproseduurikutsuja käyttämällä ja selvittää ensimmäisten blokkien sijainnin. Namenode palauttaa kaikkien blokkien datanoden osoitteen, joka sisältää kopion oikeasta blokista. Datanoden valinta perustuu verkkotopologiaan, jolloin lähimpänä clienttiä oleva datanode valitaan. MapReducen tapauksessa client voi olla itsessään datanode, jolloin se lukee paikallisen datanoden blokin kopiota varten. Kolmosvaiheessa client käyttää *read()* kutsua InputStreamille, joka tietää ensimmäisten blokkien osoitteet ja osaa yhdistää lähimpään datanodeen tiedoston lukua varten. Nelosvaiheessa dataa streamataan takaisin clientille, joka kutsuu *read()* komennolla streamia toistuvasti. Vaiheesa viisi blokki loppuu, InputStream sulkee yhteyden ja etsii parhaan datanoden seuraavan blokin lukua varten. Vaihto tapahtuu clientin kannalta huomaamattomasti. Kohdassa kuusi client on lopettanut lukemisen ja se kutsuu InputStreamia *close()* komennolla. (White 2015, 87)

Virhetilanteessa kommunikoinnissa datanodelle, InputStream etsii seuraavaksi lähimmän datanoden kyseiselle blokille ja muistaa vikaantunen noden turhien

uusintakyselyjen välttämiseksi. InputStream tarkastaa myös tarkistussumma-arvot ja korruptoituneet datanodet raportoidaan namenodelle. (White 2015, 88)

Tämän mallin parhaimpana puolina on skaalautuvuus usean yhtäaikaisen clientin datanluku, koska liikenne on jakautunut klusterin ympäri eri datanodeille. Namenoden tehtävätkin ovat vain blokkien sijaintien välittämistä, jotka sijaitsevat namenoden muistissa toimien tehokkaasti. (White 2015, 88)

Kuviossa 7 on esitetty tiedoston kirjoitus HDFS:ään. Kirjoitusvaihe koostuu seitsemästä vaiheesta.



Kuvio 7. Clientin kirjoitus HDFS:ään.

Ensimmäisessä vaiheessa client luo tiedoston *create()* kutsulla. Tiedostojärjestelmä ottaa yhteyden namenodeen etäproseduurikutsulla ja luo uuden tiedoston tiedostojärjestelmän nimiavaruuteen. Tässä vaihessa siihen ei ole vielä liitetty mitään blokkia. Toisessa vaiheessa namenode tarkistaa, ettei tiedostoa ole jo olemassa, sekä clientin oikeudet tiedoston luomiseksi. Jos tiedoston luonti onnistuu, palauttaa tiedostojärjestelmä OutputStreamin clientille, joka hoitaa kommunikoinnin datanodelle ja namenodelle. Kolmoskohdassa OutputStream jakaa clientin kirjoittaman datan paketeiksi ja muodostaa niistä jonon sisäiseen *data queue* jonoonsa. Tämän jälkeen *DataStreamer* kysyy namenodelta uusia blokkisijainteja

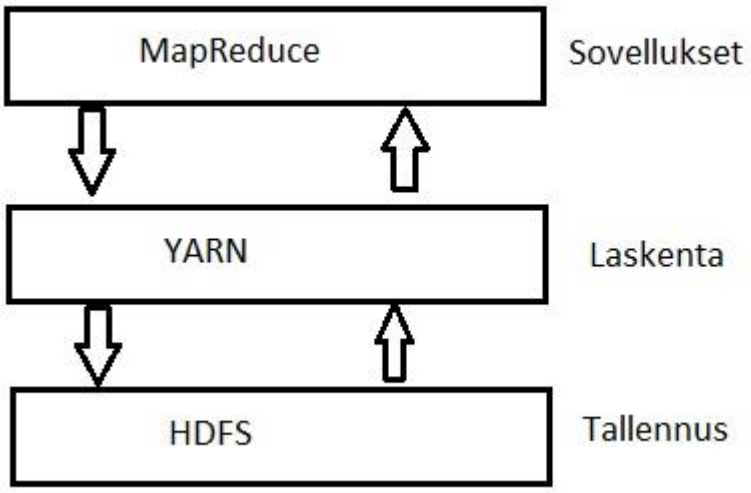
valitsemalla datanodet, joihin kopiot tallennetaan. Datanodeista muodostuu putki ja kuviossa 7 on esitetty replikointitason kolme putki. Neljännessä kohdassa *DataStreamer* streamaa paketit ensimmäiselle datanodelle, joka tallentaa paketit, sekä jakaa ne seuraavalle datanodelle. Putken seuraavat datanodet toistavat saman prosessin kuin ensimmäisenä paketit saanut datanode. Viidennessä kohdassa OutputStream pitää paketit sisäisessä vahvistusjonossaan, eli *ack queuessa*. Paketti poistuu sieltä vasta, kun kaikki putken datanodet ovat vahvistaneet paketit. Kuudennessa vaiheessa client on lopettanut datan kirjoittamisen ja se kutsuu streamia *close()* kutsulla. Silloin jäljelle jääneet paketit ajetaan putkeen odottamaan vahvistusta. Vasta vahvistusten jälkeen tapahtuu kohdan seitsemän signaalin lähetys namenodelle, joka ilmoittaa tiedoston kirjoituksen olevan valmis. (White 2015, 90)

Jos vikatilanne tapahtuu kirjoittaessa, tapahtuu ensiksi putken sulkeminen ja vahvistusjonossa olevat paktit lisätään datajonon eteen, jotta datanodet eivät menetä yhtään pakettia. Seuraavaksi toimivat datanodet saavat uuden identiteetin ja se ilmoitetaan namenodelle jolloin vikaantuneen datanoden osittainen blokki poistetaan, jos se palaa jatkossa toimintakykyiseksi. Vikaantunut datanode poistetaan putkesta ja jäljellä olevat toimivat datanodet muodostavat uuden putken. Loppu blokin datasta kirjoitetaan toimiville datanodeille. Kun namenode havaitsee alireplikoidun blokin, se muodostaa uuden noden, johon se replikoi tiedot. (White 2015, 91)

4.7 YARN

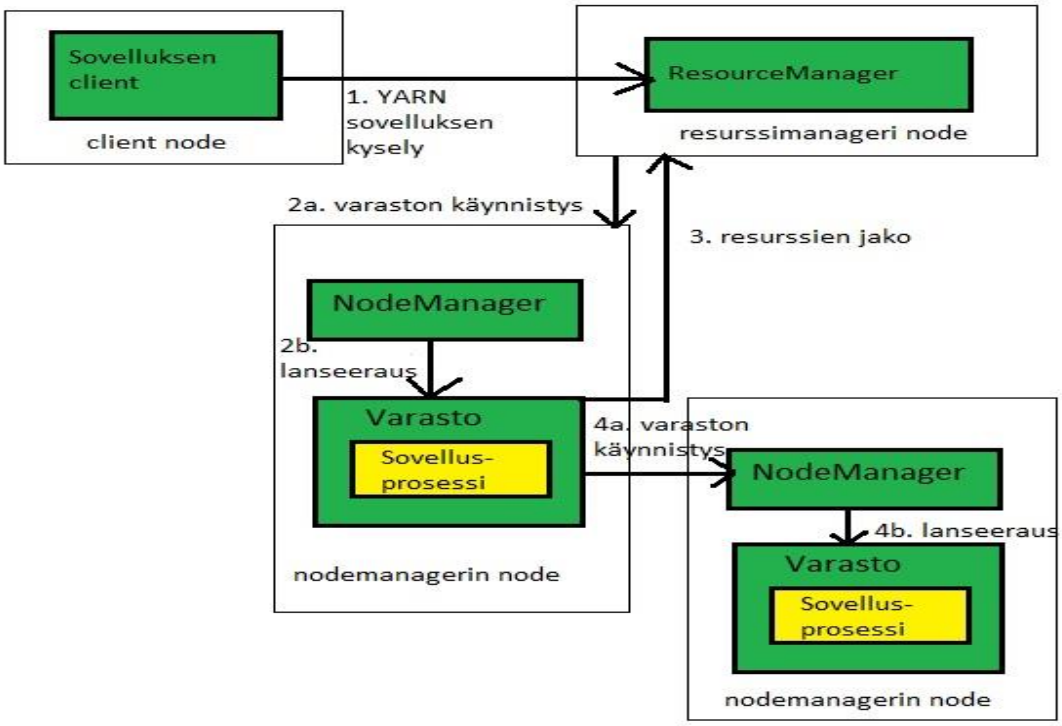
Apachen YARN on Hadoopin klusterin resurssinhallintajärjestelmä. YARN muodostuu sanoista Yet Another Resource Negotiator. Se esiteltiin Hadoopin kakkosversiossa MapReducen käytön parantamiseksi, mutta siitä on hyötyä myös muissa hajautetuissa laskentamalleissa. (White 2015, 97)

YARN tarjoaa ohjelmointirajapintoja, joilla klusterin resursseja pyydetään ja työstetään, mutta rajapintoja ei pysty yleensä käyttämään käyttäjän omalla koodilla. Sen sijaan on käytettävä korkeamman tason rajapintoja, jotka ovat tehty YARNia varten ja piilottavat resurssihallintatiedot käyttäjältä, kuten MapReduce. Kuviossa 8 on esitetty havainnollistava kuva YARNista. (White 2015, 97)



Kuvio 8. Esimerkki YARNista

YARN tarjoaa tärkeät palvelunsa kahdella eri taustaprosessityypillä, eli resurssimanagerilla ja nodemanagerilla. Resurssimanagereita on yksi jokaisella klusterilla ja ne vastaavat kaikesta resurssinhallinnasta. Nodemanageri on jokaisella klusterin nodella ja se vastaa varastojen käynnistämisestä ja monitoroinnista. Kuviossa 9 on esitetty esimerkki sovelluksesta YARNissa. (White 2015, 98)



Kuvio 9. Sovelluksen ajo YARNissa

Ensimmäisessä vaiheessa client ottaa yhteyden YARNin resurssimanageriin ja pyytää siltä *application master* –prosessin suorittamista. Toisessa vaiheessa resurssimanageri etsii nodemanagerin, joka voi lanseerata *application masterin* johonkin varastoon. *Application masterin* toiminta on täysin sovellusriippuvainen. Se voi joko suorittaa yksinkertaisen laskutoimituksen omassa varastossaan ja palauttaa vastauksen sovellukselle, tai se voi pyytää uusia varastoja suorittamaan hajautettua laskentaa, kuten kuvion 9 kohdissa kolme ja neljä. (White 2015, 98)

YARN käyttää resurssipyynnöissään joustavaa mallia, jossa pyydetty varastomäärä kertoo myös laskentakapasiteetin tarpeen. Pyynnöstä käy ilmi myös paikallisuusrajoitteet. Paikallisuus on tärkeää hajautetussa laskennassa klusterin kaistan säästämiseksi. Paikallisuusrajoitteet voivat tarkoittaa tiettyä paikallista nodea, räkkiä tai räkin ulkopuolista nodea. Aina paikallisuutta ei voi totetutta sovelluksen pyytämällä tavalla, jolloin YARN etsii lähintä nodea ensin samasta räkistä, tai lopulta mistä tahansa klusterin alueelta. Tavallinen tilanne on varaston lanseeraus HDFS blokin prosessia varten, kuten map taski. Silloin sovellus pyytää varastoa joltain nodelta, jossa blokki on replikoituna. Jos se ei onnistu, seuraavaksi node valitaan samasta räkistä, jossa replikoitu blokki on. Lopullinen vaihtoehto on etsiä node mistä tahansa klusterin alueelta. Resurssipyyntöjä voi tulla koko sovelluksen käynnissä olon ajan. Sovellukset voivat joko pyytää resursseja etukäteen, tai dynaamisesti tarpeen mukaan. (White 2015, 99)

