HAAGA-HELIA
ammattikorkeakoulu

eHUB-demo – pilvipalvelun käyttö EHR-datan käsittelyssä

Pasi Martikainen

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2016

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

5. huhtikuuta 2016

Tekijä
Pasi Martikainen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
eHUB-demo – pilvipalvelun käyttö EHR-datan käsittelyssä

Sivu- ja liitesivumäärä
33

Sairaaloiden välisen tietojenvaihdon yhteentoimivuus on ollut jo pitkään yksi suurimmista terveydenhuollon tietojärjestelmien kehittämiskohteista, ja tämä opinnäytetyö tuo tähän kehitykseen oman panoksensa.

Projektissa toteutettiin demo, jossa potilastietorekisteridataa viedään tietolähteestä suojattua verkkoyhteyttä käyttäen pilvipalveluun tilastollista laskentaa varten. Tarvittavat tilastotiedot ovat haettavissa pilvipalvelusta verkkoselaimelle ja esitettävissä sekä numeerisessa että graafisessa muodossa. Potilastietorekisteridataa pyritään käsittelemään standardimuotoisena pilvipalvelun tietovarastoon asti hyödyntäen uutta terveysdatan yhteentoimivuuden mahdollistavaa FHIR-viitekehystä.

Demo ohjelmoitiin Java 7:llä ja pilvipalvelun arkkitehtuuri rakennettiin Amazonin tarjoamista palveluista: S3, EC2 ja EMR. Pilvipalvelun osana käytettiin Jboss-sovelluspalvelinta, johon tehtiin web-palvelut rekisteridatan vastaanottamista varten ja laskennan tulosten siirtämiseen asiakkaalle. Kaikki palvelimen asiakasrajapinnat toteutettiin REST-arkkitehtuurimallin mukaisesti. Rekisteridatan tuottamiseen luotiin asiakasohjelma, joka toteutettiin JavaFX-teknologialla.

Työssä osoitettiin, että on mahdollista toteuttaa uusi terveydenhuollon tietojenkäsittelyn palveluarkkitehtuuri kustannustehokkaasti käyttäen avoimia standardeja ja moderneja pilvipalvelu- ja web-teknologioita sekä ohjelmistokehitystyökaluja. Kustannustehokkuus syntyy toisaalta maksuttomien lisenssien ja avoimien ohjelmointirajapintojen käytöstä, ja toisaalta pilvipalvelun käyttöön liittyvistä teknologisista ja taloudellisista eduista. Työssä käytetyt kehitystyökalut eivät vaadi erityisosaamista terveydenhuollon tietojärjestelmistä mahdollistaen kehitystyön nopean käynnistymisen, mikä edelleen lisää kustannustehokkuutta. Opinnäytetyöprojekti käynnistyi marraskuussa 2014, ja ensimmäinen demo esiteltiin maaliskuussa 2015.

Asiasanat
Pilvilaskenta, potilasasiakirjat, Java, FHIR

HAAGA-HELIA
ammattikorkeakoulu

Abstract

5th April 2016

Author
Pasi Martikainen

Degree programme
Business Information Technology

Report/thesis title	**Number of pages and appendix pages**
eHUB demo – processing EHR data in cloud service	33

For some time, the interoperability of data exchange between hospitals and other healthcare provider organizations has been one of the important areas of development of clinical and administrative computing in the field of health care, to which this thesis will provide its own contribution.

This project implemented demo software that first transfers clinical data from the test source to a cloud service with using secured connection. The clinical data is then used for statistical calculations within the cloud environment. Results can be fetched from the cloud and presented numerically or visually using graphs. Clinical data was handled with the guidelines of the new standard FHIR (Fast Healthcare Interoperability Resources).

The demo software used Java 7 and the cloud environment were built with components offered by Amazon Web Services (AWS): S3, EC2 and EMR. A part of the cloud service used the Jboss application server, which acted as a controller for uploading data to cloud storage, running statistical operation and transferring the results to the client. The client-side software that created the test data for the demo was built with JavaFX-technology. The interfaces of the web application for the client side were implemented by using the REST architectural model.

In the project it was shown to be possible to implement a new service architecture cost-efficiently. The cost-effectiveness is achieved with free licenses and modern web technologies, but also with the economic and technological benefits that cloud computing offers. The tools used in this project do not require special knowledge from the health care information systems. This allows for rapid initiation of the software projects which even more increases cost-efficiency.

Keywords
cloud computing, health care systems, Java, FHIR

Sisällys

1 Johdanto

Terveydenhuoltopalveluihin liittyvän tiedonsiirron yhteensopivuus on ollut jo pitkään yksi suurimmista haasteita terveydenhuollon tietojärjestelmien kehityksessä, ja tämä opinnäytetyö haluaa tuoda oman panoksensa tähän kehitystyöhön esittämällä uuden E2E-ratkaisun terveydenhuoltotiedon sujuvan siirtämisen ja prosessoinnin hallintaan. Uusilla ohjelmistopohjaisilla ratkaisuilla, jotka parantavat tietojen vaihtoa eri terveydenhuollon organisaatioiden kesken, on olemassa myös merkittävää taloudellista ja kaupallista merkitystä. Tämä tuo uusia kustannustehokkaita vaihtoehtoja terveydenhuollon palveluntarjoajille tietojenkäsittelyhaasteiden ratkaisemiseksi.

Tämä projekti pohjautuu useamman vuoden kestäneeseen esitutkimukseen, jossa opinnäytetyön esittämää ratkaisua on selvitetty sekä teknologiselta että kaupalliselta näkökannalta. Projektin kautta on pyrkimyksenä osoittaa, että käyttämällä standardiratkaisuja ja moderneja web-teknologioita, on mahdollista saada aikaan kustannustehokas ja silti laadukas ratkaisu terveydenhuollon tietojärjestelmien välisen tiedonsiirron ja tiedon prosessoinnin haasteisiin.

Kustannustehokkuus syntyy siitä, että käytetään ensinnäkin avointa terveystiedon standardia lisenssimaksujen minimoimiseksi ja toisaalta modernit, tehokkaat, avoimet ja tutut ohjelmistokehitystyökalut mahdollistavat matalan kynnyksen kehitystyöhön ryhtymiselle. Modernit työkalut, työmenetelmät ja teknologiat mahdollistavat edelleen käytännössä rajattomat mahdollisuudet uusien palveluiden kehittämiselle verrattuna suljettuihin järjestelmiin ja lisensseillä rajattuihin standardeihin, joissa kehitystyö on todennäköisesti kalliimpaa ja hitaampaa.

Projektissa käytettiin uutta avointa kansainvälistä terveystiedon yhteensopivuusstandardia FHIR (Fast Health Interoperability Resources) sekä yleisesti ja laajasti käytössä olevia teknologioita nykyaikaisten tietotekniikkapalveluiden rakentamisessa.

1.1 Tavoitteet

Projektissa toteutettiin demo, jossa EHR-dataa (electronic health record) vietiin tietolähteestä suojattua verkkoyhteyttä käyttäen pilvipalveluun tilastollista laskentaa varten, ja tarvittavat tilastotiedot olivat haettavissa pilvipalvelusta verkkoselaimelle ja esitettävissä sekä numeerisessa että graafisessa muodossa. Terveysdata pyrittiin käsittelemään standardimuotoisena pilvipalvelun tietovarastoon asti.

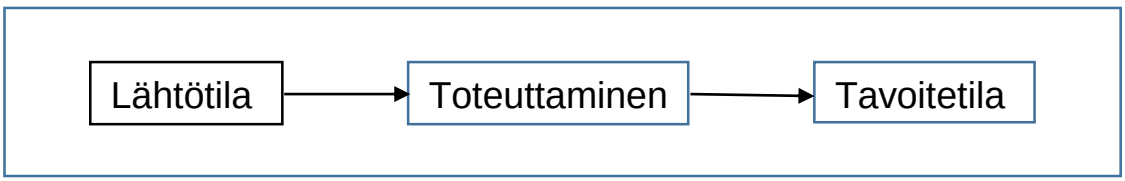
1.2 Rajaukset

Demosta rajattiin seuraavia asioita projektin ulkopuolelle:
a) Pilvipalvelun käyttäjien tunnusten ja salasanojen hallintasovellusta ei toteutettu. Demoa varten käytössä oli yksi tunnus–salasana-pari.
b) Terveysdata käyttö tiedon lähteenä oli supistettu vain demon kannalta tarvittavaan tietoon.
c) Pilvipalvelussa tehtiin tilastollista laskentaa vain yhden käyttötapauksen verran. Demossa ”tutkija” halusi tietää potilasaineiston verensokeriarvoista tehdyn jakauman prosentuaaliset osuudet.
d) FHIR-muotoisen datan jäsennin (parser) toteutettiin vain siinä laajuudessa kun oli tarpeen demossa käytettävän datan käsittelyyn.
e) Testausta ei suoritettu erikseen tässä opinnäytetyössä. Yksikkö- ja integraatiotestaukset olisivat kasvattaneet opinnäytetyön kokoa huomattavasti.

Opinnäytetyö noudattaa perinteistä mallia rakenteen suhteen. Johdannon jälkeen seuraa tietoperusta, jossa esitetään FHIR:n, pilvipalvelun ja tarvittavien web-teknologioiden keskeiset asiat demon toteuttamisen kannalta. Suunnittelu- ja toteutusosassa esitetään demon arkkitehtuuri ja eri ohjelmistokomponentit, joita tarvitaan kokonaisuuden tuottamiseksi. Lähdekoodia ja ohjelmien yksityiskohtaista toimintalogiikkaa ei esitetä tarkemmin salassapitovelvoitteen vuoksi. Lopuksi tässä raportissa on tulokset ja johtopäätökset.

1.3 Menetelmä

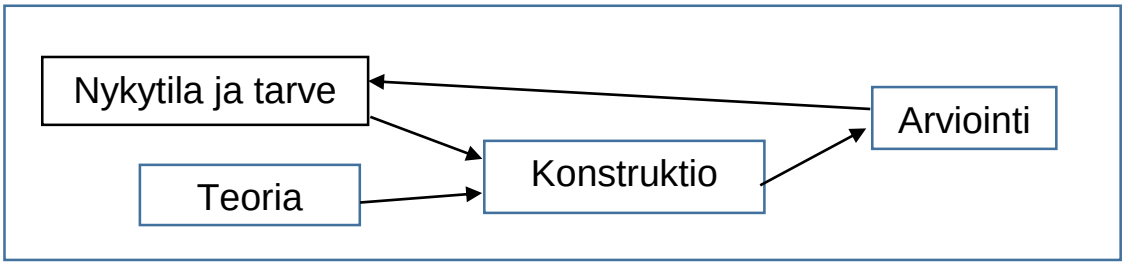
Opinnäytetyössä käytettiin demon kehittämiseksi konstruktiivista eli suunnittelutieteellistä menetelmää. Tämän tavoitteena on saavuttaa pysyvä muutos systeemissä, alkutilasta lopputilaan (Järvinen & Järvinen 2011, 103). Tämä on esitetty kuviossa 1. Tavoitetilaa kuvaa tuotettava konstruktio eli toimiva ohjelmisto.



Kuvio 1. Toteuttamisprosessi (Järvinen & Järvinen 2011, 108)

Ohjelman toteuttaminen kohti tavoitetilaa etenee iteratiivisesti ja lisäävästi, jolloin jokaisen osatuotoksen jälkeen arvioidaan sen soveltuvuutta tavoitetilan saavuttamiseksi. Tarvitta-

essa tehdään muutoksia ja kasvatetaan konstruktiota. Prosessissa joudutaan myös mahdollisesti hylkäämään aiempia osatuotoksia. Kuviossa 2 on esitetty iteratiivisen prosessin kulku karkeasti.



Kuvio 2. Iteratiivinen ohjelmiston kehittäminen

Tätä tapaa kutsutaan iteratiiviseksi ja lisääväksi kehitykseksi, IID (Iterative and Incremental Development). IID on ollut tietojenkäsittelyssä käytössä jo kauan, ainakin 1950-luvun jälkipuoliskosta lähtien. (Larman & Basili, 2003.)

2 HL7 ja EHR

EHR (Electronic Heatlh Record) on järjestelmällinen kokoelma sähköisessä muodossa olevia potilastietoja yksittäisestä ihmisestä tai populaatiosta. Potilastiedot on mahdollista jakaa useiden järjestelmien välillä. (Gunter & Terry 2005.) Tiedon jakaminen voi tapahtua muun muassa tietoverkkojen, kuten Internetin avulla. EHR voi sisältää suuren määrän erityyppistä dataa, kuten henkilötietoja, potilashistorian, tietoja lääkityksestä ja allergioista, laboratorioanalyysien tuloksia ja pankkiyhteystietoja. (topmobiletrends.com.)

HL7 (Health Level 7) on terveydenhuollon tietojärjestelmiä kehittävä kansainvälinen organisaatio, jonka tavoitteena on luoda kansainvälisiä standardeja kliiniselle ja hallinnolliselle datalle järjestelmien yhteensopivuuden parantamiseksi. HL7-organisaatio on kehittänyt standardin nimeltä HL7, joka keskittyy OSI-mallin mukaiseen sovelluskerrokseen ja on kansainvälisten standardoimisorganisaatioiden ANSI:n ja ISO:n hyväksymä. HL7-organisaatio on tuottanut useita laajassa käytössä olevia yhteensopivuusstandardeja, kuten HL7 versiot 2 ja 3 sekä CDA (Clinical Document Architecture). CDA on tiedonvaihtomalli kliiniselle datalle, ja sen pohjana on HL7 versio 3. (HL7 About.)

