Jan-Erik Sundman

Mikropalveluarkkitehtuuri ja sen käyttö sekä luotettavuuskatsaus Sanoste Oy:n järjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

28.10.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Jan-Erik Sundman Mikropalveluarkkitehtuuri ja sen käyttö sekä luotettavuuskatsaus Sanoste Oy:n järjestelmässä 32 sivua 28.10.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Simo Silander Sanoste Oy:n Teknologiapäällikkö Kari Heikkilä

Tämän insinöörityön tarkoituksena on mikropalveluiden ja mikropalveluarkkitehtuurin käyttöön ja vaikutuksiin paneutuminen sekä teoreettisen että käytännöllisen näkökulman kautta. Työ on tehty yritykselle Sanoste Oy.

Työn ensimmäinen osuus on katsaus mikropalveluihin kokonaisuudessaan. Käsitellään mikropalveluarkkitehtuurin alkuperää ja historiaa, toimintaperiaatteita, vaikutuksia sekä tekniikkaa.

Työn toisessa osuudessa tutustutaan mikropalveluarkkitehtuurin käyttöön Sanoste Oy:n järjestelmässä, käytettyihin ratkaisuihin ja rakenteisiin sekä tekniikkaan.

Työn kolmas osuus on luotettavuuskatsaus Sanoste Oy:n mikropalvelujärjestelmään. Tarkoituksena on luoda virhetilanteita, joissa eri mikropalvelut ovat käyttämättömissä, ja löytää mahdollisia kipukohtia järjestelmässä sen kautta.

Avainsanat	Mikropalvelut, Mikropalveluarkkitehtuuri, Luotettavuuskatsaus

Author	Jan-Erik Sundman
Title	Microservice Architecture and Its Use and Reliability Overview in Sanoste Systems
Number of Pages	32 pages
Date	28 October 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Simo Silander, Senior Lecturer Kari Heikkilä, CTO of Sanoste Oy

The goal of this thesis was to look into microservices and microservice architecture and their usage and evaluate their effects in software development from a theoretical and practical standpoint. This thesis was done for the company Sanoste Oy.

The study presents an overview into microservice architecture as a whole, addressing the history and origins, principles, effects and technologies of microservice architecture.

Next, the use of microservice architecture in Sanoste Oy:s system is introduced and the used solutions, structures and technologies are addressed.

The study also provides a reliability overview in Sanoste Oy:s microservice system. The goal is to create exceptions where different microservices are out of order and through that locate weak points in the system.

Keywords	Microservices, Microservice Architecture, Reliability overview

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta.
REST	Representational State Transfer. Http-protokollaan pohjautuva ohjelmointirajapintojen arkkitehtuurimalli.
JSON	JavaScript Object Notation. Tiedonvälitykseen tarkoitettu tiedostomuoto.
SOA	Service Oriented Architecture. Palvelukeskeinen ohjelmistoarkkitehtuuri.
ESB	Enterprise Service Bus. SOA:ssa käytetty älykäs kommunikaatiomekanismi.
AWS	Amazon Web Services. Joukko Amazonin tarjoamia pilvipalveluita.

1 Johdanto

Mikropalveluarkkitehtuuri (Eng. Microservice Architecture) on nuori arkkitehtuuritason ohjelmistosuunnittelutapa, jota käytetään nykyajan hajautetuissa ohjelmistojärjestelmissä enenevissä määrin.

Mikropalveluarkkitehtuurissa ohjelmisto kootaan monista toisistaan itsenäisistä pienikokoisista palveluista, ns. mikropalveluista, jotka käsittelevät omaa toimintoaan ja yhteistyössä toisten mikropalveluiden kanssa muodostavat toimivan kokonaisuuden. Mikropalveluarkkitehtuuri pyrkii tarjoamaan ratkaisuja ongelmiin, joita esiintyy perinteisen ohjelmistosuunnittelun tuloksena.

Nuoren ikänsä takia mikropalveluarkkitehtuuri on tyylinä vielä muovautuva eikä sille näin ollen löydy tarkkoja virallisia määritelmiä, vaan lähinnä yhtäläisyyksiä tunnusomaisissa piirteissä mikropalveluita toteuttavissa ohjelmistoissa. [1; 12.]

Työ tehtiin yritykselle Sanoste Oy. Työ tehtiin osittain tarkoituksella kasvattaa ymmärrystä mikropalveluista ja mikropalveluarkkitehtuurista, niiden käytöstä sekä vaikutuksista ohjelmistokehitykseen sekä teoreettisesta että käytännöllisestä näkökulmasta. Työssä tutustutaan Sanoste Oy:n mikropalveluarkkitehtuuria käyttävään ohjelmistorakenteeseen ja sen komponentteihin.

Tarkoituksena on myös testata Sanoste Oy:n järjestelmän palveluita luotettavuudeltaan, selvittää, miten itsenäisesti ne toimivat, ja oppia, miten eri virhetilanteet vaikuttavat järjestelmän toimintaan, paikantaen mahdollisia järjestelmään laajasti vaikuttavia virheitä.

Työn ensimmäinen osa on katsaus mikropalveluihin. Käsitellään mikropalveluarkkitehtuurin alkuperää, toimintaperiaatteita, vaikutuksia sekä tekniikkaa.

Työn toisessa osassa pyritään havainnollistamaan mikropalveluarkkitehtuurin toteutuksen käytännössä käsittelemällä Sanoste Oy:n kehittämän palvelun ohjelmistoarkkitehtuuria, esittäen sovelluksen rakenteen kannalta tehtyjä ratkaisuja sekä käytettyä tekniikkaa.

Työn kolmas osa on palveluarkkitehtuurin luotettavuuskatsaus, jossa testataan pari liiketoiminnan kannalta tärkeää toimintoa mahdollisissa virhetilanteissa. Mikropalveluarkkitehtuureissa eri komponenttien välisen viestinnän toimivuus ja virhetilanteiden käsittelyn tärkeys korostuvat kasvavan palvelumäärän myötä.

2 Mikropalvelut

Mikropalveluarkkitehtuuri perustuu yksittäisen ohjelmiston kokoamiseen monista toisistaan itsenäisistä pienikokoisista palveluista, ns. mikropalveluista. Jokainen palvelu käsittelee omaa tehtäväänsä ja kommunikoi muiden mikropalveluiden kanssa viestein ohjelmointirajapintojensa eli API:en välityksellä. Tällöin käytetään esimerkiksi HTTP-verbien avulla pyyntö vasteita (Eng. "request-response"), REST ja JSON. [1.] Toisin sanoen itsenäisiä palveluja käytetään mikropalveluarkkitehtuurissa ohjelmistojärjestelmän komponentteina. [2.]

Mikropalveluarkkitehtuuri on kasvattanut suosiotaan ohjelmistokehityksessä. Sitä käyttävät mm. palvelut ja yritykset kuten: SoundCloud, Netflix, Amazon, Spotify, Uber ja Zalando. [3; 4; 5; 6; 7; 8.]

Mikropalvelut mainittiin ensimmäistä kertaa vuonna 2011 ohjelmistoarkkitehtuuriin keskittyvässä työpajassa. Osallistujat käyttivät "Microservices"-nimitystä kuvaillessaan uutta arkkitehtonista suuntausta, jota monet heistä olivat viime aikoina tutkineet. [1.] Vuonna 2012 tämä sama ryhmä päätti nimetä tyylisuuntauksen tämän mukaisesti. James Lewis sekä Fred George esittelivät näitä ideoita konferensseissa vuonna 2012. [9; 10.]

Yksi ensimmäisistä mikropalveluarkkitehtuuria muistuttavista verkkopalveluista oli Netflix, missä tyyliä kutsuttiin nimellä "Fine-grained Service Oriented Architecture", eli "hienojakoinen palvelukeskeinen arkkitehtuuri". [8.]

Vastakohdan vertauskuvaksi mikropalvelusovelluksille voidaan ottaa suuret jakamattomat palvelut, ns. monoliittiset sovellukset, joissa kaikki sovelluksen komponentit ovat yhdessä "kaiken tekevässä" palvelussa. Komponentit ovat riippuvaisia sovelluksesta johon ne sisältyvät, sekä yhteisistä resursseista. Näin ollen ne ovat kovin riippuvaisia myös toisistaan. [11.]

Mikropalveluiden hyötyihin ja haittoihin paneudutaan tässä työssä myöhemmässä luvussa.

2.1 Mikropalveluarkkitehtuurin ominaispiirteitä

Mikropalveluarkkitehtuuri on niin nuori ja muovautuva arkkitehtuurityyli, että sen määritelmät elävät.

Joitakin yhteneväisyyksiä kuitenkin löytyy mikropalveluarkkitehtuuria käyttävien sovellusten joukosta. Vuonna 2014 GOTO-konferenssissa Berliinissä Martin Fowler nosti esille seuraavia mikropalveluiden tunnusomaista piirteitä:

- sovelluksen komponenttien jaottelu palveluiksi
- järjestely liiketoiminnallisten toimintojen ympärille
- tuotteita projektien sijaan
- älykkäät päätteet, tyhmät putket
- hajautettu hallinto ja tiedonhallinta
- automatisoitu infrastruktuuri
- suunniteltu vikojen varalta
- evolutiivinen suunnittelu. [12.]