YARNissa sovellusten ikä vaihtelee paljon, sillä osa sovelluksista on muutaman sekunnin mittaisia ja toiset voivat kestää kuukausien ajan. Siksi parempi tapa kategorisoida sovellukset sen perusteella, miten ne *mappaavat* käyttäjän suorittamat *jobit.* Yksinkertaisessa tapauksessa jokaisella sovelluksella on yksi *job*, kuten MapReducen tapauksessa. Toisessa mallissa yksi sovellus suoritetaan yhtä työnkulkua tai käyttäjäsessiota kohden. Tämä malli mahdollistaa varastojen uudelleenkäytön *jobien* välissä. Myös tiedon välimuistiin tallennus on vaihtoehtona. Sovellus voi myös toimia pitkäaikaisesti usealla käyttäjälle jaettuna. Silloin sovelluksen tulee pystyä koordinoimaan käyttäjän kutsuja. (White 2015, 100)

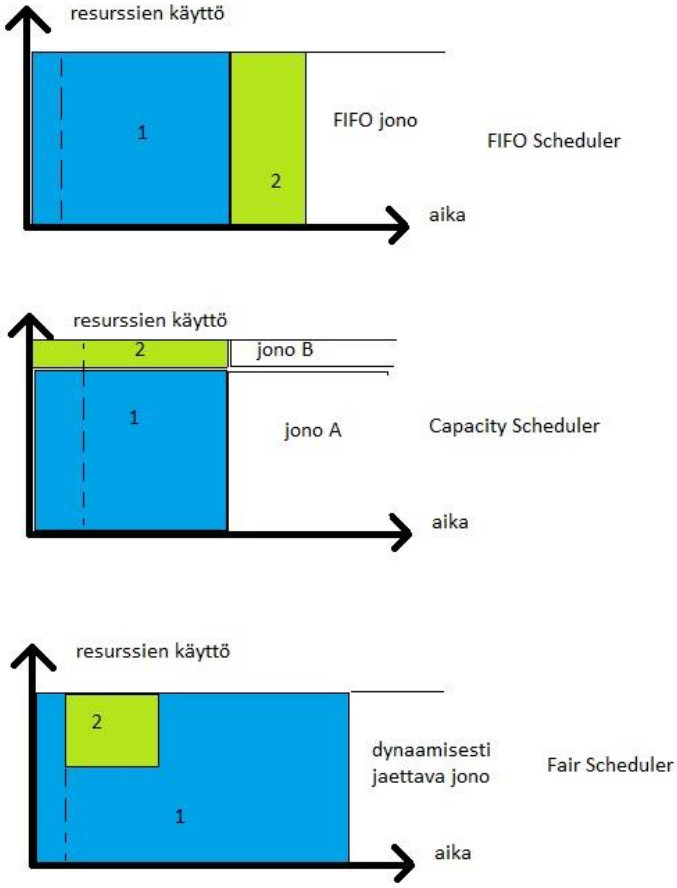
Ideaalitilanteessa YARN aikatauluttaisi kaikki pyynnöt tapahtumaan välittömästi, mikä ei kuitenkaan ole oikeasti mahdollista rajallisten resurssien takia. Varsinkin ruuhkaisella klusterilla sovellukset joutuvat odottamaan pyyntöjen toteuttamista. Ei

ole olemassa yhtä oikeaa tapaa aikatauluttaa sovelluksia, joten YARN tarjoaa usean vaihtoehdon aikatauluttajiksi ja konfiguroitavia käytäntöjä. YARNissa on tarjolla kolme aikatauluttajaa, jotka ovat FIFO, Capacity ja Fair Scheduler. (White 2015, 104)

FIFO toimii periaatteella, jossa sovellukset suoritetaan niiden jättämisjärjestyksen mukaisesti. Ensimmäisen sovelluksen pyynnöt suoritetaan loppuun ennen siirtymistä jonon seuraavaan pyyntöön. Vaikka FIFO on helppo ymmärtää, eikä vaadi konfigurointia, ei se ole hyvä valinta jaetuissa, suurissa klustereissa, jossa sovellukset joutuvat aina odottamaan vuoroaan. (White 2015, 104)

Capacity Scheduler käyttää erillistä jonoa, joka sallii pienempien *jobien* suorittamisen välittömästi pyynnöstä. Klusterin kokonaiskäytön kannalta tämä ei ole paras tapa, sillä kaikki jonon kapasiteetti on varattu suoritettaville *jobeille*. Silloin suurten *jobien* suoritus on hitaampaa kuin FIFOssa. (White 2015, 104)

Fair Schedulerissa on dynaaminen resurssien balanssin hallinta kaikkien *jobien* kesken. Jos suoritetaan vain yhtä isoa *jobia*, saa se käyttöön kaiken laskentatehon klusterissa. Jos vierelle käynnistyy toinen *job*, se saa puolet resursseista käyttöönsä tasapuolisen resurssijaon vuoksi. Resurssien jaossa on aina pientä viivettä, sillä varastoja täytyy ensin tyhjentää edelliseltä *jobilta.* Kuviossa 10 on kuvattu suurten ja pienten *jobien* aikataulutusta eri menetelmillä. Kuviossa ajan ollessa nolla, on ensimmäinen *job* –pyyntö jätetty ja katkoviivan kohdalla toinen. (White 2015, 104)



Kuvio 10. YARNin aikataulutusmenetelmät.

YARNissa on mahdollista käyttää myös viivästettyä aikataulutusta. Siitä on hyötyä tilanteissa, joissa klusteri pyytää tiettyä nodea, joka on sillä hetkellä varattu. Normaalisti tilanteessa aikatauluttaja etsisi samasta räkistä toista nodea, mutta käytännössä parin sekunnin odottamisella voidaan saada pyydetty node vapaaksi ja samalla kasvattaa klusterin tehokasta toimintaa. Viivästettyä aikataulutusta tukee Capacity Scheduler ja Fair Scheduler. (White 2015, 111)

4.8 Ambari

Ambari on avoimen lähdekoodin Hadoop – klusterin monitorointityökalu. Sitä käytettään selaimen avulla graafisesti. Ambari tukee useita Hadoop –komponenttia, kuten HDFS, MapReduce, Hive, Pig, HBase ja ZooKeeper. Ambarin avulla voi myös hallita ja asentaa Hadoop –palveluita etäpalvelimille. Uuden noden lisääminen ei ole yksinkertaista, jossa auttaa Ambarin ohjattu asennustoiminto. Ambarilla onnistuu myös keskitetty konfiguraatiomuutosten ajo jokaiselle klusterin nodelle, kuten myös palveluiden käynnistäminen ja sammutus. (Wadkar jne. 2014. Appendix C)

Ambarilla ei toistaiseksi pysty poistamaan nodeja, vaan silloin täytyy turvautua APIen käyttöön.

4.9 Docker

Docker on avoimien standardien alusta, jossa ohjelmistoja voidaan kehittää, paketoida ja tehdä niistä liitettäviä toisiin ohjelmistoihin. Dockerin avulla helpotetaan sovellusten riippuvuuksia, kuten kirjastot, lähdekoodi ja järjestelmätyökalut. Docker pakkaa kaikki tarvittavat ohjelmat ja riippuvuudet yksittäiseen yksikköön, eli Docker imageen. Sitä voidaan ajaa niin PC:llä, virtuaalikoneissa, datakeskuksissa kuin pilvessäkin. Docker imaget eroavat tavallisesta virtuaalikoneimageista siten, että docker ohjelmisto suoritetaan yhden käyttöjärjestelmän kernelin sisällä, mutta eristetyssä docker säiliöissä. Jokaisella docker säiliöllä on oma tiedostojärjestelmä ja ympäristömuuttujat. Säiliöt ovat kuitenkin erillään isäntäkäyttöjärjestelmästä ja muista säiliöistä. (Vohra 2016, Chapter 1)

Docker säiliöt sisältävät kaiken tarvittavan ohjelman suorittamiseksi, mutta joskus tiedostoja voidaan kopioida myös isäntäkäyttöjärjestelmästä. On myös mahdollista, että docker säiliöt linkitetään keskenään, jolloin ympäristömuuttujia ja ohjelmia toisesta säiliöstä saadaan jaettua. (Vohra 2016, Chapter 1)

Imagen luonnissa Docker käyttää Dockerfileä. Se sisältää kaiken tiedon ohjelmien lataussijainneista, tarvittavista komennoista, verkkoporteista, tiedostojen lisäämisestä tiedostojärjestelmään ja ympäristömuuttujien asetuksista. (Vohra 2016, Chapter 1)

5 InfoSphere BigInsights asennus

5.1 Vaatimukset

Valitsin asennettavaksi alustaksi *IBM BigInsights Quick Start Edition*in, jossa on seuraavat vaatimukset toimiakseen:

- Red Hat Enterprise Linux (RHEL) 7.x – 64bit

- Docker 1.8.1
- 4-ydin prosessori
- usean noden tapauksessa vähintään 5 nodea
- minimissään 16GB RAM
- minimissään 30GB kovalevytilaa
- root-oikeudet kaikilla nodeilla
- Internet-yhteys

5.2 Asennusvaiheet

Alkuvaiheessa quick startin koneet olivat määritetty käyttämään englantilaisia näppäimiä, joka muutettiin suomalaiseksi komennolla *system-config-keyboard.* Sitten muutettiin jokaisen noden hostname, sekä määriteltiin IP-osoitteet: *system-config-network*. Hostnameksi määriteltiin nodex.haatamii.com, jossa x korvataan kunkin noden numerolla siten, että masternode on nimeltään node1.haatamii.com.