EHR-datan järjestelmien välistä yhteensopivuutta tarkasteltaessa on huomioitava erityisesti kolme seuraavaa osatekijää: tekninen, semanttinen ja prosessien välinen yhteensopivuus. Tekniset haasteet eli tiedon siirtäminen paikasta toiseen on ratkaistava. Semanttinen yhteensopivuus varmistaa taas sen, että data tulkitaan eri järjestelmissä samalla tavoin. Lopulta eri toimintaprosessien on kyettävä toimimaan yhdessä. (HL7 IntroToHL7.)

2.1 HL7 Versiot 2 ja 3

HL7 versio 2 on HL7:n kliinisen datan siirtoon luotu standardi, joka on saavuttanut suuren suosion, ja on maailmanlaajuisesti yksi yleisimmistä tuotantokäytössä olevista yhteensopivuusstandardeista. HL7 versio 2 standardia on myös sovellettu muihin tarkoituksiin kuin terveysdatan käsittelyyn. HL7 versio 2 käyttää kahden järjestelmän väliseen tiedonvaihtoon viestejä, jotka koostetaan useista terveysdataa sisältävistä lohkoista, ja sitä voidaan käyttää myös toimenpidepyyntöjen lähetykseen (potilassiirrot, laboratoriotutkimukset). Versio 2 tarjoaa myös mahdollisuuden käyttää yhdensuuntaista viestintää, kuten kommentteja. (FHIR: HL7 Version 2.)

HL7 versio 3 (v3) on alun perin suunniteltu olevan seuraavan sukupolven viestitysstandardi HL7 versio 2:n jälkeen ja se on tuonut mukanaan uuden tietotyyppimallin RIM (Reference Information Model). HL7 v3 määrittelee myös dokumenttien käytön vaihtoehtoisena

arkkitehtuurina aiemmalle viestipohjaiselle HL7 v2-standardille terveydenhuollon informaation jakamiseen. HL7 v3 standardi on myös tuotantokäytössä maailmanlaajuisesti ja sitä käyttävät erityisesti EHR:n kanssa toimivat organisaatiot, mutta se ei ole kuitenkaan saavuttanut suurta läpimurtoa verrattuna versio 2:een. (FHIR: HL7 Version 3.)

2.2 HL7 FHIR

FHIR (Fast Healtcare Interoperability Resources) on uuden sukupolven terveysdatan yhteensopivuusstandardi, jota kehitetään HL7-organisaation alaisuudessa. Se yhdistää parhaimmat puolet HL7:n versio 2:sta ja 3:sta sekä CDA:sta, ja käyttää uusimpia web-teknologioita, kuten esimerkiksi REST-rajapintaa ja XML- ja JSON-pohjaisia tietorakenteita. FHIR:iin perustuvat ratkaisut koostuvat komponenteista, joita kutsutaan resursseiksi (Resources). Resurssit voidaan koostaa isommiksi kokonaisuuksiksi, ja tällä tavoin pyritään täyttämään kokonaisvaltaisesti ne tarpeet, joita kliinisessä ja hallinnollisessa työssä esiintyy. FHIR on suunniteltu toimimaan useissa eri ympäristöissä, kuten mobiilisovelluksissa, pilvipalveluissa, EHR-resursseihin pohjautuvassa tiedon jakamisessa ja isojen terveydenhuollon organisaatioiden välisessä viestinnässä. FHIR on tällä hetkellä vielä kehitysvaiheessa ja HL7:llä on tavoitteena tehdä siitä täydellinen määrittely myöhemmin. (FHIR: Introducing HL7 FHIR.)

2.2.1 Tietotyypit

FHIR-standardi määrittelee kokoelman tietotyyppejä, joita käytetään resurssien elementeissä. Tietotyyppejä on kaksi eri kategoriaa: primitiiviset ja kompleksit tietotyypit. Primitiiviset tietotyypit sisältävät vain yksittäisen arvon, eikä niillä ole lapsielementtejä. Nämä tietotyypit on määritelty XML-skeeman perusteella, ja ne ovat instant, time, date, dateTime, decimal, integer, string, uri, boolean ja base64binary. JSON-dokumenteissa nämä tietotyypit on määritelty boolean-, number- ja string-tyyppisinä muuttujina.

Kompleksit tietotyypit sisältävät lapsielementtejä, jotka on määritelty standardissa. Näitä ovat ratio, sampledData, period, quantity, range, coding, attachment, codeableConcept, humanName, address, contactPoint, identifier, timing, signature, annotation sekä repeat. Nämä puolestaan sisältävät attribuutteja, jotka ovat primitiivisiä tietotyyppejä. Komplekseille tietotyypeille voidaan tehdä tiukka rakenteenmäärittely, jossa myös primitiivisten tietotyyppien arvoalueet voivat olla rajattuja. (FHIR – Data Types.)

2.2.2 Resurssit

FHIR määrittelee useita erilaisia resurssityyppejä, joita voidaan käyttää tiedon vaihtoon ja tallennukseen. Resurssi on objekti, jolla on yksilöivä identiteetti, johon voidaan yksikäsitteisesti viitata. Se voidaan identifioida yhdeksi niistä resurssityypeistä, jotka standardi määrittelee. Resurssi sisältää joukon rakenteista dataa resurssityyppien määrittelyiden mukaisesti. Se sisältää helposti luettavissa olevan XHTML esityksen resurssin sisällöstä, ja sillä on tunnistettavissa oleva versio, joka muuttuu resurssin sisällön muuttuessa. (FHIR Resource Definitions 2014.)

Tarkastellaan lähemmin yhtä näistä kliinisistä resursseista, joka on nimeltään ”Observation”, havainto. Se on keskeinen asia terveydenhuollossa, sillä se tukee diagnoosin tekoa, seuraa terveydentilan kehittymistä ja määrittelee havaintojen viitearvoja. Useimmat havainnot ovat yksinkertaisia nimi–arvo-pareja, mutta jotkut havainnot liittyvät yhdeksi loogiseksi kokonaisuudeksi tai jopa moniosaiseksi havainnoksi. (FHIR Resource Observation - Content 2014.)

Seuraavassa kuviossa on esimerkki vähimmäisvaatimukset täyttävästä Observation-resurssista.

```
<Observation xmlns="http://hl7.org/fhir">
   <name>
      <coding>
         <system value="http://loinc.org"/>
         <code value="2339-0"/>
         <display value="Glucose [Mass/volume] in Blood"/>
      </coding>
   </name>
   <valueQuantity>
      <value value="6.3"/>
      <units value="mmol/l"/>
      <system value="http://unitsofmeasure.org"/>
      <code value="mmol/l"/>
   </valueQuantity>
   <status value="final"/>
   <reliability value="ok"/>
</Observation>
```

Kuvio 3. Observation-resurssi (FHIR Resource Observation - Content 2014)

2.2.3 Tietoturva ja FHIR

FHIR ei ole tietoturvaprotokolla, eikä se määrittele mitään tietoturvaan liittyvää toiminnollisuutta. Kuitenkin standardi määrittelee käytettävät protokollat ja mallit tiedonsiirrolle, joiden yhteys tietoturvaan liittyvien asioiden kanssa on määritelty muualla. FHIR-standardi

suosittelee, että kaikki tiedonvaihto tuotantokäytössä tulisi suojata käyttäen TLS/SSL-protokollaa, esimerkiksi https:ää. Käyttäjät ja asiakassovellukset voidaan todentaa halutulla tavalla. Tuotantokäytössä FHIR-standardiin pohjautuva järjestelmä tarvitsee komponentin, joka hallinnoi käyttäjiä, käyttäjien todentamista ja valtuutusta. Turvallisuusjärjestelmä koostuu kolmesta osasta, joista ensimmäinen on käyttäjien todennus. Toinen osa on pääsyn hallinta, jonka perusteella päätetään, mihin resursseihin käyttäjällä on pääsyoikeus. Kolmas osa on tapahtumaloki, johon tallennetaan tietoja käyttäjien kirjautumisista ja valvotaan mahdollisia järjestelmään tunkeutumisia ja asiatonta käyttöä. (FHIR Security 2014.)

3 Pilvipalvelut

Yhdysvaltojen kansallinen standardoimisorganisaatio NIST (National Institute of Standards and Technology) määrittelee pilvipalveluille 5 tyypillistä piirrettä:
Itsepalvelullisuus (on demand self-service). Asiakas voi käyttää tarjottuja resursseja, kuten palvelinaikaa ja tiedon tallennuskapasiteettia ilman, että se vaatii palveluntarjoajan puuttumista asiaan.
Pääsy palveluihin laajalti (broad network access). Pilvipalvelu on käytettävissä tietoverkon välityksellä eri päätelaitteilla, esimerkiksi kannettavilla tietokoneilla, työasemilla ja mobiililaitteilla.
Resurssien yhteiskäyttö (resource pooling). Palveluntarjoajan tarjoamat resurssit on jaettu useiden asiakkaiden käyttöön. Eri pilvipalvelun osasista voidaan koostaa kokonaisuuksia asiakkaan tarpeiden mukaan.
Nopea joustavuus (rapic elasticy). Resursseja voidaan joustavasti varata ja vapauttaa, ja joissain tapauksissa palvelu voi skaalautua automaattisesti tarpeen mukaan.
Käytön tarkka mittaaminen (measured service). Pilvipalvelu automaattisesti valvoo ja optimoi resurssien käyttöä. Resurssien käyttöä valvotaan, ohjataan ja siitä raportoidaan siten, että sekä palveluntarjoaja että käyttäjä saavat tietoa palvelun käytöstä. (Mell & Grance 2011, 2.)

3.1 Pilvipalvelumallit

Pilvipalvelut voidaan jakaa kolmeen päätyyppiin niiden toteutustavan mukaan: Iaas, Saas ja Paas.

IaaS (Infrastructure-as-a-Service) on ratkaisu, joka on käytetyin ja kehittynein pilvipalvelujen markkina-alueella. Palvelu tarjoaa mukautuvan tietojenkäsittelyn infrastruktuurin tarpeen mukaan, ja siihen kuuluu muun muassa verkkolaitteet, kuormantasaajat, tietokanta- ja www-palvelimet. (Kavis 2014, 46.)

PaaS (Platform-as-a-Service) on ratkaisu, joka tarjoaa sovellusten kehitys- ja käyttöympäristön pilvipalvelussa. Palvelu rakentuu väliohjelmistosta (middleware), joka toimii alustana sovelluksille. Väliohjelmiston eräs tärkeimmistä tehtävistä on sovellusten hallinta. PaaS-toteutukset eivät tarjoa mahdollisuutta muokata alustana olevan käyttöjärjestelmän asetuksia. (Kavis 2014, 48.)

Pääasiallinen PaaS-palvelumallin käyttöalue on sovelluskehitys, jossa useampi ohjelmistokehittäjä tai muu käyttäjä ovat yhteistyössä ja kehitys- ja testausympäristö on automatisoitu. PaaS ei ole erityisen käyttökelpoinen, kun sovelluksen tulee olla siirrettävissä tai kun asiakkaan omaa kehitysympäristöä käytetään tai sovelluksen vaatimaa laitteistoa tai ohjelmistoa on räätälöitävä suorituskyvyn optimoimiseksi. (Marinescu 2013, 13.)

Saas (Software as a Service) on palvelumalli, jossa sovellus on tarjottu palveluna asiakkaalle, joka käyttää sitä internetin välityksellä. Palvelun käyttäjän ei tarvitse tietää, missä sovellus on fyysisesti käynnissä tai mikä käyttöjärjestelmä on sen alustana. Käyttäjän ei tarvitse myöskään asentaa mitään sovelluksia itse. (Velte, Velte & Elsenpeter. 2010, 11.)

3.2 Pilvityypit

Pilvipalvelut voidaan jakaa kolmeen päätyyppiin, joita ovat yksityinen, julkinen ja hybridipilvi. Yhteisöpilvi on neljäs tyyppi, joka on yksityisen pilven erikoistapaus.

Julkiset pilvet olivat ensimmäinen tyyppi pilvipalveluja, joita kehitettiin ja tarjottiin käyttäjille. Ne tarjoavat ratkaisuja, joilla voidaan optimaalisesti pienentää tietojärjestelmän kustannuksia ja tarjota mahdollisuus hallita epätasaista kuormaa paikallisessa infrastruktuurissa. Tämä pilvityyppi on muodostunut kiinnostavaksi vaihtoehdoksi pienille yrityksille, jotka voivat käynnistää liiketoiminnan ilman, että ne joutuvat tekemään isoja laitteistohankintoja alkuvaiheessa. Yksi tärkeistä ominaisuuksista on se, että pilvipalvelun resurssien käyttö voi kasvaa tai vähentyä vuokraamalla virtuaalista infrastruktuuria tai tilaamalla sovelluspalveluja tarpeen mukaan. (Green 2013, 125.)

Yksityinen pilvi on virtuaalinen hajautettu järjestelmä, joka perustuu yksityiseen infrastruktuuriin ja tarjoaa organisaation sisällä käyttäjille skaalautuvia tietokoneresursseja. Pilviresurssien käytön hinnoittelu tehdään eri tavalla verrattuna julkisiin pilviin, joissa on käytössä laskutus käytön mukaan. Organisaation eri yksiköitä voidaan laskuttaa sen mukaan, mikä on niiden osuus yksityisen pilven käytöstä. Tällä pilvipalvelumallilla on se etu, että liiketoiminnan kannalta tärkeät ja arkaluontoiset tiedot voidaan säilyttää organisaation sisällä omassa tietojärjestelmässä. Tämän lisäksi olemassa olevia resursseja voidaan paremmin hyödyntää, koska yksityinen pilvi voi kootusti tarjota niitä vaihtelevalle määrälle käyttäjiä. (Green 2013, 126.)