Sovelluksen komponenttien jaottelu palveluiksi

Perinteisesti ohjelmistosuunnittelussa jaottelu komponentteihin on toteutettu jakamalla ohjelmisto esimerkiksi kirjastoihin, pakkauksiin tai moduuleihin, joita voidaan käyttää uudelleen muissa ohjelmistoissa, joissa komponentit ladataan muistiin niitä käytettäessä. Mikropalveluiden prosesseissa käytetään myös muistista kutsuttavia komponentteja kuten kirjastoja, mutta oman sovelluksen komponentteihin jako tapahtuu jakamalla ohjelmisto itsenäisiin, toisistaan eristettyihin palveluihin, joita käytetään funktiokutsujen sijaan viestein kommunikoiden. Voidaan käyttää esimerkiksi HTTP-verbejä, REST:iä ja JSON:ia.

Järjestely liiketoiminnallisten toimintojen ympärille

Kuten nimestä voi arvata, mikropalvelut ovat yleensä kooltaan pieniä. Etuliite "mikro" ei kuitenkaan viittaa suoranaisesti palvelun kokoon, vaan sen tarjoamaan toiminnallisuuden laajuuteen. Yksittäinen mikropalvelu keskittyy sille tarkoitettuun toimintoon, mikä perustuu sovellus- ja liiketoiminnallisiin valmiuksiin. [13.]

Tuotteita projektien sijaan

Mikropalveluarkkitehtuuria noudattava ohjelmistokehitys yleensä välttää tavanomaista mallia sovelluksista projekteina, joissa tarkoituksena on toimittaa ohjelma jollekin toiselle taholle, jonka jälkeen nähdään projektin valmistuvan ja vastuun siirtyvän ohjelmaa hallinnoiville osapuolille.

Mikropalveluarkkitehtuuria noudattavat ohjelmistokehitystiimit omistavat usein kehityksessä olevan tuotteen sen koko elinkaaren ajan ja ovat vastuussa sen koko tuotanto- ja hallintatoiminnasta. Tämä kokonaisvaltainen ajattelu tuotetusta palvelusta on yhteydessä sovelluksen jaotteluun palveluiksi liiketoiminnallisten toimintojen mukaan, palvelut muuttuvat ja heijastavat liiketoimintaa sen kehittyessä. [1.]

Älykkäät päätteet, tyhmät putket

Mikropalveluarkkitehtuuri on rakennettu palvelukeskeisen arkkitehtuurin, eli SOA:n, käsitteiden päälle ja on sen erikoistunut toteutustapa. [11.]

SOA on erittäin laaja käsite, ja jotkut ovatkin jo toteuttaneet mikropalveluarkkitehtuuria muistuttavaa suunnittelua kutsuen sitä SOA:ksi. Palvelukeskeisessä arkkitehtuurissa integroidaan eri sovelluksia joukosta itsenäisiä palveluita tarpeen mukaan. Palveluiden välisessä kommunikaatiossa käytetään kuitenkin tyypillisesti huomattavan älykästä viestintäkerrosta, joka vastaa mm. viestien ohjauksesta, reitityksestä, muuntamisesta ja turvallisuudesta. Esimerkki tällaisesta älykkäästä kommunikaatiomekanismista on Enterprise Service Busilla (Lyh. ESB). [1.]

SOA:ssa yleisesti käytetyn palveluiden integraation ja mikropalveluiden arkkitehtonisen suunnittelun ero voidaan esittää vertaamalla orkestraatiota koreografiaan. Orkestraatio vaatii johtajan, eli keskitetyn palvelun, joka valvoo ja ohjaa muiden palveluiden tekemisiä

sekä yhdistää kaikki palvelut kokonaisuudeksi. Koreografia ei vaadi johtajaa, vaan palvelut ovat tekemisissä toistensa kanssa yhteisesti ymmärrettävällä viestintäprotokollalla ja globaalisti määritetyillä toimintamekanismeilla. [2; 11.]

Palvelukoreografian painotus ei ole mikropalveluarkkitehtuurin erikoisuus ja onkin ollut käytössä aikaisemmissa palvelukeskeisen arkkitehtuurin implementaatioissa, mutta palveluorkestraation helppokäyttöisyys monimutkaisten järjestelmien hallinnassa johti palveluorkestraation suosioon palvelukeskeisten arkkitehtuurien alkuvaiheissa. [11.]

Perinteisestä palvelukeskeisestä arkkitehtuurista eroten, mikropalveluiden välisessä viestintäkerroksessa ei ole älykkyyttä vaan päätepisteet pyritään rakentamaan mahdollisimman älykkäiksi. Mikropalveluissa pyritään myös vähentämään palveluiden kompleksisuutta sekä kokoa, jotta palveluiden välinen vastuu jakautuisi mahdollisimman tasapuolisesti ja jotta yksittäinen palvelu toteuttaisi mahdollisimman tehokkaasti sille tarkoitettua toiminnallisuutta. [11.]

Hajautettu hallinto ja tiedonhallinta

Mikropalveluarkkitehtuurissa suunnitellaan yksittäinen sovellus joukoksi itsenäisiä, keskenään kommunikoivia, palveluita. Tällainen hajautettu järjestelmä mahdollistaa sen, että eri palvelut voivat toteuttaa tarvittavan toiminnon käyttäen mitä tahansa työkaluja ja teknologioita; järjestelmä on agnostinen ohjelmointikielien ja standardien suhteen, kunhan kommunikaatio on yhteneväistä. Hajautettu hallinto näkyy mm. aikaisemmin mainitussa sovelluskehitystiimin kokonaisvaltaisessa vastuussa liittyen sen kehittämään ja hallinnoimaan tuotteeseen.

Tiedonhallinta on myös hajautettuna eri palveluiden kesken. Yleisesti palvelulla on oma tietokantansa, jota se hallinnoi. Tämä tietokanta voi joko olla erillinen ilmentymä järjestelmässä käytettävästä tietokantateknologiasta tai sitten aivan erillistä tietokantateknologiaa. [1.]

Automatisoitu infrastruktuuri

Automatisoitu infrastruktuuri ei ole mikropalveluille ainutlaatuista, mutta suurin osa mikropalveluarkkitehtuuria noudattavista sovelluskehitystiimeistä käyttävät erilaisia infrastruktuurin automatisaatiokäytäntöjä kuten jatkuvaa integraatiota (Eng. Continuous Integration) sekä jatkuvaa toimitusta (Eng. Continuous Delivery).

Infrastruktuuriautomatisaatiota tukevien pilvipalveluiden kuten esimerkiksi Amazon Web Servicesin kehitys on helpottanut mikropalveluiden tuottamista, toimittamista ja hallinnoimista. [1.]

Suunnittelu vikojen varalta

Järjestelmän ollessa kokoelma toisistaan itsenäisesti toimivia mikropalveluita on mahdollisuus virhetilanteisiin suurempi kasvaneesta palvelumäärästä johtuen. Mikä tahansa palveluista voi olla nurin tai toimia virheellisesti, ja sen kanssa yhteistyötä tekevän palvelun on käsiteltävä virhetilanne niin hienovaraisesti kuin mahdollista luomatta uutta virhetilannetta itselleen. Lisääntynyt kompleksisuus vaatii lisääntynyttä laadunvarmistusta ja automaattitestausta sekä jatkuvaa valvontaa.

Evolutiivinen suunnittelu

Jaotellessa järjestelmän liiketoiminnan kannalta tärkeitä toimintoja heijastaviin mikropalveluihin keskitytään myös jaottelemaan palvelut tukemaan tulevaa kehitystä. Jaotellaan palvelut sellaisiin osiin, joissa osien palveluiden uusiminen ja korvaaminen sekä uudelleenkäyttö on mahdollisimman sujuvaa. [1.]

2.2 Miksi mikropalvelut?

Mikropalveluarkkitehtuurin hyötyihin kuuluvat monet samat aspektit kuin perinteiseen palvelukeskeiseen arkkitehtuuriin.

Palvelukeskeistä arkkitehtuuria käyttävässä järjestelmässä monimutkaisemmat palvelut koostuvat monista pienemmistä ja yksinkertaisemmista palveluista. Palvelut voivat olla monen järjestelmän käytössä, ja palveluista voidaan käynnistää uusia instansseja tarpeen mukaan, esim. kasvavan kuorman hallitsemiseksi ja järjestelmäkuorman tasapainottamiseksi. Palvelut käyttävät yhteisesti sovittuja rajapintoja ja kommunikoivat käyttäen sovittuja viestintäprotokollia.

Palvelukeskeisen arkkitehtuurin hyödyllisin aspekteihin kuuluvat näin ollen mm.

- modulaarisuus ja palveluiden uudelleenkäytettävyys
- dynaamisuus ja skaalautuvuus

- järjestelmän ja sen kehityksen hajautus
- integroinnin mahdollisuus myös vanhempiin järjestelmiin. [11.]

Mikropalveluiden koot ovat tyypillisesti pienempiä, ja palvelut keskittyvät yksittäisiin toimintavalmiuksiin. Palvelut ovat täysin itsenäisiä toisistaan ja viestivät toisten palveluiden kanssa ainoastaan rajapintojensa välityksellä.