Alussa on tärkeää lisätä jokaisen noden */etc/hosts* –tiedostoihin kaikki tarvittavat ip osoitteet. Tämä on ainoa tapa saada ambarin kautta tapahtuva klusterien asennus toimimaan oikein. Quickstartin koneilla */etc/hosts* –tiedosto generoituu automaattisesti käynnistyksen ohella erillisestä *setHostname.sh* tiedostosta, jota pitää myös muokata pysyvän muutoksen aikaansaamiseksi. Kuviossa 11 nähdään */etc/hosts* tiedoston tarpeelliset ip:t.

```
[root@node1 ~]# cat /etc/hosts
# File is generated from /home/virtuser/setHostname.sh
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost
192.168.44.2    node1.haatamii.com
192.168.44.3    node2.haatamii.com
192.168.44.4    node3.haatamii.com
```

Kuvio 11. /etc/hosts -tiedosto

Kun ip:t oli listattu jokaisen koneen *hosts* –tiedostossa, luotiin SSH-avaimet, jotta asennusvaiheessa ambari voi ottaa salasanattomasti yhteyden kaikkiin klusterin koneisiin. Avainten luonnin jälkeen julkinen avain kopioitiin kaikille nodeille

komennolla: *ssh-copy-id –i ~/.ssh/id_rsa.pub root@nodex.haatamii.com*, jossa x korvataan kunkin noden numerolla. Tärkeä asia ottaa huomioon oli myös *.ssh* hakemiston oikeuksien muuttaminen arvoon 700 ja *authorized_keys* hakemiston muuttaminen arvoon 600

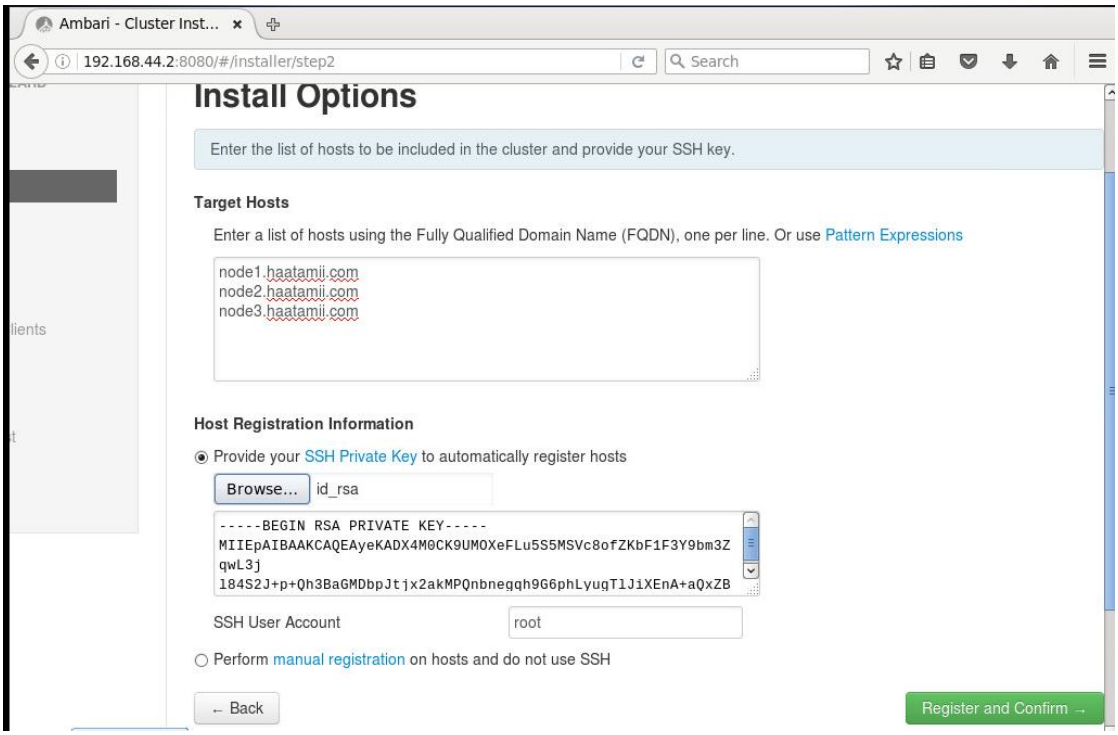
Seuraavassa vaiheessa luotiin docker.repo /etc/yum.repos.d/ -hakemistoon, johon lisättiin rivit:

```
[docker]
name=docker
baseurl=http://yum.dockerproject.org/repo/main/centos/7/
path=/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
```

Sitten asennettiin docker jokaiselle nodelle: *yum install docker-engine*. Nyt tuli ladata *qse-docker-deployer* –tiedosto IBM:n sivuilta, joka sisältää skriptejä, joilla voidaan määritellä klusteri toimimaan halutusti. Vakiona skripteissä oli vaatimuksena 16GB RAM, RedHat 7.x, ja minimissään viiden noden klusteri. Näitä kohtia tuli muokata halutun laiseksi *functions* –tiedostossa, joka sisältyi ladattuun *qse-docker-deployer* –pakettiin. Muutin RAM –vaatimukseksi 12GB, RedHat –versioksi 6.x ja klusterin kooksi kolme. Muutokset ovat havaittavissa liitteessä 1, johon merkitsin punaisella muutetut kohdat. Seuraavaksi luotiin hosts.txt –tiedosto, johon syötettiin rivi kerrallaan jokaisen noden hostname. Tärkeää oli laittaa masternode ensimmäiselle riville, jotta asennus onnistuu ja skriptit menevät läpi. *cleanHosts.py* –niminen skripti hoitaa mahdollisesti olemassa olevien docker säiliöiden poiston imageilta, jotka on listattu *hosts.txt*:ssä. Komento toimii seuraavasti: *python cleanHosts.py hosts.txt.* Sitten ajettiin *deploy.sh* –skripti, jolla saatiin ladattua image kaikille nodeille, sekä luotua säiliöt. Se myös käynnistää ambari-serverin masternodella ja ambari-agentin muilla nodeilla. Komento on *./deploy.sh hosts.txt*. Aivan skriptin lopussa tulee virheilmoitus: *bash: /tmp/run.sh Permission denied.* Tästä pääsee eroon muuttamalla chmodilla *run.sh* tiedosto ajettavaan muotoon, jolloin skripti suorittaa itsensä loppuun.

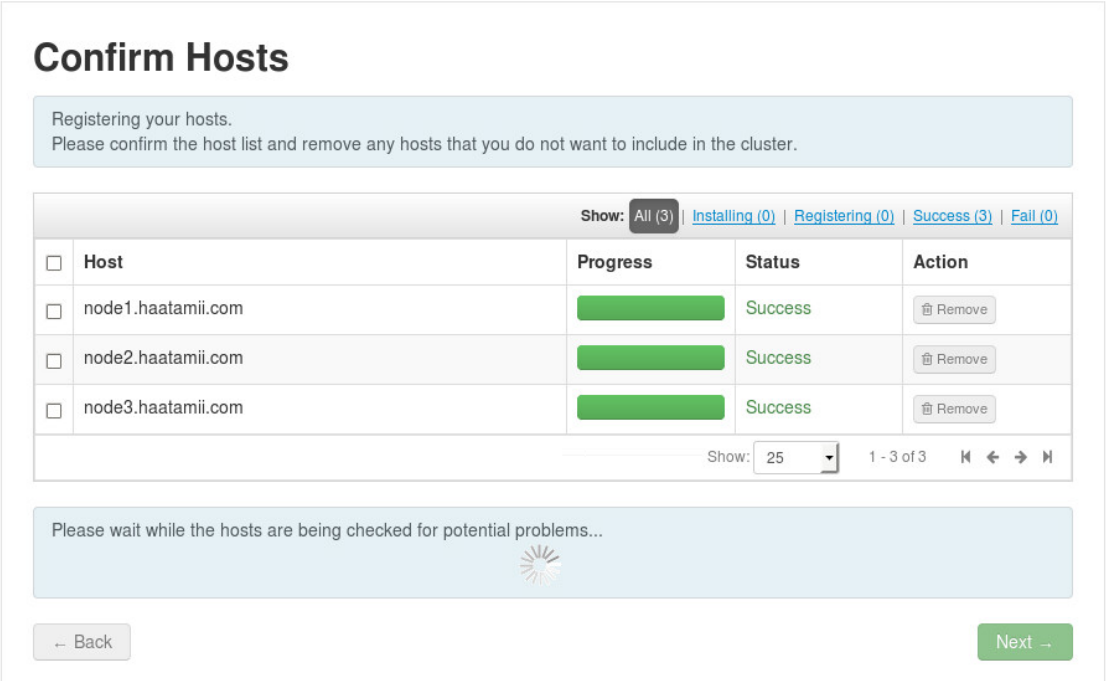
Tästä eteenpäin asennus jatkui selaimella ambarin asennuswizardia käyttäen. Osoitepalkkiin tulee syöttää ip 192.168.44.2:8080, jotta ambariin päästään käsiksi.

Etusivulla näkyy painike, jota painamalla käynnistyy ohjattu asennustoiminto. Ensimmäisenä täytyy määrittää klusterin nimi, jonka jälkeen ilmestyy kuvion 12 kaltainen vaihe. Tässä on tärkeää listata nodet oikeaan järjestykseen ja valita aiemmin luotu privaatti avain, jonka avulla ssh-yhteyksiä voidaan ambarilla muodostaa muihin nodeihin.



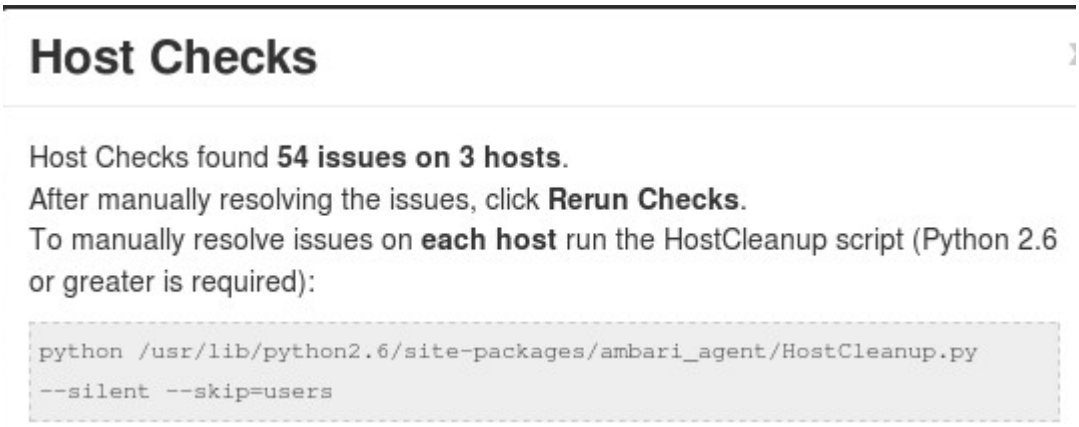
Kuvio 12. Asennuksen toinen vaihe

Kuviossa 13 nähdään, kuinka ssh-avainten tarkistus ja yhteyden muodostus on tapahtunut. Samalla ambari tarkistaa vielä ylimääräisiä ongelmia ennen asennuksen aloittamista ja roolien määrittämistä eri nodeille.