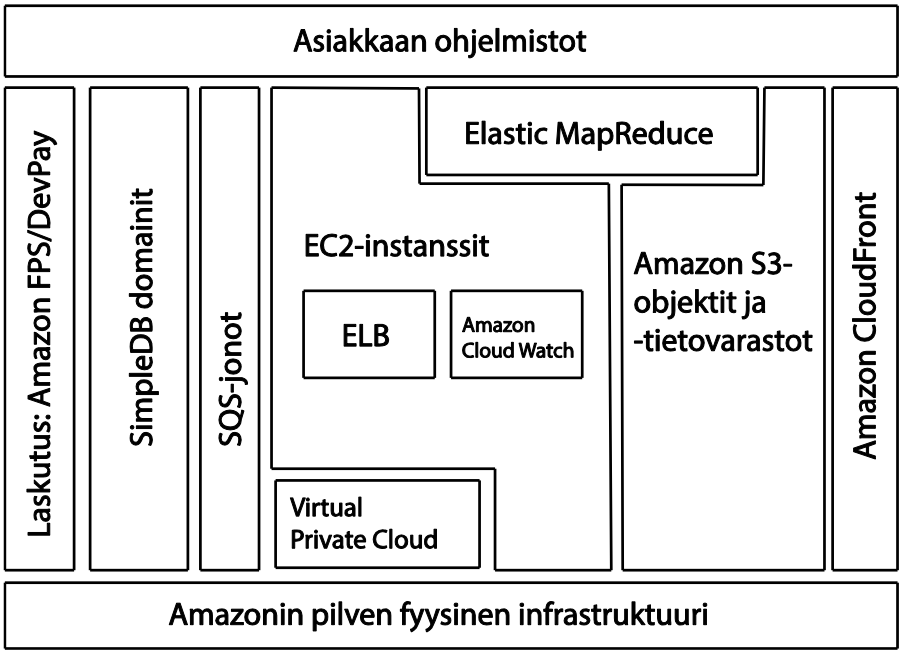
Hybridipilvet mahdollistavat yritykselle käyttää jo olemassa olevaa tietojärjestelmää arkaluontoisten tietojen hallintaan ja samalla käyttää ulkoisia resursseja, joiden käyttö kasvaa ja vähenee kulloisenkin tarpeen mukaan. Järjestelmä voidaan toteuttaa esimerkiksi siten,

että julkisessa pilvessä tehdään vain toimintoja, joihin ei liity isoja riskejä kun taas kriittiset toiminnot tehdään yksityisessä pilvessä. (Green 2013, 129.)

Yhteisöpilvi on yksityisen pilven kaltainen palvelumalli, mutta siinä palvelua käyttää rajattu joukko organisaatioita, joilla on yhteinen tarve pilvipalvelun käytölle. Tällä saavutetaan yksityisen pilven edut, ja samalla voidaan myös säästää kustannuksissa kun resurssit on jaettu organisaatioiden kesken. Yhteisöpilvellä saavutetaan myös parempi tietoturva kuin julkisella pilvellä, mikä voi olla merkittävä asia esimerkiksi terveydenhuollon hajautetuissa järjestelmissä. (Green 2013, 131–132.)

3.3 Amazon Cloud

Amazonin julkisen pilvipalvelun (AWS) juuret ulottuvat vuoteen 2002, kun Amazon teki päätöksen käyttää isoksi kasvanutta infrastruktuuriaan palvelujentarjontaan ohjelmistokehittäjille. AWS on kasvanut suureksi ja monipuoliseksi kokonaisuudeksi reilun kymmenen vuoden aikana. Kuviossa 4 on esitetty Amazonin pilvipalvelun arkkitehtuuri pääpiirteissään.



Kuvio 4. Amazonin pilvipalvelun arkkitehtuuri (Barr, J. 2010)

Amazon pilvi sisältää suuren joukon palveluja, joista keskeisimpiä ovat EC2, S3 ja ELB. *EC2 (Elastic Compute Cloud)* on palvelu, joka tarjoaa skaalautuvaa laskentaresurssia, eli virtuaalipalvelimia Amazonin datakeskuksissa. Näiden avulla asiakkaan on mahdollista rakentaa omia palvelinjärjestelmiä. (Amazon EC2, User Guide for Microsoft Windows 2015.)

S3 (Simple Storage Service) on tietovarasto Internetiä varten. Tieto tallennetaan säiliöihin, joita kutsutaan nimellä bucket. Näihin säiliöihin voidaan siirtää Internetistä tietoa, sitä voidaan hakea, luoda ja poistaa tarpeen mukaan. (Amazon S3: Developer Guide 2015, 2-4.)

ELB (Elastic Load Balancing) automaattisesti jakaa tulevan verkkoliikenteen useiden EC2-instanssien kesken. Mikäli yksittäinen EC2-instanssi sammuu, ELB kykenee reitittämään uudelleen kuorman jäljellä oleville instansseille. (Elastic Load Balancing, Developer Guide 2015.)

Muita ydinpalveluja ovat EMR, CloudFront, SQS-jonot ja SimpleDB. Näistä Amazon Elastic MapReduce (EMR) on palvelu, jonka avulla voidaan käsitellä suuria määriä dataa tehokkaasti. EMR käyttää Hadoop-viitekehystä, joka on yhdistetty useisiin Amazonin pilvipalvelun komponentteihin. Palvelun avulla voidaan suorittaa esimerkiksi tiedon louhintaa, lokitiedostojen analytiikkaa, tieteellisiä simulaatioita ja internetin indeksointia. (Amazon Elastic MapReduce Documentation 2016.)

CloudFront on palvelu, joka nopeuttaa staattisten ja dynaamisten web-sivujen komponenttien hakuaikoja sivujen käyttäjille. Tällaisia komponentteja ovat esimerkiksi html-, css-, php- ja kuva-tiedostot. Kun käyttäjä lataa sivuja Amazonin datakeskuksista, CloudFront pyrkii reitittämään liikenteen nopeimmalla mahdollisella tavalla. (Amazon CloudFront Documentation 2016.)

Amazon Simple Queue Service (Amazon SQS) on viestijonopalvelu, joka käsittelee tiedonsiirtoa eri pilvipalvelun komponenttien välillä. Viestijonot toimivat tilapäisinä tietovarastoina viesteille, joka odottavat prosessointia. (Amazon Simple Queue Service Documentation 2016.)

SimpleDB on indeksoitu tietovarasto, joka ei käsittele dataa relaatiotietokantojen tapaan. Se on optimoitu nopeisiin hakuaikoihin ja hyvään saatavuuteen, sillä tietovarasto on replikoitu eri datakeskusten välillä. (Amazon SimpleDB 2016.)

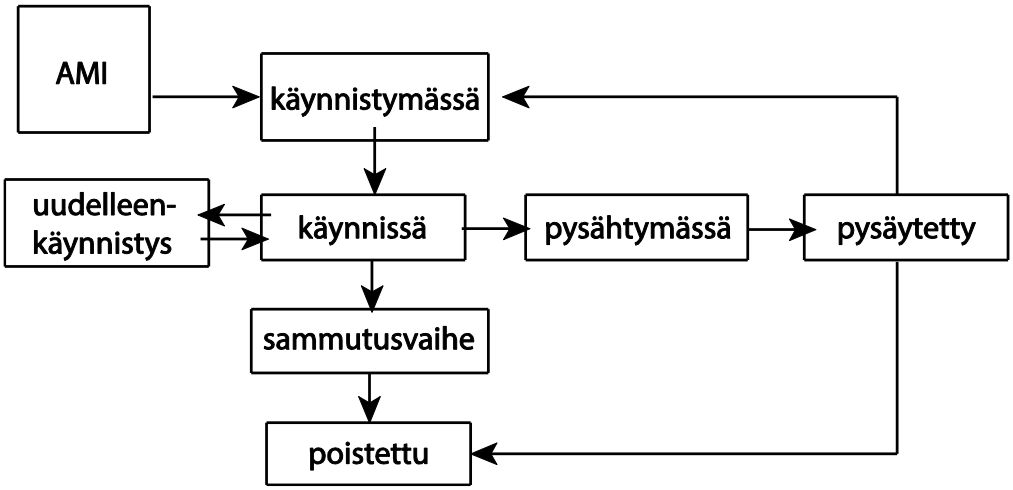
3.3.1 EC2

Amazonin Elastic Compute Cloud (EC2) mahdollistaa skaalautuvan tietojenkäsittelykapasiteetin osana Amazonin pilvipalvelua, jolloin asiakkaat voivat siirtää laskentatarvettaan joustavasti pilvipalveluun. EC2 sisältää virtuaalipalvelimia, joita voidaan ottaa käyttöön tarpeen mukaan. Niitä varten AWS:ssä on esiasennettuja palvelinkuvia, joista käytetään ter-

miä AMI (Amazon Machine Image). Amazonin pilvipalvelu tukee sekä Linux- että Windows-käyttöjärjestelmiin pohjautuvia virtuaalipalvelinympäristöjä. (Amazon EC2, User Guide for Microsoft Windows 2015, 1).

Virtuaalipalvelimet käyttävät julkisen avaimen salaustekniikkaa kirjautumisen yhteydessä. Tässä tekniikassa julkisella avaimella salataan dataa, tässä tapauksessa salasana, ja vastaanottaja käyttää omaa yksityistä avainta salasanan suojauksen purkamiseen. Julkinen ja yksityinen avain muodostavat avainparin, jossa käytetään 2048 bittistä SSH-2 RSA-algoritmia. EC2-virtuaalipalvelin tallentaa julkisen avaimen, ja palvelun käyttäjällä on hallussaan yksityinen avain. (Amazon EC2 Key Pairs).

Alla olevassa kuviossa 5 on esitetty virtuaalipalvelimen elinkaari, joka alkaa palvelinkuvan valinnalla ja käynnistämisellä (launch). Palvelimien tiloja voidaan valvoa Amazonin pilvipalvelun käyttäjäkohtaisesta hallintapaneelista. Käynnistymistilan (pending) jälkeen palvelimelle voi kirjautua sisään avainparia käyttämällä.



Kuvio 5. Virtuaalipalvelimen elinkaari

Toiminnassa olevan palvelimen voi tarvittaessa käynnistää uudelleen tai pysäyttää palvelimen käyttöjärjestelmätason komennoilla. Amazonin käyttäjäkohtaisessa hallintapaneelissa virtuaalipalvelimen voi poistaa ”terminate”-toiminnolla, jolloin myös kaikki palvelimessa oleva tieto häviää. (Amazon EC2, User Guide for Microsoft Windows 2015, 213.)

3.3.2 S3

Amazon S3 on palvelu, joka mahdollistaa tiedon tallentamisen ja haun ajasta ja paikasta riippumatta. S3-tietovarastot voivat sijaita fyysisesti eri Amazonin datakeskuksissa ympäri maailmaa. Näitä paikkoja ovat muun muassa EU:n alueella Irlanti ja Saksa, tarkemmin

Frankfurt am Main. Palvelun voi valita eri alueilta, ja tästä johtuen palvelun nopeus ja hinnoittelu vaihtelevat. S3-palvelussa yksittäistä tietosäiliötä kutsutaan nimellä ”bucket”, ja niitä voi olla useita yhdellä käyttäjällä. Arkkitehtuuri on suunniteltu tukemaan useita eri ohjelmointirajapintoja kuten Javan AWS SDK:ta, jonka avulla on mahdollista toteuttaa kaikki toiminnollisuus S3:n käytön suhteen. Se tarjoaa sekä SOAP- että REST-rajapinnat, tosin tällä hetkellä http-protokollaa käyttävä SOAP-rajapinta on vanhentunut ja on käytettävissä enää https:n kanssa. (Amazon S3: Developer Guide 2015, 2-4.)

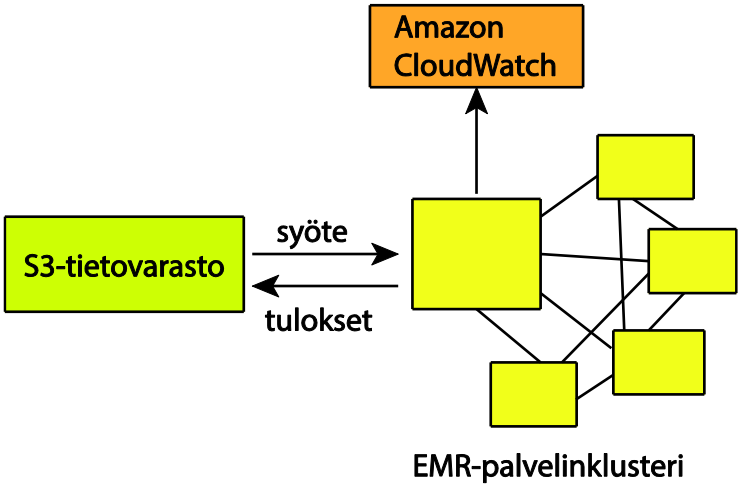
Palvelu tarjoaa ohjelmointirajapinnat, joiden avulla voidaan luoda ja hallinnoida tietosäiliöitä, joita on enimmillään 100 kappaletta kullekin pilvipalvelun käyttäjätunnukselle. Tietosäiliötä luotaessa on sille annettava nimi ja määritettävä missä datakeskuksessa se sijaitsee. Jokaiseen säiliöön on mahdollista tallentaa objekteja periaatteessa lukematon määrä. Säiliön voi luoda joko ohjelmallisesti käyttäen ohjelmointirajapintoja tai Amazonin hallintapaneelin kautta. (Amazon S3: Developer Guide 2015, 55.)