Järjestelmän ylläpito sekä laajennettavuus helpottuvat yksittäisten palveluiden ollessa muokattavissa, vaihdettavissa ja lisättävissä ilman, että muihin palveluihin tai viestintäkerrokseen joudutaan tekemään muutoksia. [11.]

Järjestelmänarkkitehtuurin suunnitteleminen liiketoiminnan kannalta tärkeiden toimintojen mukaan mikropalveluiksi mahdollistaa myös organisaatiossa tuotekehityksen jaottelun tiimeiksi palveluiden mukaan sen sijaan, että jaoteltaisiin tiimit puhtaasti erikoisosaamisen mukaan. Täten syntyy itsenäisesti toimivia ns. ”Full Stack” -tiimejä, eli tiimejä, jotka osaavat palvelun kehityksen kaikki osa-alueet, jotka ovat täysin vastuussa oman palvelun kehityksestä ja käyttöönotosta alusta loppuun. [1.]

Joitakin mikropalveluarkkitehtuurin koettuja hyötyjä voidaan osoittaa vertaamalla mikropalveluita käyttävää sovellusta tyypilliseen monoliittiseen sovellukseen, missä palvelu sisältää sovelluksen koko toiminnallisuuden.

Kun sovellus on monoliittinen ja kaikki toiminnallisuus on saman prosessin käsiteltävänä:

- Skaalaus vaatii koko sovelluksen skaalausta. Kasvava kuorma vaatii uusien instanssien käynnistämistä koko sovelluksesta, vaikka lisäresurssien tarve koskee ainoastaan tiettyjä moduuleja. (Kts. kuva 1.)
- Muutokset ja päivitykset vaativat koko sovelluksen uudelleenrakentamista ja -käynnistämistä vaikka muutokset koskisivat vain tiettyjä moduuleja, johtaen huomattaviin häiriöaikoihin sekä kehityksen, testauksen ja ylläpidon vaikeutumiseen ja hidastumiseen.
- Kehitys, ylläpito ja vikojenetsintä on vaikeaa sovelluksen ollessa isokokoinen ja monimutkainen.
- Järjestelmä on konservatiivinen ohjelmointikielen ja ohjelmistokehyksen suhteen. Kehitys on sidottu samaan ohjelmointikieleen ja samoihin ohjelmistokehyksiin kuin alkuperäisessä sovelluksessa. [1; 11.]

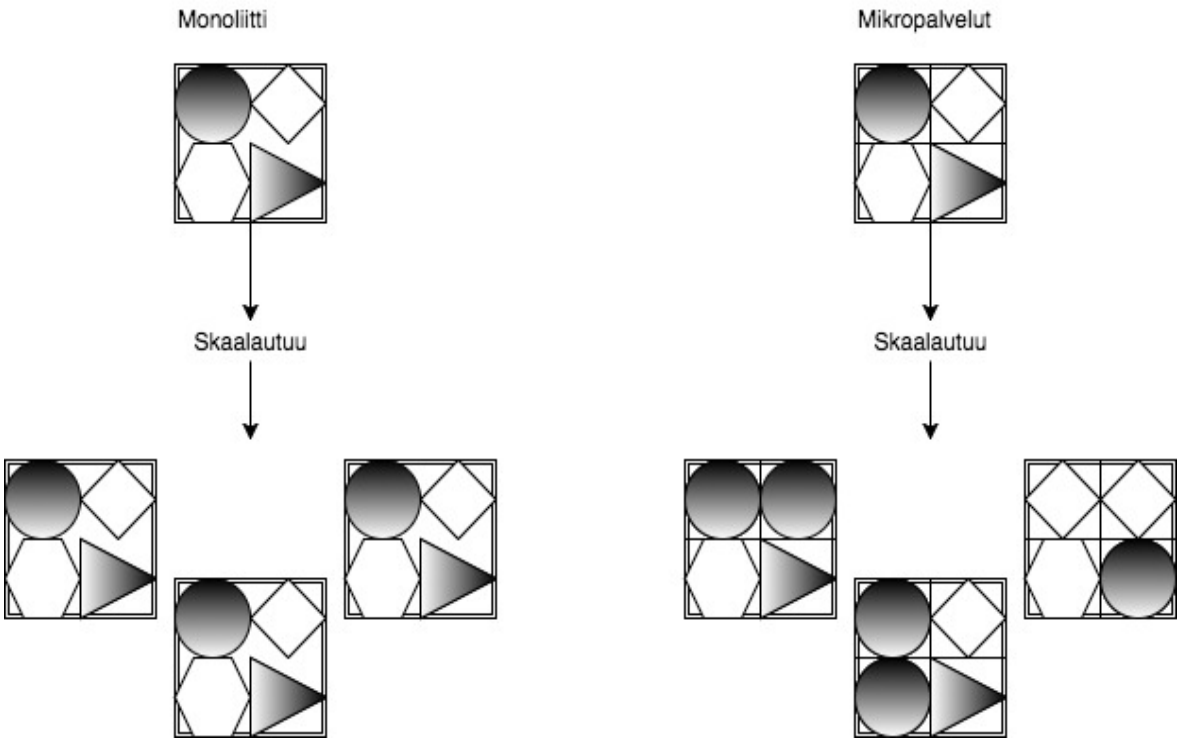
Kun sovelluksen toiminnallisuus jaetaan itsenäisiin toimintospesifisiin mikropalveluihin:

- Skaalaus ei vaadi koko sovelluksen skaalausta. Tietyn osan sovelluksesta ollessa kuorman alla, voidaan tarpeen mukaan käynnistää lisäinstansseja siitä mikropalvelusta, joka vaatii lisäresursseja. (Kts. kuva 1.)

- Muutokset ja päivitykset vaativat ainoastaan sen mikropalvelun uudelleenrakentamista ja -käynnistämistä jota muutos koskee. Vähentää tai jopa eliminoi häiriöaikoja sekä helpottaa ja nopeuttaa kehitystä, testausta ja ylläpitoa.

- Kehitys, ylläpito ja vikojenetsintä helpottuvat palveluiden ollessa pienempiä ja yksinkertaisempia.

- Järjestelmä on agnostinen ohjelmointikielen ja ohjelmistokehyksen suhteen. Kehitys ei ole sidottu mihinkään tiettyyn kieleen tai kehykseen kun mikropalvelut tarjoavat toiminnallisuutensa yhteisesti ymmärrettävien viestintäprotokollien ja rajapintojen välityksellä.



Kuva 1. Monoliittisen järjestelmän skaalautuvuus verrattuna mikropalveluihin. Monoliitti replikoidaan kokonaisuudessaan, kun yksittäisiä mikropalveluita voidaan sen sijaan hajauttaa usealle palvelimelle ja lisätä instansseja tarvittaessa.

3 Mikropalveluarkkitehtuuri Sanoste Oy:n sovelluksessa

3.1 Yleisesti Sanoste Oy:n tarjoamista palveluista

Sanoste Oy on sosiaali- ja terveydenhuoltoalalla toimiva nuori teknologiayritys. Yrityksen tarjoaman järjestelmän avulla pystytään tarjoamaan joustavasti ja helposti etäpalveluita ikääntyneille.

Palvelut toimitetaan palveluiden tuottajilta reaaliaikaisen video- ja ääniyhteyden välityksellä loppukäyttäjien omiin laitteisiin. Tämä mahdollistaa tärkeän vuorovaikutuksen palvelun tarjoajan sekä käyttäjien välille kuten esimerkiksi henkilökohtaisen neuvonnan ja ohjauksen. Palveluilla on ajoitettuja ilmentymiä, joita kutsutaan ”ohjelmiksi”. Loppukäyttäjille voidaan ostaa palveluita Sanosten SanoStore-verkkokaupasta, jolloin asiakkaille luodaan tilaus valittuun palvelun ohjelmaan.

Järjestelmä on suunniteltu niin, että se on integroitavissa monen eri kanavakumppanin kanssa. Kanavakumppaneihin kuuluvat mm. eri laitetoimittajat. Palveluntuottajien tarjoamia palveluita voidaan toisin sanoen tarjota monessa eri päätejärjestelmässä yksityisistä toimijoista kunnallisiin. Palveluntuottajat voivat olla niin yrityksiä kuin yksityisiä henkilöitä.