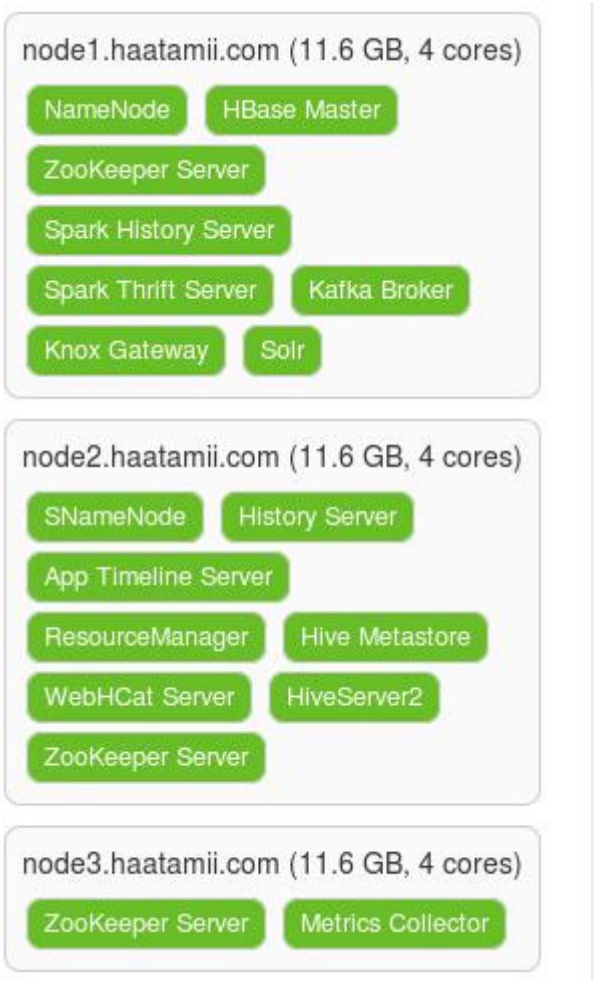
Kuvio 13. Hostien tarkastus

Tarkastuksen aikana erilaisia ongelmia ilmeni, kuten kuviosta 14 havaitaan, mutta niistä päästään eroon suorittamalla komento: *python /usr/lib/python2.6/site-packages/ambari_agent/HostCleanup.py –silent –skip=users.*



Kuvio 14. Ilmoitus mahdollisista ongelmista

Seuraavassa vaiheessa valittiin kullekin nodelle roolit. Tässä valitsin suositut asetukset ja roolit asettuivat kuvion 15 mukaisesti.



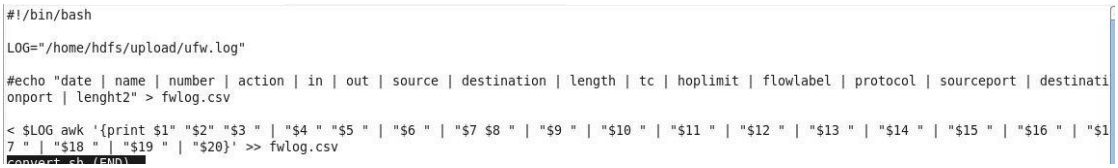
Kuvio 15. Valitut roolit jokaiselle nodelle

Tämän jälkeen asennus suoritti itsensä loppuun. Ainoastaan knoxille ja HBaselle piti asetuksista käydä syöttämässä salasana, jolloin palvelut saatiin käynnistymään. Ongelmia ilmeni myös nodejen ajan epätarkkuudesta, koska en saanut NTP:tä toimimaan. Asetinkin ajat manuaalisesti jokaiselle nodelle, jolloin palvelut suostuivat käynnistymään. Datanoden käynnistyksessä ilmeni ongelmia, sillä jostain syystä */hadoop/hdfs/data/current/VERSION* –tiedostossa clusterID oli node2:ssa ja node3:ssa väärä. Kun tämän arvon kopioi node1:ltä muilla nodeille, alkoi datanodet käynnistyä suunnitellusti.

5.3 Automaattisen skriptin luonti

Kun nodet oli saatu toimimaan, luotiin automaattinen skripti, joka kopioi ubuntu –koneelta viimeisimmän palomuurilokin ja lisää sen hiveä käyttäen hadoopiin. Aluksi luotiin ssh-avaimet Ubuntu-konetta varten node2:lla, jotta salasanaton ssh-yhteys toimii Ubuntu koneelta node2:lle palomuurilokien kopiointia varten.

Hiveen voi tuoda dataa .csv –muodossa, joten nyt palomuurilokit muutettiin .log muodosta .csv:ksi. Loin skriptin convert.sh, joka nähdään kuviossa 16. Siinä luodaan jokaiselle palomuurin tietokentälle omat sarakkeet ja tallennetaan tieto fwlog.csv:nä. Tiedostot ovat tallennettuina node2:n */home/hdfs/upload/* -hakemistossa.



Kuvio 16. Lokien muunto .csv –muotoon

Nyt voitiin siirtyä hiveen ja luoda table palomuurilokeille. Kuvio 17 näyttää tablen luontikomennon.

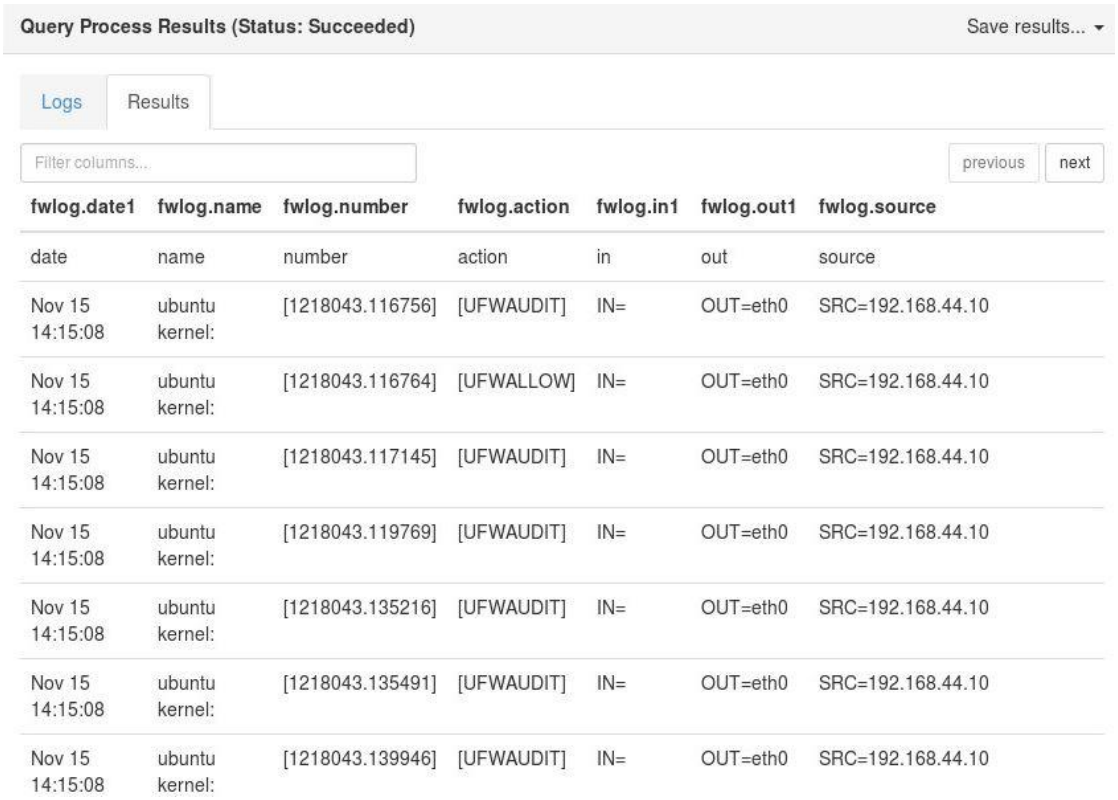
```
hive> CREATE TABLE Fwlog (date1 string, name string, number string, action string, in1
string, out1 string, source string, destination string, lenght string, tc string, hopli
mit string, flowlabel string, protocol string, sourceport string, destinationport strin
g, lenght2 string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';
```

Kuvio 17. Tablen luonti hiveen

Kun loki on saatu .csv –muotoon, se voidaan lisätä hiven tableen komennolla:

LOAD DATA LOCAL INPATH '/home/hdfs/upload/fwlog.csv' OVERWRITE INTO TABLE Fwlog;

Kuviosta 18 voidaan havaita lokien onnistunut tuonti hadoopiin. Ambarissa hiven query editoriin annettiin komento: *SELECT * FROM Fwlog;*



Query Process Results (Status: Succeeded) Save results...

Logs Results

Filter columns... previous next

fwlog.date1	fwlog.name	fwlog.number	fwlog.action	fwlog.in1	fwlog.out1	fwlog.source
date	name	number	action	in	out	source
Nov 15 14:15:08	ubuntu kernel:	[1218043.116756]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.116764]	[UFWALLOW]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.117145]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.119769]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.135216]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.135491]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10
Nov 15 14:15:08	ubuntu kernel:	[1218043.139946]	[UFWAUDIT]	IN=	OUT=eth0	SRC=192.168.44.10

Kuvio 178. Tulokset hiven querylle.

Lopuksi palomuurilokien kopiointi ja muuntaminen oikeaan muotoon, sekä tuonti hiveen tehtiin automaattiseksi käyttäen crontabia. Aluksi käynnistettiin crontab komennolla *crontab –e*. Sinne määriteltiin copy_logs2.sh –skriptin suoritus joka päivä kello 23.00. Crontabiin kirjoitettiin siis: *0 23 * * * /home/user1/scripts/copy_logs2.sh*.

Kuviossa 19 nähdään *copy_2.logs.sh* –skriptin sisältö. Aluksi logrotate siirtää uusimman palomuurilokin ufw.log.1 –tiedostoon ja nollaa alkuperäisen lokin uusia lokimerkintöjä varten. Tällä estetään samojen tietojen tuonti useaan kertaan. Logrotate säilyttää viisi uusinta lokitiedostoa ennen niiden ylikirjoitusta. Seuraavaksi otetaan ssh-yhteys node2:lle ja siirretään ufw.log.1 sinne, jossa se muutetaan .csv:ksi ja ladataan hiveen.

```
#!/bin/bash



TARGET_DIR=/home/hdfs/upload

logrotate -f /home/user1/scripts/firewall_logs
scp /var/log/ufw.log.1 root@node2.haatamii.com:${TARGET_DIR}/ufw.log

ssh root@node2.haatamii.com -C "
cd ${TARGET_DIR}
rm fwlog.csv
./convert.sh
sudo -u hdfs hive -e 'LOAD DATA LOCAL INPATH '\''${TARGET_DIR}/fwlog.csv'\'' INTO TABLE Fwlog;'"
```

Kuvio 19. Automaattinen lokien siirto ubuntulta hiveen

6 YHTEENVETO

6.1 Pohdinta

Opinnäytetyö aloitettiin helmikuussa 2016. Aluksi tutustuttiin big dataan yleisesti Immo Salon Big data ja pilvipalvelut –kirjaa lukemalla. Sen avulla sain hyvän käsityksen big datasta ja sen käyttötarkoituksista. Teoriaosuuden tekstiin ja aikataulussa pysymiseen sen osalta olen tyytyväinen.

Testijärjestelmän pystyttäminen ei kuitenkaan sujunut ongelmitta ja aikaa hukkaantui paljon järjestelmän saamiseen toimintakuntoiseksi. Osittain vika on omassa tietämättömyydessä, mutta myöskään IBM:n ohjeet eivät olleet kovin selkeitä, ja joiltain kohdin jopa ristiriitaisia. Lopulta järjestelmä saatiin kuitenkin toimimaan, jolloin pystyttiin demoamaan sen toimintaa tuomalla palomuurilokia järjestelmään. Harmillisesti en saanut järjestelmää toimintakuntoon haluamassani aikataulussa, jolloin Hadoopin ja Mapreducen käytännön toiminnan näkeminen jäi liian vähäiseksi.

