Haaga-Helia
ammattikorkeakoulu Oy

Kuhina-palvelun siirto Microsoft Azureen

Aleksi Rossi

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä
Aleksi Rossi

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Kuhina-palvelun siirto Microsoft Azureen

Sivu- ja liitesivumäärä
28 + 1

Tämä opinnäytetyöprojekti on Roger Studio Oy:lle toteutettava toimeksianto, jonka tarkoituksena on siirtää yrityksen luoma Kuhina-sovellus Microsoft Azureen isännöitäväksi App Service -palvelulla.

Opinnäytetyön tavoitteina on parantaa sovelluksen skaalautuvuutta, tietoturvaa ja ylläpidettävyyttä. Sovelluksen aikaisempi arkkitehtuuri, jossa palvelu isännöidään yhdellä palvelimella, koettiin ongelmalliseksi tietoturvan ja skaalautuvuuden osalta. Siirtämällä sovellus Microsoft Azureen voidaan palvelusta tehdä dynaamisesti skaalautuva ja kyetään fyysisesti erottamaan sovelluksen komponentit toisistaan.

Projektin aikana sovelluksen arkkitehtuuri muokattiin Azuren App Service -palvelulle sopivaksi, jonka jälkeen sovelluksen komponentit siirrettiin yksi kerrallaan Azureen. Prosessi dokumentoitiin ja työstä saatavat hyödyt kuvattiin. Työllä myös mahdollisestetaan sovelluksen kehittäminen usealle asiakkaalle sopivaksi käyttäen yhtä ympäristöä.

Työn ajankohta sijoittui keväälle 2017. Työ valmistui projektisuunnitelmassa määritetyn aikataulun puitteissa. Kaikki määritetyt tavoitteet saavutettiin ja työ mahdollistaa sovelluksen jatkokehityksen miltei globaaliksi palveluksi.

Asiasanat
Microsoft Azure, Web App, PaaS, Kuhina

Sisällys

1 Johdanto

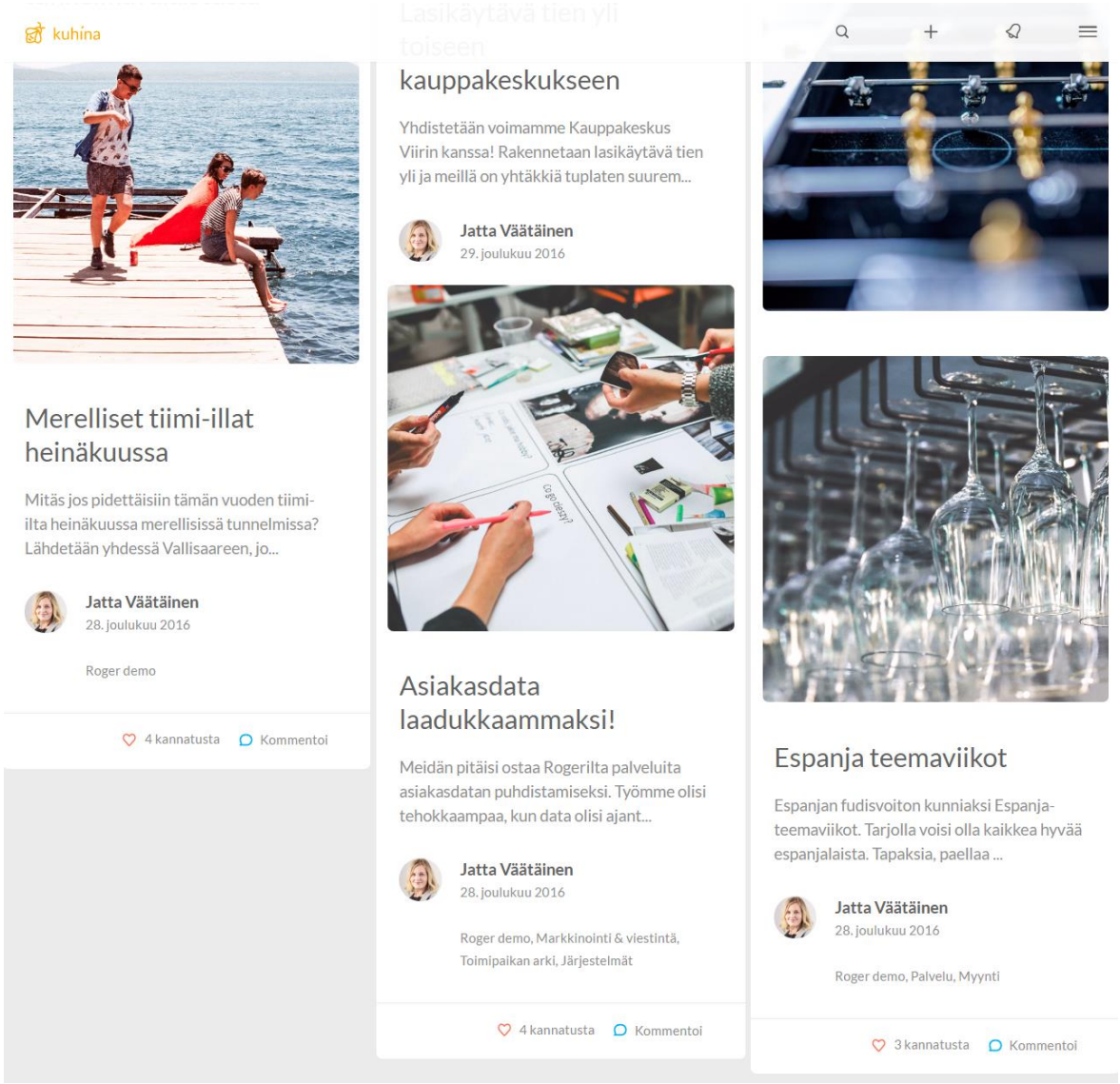
Opinnäytetyön päämääräinen tarkoitus on uusia Kuhina-palvelun arkkitehtuuri PaaS alustoille sopivaksi ja siirtää se tavallisesta yhdellä palvelimella toimivasti ympäristöstä Microsoft Azure-pilvipalveluun hyödyntäen alustan App Service -palvelua. Työ toteutetaan toimeksiantona Roger Studio Oy:lle.