3.2 Järjestelmän arkkitehtuuri

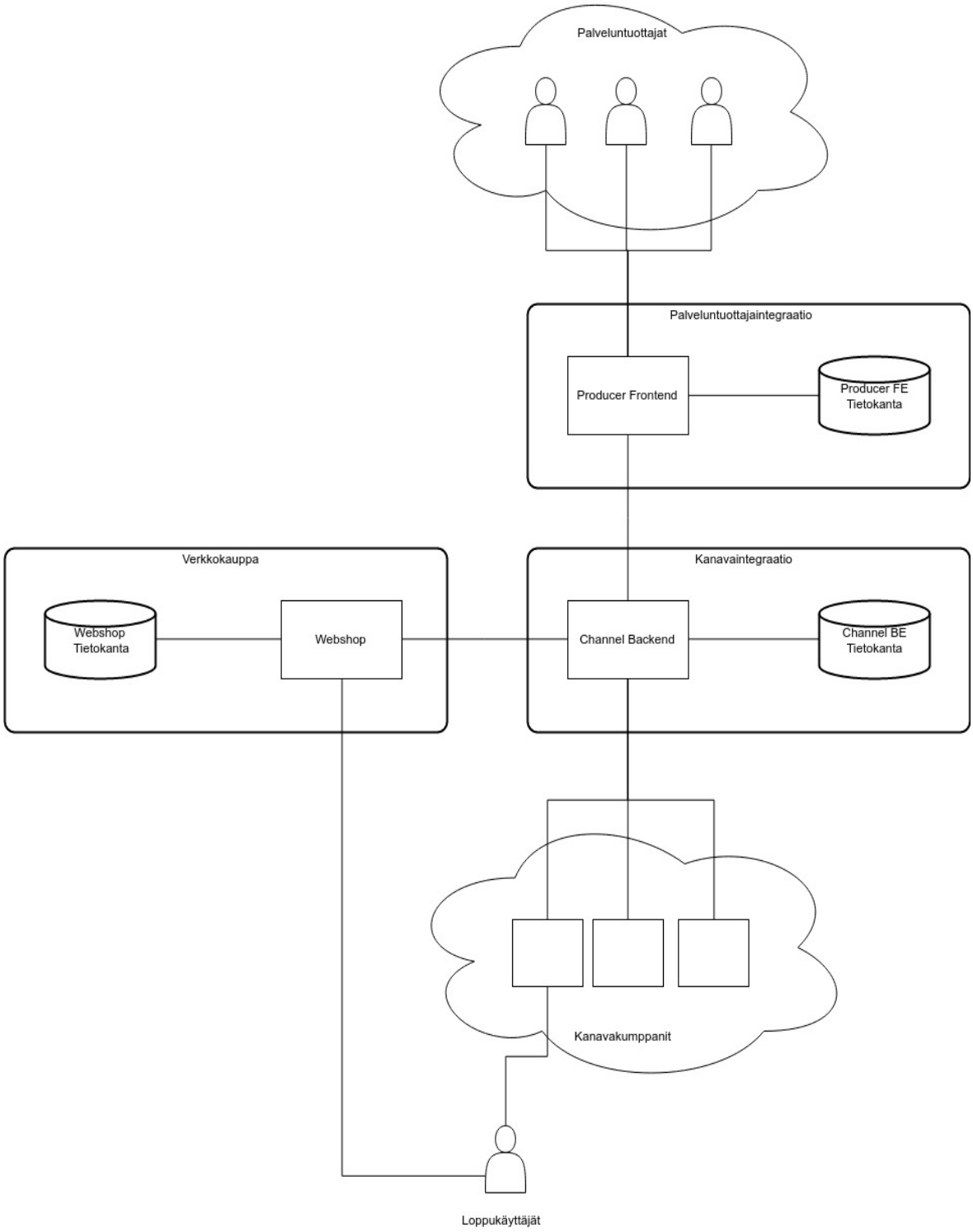
Sanosten järjestelmä on suunniteltu ja rakennettu löyhästi mikropalveluarkkitehtuuria noudattaen. Järjestelmä on jaettu liiketoiminnan kannalta tärkeiden toimintojen mukaan mikropalveluihin.

Tuottajia ja kanavakumppaneita integroiva järjestelmä "Sassy" on jaettu pääasiassa kolmeen mikropalveluun: palveluntuottajia sekä niiden palveluita ja palveluiden ohjelmia hallinnoivaan mikropalveluun, kanavaintegraatioita ja videosessioita hallinnoivaan mikropalveluun sekä mikropalveluun, jolla toimii loppukäyttäjille suunnattu verkkokauppa. (Kts. kuva 2.)

Palveluntuottajia käsittelevää mikropalvelua kutsutaan tässä nimellä "Producer Frontend", ja vastaavasti kanavakumppaneita käsittelevää palvelua nimellä "Channel Backend".

Näille molemmille mikropalveluille löytyy omat tietokantainstanssit. Kokonaisuutena tätä mikropalveluita sisältävää integraatiojärjestelmää viitataan tässä nimellä "Sassy" tai "Sassy-järjestelmä".

Näiden kahden komponentin lisäksi käytössä on Sanosten verkkokauppa ”SanoStore”, johon viitataan tässä nimellä ”Webshop”. Webshopin kautta kanavakumppaniemme asiakkaat, eli loppukäyttäjät, voivat ostaa itsellensä palveluntuottajiemme tarjoamia palveluita. Webshop kommunikoi kanavaintegraatiosta vastuussa olevan Channel Backendin kanssa.



Kuva 2. Sassy-integraatiojärjestelmän pääasiallinen arkkitehtuuri on jaoteltu liiketoiminnan kannalta tärkeiden toimintojen mukaan mikropalveluiksi: Producer Frontend, Channel Backend sekä Webshop

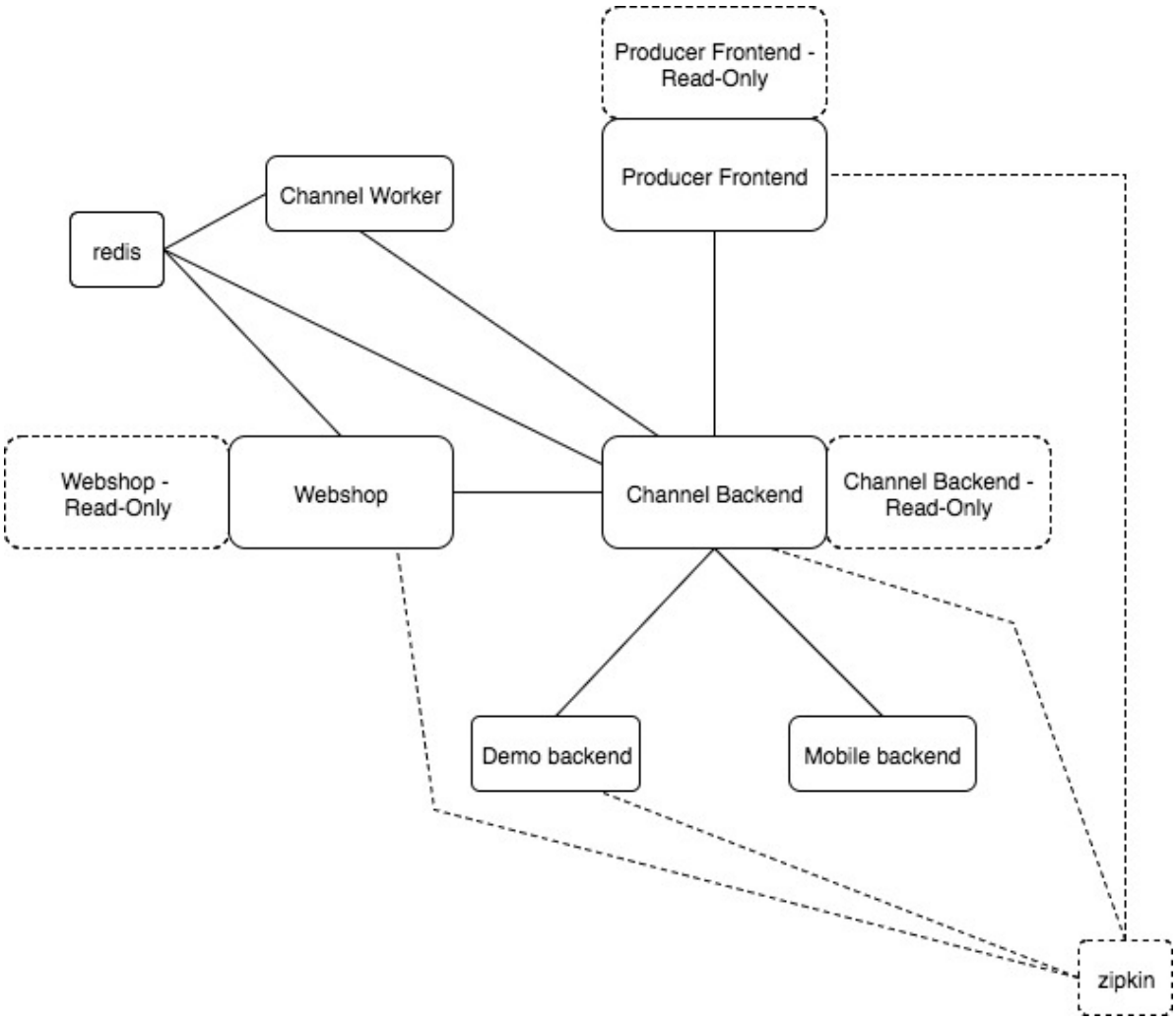
Kanavakumppanien loppukäyttäjillä voi olla usean eri palvelutuottajien palveluita. Sama pätee näin ollen myös toiseen suuntaan, eli palveluntuottajan tarjoamiin palveluihin voi liittyä loppukäyttäjiä usealta eri kanavakumppanilta.