Oman ymmärryksen ja tulevaisuuden työmahdollisuuksien takia oli tärkeää saada opinnäytetyössä kokemusta big datan alueelta, sillä se tulee jatkossa nousemaan suurempaan rooliin IT-maailmassa.

6.2 Jatkokehitys ja parannusehdotukset

Jatkossa järjestelmää voidaan käyttää datan analysointiin, kun käyttöönoton ongelmat on vihdoin selvitetty. Omalta osaltani big datan, Hadoopin ja Mapreducen ominaisuuksia ja käyttömahdollisuuksia on vasta raapaistu pinnalta. Kun järjestelmä

on nyt toimintakuntoinen, voisi sinne tuoda esimerkiksi lokitietoja monesta eri lähteestä ja niiden analysointia voisi kokeilla suuremmassa mittakaavassa. Samalla voidaan testata Hiven lisäksi muitakin vaihtoehtoja datan tuomiselle.

Lähteet

Hurwitz, J., Nugent, A., Halper, N., Kaufman, M. 2013. Big Data for Dummies. Wiley

Salo, I. 2014. Big data ja pilvipalvelut. Docendo

Schneider, Robert D. 2012. Hadoop for Dummies. IBM

Vohra, D. 2016. Pro Docker. Apress

Wadkar, S., Venner, J., Siddalingaiah, M. 2014. Pro Apache Hadoop. Apress

White, T. 2015. Hadoop: The Definitive Guide 4th Edition. O'Reilly.