Kaikki S3-palvelussa sijaitsevat objektit ovat käyttäjäkohtaisia, ja vain objektien omistajalla on pääsy niihin. Tilin omistaja voi myös antaa muille pilvipalvelun käyttäjille oikeudet tallentaa objekteja tietosäiliöön. Tässä tapauksessa tallennuksen tekevä käyttäjä saa omistusoikeuden objekteihin. (Amazon S3: Developer Guide 2015, 280.)

Kun S3 vastaanottaa palvelupyynnön, esimerkiksi tietosäiliöön tai yksittäiseen objektiin kohdistuvan operaation, se tarkistaa ensin tilanteen mukaiset pääsyoikeudet, käyttäjäoikeudet sekä resurssikohtaiset oikeudet. Näihin perustuen palvelu päättää onko pyynnön lähettäjällä oikeus resurssien käyttöön. (Amazon S3: Developer Guide 2015, 292.)

3.3.3 Elastic MapReduce

Amazon Elastic MapReduce (EMR) on palvelu, joka on tarkoitettu ison datamäärän analysointiin ja prosessointiin. Tämän mahdollistaa tekniikka, jossa laskenta jaetaan klusterissa virtuaalipalvelimien välillä. Klusteria ohjataan Hadoop-viitekehyksellä, joka perustuu avoimen lähdekoodin projektiin. Hadoop käyttää hajautettua prosessointiarkkitehtuuria nimeltä MapReduce, jossa tehtävät on jaettu joukolle palvelimia. Virtuaalipalvelimista yksi on isäntäsolmukone ja loput ovat orjasolmukoneita, joiden lukumäärä on säädettävissä tarpeen mukaan myös laskennan aikana. Amazon on tehnyt MapReduce-ohjelmistoon omia muutoksia, ja tämä versio toimii EC2-instanssien kanssa. Instanssit ovat virtuaalisia Linux-palvelimia ja S3-tietovarastoa käytetään laskentadatan lähteenä ja tulosten tallentamiseen (kuvio 6). CloudWatch-palvelua käytetään klusterin toiminnan valvontaan ja mahdollisten virhetilanteiden raportointiin. (Amazon EMR, Developer Guide 2015.)



Kuvio 6. Elastic MapReduce -arkkitehtuuri (Amazon EMR, Developer Guide 2015, 2)

EMR-klusterin elinkaari alkaa varaamalla ensin palvelinresurssit käyttöön. Tässä vaiheessa klusterin tila on ”starting”. Tämän jälkeen suoritetaan käyttäjän määrittelemät esilatausohjelmat (bootstrapping) ja siirrytään ”running” tilaan. Esilatausohjelmat ovat skriptejä, joilla voidaan muuttaa käynnistyvän klusterin asetuksia.

Käynnissä olevassa klusterissa kaikki käyttäjän määritetyt laskentatehtävät (step) ajetaan peräkkäin. Mikäli klusteri on asetettu pysymään toiminnassa laskennan jälkeen, se siirtyy ”waiting”-tilaan ja jää odottamaan mahdollisesti tulevia uusia laskentapyyntöjä. Muussa tapauksessa klusteri sammuu automaattisesti viimeisen tehtävän jälkeen. Laskennan tulokset on tätä ennen siirretty S3-tietovarastoon talteen, ja kaikki klusterissa oleva data häviää. (Amazon, Life Cycle of a Cluster 2016.)

MapReducessa on kaksi vaihetta, joista ensimmäinen map-funktio tehdään jokaiselle datan osalle erikseen ja hajautetusti. Data on esitetty avain–arvo-pareina. Jokainen orjasolmukone lukee osan syötteen tietoalkioista ja tuottaa tuloslistan seuraavalla tavalla:

map-funktio: (avain1, arvo1) -> lista(avain2, arvo2)

Tämä map-funktion tulos tallennetaan väliaikaistiedostoihin joko välimuistiin tai levylle.

Reduce-funktio lukee nämä väliaikaistiedostot ja yhdistää tietoalkiot, joissa on sama avain. Tämän funktion tulos on lista arvoja:

reduce-funktio: (avain2, lista(arvo2)) -> lista(avain3, arvo3)
(Zinn, Bowers, Köhler & Ludäscher 2010.)

MapReduce-viitekehys voi vaihtoehtoisesti lajitella arvoja ennen kuin tulos siirretään reduce-funktiolle. Viitekehyksessä on mahdollista käyttää myös combiner-funktiota map- ja reduce-vaiheiden välissä, jolla voidaan tehostaa map-funktion toimintaa dataa ryhmittämällä. Reduce-funktioon liittyy shuffle- ja sort-vaiheet, joiden toimintaan voidaan vaikuttaa ohjelmallisesti. Esimerkiksi lajitteluun on mahdollista tehdä räätälöity algoritmi. (Miner & Shook 2013, 6.)

4 Asiakassovellusten teknologioita

Tässä luvussa esitetään suosittuja moderneja tekniikoita, joita käytetään pilvipalvelun asiakassovelluksissa. Näillä tekniikoilla pyritään erottamaan palvelimen ja asiakkaan toiminnollisuus toisistaan erillisiksi kokonaisuuksiksi. Luvussa käsitellään myös palvelimen ja asiakkaan välistä rajapintaa ja sen piirteitä.

4.1 RESTful Web Services

REST on Roy Fieldingin 2000-luvun vaihteessa kehittämä ohjelmistoarkkitehtuurimalli, jonka tavoitteena on säilyttää yhteentoimivuus hajautettujen järjestelmien välillä siten, että osien sisäisiä toimintoja voidaan kehittää toisistaan riippumatta. REST ei ota kantaa eri komponenttien toteutukseen tai protokollien syntaksiin vaan keskittyy eri komponenttien rooleihin ja niiden vuorovaikutusta koskeviin rajoituksiin.(RestAPI tutorial.)

Fielding esittelee väitöskirjassaan kuusi rajoitusta, joista ensimmäinen koskee asiakas-palvelin-arkkitehtuurimallia. Erottamalla käyttöliittymä ja tiedon varastointi toisistaan parannetaan käyttöliittymän siirrettävyyttä eri järjestelmiin. Tästä seuraa myös se, että palvelinkomponentteja voidaan yksinkertaistaa, joka taas lisää järjestelmän skaalautuvuutta. Käyttöliittymää ja palvelinkomponentteja voidaan kehittää omina projekteinaan toisistaan erillään.(Fielding 2000, 78.)

Toinen rajoitus määrittelee, että asiakas-palvelin-yhteyden on oltava tilaton, eli palvelimella ei ylläpidetä tietoja yksittäisistä istunnoista, vaan asiakkaan pyyntö palvelimelle tulee sisältää kaikki tarvittava tieto vastauksen tuottamiseen. Istunnon tila voidaan säilyttää vain asiakkaan puolella. Tämä parantaa skaalautuvuutta, sillä palvelimen ei tarvitse tallentaa tietoja eri pyyntöjen välillä. Palvelimen tilattomuus myös helpottaa vikatilanteista toipumisessa, joka osaltaan lisää luotettavuutta.(Fielding 2000, 78.)

Verkon tehokkuutta voidaan parantaa, kun asiakaspuolella käytetään välimuistia. Tämä edellyttää, että palvelimen vastauksessa asiakkaalle on ilmoitettava, voidaanko datan kanssa käyttää välimuistia vai ei. Jos välimuistin käyttö on sallittua, saa asiakas oikeuden käyttää uudelleen aiemmin saadun vastauksen dataa uuden pyynnön yhteydessä, ja näin vähentäen liikennettä asiakkaan ja palvelimen välillä. (Fielding 2000, 79.)

Yhdenmukainen rajapinta on keskeinen ominaisuus REST-arkkitehtuurimallissa, ja sen käyttö komponenttien välillä yksinkertaistaa järjestelmän arkkitehtuuria. Haittapuolena tästä on se, että yhdenmukainen rajapinta heikentää tehokkuutta, sillä se on suunniteltu

tehokkaaksi hypermedian siirtoon, eikä yksittäisen sovelluksen erityisiä tarpeita tiedonsiirrolle välttämättä tällöin oteta huomioon. (Fielding 2000, 81.)

Viides rajoitus tässä arkkitehtuurimallissa on kerroksellinen järjestelmä, joka tarkoittaa sitä, että asiakkaan ja kohdepalvelimen välissä voi olla useita välitasoja, jotka siirtävät tietoa päätepisteiden välillä. Tiedonsiirron välitasot voivat parantaa järjestelmän skaalautuvuutta, koska ne mahdollistavat palveluiden kuormantasauksen. Tämä johtaa toisaalta siihen, että asiakas ei kykene ”näkemään” välitasojen yli, minkä palvelimen kanssa se on lopulta vuorovaikutuksessa. (Fielding 2000, 81.)

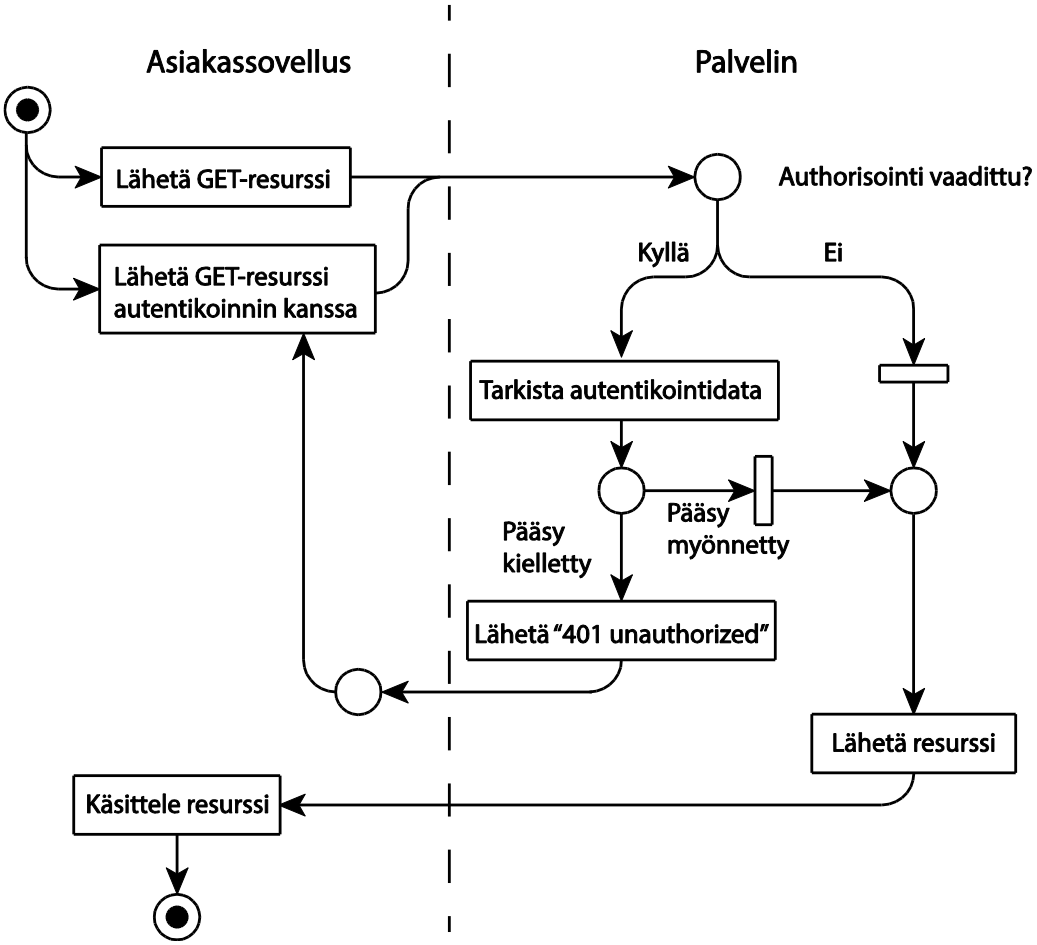
Viimeinen rajoitus on koodin lähettäminen pyynnöstä (code-on-demand), jolla tarkoitetaan sitä, että asiakkaan toiminnollisuutta voidaan muuttaa lähettämällä sille palvelimelta ajettavaa koodia. Tätä tekniikkaa on käytetty verkkosivuilla esimerkiksi Java-sovelmissa ja verkkoselaimessa JavaScript-koodia ajettaessa. Tämä on ainut optionaalinen rajoitus REST-arkkitehtuurimallissa. (Fielding 2000, 84.)

4.2 Basic-autentikointi

Usein verkkopalveluihin liittyvä tiedonsiirto on yksityistä, ja tästä johtuen internetin yli siirrettävä informaatio on tarpeen salata muilta käyttäjiltä. Tällöin palveluihin tarvitaan pääsynhallintamekanismi ja käyttäjän tunnistus on tähän ratkaisu. REST-rajapinnassa on kyse http-protokollaa käyttävästä liikenteestä ja Basic-autentikointi on kehitetty tätä protokollaa varten. (Gourley & Totty 2002, 277.)