Nykyisen toimintamallin mukaan kanavakumppanit hallitsevat omia loppukäyttäjiään, heidän laitteitaan sekä näiden kanssa kommunikoinnin, ja näin ollen Sassy-järjestelmä ei siis ole suorassa yhteydessä loppukäyttäjiin. Kanavakumppanien ei tarvitse olla yhteydessä palveluntuottajiin tai tarjota näille rajapintaa palveluiden tarjoamista varten. Niin kanavakumppanit kuin palveluntuottajat ovat yhteydessä Sassy-järjestelmään, joka hoitaa näiden toimijoiden välisen integraation sekä tarjoaa näille tarvittavat rajapinnat. Loppukäyttäjien ja palveluntuottajien välinen videoyhteys tapahtuu käyttäen Tokbox-palvelua.

Näiden kolmen pääasiallisen mikropalvelun eli Channel Backendin, Producer Frontendin sekä Webshopin lisäksi on joitakin mikropalveluita. Näihin kuuluvat mm. sisäiseen testaukseen tarkoitettu demopalvelu ”Demo Backend” sekä pääasiallisten mikropalveluiden avuksi toimivia palveluita kuten esimerkiksi Channel Backendin asynkronisessa viestittelyssä auttava ”Channel Worker”. (Kts. kuva 3.)

Taustalla pyörivät myös omissa mikropalveluissaan palveluiden toimintaa ja valvontaa tukevia palveluita kuten muistinvarainen tietokanta ja viestinvälittäjäpalvelu Redis sekä kehitys- ja testiympäristössä toimiva järjestelmän vianjäljityspalvelu Zipkin. Kehitys- ja testiympäristöissä on myös pääasiallisten mikropalveluiden lisäksi niiden ”Read-only” -versiot käynnissä testausta varten. Nämä palvelut käyttäytyvät kuten pääasialliset palvelut käyttäytyisivät järjestelmäpäivitystilanteessa, eli ainoastaan tietokannan lukuoperaatiot ovat sallittuja ja käyttäjälle kerrotaan järjestelmäpäivityksestä sellaisissa tilanteissa, kun hän yrittää tehdä muita toimenpiteitä. Muissa tapauksissa palvelut käyttäytyvät kuten alkuperäiset palvelut.

Kaikkien näiden lisäksi yhdessä mikropalvelussa toimii, vielä kehitysvaiheessa oleva, mobiilsovelluksen taustajärjestelmä ”Mobile Backend”. Demo Backend sekä Mobile Backend ovat käytännössä aikaisempaan yrityksen toimintamalliin viitaten ”kanavakumppaneita”. Demo Backend simuloi ulkoista kanavakumppania, ja Mobile Backend tulee olemaan Sanoste Oy:n oma kanava palveluidensa toimittamista varten loppukäyttäjien omiin mobiililaitteisiin. (Kts. kuva 3.)



Kuva 3. Pääasiallisten mikropalveluiden yhteydessä toimivat, järjestelmää tukevat tai laajentavat mikropalvelut. Katkoviivoin merkityt mikropalvelut ovat käytössä ainoastaan kehitys- ja testiympäristössä.

3.2.1 Producer Frontend

Producer Frontend tarjoaa palveluntuottajille selaimesta käytettävän graafisen rajapinnan, josta palveluntuottajan toimihenkilö kuten ohjaaja voi hallinnoida erilaisia videoistuntoon ja sen osallistujiin liittyviä toimintoja. Näihin kuuluvat mm. videoistuntojen aloittaminen/soitto sekä niiden lopettaminen, istunnon aikana tarvittavat toiminnot kuten oman ja osallistujien videokuvan näkeminen, videoistunnosta poistuminen sitä lopettamatta, osallistuvien käyttäjien tietojen kuten nimen näkeminen taikka hätätilanteen sattuessa yhteystietojen näyttäminen. Producer Frontendin ei tarvitse hallita tai tietää, minne tai miten palvelut toimitetaan vaan tämän hoitaa Channel Backend.

3.2.2 Channel Backend

Channel Backend on vastuussa kanavaintegraatiosta ja videoistuntojen välittämisestä. Channel Backendin ei tarvitse tietää palveluiden sisällöstä tai kuvauksista, vaan toimittaa Producer Frontendlta saadut tiedot eteenpäin.

Järjestelmä on Channel Backendin tarjoaman ohjelmointirajapinnan kautta yhteydessä kanavakumppaneiden ohjelmointirajapintoihin. Channel Backendin tarjoamaan ohjelmointirajapintaan viitataan tästä lähtien nimellä "Channel API".

Channel API on kaksisuuntainen REST-rajapinta, joka käsittelee kanavakumppanien pyyntöjä sekä lähettää näille ilmoituksia. Niin Channel API:n kuin muiden rajapintojen tiedonvälitykseen käytetään JSON-tiedostomuotoa.

Channel API:n piiriin kuuluu seuraavia osa-alueita:

- todennus
- tarjolla olevien palveluiden / tuoteluettelon tiedustelut
- tilauksien käsittely
- palveluiden toimitus.

Näiden eri osa-alueiden käsittelyä varten on olemassa Channel-APIn rajapinnat: Authentication API, Catalogue API, Subscription Management API sekä Service Delivery API.

Todennus ja Authentication API

Todennus toimii käyttäen käyttöoikeustietueita (Eng. "access token"). Jotta todennusta kaipaava toimija saa käyttöoikeustietueen, tämän on kirjauduttava onnistuneesti sisään. Tämä tapahtuu HTTP POST -pyynnöllä Authentication API:lle, joka sisältää tunnistautumiseen tarvittavat tiedot. Tunnistetietojen ollessa kelvollisia rajapinta palauttaa käyttöoikeustietueen sekä sen mahdollisen erääntymisajan.

Jotta toimija voisi olla yhteydessä kaikkiin muihin Channel API:n rajapintoihin, on tällä oltava voimassa oleva käyttöoikeustietue liitettynä rajapintapyyntöihin.

Palvelutiedot ja Catalogue API

Catalogue API:lta toimija voi saada tietoa tarjolla olevista palveluista, näiden ohjelmista, palveluntuottajista sekä kategorioista. Esimerkiksi halutessa tietoja palveluntuottajista HTTP-verbiä GET käyttävä pyyntö suunnattaisiin Catalogue-rajapinnan Producer API:in.

Vastaukseksi pyynnölle saataisiin JSON:ia sisältävä vaste, joka sisältää kaikkien palveluntuottajien tiedot olettaen, että pyynnön mukana lähetettävä käyttöoikeustietue on kelvollinen.

Vasteen sisältävä JSON olisi muodoltaan seuraavanlainen:

```
{
  producers: [{
       id: <integer>,
       uri: <resource uri>,
       services: [{
            id: <integer>,
            uri: <resource uri>
       }, ...],
       name: <string>,
       description: <string> or null,
       logo_url: <string> or null,
       receipt_info: <string>
  }, ...]
}
```

Kuva 4. Catalogue API:n Producer rajapinnalta saadun vasteen sisältämän JSON-tiedoston sisällön muoto

Samankaltainen logiikka pätee viestintään Catalogue API:n muiden rajapintojen kanssa.

Tilausten käsittely ja Subscription Management API

Subscription Management API tarjoaa rajapinnan ohjelmatilausten hakemiseen, muokkaamiseen, lisäämiseen ja poistamiseen.

Palveluiden toimitus ja Service Delivery API

Service Delivery API tarjoaa rajapinnan palveluiden toimitukseen liittyviin toimintoihin. Näihin kuuluvat mm. videosessioiden aloitus, lopetus sekä tietojen noutaminen käynnissä olevista ja menneistä sessioista.

3.2.3 Webshop

Webshop saa tiedot tarjolla olevista palveluista, palveluntuottajista ja käyttäjistä Channel-kanavaintegraatiopalvelun API-rajapinnasta, johon se on yhteydessä tasaisin väliajoin, päivittäen esimerkiksi ohjelma- ja palvelumuutoksia verkkokauppaan tai ilmoittaen uusista tilauksista.

Ostaakseen palveluita välitettäväksi omalle laitteelleen kanavakumppanien asiakkaiden täytyy tunnistautua verkkokaupassa. Kanavakumppanin asiakkaiden ja SanoStoren välille on luotava yhteys, jotta palvelut päätyvät loppukäyttäjän laitteeseen.

Kanavakumppanin ja SanoStoren välinen tunnistautuminen tapahtuu seuraavasti:

1. Asiakas siirtyy kanavakumppanin sivuilta SanoStoreen ohjaavan linkin kautta, jonka mukana välittyy SanoStorelle tieto siitä, minkä kanavakumppanin asiakas on kyseessä. Kyseiseen ohjaukseen viitataan tästä lähtien nimellä "Ingress".

2. SanoStore tunnistaa kanavakumppanin, joka on yhdistetty kyseiseen Ingress-linkkiin.

3. Jos käyttäjä ei ole jo kirjautunut SanoStoreen entisestään, häntä pyydetään kirjautumaan taikka luomaan käyttäjätunnukset SanoStoreen.

4. Käyttäjä ohjataan kanavakumppanin tunnistautumispäätteeseen, jossa tämä autentikoidaan käyttäen OAuth 2.0 -protokollaa. Jos käyttäjä on kirjautunut jo kanavakumppanin sivuille, tämä tapahtuu yleensä huomaamattomasti. Vaihtoehtoisesti kanavakumppani kysyy asiakkaaltaan lupaa jakaa tietojansa SanoStorelle. Tapauksessa jossa käyttäjä ei ole kirjautunut kanavakumppanin sivuille, tätä pyydetään kirjautumaan.