Liitteet

Liite 1. functions.py, joka tarkistaa järjestelmän vaatimukset

```
#!/usr/bin/python
#-------------------------------------------------------------
# IBM Confidential
# OCO Source Materials
# (C) Copyright IBM Corp. 2010, 2015
# The source code for this program is not published or
# otherwise divested of its trade secrets, irrespective of
# what has been deposited with the U.S. Copyright Office.
#-------------------------------------------------------------

import random
import string
import sys, getopt
import argparse
import os
import subprocess
import threading
import time
import multiprocessing
import re
import pycurl, json
import logging
from pprint import pprint
import cStringIO
import sys

BLUEPRINT = 'blueprint.json'
HOSTMAPPING = 'hostmapping.json'
TEMPLATE_SINGLE_BLUEPRINT = 'template_single_blueprint.json'
TEMPLATE_SINGLE_VALUEADD = 'template_single_valueadd.json'
TEMPLATE_BLUEPRINT = 'template_blueprint.json'
TEMPLATE_VALUEADD = 'template_valueadd.json'
TEMPLATE_HOSTMAPPING = 'template_hostmapping.json'
TEMPLATE_SINGLE_HOSTMAPPING = 'template_single_hostmapping.json'
CLUSTER_NAME = 'demo'

#sys.stdout = open('log.txt', 'w')

from multiprocessing.dummy import Pool as ThreadPool
from subprocess import Popen, PIPE

logging.basicConfig(format='%(levelname)s: %(message)s',file-
name='deploy.log', level=logging.DEBUG)

class Pdsh:
  def __init__(self, hosts, pool=10):
    self.hosts = hosts
    self.pool = pool
    self.results = {}
    self.error = False

  def ssh(self, host):
    cli = ["ssh", host, self.cmd]
```

```
    process = Popen(cli, stdout=subprocess.PIPE, stderr=subpro-
cess.PIPE)
    (stdout, stderr) = process.communicate()
    if process.returncode!= 0:
      logging.error("Error running %s on %s." % (self.cmd, host))
      logging.debug("host: %s", host)
      logging.debug(stdout)
      logging.debug(stderr)
      self.error = True
    else:
      logging.info("Success running %s on %s." % (self.cmd, host))
    self.results[host] = process.returncode

  def run(self, cmd):
    self.cmd = cmd
    pool = ThreadPool(self.pool)
    pool.map(self.ssh, self.hosts)
    pool.close()
    pool.join()
    return self.results

def word_replace(text, replace_dict):

    rc = re.compile('|'.join(map(re.escape, replace_dict)))
    def translate(match):
         return replace_dict[match.group(0)]
    return rc.sub(translate, text)

def generateBlueprint(templateblueprint):
    fin = open(templateblueprint, "r")
    str1 = fin.read()
    fin.close()
    replace_dict = {
          "${MGMT1}" : "bi-hadoop-dev-008.services.dal.blue-
mix.net",
          "MGMT2" : " ",
          "WORKER1" : " ",
          "WORKER2" : " ",
          "WORKER3" : " "
    }
    str2 = word_replace(str1, replace_dict)
    fout = open(BLUEPRINT, "w")
    fout.write(str2)
    fout.close()

def generateBlueprint(hostarray,mode):
          nodesize=len(hostarray)
          templateblueprint = TEMPLATE_SINGLE_BLUEPRINT
          replace_dict = {}
          if mode == "iop" :
                    if nodesize == 1:
                              print "1"
                              MGMT_HOST=hostarray[0]
                              templateblueprint =
TEMPLATE_SINGLE_BLUEPRINT
                              replace_dict = {
                                        "${MGMT1}" : MGMT_HOST
                              }
                    elif nodesize == 3:
```

```
                                templateblueprint =
TEMPLATE_BLUEPRINT
                                nodesize=len(hostarray)
                                MGMT1=hostarray[0]
                                MGMT2=hostarray[1]
                                WORKER1=hostarray[2]
                                WORKER2=hostarray[3]
                                WORKER3=hostarray[4]
                                replace_dict = {
                                          "${MGMT1}" : MGMT1,
                                          "${MGMT2}" : MGMT2,
                                          "${WORKER1}" : WORKER1,
                                          "${WORKER2}" : WORKER2,
                                          "${WORKER3}" : WORKER3
                                }
                        fin = open(templateblueprint, "r")
                        str1 = fin.read()
                        fin.close()
                        str2 = word_replace(str1, replace_dict)
                        fout = open(BLUEPRINT, "w")
                        fout.write(str2)
                        fout.close()
                elif mode == "valueadd" :
                        if nodesize == 1:
                                print "1"
                                MGMT_HOST=hostarray[0]
                                templateblueprint =
TEMPLATE_SINGLE_VALUEADD
                                replace_dict = {
                                          "${MGMT1}" : MGMT_HOST
                                }
                        elif nodesize == 3:
                                templateblueprint =
TEMPLATE_VALUEADD
                                nodesize=len(hostarray)
                                MGMT1=hostarray[0]
                                MGMT2=hostarray[1]
                                WORKER1=hostarray[2]
                                WORKER2=hostarray[3]
                                WORKER3=hostarray[4]
                                replace_dict = {
                                          "${MGMT1}" : MGMT1,
                                          "${MGMT2}" : MGMT2,
                                          "${WORKER1}" : WORKER1,
                                          "${WORKER2}" : WORKER2,
                                          "${WORKER3}" : WORKER3
                                }
                        fin = open(templateblueprint, "r")
                        str1 = fin.read()
                        fin.close()
                        str2 = word_replace(str1, replace_dict)
                        fout = open(BLUEPRINT, "w")
                        fout.write(str2)
                        fout.close()



def registerBlueprint(hostname, blueprintname):
   url = "http://"+hostname+":8080/api/v1/blueprints/"+blueprint-
name+"?validate_topology=false"
```

```
   logging.debug('url is %s', url)
   logging.debug('blueprintname is %s', blueprintname)
   logging.debug('json filei %s', BLUEPRINT)
   with open(BLUEPRINT) as data_file:
    data = json.load(data_file)
   datadump = json.dumps(data)
   logging.debug('data dump %s', datadump)
   resp=000
   while resp == 000 :
          c = pycurl.Curl()
          c.setopt(pycurl.URL, url)
          c.setopt(pycurl.HTTPHEADER, ['X-Requested-By: ambari'])
          c.setopt(pycurl.USERPWD, "%s:%s" % ('admin', 'admin'))
          c.setopt(pycurl.POST, 1)
          c.setopt(pycurl.POSTFIELDS, datadump)
          try:
                    c.perform()
                    resp=c.getinfo(c.RESPONSE_CODE)
          except:
                    logging.warn("Waiting on ambari-server")
                    time.sleep(10)

   if resp == 201:
                    logging.info("Registered the blueprint suc-
cessfully")
   else:
                    error=c.getinfo(c.ERROR)
                    logging.error('Failed registering the blue-
print')
                    logging.debug('Error Code %ss', resp)
                    logging.debug('Error %s', error)
                    sys.exit()

def generateHostmapping(hostarray,mode):
          nodesize=len(hostarray)
          templateHostMapping = TEMPLATE_SINGLE_HOSTMAPPING
          replace_dict = {}
          if nodesize == 1:
                    MGMT_HOST=hostarray[0]
                    templateHostMapping =
TEMPLATE_SINGLE_HOSTMAPPING
                    replace_dict = {
                              "${MGMT1}" : MGMT_HOST
                    }
          elif nodesize == 3:
                    templateHostMapping = TEMPLATE_HOSTMAPPING
                    nodesize=len(hostarray)
                    MGMT1=hostarray[0]
                    MGMT2=hostarray[1]
                    WORKERS=""
                    for x in xrange(2, nodesize):
                              WORKERS=WORKERS+",{
\"fqdn\":\""+hostarray[x]+"\" }"
                    if WORKERS.startswith(","):
                              WORKERS = WORKERS[1:]
                    replace_dict = {
                              "${MGMT1}" : MGMT1,
                              "${MGMT2}" : MGMT2,
                              "${WORKERS}" : WORKERS
                    }
```

```
            fin = open(templateHostMapping, "r")
            str1 = fin.read()
            fin.close()
            str2 = word_replace(str1, replace_dict)
            fout = open(HOSTMAPPING, "w")
            fout.write(str2)
            fout.close()

def installCluster(mgmtnode,clustername):
            url = "http://"+mgmtnode+":8080/api/v1/clusters/"+clus-
ternam e
            #global LINK
            LINK = ''
            logging.debug('url is %s', url)
            logging.debug('hostmapping json is %s', HOSTMAPPING)
            logging.debug("MGMT Node %s",mgmtnode)
            logging.debug("clustername is %s",clustername)
            response = cStringIO.StringIO()
            with open(HOSTMAPPING) as data_file:
                        data = json.load(data_file)
            datadump = json.dumps(data)
            resp=000
            c = pycurl.Curl()
            c.setopt(pycurl.URL, url)
            c.setopt(pycurl.HTTPHEADER, ['X-Requested-By:ambari'])
            c.setopt(pycurl.USERPWD, "%s:%s" % ('admin', 'admin'))
            c.setopt(pycurl.POST, 1)
            c.setopt(pycurl.POSTFIELDS, datadump)
            c.setopt(c.WRITEFUNCTION, response.write)
            c.perform()
            #resp=pycurl.HTTP_CODE
            resp=c.getinfo(c.RESPONSE_CODE)
            logging.debug("Full response is %s", response.get-
value())
            logging.debug("====================================")
            logging.debug("resp is %s",resp)
            if resp == 202:
                        logging.info("Request accepted. Installation
is in progress...")
                        r = re.compile('\"href\" : "(.*?)\",')
                        m = r.search(response.getvalue())
                        if m:
                                    LINK = m.group(1)
                                    logging.debug("The link is %s",
LINK)
            else:
                        error=c.getinfo(c.ERROR)
                        logging.error('Failed to initiate install re-
quest. ')
                        logging.debug('Error Code %s', resp)
                        logging.debug('Error %s', error)
                        sys.exit()
            return LINK


def checkprogress(LINK):
            logging.debug("Status is %s",LINK)
            temp_percent=0
            print "\nProgress : "
            print " "
```

```
          loop_time=0
          progress_percent=0
          while (progress_percent != 100):
                    response = cStringIO.StringIO()
                    c = pycurl.Curl()
                    c.setopt(pycurl.HTTPHEADER, ['X-Requested-
By:ambari'])
                    c.setopt(pycurl.USERPWD, "%s:%s" % ('admin',
'admin'))
                    c.setopt(pycurl.URL, LINK)
                    c.setopt(pycurl.WRITEFUNCTION, re-
sponse.write)
                    c.perform()
                    c.close()
                    progress = ""
                    r = re.compile('\"progress_percent\" :
(.*?),')
                    m = r.search(response.getvalue())
                    if m:
                              progress = m.group(1)
                    progress_percent= int(float(progress))
                    output = "\r{0}% ... time spent {1} sec-
onds".format(progress_percent, loop_time)
                    sys.stdout.write("\r" + output)
                    sys.stdout.flush()
                    time.sleep(30)
                    loop_time = loop_time + 30
          print "\n"

def checkStatusCode(output, CMD) :
          match = re.search(r'HTTP/1.1 \d\d\d', output)
          statuscode = 0
          if match:
                    statuscode = (match.group()).rsplit(None,
1)[-1]
          if statuscode == "200" or statuscode == "201" or sta-
tuscode == "202" :
                    print 'Done'
          else :
                    print 'Failed to run the request. not accept-
ed'
                    print CMD

def checkFinalStatus(LINK) :
          logging.debug("Status is %s",LINK)
          response = cStringIO.StringIO()
          c = pycurl.Curl()
          c.setopt(pycurl.HTTPHEADER, ['X-Requested-By:ambari'])
          c.setopt(pycurl.USERPWD, "%s:%s" % ('admin', 'admin'))
          c.setopt(pycurl.URL, LINK)
          c.setopt(pycurl.WRITEFUNCTION, response.write)
          c.perform()
          c.close()
          status = ""
          r = re.compile('\"request_status\" : \"(.*?)\",')
          m = r.search(response.getvalue())
          if m:
                              status = m.group(1)
          if status == "COMPLETED" :
```

```
                    logging.info("Completed set up and starting
of services for cluster %s", CLUSTER_NAME)
          elif status == "ABORTED" :
                    logging.warn('Setup was aborted. Please check