Basic-autentikointi on yksinkertaisin tunnistamismenetelmä. Vaikka salasanat on tallennettu salattuina palvelimella, asiakaspuolen sovellus lähettää käyttäjätunnuksen ja salasanan ilman salausta tässä menetelmässä. Tästä seuraa, että kuka tahansa voi saada haltuun ne, varsinkin kun ne lähetetään aina jokaisessa http-pyynnössä. Tämä lisäksi vastaus http-pyyntöön lähetään myös salaamattomana. Ratkaisu tähän ongelmaan on TSL/SSL-protokollan käyttö basic-autentikoinnin yhteydessä, jolloin koko http-liikenne salataan. (Gröne, Knöpfel, Kugel & Schmidt 2008, 18–20.)

Kuviossa 7 on esitetty basic-autentikoinnin kulku asiakassovelluksen ja palvelimen välillä. Mikäli palvelin vaatii käyttäjän tunnistusta, tarkistetaan http-pyynnön otsakkeissa mahdollisesti tuleva käyttäjätunnus- ja salasanapari. Jos niitä ei löydy, palvelin lähettää vastauksen ”401 unauthorized”, ja pyytää tunnistautumaan. Asiakassovellus voi tämän jälkeen yrittää uudelleen lähettää pyynnön tunnistautumistietojen kanssa. Mikäli palvelin hyväksyy pyynnön tämän perusteella, lähettää se asiakassovellukselle pyydetyn resurssin.



Kuvio 7. Basic-autentikoinnin kulku, asiakkaana sovellus (Gröne ym. 2008, 19.)

4.3 JSON- ja XML-dokumentit viestinvälityksessä

JSON (JavaScript Object Notation) on yksinkertainen tiedonvälitysformaatti ja sitä pystytään tuottamaan ja lukemaan myös ilman tietokoneita. JSON pohjautuu JavaScript-ohjelmointikielen Standardi ECMA-262:n kolmanteen versioon vuodelta 1999. Se on tekstimuotoista ja täysin ohjelmointikielistä riippumaton, tosin pohjautuu käytäntöihin, jotka ovat tuttuja C-kieleen pohjautuvissa kielissä, kuten Java, C#, Perl ja Python. Nämä ominaisuudet tekevät JSON-formaatista ihanteellisen viestinvälityskielen. (Introducing JSON.)

JSON käyttää neljää alkeistyyppiä, jotka ovat merkkijono, numero, Boolean-arvo ja null. Tämän lisäksi on käytössä rakenteiset tyypit, joita ovat objektit ja taulukot. (Crockford 2006.)

XML (Extensible Markup Language) on joustava tekstipohjainen formaatti, joka on johdettu SGML-kielestä (ISO 8879). Alun perin se on kehitetty vastaamaan haasteisiin, joita

laajat elektroniset julkaisut ovat tuoneet. XML myös on tärkeässä osassa, kun dataa siirretään internetin välityksellä. (W3.org: Extensible Markup Language.)

XML-kielen kehittämisessä on otettu huomioon muun muassa seuraavia seikkoja:

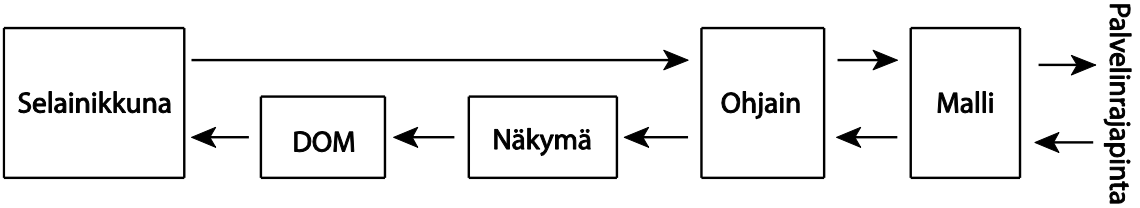
1. XML-dokumentti tulee olla sellaisenaan käytettävissä internetin siirroissa.
2. XML:n tulee tukea laajaa joukkoa eri sovelluksia.
3. XML:n tulee olla yhteensopiva SGML:n kanssa.
4. XML-dokumenttien tuottaminen sovelluksissa tulee olla helppoa.
5. XML:n vaihtoehtoisia ominaisuuksia tulisi olla mahdollisimman vähän.
6. XML-dokumentit tulee olla ihmisten luettavissa ja kohtuullisen selkeitä.

(W3.org: Extensible Markup Language.)

4.4 AngularJS

AngularJS on asiakaspuolen (client-side) JavaScript-viitekehys, jonka kehittivät Misko Hevery ja Adam Abrons vuonna 2009. Taustalla oli käsitys, että deklaratiivinen ohjelmointi on paljon parempi vaihtoehto käyttöliittymien suunnitteluun, kun taas imperatiivinen ohjelmointi soveltuu hyvin sovelluksen toimintalogiikan toteuttamiseen. Tähän tavoitteeseen on pyritty laajentamalla perinteistä verkkosivuilla käytettävää HTML-kieltä AngularJS:n toiminnollisuudella. AngularJS pohjautuu alun perin Malli-Näkymä-Ohjain-arkkitehtuuriin (Model-View-Controller, MVC), mutta nykyään viitekehyksen kehittäjät ovat käyttäneet myös käsitettä MVW, Model-View-Whatever, missä W tarkoittaa ”mikä vain toimii”. Tyypillinen AngularJS-viitekehystä käyttävä sovellus sisältää näkymän, mallin ja ohjaimen, mutta tärkeitä komponentteja ovat myös direktiivit, palvelut ja suodattimet (filters). (Branas, 8–9.)

MVC-arkkitehtuurin lähtökohta on, että sovelluksen käyttämä data, sen esittäminen ja sovelluksen toimintalogiikka erotetaan toisistaan omiksi osasiksi. AngularJS-sovelluksessa näkymä on Document Object Model (DOM), ohjaimia edustavat JavaScript-luokat, ja data on varastoitu JavaScript-muuttujiin. (Green & Seshadri, 3.) Kuvio 8 esittää AngularJS:n mukaisen MVC-arkkitehtuurin.



Kuvio 8. AngularJS:n toteutus MVC-arkkitehtuurista (Freeman 2014, 48)

Toimintalogiikka, joka käsittelee datan tallennusta ja hakemista, on osa mallia. Logiikka, joka hallitsee datan esittämisen käyttäjälle, on osa näkymää. Ohjain on näiden kahden

osan välillä, ja yhdistää ne toisiinsa. Ohjain reagoi käyttäjän toimintaan päivittäen dataa mallissa ja tarjoamalla se näkymänä käyttäjälle. (Freeman 2014, 22.)

Direktiivit ovat AngularJS:lle ominainen piirre, joka mahdollistaa HTML:n toiminnollisuuden laajentamisen. AngularJS-kirjaston mukana tulee kokoelma valmiiksi määriteltyjä direktiivejä, jotka alkavat ng-etuliitteellä. Näistä ensimmäinen on ng-app, joka aloittaa AngularJS-sovelluksen, eli määrittelee juurielementin HTML-dokumentissa. Se automaattisesti alustaa tai esilataa sovelluksen, kun AngularJS:n sisältämä verkkosivu ladataan. Tätä direktiiviä käytetään myös lataamaan AngularJS:n moduuleja. Direktiivi ng-init alustaa sovelluksen datan, ja sitä käytetään arvojen sijoittamiseen muuttujiin, joita sovellus käyttää. Ng-model-direktiivi kytkee toisiinsa mallin ja AngularJS:ssä käytettävän muuttujan. Direktiivi ng-repeat iteroi JavaScriptin tietorakenteen, yleensä taulukon, ja kutakin tietorakenteen alkiota kohden HTML-elementti toistetaan näkymässä. (AngularJS - Directives 2016.)

4.5 D3.js

D3.js on JavaScript-kirjasto, jonka ensimmäisen version kehitti Mike Bostock yhteistyössä Jeffery Heerin ja Vadim Ogievetskyn kanssa Stanfordin yliopistossa. Kirjasto on vapaasti saatavilla oleva laajennus JavaScriptiin ja sen perusajatuksena on tarjota keinot datan liittämiseen DOM-malliin (Document Object Model) sekä muokata web-sivua perustuen tähän dataan ja tuottaa vuorovaikutteisia graafisia esityksiä selainympäristössä. D3-kirjasto tukeutuu yleisiin web-standardeihin kuten HTML, SVG ja CSS. (Bostock, Ogievetsky & Heer 2011.)

Skaalautuvaan vektorigrafiikkaan perustuva SVG on tekstipohjainen formaatti, ja siinä jokainen kuva on määritelty HTML-syntaksin tapaisesti, kuten alla olevasta esimerkistä voidaan nähdä. Ympyräelementti määritellään esimerkiksi seuraavasti:

```
<svg width=”50” height=”50”>
<circle cx=”25” cy=”25” r=22” fill=”blue” stroke=”grey stroke-width=”2”/>
</svg>
```

SVG-formaatilla tuotettuja graafisia elementtejä käytetään muun muassa hyödyksi datan visualisoinnissa, ja niiden avulla voidaan piirtää esimerkiksi ympyröitä, joita voidaan käyttää tilastollisen datan esittämismuodoissa kuten piirakkakuvioissa. (Murray, 49–50.)

Datan visualisointi on prosessi, jossa kytketään dynaamisesti tietoa dokumentin, usein html-sivu, visuaalisiin elementteihin. D3-kirjaston avulla tämä tehdään valitsemalla DOM-

elementit ja sen jälkeen lisäämällä niihin attribuutteja. Valinta tuottaa taulukon (array) elementtejä kohteena olevasta dokumentista. D3 käyttää CSS3-muotoilukieltä elementtien valinnassa. Valinta voi perustua esimerkiksi tunnisteeseen (tag), luokkaan tai attribuuttiin. Valinnoissa voidaan käyttää myös Boolen-logiikan operaattoreita: OR ja AND. Kirjasto tarjoaa kaksi metodia valintaan: d3.select ja d3.selectAll, joista select-metodi palauttaa ensimmäisen elementin, joka täyttää valinnan ehdot, ja selectAll palauttaa taulukon elementtejä. (Bostock 2016.)

Valittuihin elementteihin tehdään tarvittavat muutokset tämän jälkeen. Elementtien attribuuttien arvoja voidaan muuttaa attr-operaattorilla ja luokka-attribuuttien arvoja voidaan muuttaa classed-operaattorilla. Append-metodilla voidaan lisätä uusi elementti valitun elementin osaksi, ja tämän jälkeen palautetaan alkuperäinen elementti sisältäen tehdyn uuden elementin. Näin voidaan esimerkiksi lisätä tilastografiikkaa verkkosivulle. Elementtejä on myös mahdollista päivittää remove- ja insert-metodeja käyttäen. (Bostock 2016.)

5 Demon suunnittelu ja toteutus

Tässä opinnäytetyössä toteutettiin demo, jossa EHR-resursseja viedään tietolähteestä suojattua verkkoyhteyttä käyttäen pilvipalveluun tilastollista laskentaa varten, ja tämän jälkeen tilastotiedot ovat haettavissa pilvipalvelusta web-selaimelle ja esitettävissä sekä numeerisessa että graafisessa muodossa. EHR-data pyrittiin käsittelemään FHIR-standardin määrittelemässä muodossa pilvipalvelun tietovarastoon asti. Testidata koostui verensokeriarvoista, joiden vaihteluväli oli 4 – 16 mmol/l. Nämä arvot simuloivat sokerirasituskokeen mittauksia. Jakauma jaettiin kolmeen kategoriaan. Niiden rajat asetettiin Käypähoitosuosituksen perusteella siten, että mittausten ensimmäinen kategoria on alle 7,8 mmol/l (normaali), toinen 7,8 – 11 mmol/l (heikentynyt glukoosin sieto) ja kolmas yli 11 mmol/l (diabetes).

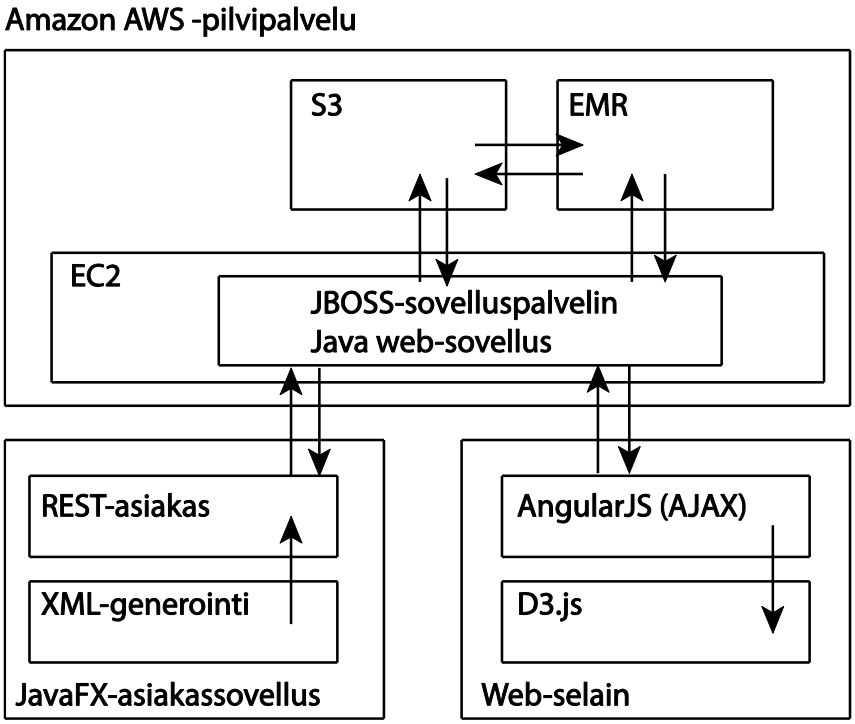
Amazonin pilvipalvelua käytettiin demon alustana, koska siitä oli jo aiempaa kokemusta hyödynnettäväksi tässä opinnäytetyössä. Sovellukset ohjelmoitiin Javalla 7:llä käyttäen Netbeans 8.1 sovelluskehitintä, johon lisäksi oli asennettu Maven ja tarvittavat Java-kirjastot kuten Amazonin oma AWS SDK for Java. Muita demossa käytettäviä teknologioita olivat XML/JSON, HTML5, AngularJS, D3.js, RESTEasy ja MapReduce.