5. Käyttäjä ohjataan takaisin SanoStorelle OAuth 2.0-tunnistautumisvasteen kanssa. Onnistuneen tunnistautumisen vaste sisältää onnistumisviestin lisäksi tarvittavat valtuutustiedot (Eng. "Authorization grant").[14.] Epäonnistuneen tunnistautumisen palauttaman vasteen tapauksessa käyttäjälle ilmoitetaan epäonnistuneesta tunnistautumisesta.

6. SanoStore hakee kanavakumppanilta käyttäjän käyttöoikeustietueen käyttäen OAuth-vasteesta saatuja valtuuksia.

3.3 Järjestelmässä käytetyt teknologiat

Järjestelmässä on käytössä monta eri teknologiaa, jotka mahdollistavat mikropalveluarkkitehtuurin toteutuksen, hallinnan, toimituksen sekä ylläpidon. Seuraavaksi nostan esille tärkeimpiä järjestelmässä käytettyjä teknologioita.

3.3.1 Python

Vaikka mikropalveluarkkitehtuuri mahdollistaa monen eri ohjelmointikielen käytön palvelukohtaisesti, olemme päätyneet käyttämään pääasiallisesti Python-ohjelmointikieltä kehittäessämme palvelujemme taustajärjestelmiä. Python on korkean tason tulkattava ohjelmointikieli, joka on syntaksiltaan yksinkertainen sekä helposti omaksuttava. Pythonille löytyy useita kirjastoja ja ohjelmistokehyksiä, kuten Django ja Flask, joita käytetään Sanosten järjestelmässä.

3.3.2 Docker

Docker on avoimen lähdekoodin työkalu, joka lisää ohjelmiston abstraktiotasoa automatisoimalla ohjelmiston käyttöönoton isoloiduissa virtuaali-instansseissa käyttäen Linuxista löytyviä resurssi-isolaatioon liittyviä toiminnallisuuksia. Nämä instanssit ovat eristäytyneitä ympäristöstään, ja isoloidussa instanssissa toimivat prosessit eivät voi nähdä tai vaikuttaa prosesseihin, jotka toimivat toisissa isoloiduissa instansseissa. Jokainen instanssi sisältää kaiken, minkä se tarvitsee toimiakseen.

Eristäessä ohjelmistoa sen ympäröimästä järjestelmästä tällä tavoin voidaan varmistaa sen toiminta monessa eri ympäristössä, tuotekehityksestä tuotantoon. Tällaisia isoloituja instansseja kutsutaan myös nimellä "container".

Docker-containerit ovat instansseja Docker-vedoksista (Eng. ”Docker image”). Vedoksissa on säilötty juuritiedostojärjestelmän muutoksia sekä ohjelmiston ajon aikana käytettäviä toteutukseen tarvittavia muuttujia.

Vedos ei ikinä muutu eikä sillä ole tilaa. Se on myös on kirjoitussuojattu. Vedoksesta luotu tilallinen container-ilmentymä voi sen sijaan muuttua toimiessaan muuttuvassa toimintaympäristössä.

Docker mahdollistaa useamman isoloidun instanssin samanaikaisen olemassaolon, mikä tekee siitä tärkeän työkalun mm. mikropalvelujen kehityksessä ja käyttöönotossa. Docker mahdollistaa tällaisten hajautettujen, Linux-instansseihin pohjautuvien sovellusten koonnin, ajamisen, testaamisen sekä käyttöönoton. [15.]

Amazon Web Services tukee myös Dockeria palvelullaan Amazon ECS. Amazon ECS on nopea ja skaalautuva työkalu Amazonin pilvipalvelussa toimivien Docker-containereiden hallintaan.

3.3.3 PostgreSQL

PostgreSQL (kutsutaan myös nimellä Postgres) on kehittynyt, avoimen lähdekoodin olio-relaatiotietokantahallintajärjestelmä tai ORDBMS (Eng. ”Object-Relational Database Management System”).

PostgreSQL on kehittyneempi RDBMS ominaisuuksiltaan kuin tunnetumpi MySQL, vaikkakin jonkin verran hitaampi. Se on luotettava, noudattaa SQL-standardeja, se on laajennettavissa ohjemallisesti ja tukee näin ollen mukautettuja, kompleksisia toimenpiteitä.

PostgreSQL:lä on osaava tukiyhteisö ja sille löytyy paljon kolmansien osapuolien kehittämiä laajennuksia.[16.]

Amazon Web Services tukee PostgreSQL-instansseja Amazon RDS -palvelussaan [17].

3.3.4 Amazon Web Services

Sanoste Oy:n palvelut toimivat Amazon Web Services (Lyh. AWS) hosting-palveluja käyttäen. Käytössä olevista palveluista mainittakoon EC2 Container Service (Amazon ECS), Relational Database Service (Amazon RDS) sekä Simple Storage Service (Amazon S3).

Mikropalveluita sisältävät Docker-containerit toimivat Amazon ECS -palvelussa ja näiden tietokannat ovat Amazon RDS -palvelussa. Kaikki tarvittavat tiedostot kuten esimerkiksi mediatiedostot ovat tallennettuna Amazon S3-pilvipalvelussa.

3.3.5 Shippable ja Quay

Shippable on Docker-containereita, Python-ohjelmointikieltä sekä PostgreSQL-tietokantoja tukeva pilvipalvelu, joka tarjoaa jatkuvaa integraatiota, jatkuvaa toimittamista sekä automaatiotestausta mm. yrityksen käytössä oleville Bitbucket-repositorioille.

Quay on yksityinen Docker-rekisteri, jonne voi tallentaa Docker-vedoksia. Valmiiksi tallennetut Docker-vedokset nopeuttavat ohjelmiston koontia, ja Sanoste käyttää Quayta lähes kaikissa integraatio- ja toimitusprosesseista, automaatiotesteistä tuotantoon vientiin saakka.

4 Liiketoiminnan kannalta tärkeiden toimintojen luotettavuuskatsaus

4.1 Yleinen kuvaus

Mikropalveluja kehittäessä on otettava huomioon lisääntyneen palvelumäärän tuomat mahdolliset virhetilanteet. Yksi tai useampi instanssi voi olla poissa toiminnasta, tai viestit palveluiden välillä eivät välity. Optimaalisessa tilanteessa kaikki mikropalvelut työskentelevät itsenäisesti, ottavat huomioon, että muut mikropalvelut voivat olla toimimattomassa tilassa ja käsittelevät sulavasti sellaiset tilanteet, joissa tarvitsevat toisen, mahdollisesti toimimattoman palvelun toimintoja.

Toiminnan varmistamiseksi ja kipukohtien paikantamiseksi tehtiin luotettavuuskatsaus. Luotettavuuskatsauksesta saaduilla tiedoilla voimme esimerkiksi lisätä tarvittaviin kohtiin virhetarkistuksia taikka yksinkertaistuja, asiakkaalle ymmärrettäviä ilmoituksia virhetilanteiden sattuessa. Luotettavuuskatsaus tehtiin testaamalla seuraavia liiketoiminnan kannalta tärkeitä toimintoja:

1. Käyttäjä ostaa Sanosten verkkokaupasta palvelun.
2. Palveluntarjoaja käynnistää videoistunnon.

Joitakin mahdollisia virhetilanteita ylläolevia toimintoja tehdessä ovat:

A. Channel Backend ei ole käynnissä.
B. Producer Frontend ei ole käynnissä.
C. Webshop ei ole käynnissä.

4.2 Menetelmät

Toimintojen testaamiseksi eri virhetilanteissa määriteltiin tyypilliset vuot toimintojen käyttötapauksille sekä laajennetut käyttötapaukset virhetilanteiden testaamista varten.

4.2.1 Käyttötapaukset

Käsiteltävissä olevat käyttötapaukset ovat kaksi liiketoiminnan kannalta tärkeää toimintoa. Ensimmäisessä käyttötapauksessa loppukäyttäjä ostaa palvelun verkkokaupasta.

Toisessa käyttötapauksessa palveluntarjoaja käynnistää palvelun, eli videopuhelun, ja palvelu toimitetaan loppukäyttäjälle.

Nämä käyttötapaukset kuvaavat, miten näiden tilanteiden kuuluisi toimia ilman virhetilanteita.

1. **Käyttäjä ostaa Sanosten verkkokaupasta palvelun.**
    1. Käyttäjä siirtyy Sanosten verkkokauppaan.
    2. Käyttäjä näkee saatavilla olevat tuotteet tuoteluettelossa.
    3. Käyttäjä lisää tuotteen ostoskoriin.
    4. Käyttäjä siirtyy kassalle.
    5. Käyttäjä lisää tai kieltäytyy lisäämästä hätäyhteystietoja
    6. Käyttäjä siirtyy maksuun.
    7. Käyttäjän maksu hyväksytään ja tilaus viedään loppuun. Käyttäjän tilaus näkyy järjestelmässä, ja hänet on lisätty tilattuun ryhmään. Maksun onnistuttua käyttäjä ohjataan sivulle, jossa hän näkee yhteenvedon tilauksestaan.