logs')
          else :
                    logging.error('Setup failed. Please Check
logs')

def executeshellcommands(CMD, hostname):
        cmd = "ssh " + hostname + " \"" + CMD + "\""
        logging.debug("executeshellcommands(). cmd is: %s", cmd)
          proc = subprocess.Popen([cmd], stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
          (out, err) = proc.communicate()
          retCode = proc.returncode
          logging.debug("executeshellcommands(). retCode: %s",
retCode)
          #hetty
        #logging.debug("executeshellcommands(). out: " + str(out))
          #logging.debug("executeshellcommands(). err: " +
str(err))
          logging.debug("executeshellcommands(). out: %s", out)
          logging.debug("executeshellcommands(). err: %s", err)
        return (retCode, out, err)

def setupknox(hostname):
    logging.info("Knox setup for BigInsights valueadds")
    CMD1 = "docker exec iop-hadoop script -q -c '/usr/bin/knox-
setup-va.exp' /dev/null"
    CMD2 = "docker exec -i iop-hadoop /bin/sh -c \'su -l knox -c
\\\"/usr/iop/current/knox-server/bin/ldap.sh start\\\"\'"
    (retCode, out, err) = executeshellcommands(CMD1, hostname)
    if retCode != 0:
        logging.error('Fail to setup Knox for BigInsights')
    (retCode, out, err) = executeshellcommands(CMD2, hostname)
    if retCode != 0:
        logging.error('Fail to start up demo LDAP')

def addservice(service,MGMT_HOST):
          logging.info('Adding %s service',service)
          addCMD = "docker exec iop-hadoop curl -u admin:admin -H
\\\"X-Requested-By:ambari\\\" -i -X POST -d \'{\\\"Ser-
viceInfo\\\":{\\\"service_name\\\":\\\"\'"+service+"\'\\\"}}\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/service
s --silent -w \\\"%{http_code}\\\""

          (retCode, output, err) = executeshellcommands(addCMD,
MGMT_HOST)
          checkStatusCode(output, addCMD)

def addBigSQLComponents(HOST_ARR):
          addComponent("BIGSQL", "BIGSQL_WORKER", HOST_ARR[0])
          addComponent("BIGSQL", "BIGSQL_HEAD", HOST_ARR[0])

def addComponent(serviceName, componentName, MGMT_HOST):
          logging.info("adding "+componentName+" for "+service-
Name)
          addCMD = "docker exec iop-hadoop curl -u admin:admin -H
\\\"X-Requested-By:ambari\\\" -i -X POST
```

```
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/service
s/"+serviceName+"/components/"+componentName+" --silent -w
\\\"%{http_code}\\\""

        (retCode, output, err) = executeshellcommands(addCMD,
MGMT_HOST)
          checkStatusCode(output, addCMD)

def rmNologin(HOST_ARR):
    RM_NOLOGIN="docker exec iop-hadoop rm -f /run/nologin"
    proc = Pdsh(HOST_ARR)
    proc.run(RM_NOLOGIN)

def prepareBigSql(service,version,HOST_ARR):
    rmNologin(HOST_ARR)
    addservice(service,HOST_ARR[0])
    addBigSQLComponents(HOST_ARR)
    createBigSQLConfig(version, HOST_ARR[0])
    applyBigSQLConfigtoCluster(version, HOST_ARR[0])
    createBigSQLHostComponent(HOST_ARR)

def createBigSQLConfig(version,MGMT_HOST):
    logging.info("Creating bigsql-env config")

    cfgCMD="docker exec iop-hadoop curl -u admin:admin -H \\\"X-
Requested-By:ambari\\\" -i -X POST -d \'{\\\"type\\\":\\\"bigsql-
env\\\",\\\"tag\\\":\'\\\""+version+"\\\"\',\\\"proper-
ties\\\":{\\\"bigsql_hdfs_poolsize\\\":\\\"0\\\",\\\"db2_fcm_port_
number\\\":\\\"28051\\\",\\\"enable_impersona-
tion\\\":\\\"false\\\",\\\"bigsql_ha_port\\\":\\\"20008\\\",\\\"bi
gsql_hdfs_poolname\\\":\\\"autocachepool\\\",\\\"db2_port_num-
ber\\\":\\\"32051\\\",\\\"scheduler_ser-
vice_port\\\":\\\"7053\\\",\\\"public_table_ac-
cess\\\":\\\"false\\\",\\\"scheduler_ad-
min_port\\\":\\\"7054\\\",\\\"bigsql_db_path\\\":\\\"/var/ibm/bigs
ql/database\\\",\\\"ena-
ble_yarn\\\":\\\"NO\\\",\\\"bigsql_resource_percent\\\":\\\"25\\\"
,\\\"bigsql_continue_on_fail-
ure\\\":\\\"false\\\",\\\"dfs.datanode.data.dir\\\":\\\"/run/ha-
doop/bigsql\\\"}}\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/configu
rations --silent -w \\\"%{http_code}\\\""

    (retCode, output, err) = executeshellcommands(cfgCMD,
MGMT_HOST)
    checkStatusCode(output, cfgCMD)

    logging.info("Creating bigsql-users-env config")

    cfgCMD="docker exec iop-hadoop curl -u admin:admin -H \\\"X-
Requested-By:ambari\\\" -i -X  POST -d \'{\\\"type\\\":\\\"bigsql-
users-env\\\",\\\"tag\\\":\'\\\""+version+"\\\"\',\\\"proper-
ties\\\":{\\\"ambari_user_login\\\":\\\"admin\\\",\\\"am-
bari_user_password\\\":\\\"ad-
min\\\",\\\"bigsql_user_id\\\":\\\"2824\\\",\\\"ena-
ble_ldap\\\":\\\"NO\\\",\\\"bigsql_user\\\":\\\"bigsql\\\",\\\"big
sql_user_password\\\":\\\"bigsql\\\"},\\\"properties_attrib-
utes\\\":{\\\"ambari_user_password\\\":{\\\"to-
Mask\\\":\\\"true\\\"},\\\"bigsql_user_password\\\":{\\\"to-
Mask\\\":\\\"true\\\"}}}\'
```

```
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/configu
rations --silent -w \\\"%{http_code}\\\""

    (retCode, output, err) = executeshellcommands(cfgCMD,
MGMT_HOST)
    checkStatusCode(output, cfgCMD)

def applyBigSQLConfigtoCluster(version, MGMT_HOST):
          logging.info("Applying bigsql-env config")
          bigSqlcfg = "{\\\"Clusters\\\":{\\\"desired_con-
figs\\\":{\\\"type\\\":\\\"bigsql-env\\\",\\\"tag\\\":\\\""+ver-
sion+"\\\"}}}"
          applyConfigtoCluster(bigSqlcfg, MGMT_HOST)

          logging.info("Applying bigsql-users-env config")
          bigSqlcfg="{\\\"Clusters\\\":{\\\"desired_con-
figs\\\":{\\\"type\\\":\\\"bigsql-users-
env\\\",\\\"tag\\\":\\\""+version+"\\\"}}}"
          applyConfigtoCluster(bigSqlcfg,MGMT_HOST)

def applyConfigtoCluster(jSonData,MGMT_HOST):
          applyCMD="docker exec iop-hadoop curl -u admin:admin -H
\\\"X-Requested-By:ambari\\\" -i -X  PUT -d \'"+jSonData+"\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+" --si-
lent -w \\\"%{http_code}\\\""

          (retCode, output, err) = executeshellcommands(applyCMD,
MGMT_HOST)
          checkStatusCode(output, applyCMD)

def createBigSQLHostComponent(HOST_ARR):
    logging.info("Creating host component BIGSQL_HEAD")
    bigSqlhcfg="{\\\"host_compo-
nents\\\":[{\\\"HostRoles\\\":{\\\"compo-
nent_name\\\":\\\"BIGSQL_HEAD\\\"}}]}"
    if (len(HOST_ARR) > 1):
        createHostComponent (bigSqlhcfg, HOST_ARR[0], HOST_ARR[1])
    else:
        createHostComponent (bigSqlhcfg, HOST_ARR[0], HOST_ARR[0])

    logging.info("Creating host component BIGSQL_WORKER")
    bigSqlhcfg="{\\\"host_compo-
nents\\\":[{\\\"HostRoles\\\":{\\\"compo-
nent_name\\\":\\\"BIGSQL_WORKER\\\"}}]}"
          #createHostComponent (bigSqlhcfg, MGMT_HOST)
          # Check to see if single node or five node cluster
    if (len(HOST_ARR) > 1):
        for WORKER in HOST_ARR[2:]:
            createHostComponent(bigSqlhcfg, HOST_ARR[0], WORKER)
    # Create single worker on single node cluster
    else:
        createHostComponent (bigSqlhcfg, HOST_ARR[0], HOST_ARR[0])

def createHostComponent (bigSqlhcfg, MGMT_HOST, WORKER):
    hostCmd="docker exec iop-hadoop curl -u admin:admin -H \\\"X-
Requested-By:ambari\\\" -i -X  POST -d \'"+bigSqlhcfg+"\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/hosts?H
osts/host_name="+WORKER+" --silent -w \\\"%{http_code}\\\""
```

```
    (retCode, output, err) = executeshellcommands(hostCmd,
MGMT_HOST)
    checkStatusCode(output, hostCmd)

def callBigSQLInstall (HOST_ARR):
    installCmd="docker exec iop-hadoop curl -u admin:admin -H
\\\"X-Requested-By: ambari\\\" -X PUT -d \'{\\\"RequestInfo\\\":
{\\\"context\\\" :\\\"Install BIGSQL via REST\\\"}, \\\"Body\\\":
{\\\"ServiceInfo\\\": {\\\"state\\\": \\\"INSTALLED\\\"}}}\'
http://"+HOST_ARR[0]+":8080/api/v1/clusters/"+CLUSTER_NAME+"/servi
ces/BIGSQL --silent -w \\\"%{http_code}\\\""
    (retCode, output, err) = executeshellcommands(installCmd,
HOST_ARR[0])
    statuscode = output[-3:]
    logging.debug("status code is %s", statuscode)
    if statuscode == "200" or statuscode == "202" :
        logging.info("\nInstall BIGSQL request accepted. Execution
in progress...")
        trackBigSQLProgress (output,"install",HOST_ARR)
        startCmd="curl -u admin:admin -H \\\"X-Requested-By: am-
bari\\\" -X PUT -d \'{\\\"RequestInfo\\\": {\\\"context\\\"
:\\\"Start BIGSQL via REST\\\"}, \\\"Body\\\": {\\\"Ser-
viceInfo\\\": {\\\"state\\\": \\\"STARTED\\\"}}}\'
http://"+HOST_ARR[0]+":8080/api/v1/clusters/"+CLUSTER_NAME+"/servi
ces/BIGSQL --silent -w \\\"%{http_code}\\\""
        (retCode, output, err) = executeshellcommands(startCmd,
HOST_ARR[0])
        statuscode = output[-3:]
        if statuscode == "200" or statuscode == "202" :
            logging.info("Start BIGSQL request accepted. Execution
in progress...")
            trackProgress (output,"start")
        else :
            logging.error("Failed to initiate start BIGSQL service
request. not accepted")
            logging.debug(startCmd)
    else :
        logging.error('Failed to initiate install BIGSQL service
request. not accepted')
        logging.debug(installCmd)

def checkBigSQLprogress(LINK,HOST_ARR):
    logging.debug("Status is %s",LINK)
    temp_percent=0
    print "\nProgress : "
    print " "
    loop_time=0
    progress_percent=0
    onlyonce=0
    while (progress_percent != 100):
        response = cStringIO.StringIO()
        c = pycurl.Curl()
        c.setopt(pycurl.HTTPHEADER, ['X-Requested-By:ambari'])
        c.setopt(pycurl.USERPWD, "%s:%s" % ('admin', 'admin'))
        c.setopt(pycurl.URL, LINK)
        c.setopt(pycurl.WRITEFUNCTION, response.write)
        c.perform()
        c.close()
        progress = ""
        r = re.compile('\"progress_percent\" : (.*?),')
```

```
        m = r.search(response.getvalue())
        if m:
            progress = m.group(1)
            #print "Progress percentage is", progress
        #print "Response is ", response.getvalue()
        progress_percent= int(float(progress))
        #print "-------------------------",progress_percent
        output = "\r{0}% ... time spent {1} seconds".format(pro-
gress_percent, loop_time)
        sys.stdout.write("\r" + output)
        sys.stdout.flush()
        time.sleep(30)
        loop_time = loop_time + 30
        if progress_percent > 35 and onlyonce == 0:
            bigsqlWorkAround(HOST_ARR)
            onlyonce=1
    print "\n"

def bigsqlWorkAround(HOST_ARR):
    proc = Pdsh(HOST_ARR[1:])
    cpyconfig="docker exec iop-hadoop cp /root/.ssh/config
/home/bigsql/.ssh"
    chownconfig="docker exec iop-hadoop chown bigsql:hadoop
/home/bigsql/.ssh/config"
    startDBM = "docker exec iop-hadoop /bin/su -c \"db2 start
dbm\" - bigsql"
    setBigSQLADM = "docker exec iop-hadoop /bin/su -c \"db2 -s up-
date dbm cfg using SYSADM_GROUP bigsqladm\" - bigsql"
    proc.run(cpyconfig)
    proc.run(chownconfig)
    proc.run(startDBM)
    proc.run(setBigSQLADM)

def trackBigSQLProgress(cmdOutput,state,HOST_ARR):
    logging.debug("output is ")
    logging.debug(cmdOutput)
    r = re.compile('\"href\" : "(.*?)\",')
    m = r.search(cmdOutput)
    LINK = ""
    if m:
        LINK = m.group(1)
    logging.debug("The link is %s", LINK)
    checkBigSQLprogress(LINK,HOST_ARR)
    checkFinalStatus(LINK)

def callInstall (serviceName, MGMT_HOST):
    installCmd="docker exec iop-hadoop curl -u admin:admin -H
\\\"X-Requested-By: ambari\\\" -X PUT -d \'{\\\"RequestInfo\\\":
{\\\"context\\\" :\\\"Install "+serviceName+" via REST\\\"},
\\\"Body\\\": {\\\"ServiceInfo\\\": {\\\"state\\\":
\\\"INSTALLED\\\"}}}\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/service
s/"+serviceName+" --silent -w \\\"%{http_code}\\\""
    (retCode, output, err) = executeshellcommands(installCmd,
MGMT_HOST)
    statuscode = output[-3:]
    logging.debug("status code is %s",statuscode)
    if statuscode == "200" or statuscode == "202" :
          logging.info("\nInstall "+serviceName+" request ac-
cepted. Execution in progress...")
```

```
          trackProgress (output,"install")
          startCmd="curl -u admin:admin -H \\\"X-Requested-By:
ambari\\\" -X PUT -d \'{\\\"RequestInfo\\\": {\\\"context\\\"
:\\\"Start "+serviceName+" via REST\\\"}, \\\"Body\\\": {\\\"Ser-
viceInfo\\\": {\\\"state\\\": \\\"STARTED\\\"}}}\'
http://"+MGMT_HOST+":8080/api/v1/clusters/"+CLUSTER_NAME+"/service
s/"+serviceName+" --silent -w \\\"%{http_code}\\\""

          (retCode, output, err) = executeshellcommands(startCmd,
MGMT_HOST)
          statuscode = output[-3:]
          if statuscode == "200" or statuscode == "202" :
              logging.info("Start "+serviceName+" request ac-
cepted. Execution in progress...")
              trackProgress (output,"start")
          else :
              logging.error("Failed to initiate start "+service-
Name+" service request. not accepted")
              logging.debug(startCmd)

    else :
          logging.error('Failed to initiate install '+service-
Name+' service request. not accepted')
          logging.debug(installCmd)

###...Restarting the stale Services...###
def restartStaleServices(mgmtNode, clusterName):
    logging.info("Restarting stale services...")
    staleServices=["HDFS", "MAPREDUCE2", "YARN", "OOZIE", "HIVE",
"ZOOKEEPER", "HBASE", "BIGSHEETS"]
    for i in range(len(staleServices)):
          logging.info("\n\n")
          logging.info(staleServices[i] + " service restarting")
          restartService(staleServices[i], clusterName, mgmtNode)