Java-sovelluspalvelimena käytettiin Jboss 6.3 EAP:ta, jota varten luotiin Amazonin pilvipalveluun EC2-virtuaalipalvelin. Palvelimen käyttöjärjestelmäksi asetettiin Windows Server 2012 R2 palvelimen luomisen yhteydessä. Jboss 6.3 EAP asennettiin lataamalla asennuspaketti Jbossin web-sivuilta ja purkamalla se omaan alihakemistoon C-asemalla. HTTPS-yhteyttä varten luotiin itse allekirjoitettu sertifikaatti ja tehtiin asetukset sovelluspalvelimen asetustiedostoihin. Tästä on kuvaus tarkemmin luvussa 5.2. ”HTTPS-yhteyden asettaminen”.

5.1 Arkkitehtuuri

Demoa varten tuotettiin useita sovelluksia:
a) Asiakassovellus, joka tuottaa testidataa ja lähettää sen pilvipalvelulle suojattua verkkoyhteyttä pitkin käyttäen Basic-autentikointia. Tämä toteutettiin hyödyntämällä JavaFX-teknologian tarjoamia työkaluja. Sovelluksessa on omat komponentit XML-muotoisen testidatan tuottamiseen ja palvelimen REST-rajapinnan käyttämiseen.
b) Web-sovellus, joka vastaanottaa pilveen tietoa ja siirtää sen eteenpäin S3-tietovarastoon laskentaa varten.

c) Sovellus tilastolaskentaan Amazonin EMR-palvelussa, joka perustuun Hadoop-viitekehykseen.
d) Web-sovellus, joka tuottaa HTML-sivun laskennan seurantaan ja tulosten tarkasteluun. Tällä sivulla käytetään AngularJS-kirjastoa eri toimintojen toteuttamiseen. Laskentatulos esitetään taulukkona ja myös piirakkakuviona D3.js–kirjaston funktioiden avulla. Alla olevassa kuviossa 9 on esitetty eri ohjelmistokomponentit ja niiden välinen yhteys



Kuvio 9. Ohjelmistokomponentit ja niiden väliset yhteydet

5.2 HTTPS-yhteyden asettaminen palvelimella

Sertifikaatin luomiseen käytettiin Oraclen työkalua ”keytool”, joka tulee lisäohjelmana Java SDK:n asennuksen mukana. Keytool on komentorivipohjainen ohjelma, ja seuraavalla komennolla luotiin tiedosto (key store), joka sisältää tarvittavan avainparin:

.

keytool -genkeypair -alias fhirdemo -keyalg RSA -keysize 1024 -validity 365 -keystore amazon.keystore.jks -keypass keksi_tähän_salasana -storepass keksi_tähän_salasana

Ohjelman aikana kysyttiin myös sertifikaattiin asetettavat nimi- ja organisaatiotiedot. Tiedosto kopioitiin sovelluspalvelimen asetushakemistoon: C:\JBoss\standalone\configuration\ ja samassa hakemistossa olevaa asetustiedostoa standalone.xml muutettiin. Elementtien <connector> sisällöt kirjoitettiin uudelleen seuraavalla tavalla:

```
<connector name="http" socket-binding="http" scheme="http" protocol="HTTP/1.1"/>
<connector name="HTTPS" socket-binding="https" scheme="https" protocol="HTTP/1.1"
secure="true">
<ssl name="https" protocol="TLSv1" cipher-suite="RSA" certificate-key-file=
"${jboss.server.config.dir}/amazon.keystore.jks" password= "keksi_tähän_salasana" key-
alias="fhirdemo"/>
</connector>
```

Tämän jälkeen sovelluspalvelin käynnistettiin ja testattiin yhteyttä käyttäen samassa tiedostossa määritettyä porttia 8443 https-protokollalle, ja havaittiin että web-selain avasi suojatun verkkoyhteyden.

5.3 Testidatan tuottaminen

EHR-datan tuottamiseksi ja lähettämiseksi pilvipalveluun tehtiin Java-sovellus, jossa käytettiin JavaFX-teknologiaa. Käytettävä testidata koostui XML-dokumenteista, ja niiden tuottaminen toteutettiin org.w3c.dom-luokkakirjastoa käyttämällä, ja se tarjoaa rajapinnat DOM-mallin käyttöön. Document-objekti kuvaa XML-dokumenttia, ja on juurena sen puumaiselle rakenteelle sekä mahdollistaa pääsyn dokumentin elementteihin. Elementti-objektille voidaan määrittää nimi ja liittää ylemmän tason kanssa yhteen lapsielementiksi. Objektille voidaan antaa myös attribuutteja. Seuraavassa on esimerkkikoodi yhden elementin tuottamisesta ja sen liittämisestä juurielementtiin.

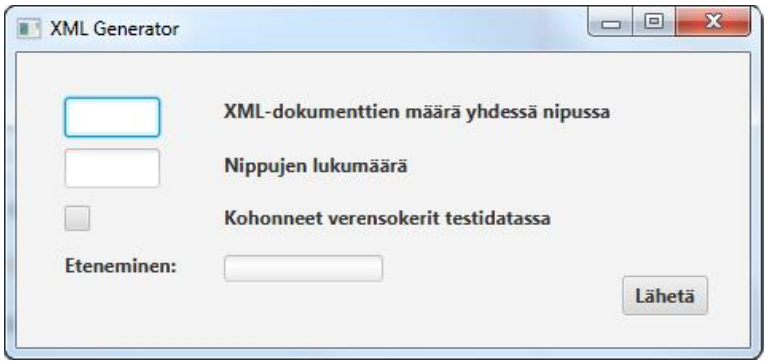
```
Document doc = docBuilder.newDocument();
Element rootElement = doc.createElement("Observation");
rootElement.setAttribute("xmlns", "http://hl7.org/fhir");
doc.appendChild(rootElement);
```

XML-dokumentit serialisoitiin String-olioiksi ennen palvelimelle lähettämistä. Tämä tehtiin käyttämällä org.w3c.dom-pakkauksen tarjoamia luokkia. Seuraavassa on esimerkkikoodi xml-dokumentin muuntamisesta:

```
DOMImplementationLS domImpl = (DOMImplementationLS) xmlDoc.getImplementation();
LSSerializer lss = domImpl.createLSSerializer();
DOMConfiguration domConfig = lss.getDomConfig();
boolean canSetParameter = domConfig.canSetParameter("format-pretty-print", true);
if (canSetParameter) {
        domConfig.setParameter("format-pretty-print", true);
}
String result = lss.writeToString(xmlDoc);
```

5.4 EHR-datan lähettäminen pilvipalveluun

Sovelluksessa oli yksinkertainen graafinen käyttöliittymä, jonka avulla lähetys voitiin käynnistää niin monta kertaa kun oli tarvetta (kuva 1). Siirron etenemistä voitiin tarkkailla Eteneminen-palkin avulla. Testiaineiston verensokeriarvojen jakaumaa voitiin muuttaa, jotta demossa nähtäisiin muutos tuloksissa eri laskentakertojen välillä. Muutos tehtiin siirtämällä normaalijakauman moodia kaksi yksikköä (mmol/l) ylöspäin, joka näin simuloi testidatan kohonneita verensokeriarvoja.



Kuva 1. Asiakassovelluksen käyttöliittymä

Jboss-sovelluspalvelimen IP-osoite oli demossa asetettava erikseen, koska Amazonin pilvessä olevalla EC2-virtuaalipalvelimella ei ollut kiinteää IP-osoitetta. Sellainen olisi ollut mahdollista hankkia, mutta sitä ei tehty tätä demoa varten kustannussyistä. Osoitetta varten tehtiin tekstimuotoinen asetustiedosto, jonka asiakassovellus luki käynnistyksen yhteydessä. Tekstitiedostoa oli helppo muuttaa sen mukaan, miten palvelimen IP-osoite vaihtui.

Tietojen lähettäminen tapahtui käyttäen suojattua verkkoyhteyttä TSL/SSL-tekniikalla. Tätä varten oli luotava asiakassovellusta varten avaintiedosto, jossa julkista avainta säilytettiin (key store).

Luvussa 5.2. luodusta avaintiedostosta vietiin julkinen avain asiakassovellusta varten komennolla:

keytool -export -keystore amazon.keystore.jks -alias fhirdemo -file sertifikaattifile.cer

Sertifikaatti-tiedostosta tuotiin uuteen avaintiedostoon julkinen avain asiakassovellusta varten. Aliaksen ja salasanan ei tarvitse olla samat kuin palvelinpuolen vastaavat parametrit:

keytool -import -alias amazonAvain -file sertifikaattifile.cer -keystore JavalleKeystore.bin -storepass keksi_tähän_salasana

Jotta Java-sovellus pystyisi käyttämään luotua avaintiedostoa, sille oli määritettävä sijainti ja salasana System-luokan staattisella setProperty-metodilla:

System.setProperty("javax.net.ssl.trustStore", "c:/Java/JavalleKeyStore.bin");
System.setProperty("javax.net.ssl.trustStorePassword", ”tähän_salasana");

Asiakassovelluksen tiedonsiirtoluokassa hyödynnettiin Javan javax.ws.rs-pakkausta, joka tarjoaa luokat ja annotaatiot käytettäväksi yhteyden luomiseen palvelimelle. Autentikointimenetelmänä käytettiin demossa Basic-autentikointia, joka lähettää enkoodatun tunnuksen ja salasanan http-pyynnön otsakkeessa.

Yhteyttä varten luotiin javax.ws.rs.client.Client –luokasta olio, jolle annettiin muodostimen parametrinä ClientRequestFilter-olio, joka ennen yhteyden muodostamista muokkaa http-pyyntöä. Tässä oliossa ylikirjoitetaan filter-metodi, jossa tunnus ja salasana ensin enkoodataan käyttäen Base64-skeemaa, ja sen jälkeen liitetään osaksi http-pyynnön otsikkoa ”Authorization”-elementtiin.

Tiedon siirrossa käytettiin PUT-metodia, joka on määritelty FHIR-standardissa. Jokaiselle resurssille, eli tässä tapauksessa XML-dokumentille annettiin yksilöivä tunnus (id), jonka perusteella URI määräytyi lähettämisessä. Koska tässä demossa ei haeta tai vastaanoteta FHIR-standardin määrittelemiä dokumentteja sovelluspalvelimelta, näitä id-parametreja ei tallennettu missään muodossa asiakassovelluksen yhteyteen.

5.5 Web-sovellus tiedon vastaanottamiseen ja siirtoon S3:een

Web-sovellus vastaanottaa XML-dokumentteja asiakassovellukselta ja siirtää ne välittömästi Amazonin S3-tietovarastoon. Sovelluksessa käytettiin REST-rajapintoja, ja annotaatioilla määritettiin kunkin resurssin nimi URI-polussa. Tässä demossa toteutettiin vain yksi resurssi: tiedon vastaanotto. Sovellus perii javax.ws.rs.core-pakkauksesta Application-luokan ja ylikirjoittaa sen getClasses-metodin. Tässä metodissa rekisteröidään ne luokat, joiden resursseja REST-rajapinta käyttää. Seuraavassa on toteutus metodista:

```
@Override
public Set<Class<?>> getClasses() {
   final Set<Class<?>> classes = new HashSet();
```

```
        classes.add(S3Resource.class);
        classes.add(S3AuthFilter.class);
        return classes;
    }
```

Luokissa olevien annotaatioiden avulla sovellus löytää resurssit, joita kutsutaan URI:n perusteella. Autentikoinnin toteuttamiksi luotiin S3AuthFilter-luokka, joka on toiminnaltaan samankaltainen kuin asiakassovelluksen ClientRequestFilter-luokan toteutus. S3AuthFilter-luokka toteuttaa ContainerRequestFilter-luokan ja lähemmin sen filter-metodin. Tässä metodissa http-pyynnön otsake tutkitaan ja siitä haetaan Authorization-elementti. Elementin sisällöstä puretaan merkkijono, joka sisältää käyttäjätunnuksen ja salasanan. Tässä demossa ei toteutettu laajempaa käyttäjähallintaa, joten vain yksi tunnus-salasana-pari oli käytössä.

Asiakassovellus kutsuu web-sovelluksen URI:ia /service/s3/upload, johon liitetään viimeiseksi elementiksi dokumentin ID. Annotaatioilla määritettiin tietotyyppi, mitä resurssi ottaa vastaan (Consumes) sekä resurssin vastauksen tyyppi (Produces). Seuraavassa on esitetty käytetyt annotoinnit:

```
@PUT
@Path("/upload/{id}")
@AuthMust
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.TEXT_HTML)
        public String saveXML(String data,@PathParam("id") String id) {
        ... metodin sisältö …
        }
```

Siirtoa varten määriteltiin nimetty annotaatio @AuthMust, jolloin edellä olevan resurssin käyttäminen vaatii autentikointia. Tätä annotaatiomäärittelyä varten luotiin oma rajapintaluokka.