2. **Palveluntarjoaja käynnistää videoistunnon.**
    1. Käyttäjä siirtyy palveluntuottajien käyttöliittymän kirjautumissivulle.
    2. Käyttäjä kirjautuu onnistuneesti sisään ja ohjataan palveluntuottajien käyttöliittymään.
    3. Kaikki palvelut ja ohjelmat, jotka ovat käyttäjän hallinnoimia, sekä ohjelmien tilaajat näkyvät käyttöliittymäsivulla.
    4. Käyttäjä painaa ”Avaa” nappia aloittaakseen videoistunnon, ja hänet ohjataan valmistautumisnäkymään.
    5. Istunto alkaa, ohjelmalle lisätään aktiivinen istunto, ohjelmaan osallistuville lähetetään kutsut istuntoon liittymiseksi.
    6. Käyttäjä ohjataan videoistunnon käyttöliittymään, missä hän voi nähdä ja kuulla osallistujien videosyötteet.
    7. Session päätyttyä käyttäjä lopettaa istunnon. Ohjelman aktiivinen istunto muutetaan entiseksi istunnoksi.

4.2.2 Laajennetut käyttötapaukset

Laajennettuihin käyttötapauksiin kuuluvat aikaisemmin määriteltyjen käyttötapauksien kaikki kohdat, Käyttötapauksiin on lisätty kolme ehtoa: Channel Backend ei ole käynnissä, Producer Frontend ei ole käynnissä sekä Webshop ei ole käynnissä. Nämä ovat merkittyjä isoin kirjaimin **A**, **B** ja **C**.

Laajennetuissa käyttötapauksissa käydään aikaisemmin määriteltyjä käyttötapauksia läpi, mutta lisäten näihin testattavissa olevat virhetilanteet.

Laajennetut käyttötapaukset ovat näin ollen:

1. **Käyttäjä ostaa Sanosten verkkokaupasta palvelun.**
    A. **Channel Backend ei ole käynnissä.**
    B. **Producer Frontend ei ole käynnissä.**
    C. **Webshop ei ole käynnissä.**

2. **Palveluntarjoaja käynnistää videoistunnon.**
    A. **Channel Backend ei ole käynnissä.**
    B. **Producer Frontend ei ole käynnissä.**
    C. **Webshop ei ole käynnissä.**

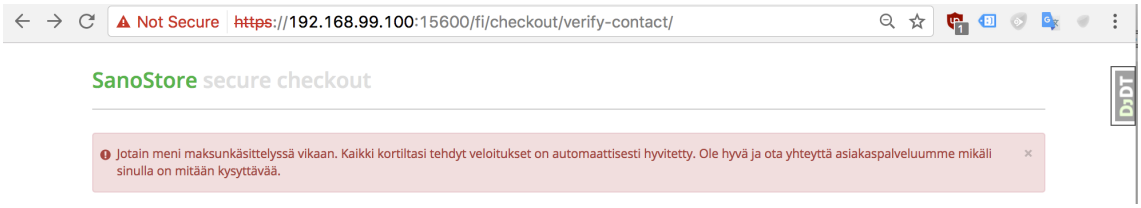
4.3 Toimintojen testaus

Virhetilanteiden sattumisen varalta eri tilanteissa käydään käyttötapaukset läpi vaihe vaiheelta ja laukaistaan testattavissa olevat virhetilanteet manuaalisesti.

Testauksen jokaisessa vaiheessa oletetaan, että virhe sattuu juuri ennen käyttötapauksen toimintoa. Simuloidut virhetilanteet aikaansaadaan sulkemalla käynnissä olevia instansseja.

Seuraavaksi käymme läpi jokaisen laajennetun käyttötapauksen ja kirjaamme testien aikana aiheutuneet virhetilanteet.

1. **Käyttäjä ostaa Sanosten verkkokaupasta palvelun.**
    A. **Channel Backend ei ole käynnissä.**
        1. Käyttäjä siirtyy onnistuneesti verkkokauppaan.
        2. Käyttäjä näkee onnistuneesti tuotekatalogin.
        3. Tuotteen lisääminen ostoskoriin onnistuu.
        4. Käyttäjä painaa nappia ”Kassalle”.
        5. Käyttäjän lisättyään tai kieltäydyttyään lisäämästä yhteystietojaan hän painaa ”Tallenna”, jolloin näkyville tulee virheilmoitus. Konsolista voimme nähdä, että yhteystietoja käsittelyn yhteydessä Webshop yrittää ottaa yhteyden Channel Backendiin yrittäessään päivittää yhteystietoja ja aiheuttaen virhetilanteen.
        6. Käyttäjä siirtyy maksuun, lisää maksutietonsa ja painaa ”Maksa”. Ennen tätä, varaus on varmistettu Channel Backend -tietokannassa.
        7. Maksun ja tilauksen käsittely keskeytyy ja näkyviin tulee samanlainen virheilmoitus kuin kohdassa 5.
           Maksupalvelu Stripestä katsoessa asiakkaan tekemä maksu on hyvitetty asiakkaan tilille, joten maksukoodi toimii halutulla tavalla, mutta epäonnistuneeseen maksuun viittaava ilmoitus ei tule näkyviin.
           Keksien välityksellä toimitettavat virheilmoitukset tulevat vasta näkyviin, kun siirrytään aikaisemmille sivuille. (Kts. kuva 5.)



Kuva 5. Virheilmoitus epäonnistuneesta maksutapahtumasta Webshopissa.

Konsolilokista näemme jälleen Webshopin pyrkivän yhteyteen Channel Backendin kanssa yrittäessään varauksien päivityksiä valmiiksi tilauksiksi.

Tarkistaessa tilauksia Channel Backendin tietokannasta, näemme, että kyseistä varausta ei ole vielä varmistettu, vaan on halutusti vielä samassa tilassa kuin kohdassa 6.

B. Producer Frontend ei ole käynnissä

1. Käyttäjä siirtyy onnistuneesti verkkokauppaan.
2. Käyttäjä näkee onnistuneesti tuotekatalogin.
3. Tuotteen lisääminen ostoskoriin onnistuu.
4. Käyttäjä painaa nappia ”Kassalle”.
5. Käyttäjän lisättyään tai kieltäydyttyään lisäämästä yhteystietojaan hän painaa ”Tallenna”, jolloin näkyville tulee virheilmoitus.

   Konsolinlokeja tutkiessa voimme nähdä, että Webshopissa syntyy virhetilanne kun tilauksia tekevä create_subscription-funktio yrittää jäsentää tyhjää vastetta. Kyseisessä funktiossa voimme nähdä, että vaste tulee Channel Backendilta.

   Konsolilokeista puolestaan voidaan nähdä virhetilanteen syntyneen Channel Backendissa ohjelmien kapasiteettia tarkistaessa.

   Kyseinen koodi vertaa ohjelman kapasiteettia tilausten määrään, ohjelman tiedot pyydetään Producer API:lta, joka palauttaa tässä tapauksessa NoneType-tyyppiä olevan olion.
6. Käyttäjä siirtyy maksuun, lisää maksutietonsa ja painaa ”Maksa”. Ennen tätä varaus on varmistettu Channel Backend tietokannassa.
7. Käyttäjän maksu hyväksytään. Maksun onnistuttua käyttäjä ohjataan sivulle, jossa hän näkee yhteenvedon tilauksestaan. Maksupalvelu Stripe vahvistaa vastaanotetun maksun, Channel Backend tietokannasta tarkistaessa varausta ei ole kuitenkaan päivitetty tilaukseksi.

C. Webshop ei ole käynnissä

Testauksessa oleva palvelu ei ole käynnissä, joten käyttötapauksen kaikki kohdat tuottavat virhetilanteen.

2. **Palveluntarjoaja käynnistää videoistunnon**

   A. **Channel Backend ei ole käynnissä**

      1. Käyttäjä siirtyy onnistuneesti palveluntuottajien käyttöliittymän kirjautumissivulle.

      2. Käyttäjä kirjautuu onnistuneesti sisään ja hänet ohjataan palveluntuottajien käyttöliittymään.

      3. Kaikki palvelut ja ohjelmat, jotka ovat käyttäjän hallinnoimia, näkyvät käyttöliittymäsivulla. Ohjelmien tilaajat eivät näy ohjelmatiedoissa.

      4. Käyttäjä painaa ”Avaa”-nappia aloittaakseen videoistunnon, ja hänet ohjataan onnistuneesti valmistautumisnäkymään.

      5. Käyttäjä aloittaa istunnon, jolloin käyttäjä näkee virheilmoituksen. Virhetilanne syntyy Producer Frontendin pyrkiessä aloittamaan videoistunnon ja lähettämään tiedon Channel API:lle käyttäen start_viewing-funktiota. Osaanottajille ei saapunut soittoja eikä ohjelmalle lisätty aktiivista sessiota tietokantaan.

      6. Käyttäjä ohjataan videoistunnon käyttöliittymään, missä hän voi nähdä ja kuulla osallistujien videosyötteet. Tämä toimii normaalisti, jos Channel Backend lopettaa toiminnan kesken istunnon.