###...Restart a service...###
def restartService(service, clust_name, mgmtNode):
    stopService(service, clust_name, mgmtNode)
    startService(service, clust_name, mgmtNode)

###...Stop Services...###
def stopService(serviceName, clusterName, hostName):
    stopCmd="docker exec iop-hadoop curl -u admin:admin -H \\\"X-
Requested-By: ambari\\\" -X PUT -d \'{\\\"RequestInfo\\\":
{\\\"context\\\" :\\\"Stop "+serviceName+" via REST\\\"},
\\\"Body\\\": {\\\"ServiceInfo\\\": {\\\"state\\\":
\\\"INSTALLED\\\"}}}\' http://"+hostName+":8080/api/v1/clus-
ters/"+clusterName+"/services/"+serviceName+" --silent -w
\\\"%{http_code}\\\""
    (retCode, output, err) = executeshellcommands(stopCmd, host-
Name)
    statuscode=output[-3:]
    logging.debug("status code is %s", statuscode)
    if statuscode != "200" and statuscode != "202" :
          logging.error("Failed to initiate stop "+serviceName+"
service request. not accepted")
          logging.debug(stopCmd)
    else:
          logging.info("Stop "+serviceName+" request accepted.
Execution in progress...")
```

```
          trackProgress(output,statuscode)

###...Start Services...###
def startService(serviceName, clusterName, hostName):
    startCmd="docker exec iop-hadoop curl -u admin:admin -H \\\"X-
Requested-By: ambari\\\" -X PUT -d \'{\\\"RequestInfo\\\":
{\\\"context\\\" :\\\"Start "+serviceName+" via REST\\\"},
\\\"Body\\\": {\\\"ServiceInfo\\\": {\\\"state\\\":
\\\"STARTED\\\"}}}\' http://"+hostName+":8080/api/v1/clus-
ters/"+clusterName+"/services/"+serviceName+" --silent -w
\\\"%{http_code}\\\""

    (retCode, output, err) = executeshellcommands(startCmd, host-
Name)
    statuscode=output[-3:]
    logging.debug("status code is %s", statuscode)
    if statuscode != "200" and statuscode != "202" :
          logging.error("Failed to initiate start "+serviceName+"
service request. not accepted")
          logging.debug(startCmd)
    else:
          logging.info("Start "+serviceName+" request accepted.
Execution in progress...")
          trackProgress(output,statuscode)

def trackProgress(cmdOutput,state):
    logging.debug("output is ")
    logging.debug(cmdOutput)
    r = re.compile('\"href\" : "(.*?)\",')
    m = r.search(cmdOutput)
    LINK = ""
    if m:
          LINK = m.group(1)
    logging.debug("The link is %s", LINK)
    checkprogress(LINK)
    checkFinalStatus(LINK)

###...Convert version string into version tuple...###
def versionTuple(version):
    return tuple(map(int, (version.split("."))))

###...Check if hostname is reachable from current host...###
def checkHostName(hostName):
  proc=subprocess.Popen("ping -c 1 " + hostName, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  (out, error) = proc.communicate()
  response = proc.returncode
  if response == 0:
    logging.info("checkHostName:passed")
    return True
  else:
    logging.warn("checkHostName: Failed")
    return False

###...Check SSH connectivity from current to all hosts listed in
the hosts file passed...###
def checkSsh(hostName):
  COMMAND="ssh -o 'PreferredAuthentications=publickey' " + host-
Name + " ls -la / > /dev/null"
```

```
  proc=subprocess.Popen(COMMAND, shell=True, stdout=subpro-
cess.PIPE, stderr=subprocess.PIPE)
  (out, error) = proc.communicate()
  retCode=proc.returncode
  if (retCode == 0):
    logging.info("checkSsh: Passed")
    statusSsh=True
  else:
    logging.warn("checkSsh: Failed")
    statusSsh=False
  return statusSsh

###...Check if OS is RHEL 7.x or CENTOS 7.x...###
def checkOS(hostName):
  COMMAND="cat /etc/system-release"
  shellOutput=subprocess.Popen(["ssh", "%s" % hostName, COMMAND],
shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  systemRelease=shellOutput.stdout.readlines()
  systemReleasetoString="".join(systemRelease)
  if systemRelease == []:
    logging.warn("OS Check Failed.")
    status_os = False
  else:
    if re.search("[Red\ Hat*release\ 6*]",systemReleasetoString):
      logging.info("OS Check: Passed")
      status_os = True
    elif re.search("[CentOS*release\ 6*]",systemReleasetoString):
      logging.info("OS Check: Passed")
      status_os = True
    else:
      logging.warn("Unsupported version of OS " + systemRe-
leasetoString)
      status_os = False
  return status_os

###...Check if root access is there...###
def checkRoot(hostName):
  COMMAND="id -u"
  proc=subprocess.Popen(["ssh", "%s" % hostName, COMMAND],
shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  (out, error) = proc.communicate()
  checkRoot=proc.returncode
  if checkRoot == 0:
    logging.info("CheckRoot: Passed")
    return True
  else:
    logging.warn("CheckRoot: Failed")
    return False

###...Check Free Space in /tmp Directory of Host...###
def checkFreeSpace(hostName):
  ###Required minimum free space for /tmp in GB
  minFreeSpace=40
  ###Required free space for /var/lib/docker in GB
  minDockerFreeSpace=40
  COMMANDChecktmp="df /tmp | awk 'NR>1' | awk '{print $4}'"
  shellOutputtmp=subprocess.Popen(["ssh", "%s" % hostName, COM-
MANDChecktmp], shell=False, stdout=subprocess.PIPE, stderr=subpro-
cess.PIPE)
  freeKBtmplist=shellOutputtmp.stdout.readlines()
```

```
  freeKBtmp="".join(freeKBtmplist)
  freeGBtmp=float(freeKBtmp)/(1024*1024)
  COMMANDCheckDocker="df /var/lib/docker | awk 'NR>1' | awk
'{print $4}'"
  shellOutputDocker=subprocess.Popen(["ssh", "%s" % hostName, COM-
MANDCheckDocker], shell=False, stdout=subprocess.PIPE, stderr=sub-
process.PIPE)
  freeKBDockerlist=shellOutputDocker.stdout.readlines()
  freeKBDocker="".join(freeKBDockerlist)

  if freeKBDocker != "":
    freeGBDocker=float(freeKBDocker)/(1024*1024)
  else:
    freeGBDocker=float(0)

  if (freeGBtmp>=minFreeSpace):
    logging.info("/tmp: Passed")
    return True
  else:
    logging.warn("/tmp: Failed. Required freespace is: " +
str(minFreeSpace) + " GB while current freespace is: " +
str(freeGBtmp) + " GB")
    return False

  if (freeGBDocker>=minDockerFreeSpace):
    logging.info("/var/lib/docker: Passed")
    return True
  else:
    logging.warn("/var/lib/docker: Failed. Required freespace is "
+ str(minDockerFreeSpace) + " GB while current freespace is " +
str(freeGBDocker) + " GB")
    return False

###...Check Total Physical Memory in the System...###
def checkSystemMemory(hostName):
  ###...required total memory in GB
  minMemory=12
  COMMAND="free | grep Mem | awk '{print $2}'"
  shellOutput=subprocess.Popen(["ssh", "%s" % hostName, COMMAND],
shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  memKBlist=shellOutput.stdout.readlines()
  memKB="".join(memKBlist)
  memGB=float(memKB)/(1024*1024)

  if (memGB>minMemory):
    logging.info("checkSystemMemory: Passed")
    return True
  else:
    logging.warn("checkSystemMemory: Failed. Minimum memory re-
quired is " + str(minMemory) + " GB while found " + str(memGB) + "
GB")
    return False

###...Check CPU Cores of the System...###
def checkCPUCores(hostName):
  minCPUCores=4
  COMMAND="cat /proc/cpuinfo | grep -c processor"
  shellOutput=subprocess.Popen(["ssh", "%s" % hostName, COMMAND],
shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  cores=shellOutput.stdout.readlines()
```

```
  cores = int("".join(cores))
  if (cores>=minCPUCores):
    logging.info("checkCPUCores: Passed")
    return True
  else:
    logging.warn("checkCPUCores: Failed. Minimum cores required is
" + str(minCPUCores) + ", found on system " + str(cores))
    return False

###...Check if Docker is installed in the System. Version is
1.10.x, and if it is running...###
def checkDocker(hostName):
  logging.info("checkDocker:")

  ###..Check if Docker is installed..###
  COMMAND="ssh " + hostName + " yum list installed 2>&1 | grep
docker &> /dev/null"
  proc=subprocess.Popen(COMMAND, shell=True, stdout=subpro-
cess.PIPE, stderr=subprocess.PIPE)
  (out, error)=proc.communicate()
  retCode=proc.returncode
  if (retCode == 0):
    logging.info("Docker Installed: Passed")
  else:
    logging.warn("Docker Installed: Failed")
    return False

  ###..Check Docker version..###
  versionCOMMAND="docker version | grep Version | awk 'NR==1' |
awk '{print $2}'"
  versionShellOutput=subprocess.Popen(["ssh", "%s" % hostName,
versionCOMMAND], shell=False, stdout=subprocess.PIPE, stderr=sub-
process.PIPE)
  versionRetCodelist=versionShellOutput.stdout.readlines()
  versionRetCode="".join(versionRetCodelist)

  if versionTuple(versionRetCode) >= versionTuple("1.10"):
    logging.info("Valid Docker Version: Passed")
  else:
    logging.warn("Valid Docker Version: Failed")
    return False

  ###..Check if Docker is active..###
  dockerStatusCheckCommand="ssh " + hostName + " service docker
status > /dev/null"
  dockerStatusShellOutput=subprocess.Popen(dockerStatusCheckCom-
mand, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
  (out, error) = dockerStatusShellOutput.communicate()
  dockerStatusRetCode=dockerStatusShellOutput.returncode
  if (dockerStatusRetCode == 0):
    logging.info("Docker Running: Passed")
    return True
  else:
    logging.warn("Docker Running: Failed")
    return False

###...Perform Prerequisite Checks...###
def prereqCheck(hostarray):
  containError = False
  for host in hostarray:
```

```
    logging.info("host: %s", host)
    if checkHostName(host) == True:
      if checkSsh(host) == True:

        if checkOS(host) == False:
          containError = True

        if checkRoot(host) == False:
          containError = True

        if checkFreeSpace(host) == False:
          containError = True

        if checkSystemMemory(host) == False:
          containError = True

        if checkCPUCores(host) == False:
          containError = True

        if checkDocker(host) == False:
          containError = True

      else:
        containError = True
        logging.error("As ssh is not working, rest of the checks
are skipped.")

    else:
      containError = True
      logging.error("As hostname couldn't be resolved, rest of the
checks are skipped.")

  if (containError == True):
    logging.error("Prerequisite check failed\n")
    return 1
  else:
    logging.info("Prerequisite check passed\n")
    return 0

###...Get the value of specified configuration setting...###
def getConfigValue(configFile, key, hostname):
  cmd = "docker exec iop-hadoop xmllint " + configFile + " | grep
\\\"<name>" + key + "</name>\\\" -C 2 | grep \'<value>\' | cut -d
\\\">\\\" -f2 | cut -d \\\"<\\\" -f1"
  (retCode, output, err) = executeshellcommands(cmd, hostname)
  logging.debug("getConfigValue(). retCode: %s", retCode)
  logging.debug("getConfigValue(). output: %s", output)
  logging.debug("getConfigValue(). err: %s", err)
  return output

def getOozieUrl(hostname):
  baseUrl = getConfigValue("/etc/oozie/conf/oozie-site.xml",
"oozie.base.url", hostname)
  baseUrl = baseUrl.lower().strip()
  logging.debug("getOozieUrl(). Oozie URL: %s", baseUrl)
  return baseUrl

###...Check if Oozie ShareLib is installed...###
def isOozieShareLibInstalled(hostname):
  baseUrl = getOozieUrl(hostname)
```

```
  cmd = "docker exec iop-hadoop curl -u admin:admin -H \\\"X-Re-
quested-By:ambari\\\" -i -X GET " \
    + baseUrl + "/v2/admin/list_sharelib --silent | grep -Po
\'[.*]?({\\\"systemMode\\\":\\\"NORMAL\\\"}).*?\'"

  (retCode, oozieShareLib, err) = executeshellcommands(cmd, host-
name)

  if oozieShareLib != "":
    logging.info("Oozie ShareLib is installed")
    return 0
  else:
    logging.warn("Oozie ShareLib is not installed")
    return 1

###...Install Oozie ShareLib if it is not already installed...###
def checkOozieShareLib(hostname):
  baseUrl = getOozieUrl(hostname)

  cmd = "docker exec iop-hadoop curl -m 10 " + baseUrl + "/v1/ad-
min/status --silent | grep -Po \'[.*]?({\\\"sys-
temMode\\\":\\\"NORMAL\\\"}).*?\'"

  (retCode, status, err) = executeshellcommands(cmd, hostname)
  logging.debug("Oozie status: %s" % status)

  if status == "":
    ###..Assumption: Oozie is started..###
    return 1

  result = isOozieShareLibInstalled(hostname)

  if result != 0:
    logging.info("Installing Oozie ShareLib...")

    cmd= "docker exec iop-hadoop hdfs getconf -confKey fs.de-
faultFS"
    (retCode, hdfsUri, err) = executeshellcommands(cmd, hostname)

    logging.debug("HDFS URI: %s" % hdfsUri)

    ###..Install Oozie ShareLib..###
    cmd = "docker exec iop-hadoop sudo -u oozie /usr/iop/cur-
rent/oozie-server/bin/oozie-setup.sh \
      sharelib create -fs " + hdfsUri + "-locallib  /usr/iop/cur-
rent/oozie-server/oozie-sharelib.tar.gz"

    (retCode, result, err) = executeshellcommands(cmd, hostname)

    logging.debug("return code from installing Oozie ShareLib:
%s", result)

    ###..Restart Oozie..###
    restartService("OOZIE", CLUSTER_NAME, hostname)

def main():
    mode="-i"
    with open ("hosts.txt", "r") as myfile:
          HOSTS=myfile.read().replace('\n', ' ')
```

```
    hostarray=HOSTS.split()
    nodesize=len(hostarray)
    version=time.strftime("%Y%m%d%H%M")
    #prepareBigSql("BIGSQL", version, hostarray[0])
    #callInstall("BIGSQL", hostarray[0])
    #generateBlueprint(hostarray, "iop")
    #registerBlueprint("bi-hadoop-dev-008.services.dal.blue-
mix.net","qse-1-nodes-42")
    #generateHostmapping(hostarray, mode)
    #link = installCluster(hostarray[0], CLUSTER_NAME)
    #checkprogress(link)
    #checkFinalStatus(link)
    #setupknox

main()
```