XML-dokumentin siirto web-sovelluksesta S3:een toteutettiin Amazonin AWS SDK:n sisältämän com.amazonaws.services.s3-pakkauksen luokkien avulla. AmazonS3Client-luokasta luotiin olio, jolle määritettiin palvelun käyttämiseen vaaditut Amazonin AWS:n käyttäjäkohtaiset avaimet: accessKey ja secretKey. PutObjectRequest-oliolle määritettiin tietosäiliön nimi (bucket), dokumentin id sekä InputStream-olio datan siirtoa varten. Tämä

olio annettiin parametrina S3Clientin putObject-metodille, jota kutsuttaessa siirto suoritettiin.

5.6 Tilastolaskenta MapReducella

MapReducea varten luotiin jar-tiedosto, jossa on Javalla ohjelmoidut algoritmit tiedon käsittelyä ja laskentaa varten. Tähän tiedostoon toteutettiin yksi luokka, joka määritteli laskennassa käytettävät luokat sekä input- ja output-formaatit. Laskentaa varten oli määriteltävä myös S3-tietovaraston hakemistot, joista laskettava data haetaan sekä tulokset kirjoitetaan.

Seuraavassa kuviossa on esitetty käytettävien luokkien rekisteröinti päämetodissa:

```
config.setOutputKeyClass(Text.class);
config.setOutputValueClass(IntWritable.class);
config.setMapperClass(JakaumaMapper.class);
config.setReducerClass(JakaumaReducer.class);
config.setInputFormat(TextInputFormat.class);
config.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(config, new Path(args[0]));
FileOutputFormat.setOutputPath(config, new Path(args[1]));
```

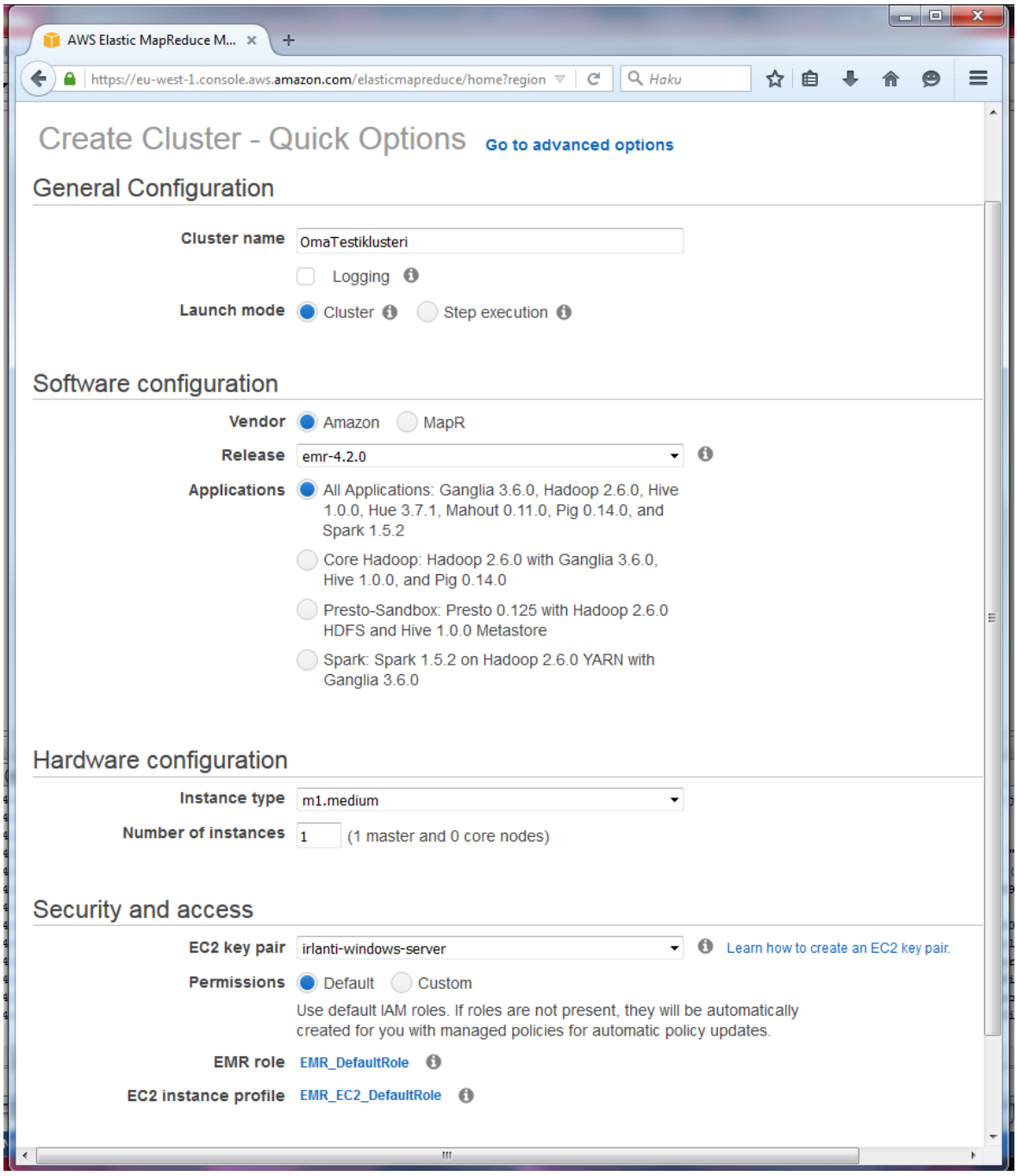
Kuvio 10. MapReducen luokkamäärittelyt

Map- ja Reduce-algoritmit ovat omissa sisäluokissaan, ja molemmat perivät org.apache.hadoop-pakkauksesta MapReduceBase-luokan. Map-algoritmi toteuttaa myös Mapper-luokan ja Reduce-algoritmi Reducer-luokan edellä mainitusta pakkauksesta.

Map-algoritmissa luetaan XML-dokumentista glukoosipitoisuuden arvo ja luokitellaan se valmiiden kategorioiden mukaan. Reduce-funktiossa nämä lukumäärät laskettiin yhteen ja kolmen arvokategorian koot tallennettiin tulostiedostoihin, jotka sijaitsivat S3-tietosäiliön ennalta määrätyssä hakemistossa. Combiner-funktiota ei käytetty laskennassa.

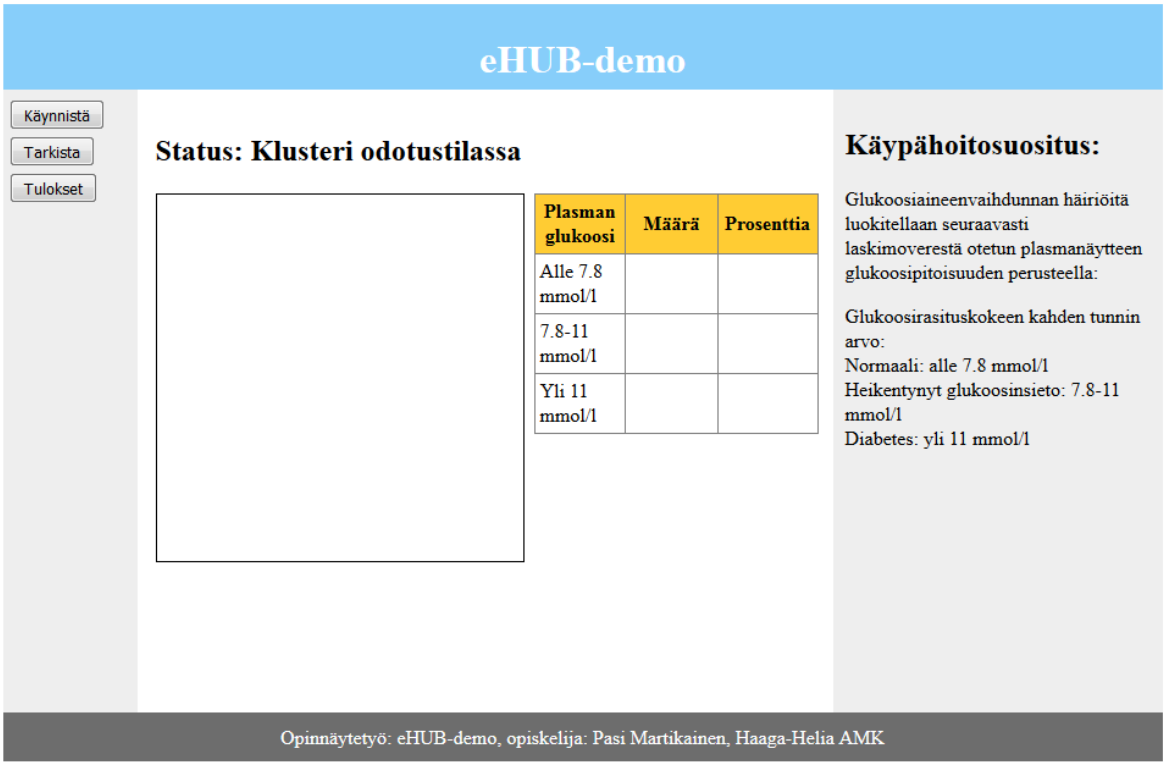
5.7 Laskennan käynnistäminen ja tuotetun tiedon esittäminen

Web-sovellus tuotti HTML5-sivun, jossa käytettiin CSS-tyylitiedostoa sivun ulkoasun määrittelyyn. Sivulta voitiin käynnistää laskenta, ja käydä hakemassa laskennan tulos sen valmistuttua. Sovellus ei itsenäisesti käynnistä pilvipalvelun klusteria vaan se on käynnistettävä manuaalisesti Amazonin pilvipalvelun hallintapaneelista. Kuvassa 2 on esitetty EMR-palvelun käyttöliittymä, jonka avulla klusterin voi luoda ja käynnistää laskentaa varten.



Kuva 2. Klusteri asetetaan käyttökuntoon hallintasivun kautta

Palvelimella on URI, jota kutsumalla laskennan voi käynnistää. Laskennan edistymistä voidaan seurata omalla toiminnolla. Kuvassa 3 on esitetty sivun käyttöliittymä.



Kuva 3. Laskennan käyttöliittymä

Web-sovellus määrittelee pilvilaskennalle input- ja output-polut sekä luokan, josta laskenta käynnistetään. Nämä määrittelyt web-sovellus hakee erillisestä xml-tiedostosta, joka on esitetty kuviossa 11. Tiedostossa on tallennettu myös pilvipalvelun avaimet sekä tieto, mitä Amazonin datakeskusta käytetään (amazon_region). Tämä demo käyttää palvelimia, jotka sijaitsevat fyysisesti Irlannissa.

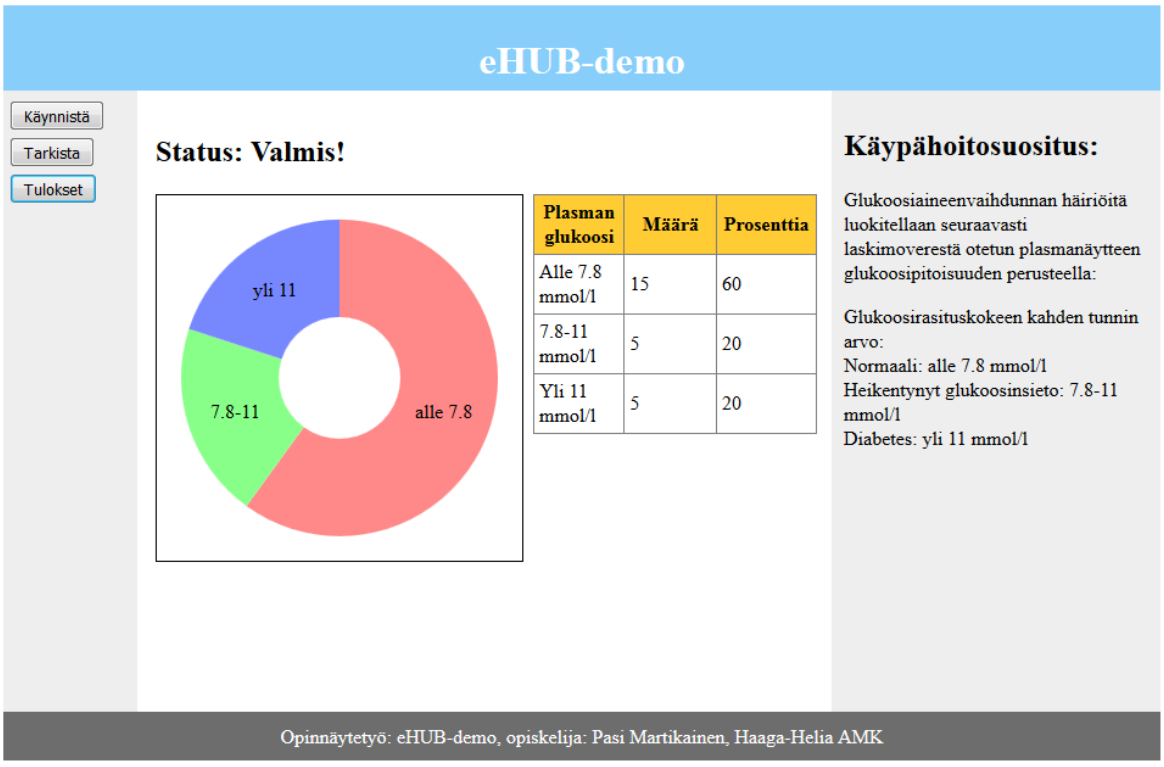
```
<?xml version="1.0" encoding="UTF-8"?>
<setup>
        <bucket>elasticbeanstalk-eu-west-1-646723725800</bucket>
        <access_key>                    </access_key>
        <secret_key>                                    </secret_key>
        <step_prefix>ehub-</step_prefix>
        <jar_name>JakaumaTesti.jar</jar_name>
        <main_class>jakaumatesti.JakaumaTesti</main_class>
        <job_folder>jobs</job_folder>
        <input_folder>input</input_folder>
        <output_folder>output</output_folder>
        <amazon_region_s3>eu-west-1</amazon_region_s3>
        <amazon_region_emr>EU_WEST_1</amazon_region_emr>
</setup>
```

Kuvio 11. Web-sovelluksen asetustiedosto

Web-sovellus lisää käynnissä olevaan EMR-klusteriin yhden laskentatehtävän (step). Kun laskenta käynnistyy, palauttaa web-sovellus selaimelle tehtävän ID:n.