      7. Session päätyttyä käyttäjä lopettaa istunnon. Käyttäjä näkee virheilmoituksen. Virhetilanne syntyy Producer Frontendin pyrkiessä lopettamaan videoistunnon ja lähettämään tiedon Channel API:lle .Ohjelman aktiivinen istunto ei muutu entiseksi istunnoksi vaan pysyy aktiivisena, ja osallistujat jäävät aktiiviseen istuntoon.

   B. **Producer Frontend ei ole käynnissä**

      Testauksessa oleva palvelu ei ole käynnissä, joten käyttötapauksen kaikki kohdat tuottavat virhetilanteen.

   C. **Webshop ei ole käynnissä**

      1. Käyttäjä siirtyy onnistuneesti palveluntuottajien käyttöliittymän kirjautumissivulle.

      2. Käyttäjä kirjautuu onnistuneesti sisään ja ohjataan palveluntuottajien käyttöliittymään.

      3. Kaikki palvelut ja ohjelmat, jotka ovat käyttäjän hallinnoimia, sekä ohjelmien tilaajat näkyvät käyttöliittymäsivulla.

      4. Käyttäjä painaa ”Avaa”-nappia aloittaakseen videoistunnon, ja hänet ohjataan valmistautumisnäkymään.

      5. Istunto alkaa, ohjelmalle lisätään aktiivinen istunto, ohjelmaan osallistuville lähetetään kutsut istuntoon liittymiseksi.

6. Käyttäjä ohjataan onnistuneesti videoistunnon käyttöliittymään, missä hän voi nähdä ja kuulla osallistujien videosyötteet.

4.4 Testauksen tulokset ja yhteenveto

Luotettavuuskatsauksessa paljastui joitakin virhetilanteita eri palveluiden ollessa toimimattomia. Eniten virhetilanteita esiintyi käyttötapauksessa 1. Käyttäjä ostaa Sanosten verkkokaupasta palvelun, jolloin Channel Backendin toiminnon lakkaaminen aiheutti kaksi virhetilannetta ja Producer Frontendin toiminnon lakkaaminen aiheutti yhden virhetilanteen. (Kts. taulukko 1.)

Tämä verkkokaupasta tapahtuva tärkeä toiminto on siis riippuvainen kahdesta muusta mikropalvelusta, eikä käsittele näissä tapahtuvia virheitä tai katkoksia hienovaraisesti.

	A. Channel Backend ei ole käynnissä.	B. Producer Frontend ei ole käynnissä.	C. Webshop ei ole käynnissä.
1. Käyttäjä ostaa Sanosten verkkokaupasta palvelun	5,7	5	X
2. Palveluntarjoaja käynnistää videoistunnon	5,7	X	-

Taulukko 1. Virhetilanteet käyttötapauksittain, kun laajennetuissa käyttötapauksissa määritellyt ongelmat A, B tai C toteutuvat. ”X” tarkoittaa kaikkien kohtien aiheuttavan virhetilanteen, ”-” tarkoittaa virhetilanteiden puuttumista. Numerot viittaavat ko. käyttötapausten kohtiin, joissa ilmeni virhetilanteita. Rivit kuvaavat käyttötapauksia ja sarakkeet laajennettujen käyttötapausten virhetilanteita.

4.5 Johtopäätökset

Järjestelmä toimi sulavasti monessa tapauksessa, joissa eri palvelut eivät olleet käynnissä. Ilmeni kuitenkin joitain tapauksia, jotka kaipaavat parannusta. Käyttäjän nähtäväksi ilmeni usein virheilmoitus, joka näytti liian paljon informaatiota loppukäyttäjälle. Tällaisiin tilanteisiin kaivataan loppukäyttäjälle sopiva virheilmoitus sekä tarkempia virhetarkistuksia kyseisiin virhekohtiin.

Varsinkin verkkokaupan virhehallintaan tulisi keskittyä ostoprosessissa, palvelun tulisi olla itsenäisempi sekä käyttäytyä hienovaraisemmin, jos muut palvelut ovat nurin.

5 Yhteenveto ja ajatuksia

Työ oli monitarkoituksellinen. Tarkoituksena oli luoda katsaus mikropalveluarkkitehtuuriin sekä tutkia Sanoste Oy:n arkkitehtonista toteutusta. Työssä oli myös tarkoitus tehdä luotettavuusarviointi Sanoste Oy:n mikropalveluja käyttävästä järjestelmästä, luoden kuvan siitä, miten virhealtis järjestelmä on, jos erinäiset palvelut ovat nurin tai toimivat väärin.

Katsastus mikroarkkitehtuuriin kokosi yhteen lyhyesti, mistä mikroarkkitehtuurissa on kyse, ja omani ymmärrys aiheesta syveni selvästi. Toivottavasti katsastuksesta on hyötyä muillekin.

Sanoste Oy:n arkkitehtuuria käsittelevä katsaus näytti, miten mikroarkkitehtuuria voidaan soveltaa kasvuyrityksen sovelluskehityksessä. Järjestelmän tutkiminen laajensi omaa ymmärrystä järjestelmästä, ja toivottavasti auttaa tulevia työntekijöitä sekä aiheesta kiinnostuneita.

Luotettavuusarvioinnissa onnistuttiin paikantamaan virhetilanteita, joiden käsittelyyn tulisi kiinnittää huomio ja joita onkin jo ryhdytty korjaamaan. Alun perin työssä oli ajatus tarkastaa myös erinäisten virhetilanteiden yhdistelmiä sekä muidenkin virhetilanteiden vaikutuksia kuin tilanteita, joissa palvelut ovat poissa käytöstä, mutta aika ja resurssit eivät riittäneet niiden sisältämiseen tähän työhön. Näitä tutkimuksia on jatkettava tulevaisuudessa.

Vaikeuksia työhön aiheutti järjestelmän jatkuva, joskus nopeakin muutos. Toinen työn tekemistä vaikeuttava seikka oli se, että suuri osa analyysistä koski ohjelmistokoodia, jota ei haluttu asettaa näytille julkisesti.

Työssä onnistuttiin saavuttamaan asetettuja tavoitteita, ymmärrys mikropalveluista ja yrityksen järjestelmästä kasvoi henkilökohtaisesti ja työyhteisössä. Myös luotettavuustutkimuksessa paikannettiin kohtia järjestelmässä, jotka kaipaavat parannusta.

Lähteet

1 Martin Fowler, James Lewis. 2014. Microservices. Verkkojulkaisu. https://martinfowler.com/articles/microservices.html Luettu 14.10.16.

2 Nguyen Quang Tung. 2015. Microservice Architecture Style. Esitelmä. https://ftri.fpt.edu.vn/wp-content/uploads/2015/03/Microservice-Architecture-v0.8.pdf Luettu 14.10.16.

3 Phil Calçado. 2014. Building Products at Soundcloud – Part 1: Dealing with the Monolith. Verkkojulkaisu. https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-1-dealing-with-the-monolith Luettu 16.6.2017.

4 Chris Munns. 2015. Microservices at Amazon. Esitelmä. https://www.slideshare.net/apigee/i-love-apis-2015-microservices-at-amazon-54487258 Luettu 16.6.2017.

5 Kevin Goldsmith. 2016. Microservices @ Spotify. Esitelmä. https://blog.kevingoldsmith.com/2016/03/18/microservices-at-spotify-from-goto-berlin/ Luettu 16.6.2017.

6 Einas Haddad. 2015. Service-Oriented Architecture: Scaling the Uber Engineering Codebase as we Grow. Verkkojulkaisu https://eng.uber.com/soa/ Luettu 16.6.2017.

7 Zalando: Restful-API-Guidelines. Verkkojulkaisu. https://zalando.github.io/restful-api-guidelines/ Luettu 16.6.2017.

8 Allen Wang, Sudhir Tonse: Announcing Ribbon. 2013. Tying the Netflix Mid-Tier Services Together. Verkkojulkaisu. https://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html Luettu 16.6.2017.

9 James Lewis. 2012. Micro Services – Java the Unix way. Esitelmä. https://2012.33degree.org/pdf/JamesLewisMicroServices.pdf Luettu 14.10.2016.

10 Fred George. 2012. MicroService Arcitechture. Esitelmä. https://www.slideshare.net/fredgeorge/micro-service-architecure Luettu 14.10.2016.

11 Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara ym. 2016. Microservices: yesterday, today, and tomorrow. Tieteellinen julkaisu. Luettu 14.10.16.

12 Martin Fowler. 2016. GOTO 2014 - Microservices. Esitelmä. https://www.youtube.com/watch?v=wgdBVIX9ifA. Luettu 21.10.16.

13 Bob Familiar. 2015. Microservices, IoT, and Azure. Kirjajulkaisu. Luettu 21.10.16.

14 Dick Hardt. The OAuth 2.0 Authorization Framework. 2012. Verkkojulkaisu. https://tools.ietf.org/html/rfc6749#section-1.3 Luettu 24.3.2017.

15 Amazon EC2 Container Service Developer Guide (API Version 2014-11-13) – Docker Basics. 2014. Verkkojulkaisu. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html Luettu 19.5.2017.

16 SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems. 2014. Verkkojulkaisu. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems Luettu 7.10.2017

17 Amazon Relational Database Service (RDS). 2017. Verkkojulkaisu. https://aws.amazon.com/rds/ Luettu 7.10.2017.