Kaikki tiedonvaihto verkkoselaimen ja web-sovelluksen välillä tapahtuu JSON-olioita siirtämällä, ja yhteydessä käytetään AJAX-tekniikkaa. Verkkosivulla on kolme painiketta, joilla käytetään edellä mainitun AngularJS-kirjaston avulla tehty ohjainta. Sivun HTML-koodissa on painikkeiden määrittelyssä viittaus ohjaimessa olevaan funktioon, esimerkiksi käynnistä-painike viittaa start-funktioon: <button data-ng-click="start()">Käynnistä</button>. Ohjaimen funktiot ovat start, check ja get. Start käynnistää laskennan lähettämällä web-sovelluksella http-pyynnön, ja saa vastauksena JSON-olion, joka sisältää tiedot kyseisen laskennan yksilöivistä tunnisteista. Näiden tunnisteiden avulla voidaan myöhemmin hakea tiedot laskennan etenemisestä ja tuloksista. Check-funktio hakee tiedot laskennan sen hetkisestä tilasta ja get-funktio hakee laskennan päätyttyä tulokset JSON-muodossa, päivittää sivulla olevan taulukon sekä kutsuu funktiota, joka piirtää piirakkakuvion. Piirakkakuvion piirtämisessä käytetään D3-JavaScript-kirjastoa.

Laskennan valmistuttua käydään hakemassa tulos ja se esitetään sekä taulukkomuodossa että piirakkakuviona (kuva 4).



Kuva 4. Laskennan tulos esitetään taulukkona sekä graafisesti

6 Tulokset ja pohdinta

Opinnäytetyössä rakennettiin Proof-of-Concept –demo, jonka tarkoituksena oli osoittaa, että on mahdollista luoda terveysdatan prosessointiin tarkoitettu pilvipalvelu kustannustehokkaasti käyttäen standardoituja rajapintoja ja nykyaikaisia web-teknologioita. Kustannustehokkuus muodostuu pääasiassa seuraavista seikoista: maksuttomien lisenssien ja avointen rajapintojen käytöstä, pilvipalveluympäristöön liittyvien teknologisten ja taloudellisten etujen hyödyntämisestä, ja nykyaikaisten ohjelmistokehitysmenetelmien käyttämisestä.

Opinnäytetyön edetessä havaittiin, että FHIR-standardi on käyttökelpoinen osana pilvipalveluarkkitehtuuria, joka mahdollistaa eri sairaaloissa käytettyjen järjestelmien luontevan yhteentoimivuuden. FHIR on joustava tiedon vaihtoon ja tallennukseen kehitetty standardi, joka sallii mahdollisuuden määritellä uusia resurssityyppejä – myös muunlaisen kuin terveysdatan käsittelyyn. Näin ollen FHIR-standardin avulla on periaatteessa mahdollista kehittää myös muita kuin terveysdataan liittyviä sovelluksia.

Tässä työssä käytettiin Amazonin pilvipalvelua, mutta demo-ohjelmisto on siirrettävissä myös muihin ympäristöihin. Amazonin ohjelmointirajapinnat S3-tietovarastoon ja EMR-laskentapalveluun ovat pilvipalvelukohtaisia, ja nämä toiminnot olisi toteutettava uudestaan, mikäli demo siirrettäisiin toiseen pilvipalveluympäristöön, esimerkiksi Microsoftin Azureen tai Google App Engineen. Testidataa tuottava asiakassovellus tehtiin JavaFX-viitekehyksen avulla, mutta avoimet rajapinnat olisivat mahdollistaneet myös muitakin toteutustapoja kuten esimerkiksi Python-ohjelmointi tai C++-kehitystyökalut.

Amazonin Elastic MapReducessa käytettiin testidatan lukemiseen Hadoop-kirjaston valmista TextInputFormat-luokkaa. Aika myöhäisessä vaiheessa opinnäytetyön tekemistä esiin nousi myös mahdollisuus tehdä täysin räätälöity lukija XML-tiedostoja varten, mikä parantaisi luultavasti laskennan suorituskykyä. Tämä voisi olla yksi seuraavista kehityskohteista tämän arkkitehtuurin kehittämissä. Toinen kokeilemisen arvoinen asia olisi korvata Basic-autentikointi esimerkiksi OAuth2-autentikointimenetelmällä. Tässä menetelmässä voitaisiin käyttää kolmannen osapuolen valtuutuspalvelinta, esimerkiksi Googlea tai Microsoftia, ja näin käyttäjien tunnistuspalvelu saataisiin skaalautumaan paremmin.

Demossa esitelty arkkitehtuuri todettiin toimivaksi ja voitiin havaita, että se mahdollistaa uusien laskennan käyttötapauksien kehittämisen terveysdatalle kustannustehokkaasti ja joustavasti. Amazonin pilvipalvelu ja Elastic MapReduce vaikuttavat siihen, että laskennan

käynnistyminen kestää noin minuutin riippumatta datan määrästä, ja tämä heikensi käyttäjäkokemusta. Projektissa ohjelmoitiin Javalla map- ja reduce-metodit Hadoop-kirjaston avulla, mutta muita, ehkä helpompiakin, tapoja olisi ollut toteuttaa laskennan algoritmit, kuten PIG tai Hive. Koska Hadoop perustuu datan levylle kirjoittamiseen ja lukemiseen, syntyy tästä ylimääräistä viivettä, ja täten in-memory–analytiikkaratkaisut, kuten esimerkiksi Spark-viitekehys, voisi nopeuttaa laskentaa.

Opinnäytetyöprosessi alkoi ripeästi ja tässä työssä käytettyjen tekniikoiden osittainen tunteminen jo entuudestaan oli suureksi hyödyksi. Ensimmäinen toimiva demo toteutettiin lähes alkuperäisen aikataulun mukaisesti, ja muutenkin työ edistyi sujuvasti. Seminaarien jälkeen itse opinnäytetyöraportin viimeistely viivästyi merkittävästi projektin ulkopuolisten syiden vuoksi.

Vaikka osa käytetyistä tekniikoista oli jo tuttuja, antoi tämä projekti paljon uusia haasteita ja mahdollisuuden oppia suosittuja ja laajassa käytössä olevia web-teknologioita, kuten RESTful Services ja AngularJS. Opinnäytetyön puitteissa päästiin myös perehtymään syvällisesti pilvipalveluiden toteuttamiseen ja Big Data -ratkaisuihin antaen näin lisäymmärrystä opinnäytetyön tekijälle näiden yhä suurempaa suosiota saavuttavien tietojenkäsittelyalueiden saralla.

Lähteet

Amazon CloudFront Documentation 2016. Luettavissa: http://aws.amazon.com/documentation/cloudfront/. Luettu 24.3.2016

Amazon EC2 Key Pairs 2016. Luettavissa: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html. Luettu 24.3.2016

Amazon EC2, User Guide for Microsoft Windows 2015. Luettavissa: http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-wg.pdf. Luettu 7.2.2016

Amazon Elastic MapReduce Documentation 2016. Luettavissa: http://aws.amazon.com/documentation/elastic-mapreduce/. Luettu 24.3.2016

Amazon ELB, Developer Guide 2015. Luettavissa: http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-dg.pdf. Luettu 6.2.2015

Amazon EMR, Developer Guide 2015. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-dg.pdf. Luettu 7.2.2015

Amazon, Life Cycle of a Cluster 2016. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/ProcessingCycle.html. Luettu 24.3.2016

Amazon S3, Developer Guide 2015. Luettavissa: http://docs.aws.amazon.com/AmazonS3/latest/dev/s3-dg.pdf. Luettu 7.2.2015

Amazon Simple Queue Service Documentation 2016. Luettavissa: http://aws.amazon.com/documentation/sqs/. Luettu 24.3.2016

Amazon SimpleDB 2016. Luettavissa: https://aws.amazon.com/simpledb/. Luettu 24.3.2016

AngularJS - Directives 2016. Luettavissa: http://www.tutorialspoint.com/angularjs/angularjs_directives.htm. Luettu 26.3.2016

Barr, J. 2010. New Whitepaper: Architecting for the Cloud: Best Practices. Luettavissa: https://aws.amazon.com/blogs/aws/new-whitepaper-architecting-for-the-cloud-best-practices/. Luettu 25.3.2016

Bostock, M. 2016. Data-Driven Documents. Luettavissa: https://github.com/mbostock/d3. Luettu 21.3.2016

Bostock, M., Ogievetsky, V. & Heer, J. 2011. D3: Data-Driven Documents. Luettavissa: http://vis.stanford.edu/files/2011-D3-InfoVis.pdf. Luettu 21.3.2016

Branas, R. 2014. AngularJS Essentials. Packt Publishing. Birmingham.

FHIR: Data Types. Luettavissa: http://www.hl7.org/implement/standards/fhir/datatypes.html. Luettu 20.3.2016.

FHIR, HL7 Version 2. Luettavissa: http://www.hl7.org/implement/standards/fhir/comparison-v2.html. Luettu 10.12.2015

FHIR: HL7 Version 3. Luettavissa: http://www.hl7.org/implement/standards/fhir/comparison-v3.html. Luettu 10.12.2015

FHIR: Introducing HL7 FHIR. Luettavissa: http://www.hl7.org/implement/standards/fhir/summary.html. Luettu 10.12.2015

FHIR Resource Definitions 2014. Luettavissa: http://www.hl7.org/implement/standards/fhir/resources.html. Luettu 5.3.2015

FHIR Resource Observation - Content 2014. Luettavissa: http://www.hl7.org/implement/standards/fhir/observation.html. Luettu 5.3.2015

FHIR Security 2014. Luettavissa: http://www.hl7.org/implement/standards/fhir/security.html. Luettu 5.3.2015

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Luettavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Luettu 20.3.2015

Freeman, A. 2014. Pro AngularJS. Springer Science+Business Media New York. New York.

Gourley, D. & Totty, B. 2002. HTTP the definitive guide. 1. painos. O'Reilly Media Inc. Sebastobol

Green, B. & Seshadri, S. 2013. AngularJs. O'Reilly Media Inc. Sebastopol.

Green, T. 2013. Mastering Cloud Computing. Elsevier Inc. Waltham.

Crockford D. 2006. RFC 4627, JSON. Luettavissa: http://www.ietf.org/rfc/rfc4627.txt. Luettu 30.1.2015

Gröne, B., Knöpfel, A., Kugel, R. & Schmidt, O. 2008. The Apache Modeling Project. Luettavissa: http://www.fmc-modeling.org/download/projects/apache/the_apache_modelling_project.pdf. Luettu 30.01.2015

Gunter, T. & Terry, N. 2005. Journal of Medical Internet Research, Jan-Mar, vol. 7(1). Luettavissa: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1550638/. Luettu 11.12.2015

HL7 About. Luettavissa: http://www.hl7.org/about/index.cfm?ref=nav. Luettu 10.12.2015

HL7 IntroToHL7. Luettavissa: http://www.hl7.org/documentcenter/public/training/IntroToHL7/player.html. Luettu 10.12.2015

Introducing JSON. Luettavissa: http://json.org/. Luettu 1.2.2015

Järvinen, P. & Järvinen, A. 2004 (2. painos: 2011). Tutkimustyön metodeista. Opinpajan kirja, Tampere.

Kavis, M. 2014. Architecting the cloud: design decisions for cloud computing service models. Wiley. New Jersey.

Larman & Basili. 2003. Computer, Kesäkuu 2003. Luettavissa: http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf. Luettu 11.2.2015

Marinescu, D. 2013. Cloud Computing – Theory and Practice. 1. painos. Elsevier. Amsterdam.

Mell P. & Grance, T. 2011. The NIST Definition of Cloud Computing 2011. Luettavissa: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Luettu 3.1.2015

Miner, D. & Shook, A. 2013. MapReduce Design Patterns. O'Reilly Media Inc. Sebastobol.

Murray, S. 2013. Interactive Data Visualization for the Web. O'Reilly Media Inc. Sebastobol

RestAPI tutorial. Luettavissa: http://www.restapitutorial.com/lessons/whatisrest.html. Luettu 30.1.2015

Topmobiletrends.com. 2014. Mobile Tech Contributions to Healthcare & Patient Experiences. Luettavissa: http://topmobiletrends.com/mobile-technology-contributions-patient-experience-parmar/. Luettu 1.2.2015

Velte, A., Velte, T. & Elsenpeter R. 2010. Cloud Computing: A practical approach. 1. painos. The McGraw Hill Companies. New York.

W3.org, Extensible Markup Language. 2015. Luettavissa: http://www.w3.org/TR/xml/. Luettu 30.1.2015

Zinn, D., Bowers, S., Köhler, S. & Ludäscher, B. 2010. Parallelizing XML processing pipelines via MapReduce. Journal of Computer and System Sciences. Luettavissa: http://www.sciencedirect.com/science/article/pii/S0022000009001226. Luettu 25.3.2016

Liitteet