Opinnäytetyön tehtävä on uudistaa palvelun arkkitehtuuri, suorittaa sen siirto Azureen, dokumentoida tämä prosessi sekä kuvata tästä seuraavat hyödyt. Tämän lisäksi tavoitteisiin kuuluu kartoittaa pilvipalveluiden etuja, verrata IaaS ja PaaS -palveluiden tarjoamia hyötyjä ja perustella valittujen teknologioiden käyttö.

Työ on rajattu kuvaamaan Azurea yleisellä tasolla sekä käsittelemään yksityiskohtaisemmin käytettyä App Service-palvelua. Kuhina-palvelu esitellään pinnallisesti mutta teknisempää kuvausta palvelun toiminnasta ei anneta.

Azure on Microsoftin maailmanlaajuinen julkinen pilvipalvelu, joka toimii alustana mm. virtuaalipalvelimille, tietokannoille, varastotilalle ja monelle muulle palvelulle kuten esimerkiksi Microsoft Dynamics CRM:lle.

Kuhina on yhteisöllinen ideointi ja palaute -palvelu, jonka on tarkoitus mahdollistaa nopea ja kollektiivinen ideointi, välitön palaute ja suora kommunikaatio päättävän tahon kanssa.



Kuva 1. Kuhina-palvelun verkkokäyttöliittymä.

Kuhinan ulkoasu ja toiminnallisuus on hakenut inspiraatiota sosiaalisen median palveluista ja sen tarkoitus on tehdä kommunikaatiosta ja ideoinnista mahdollisimman helppoa ja suoraa. Käyttöliittymässä kaikki käyttäjät voivat ”kuhista” omia ideoitaan, jotka tulevat kaikille näkyviin ja kommentoitavaksi. Uusista ideoista lähtee viesti suoraan päättävälle taholle, joka voi kommentoida ja ilmoittaa otetaanko idea käyttöön. Kuhinasta voit lukea lisää seuraavasta osoiteesta: https://rogerstudio.fi/kuhina/

Opinnäytetyössä aluksi käsitellään työn tietotaustat, tarkoituksena selventää miksi juuri näihin teknologioihin ja ratkaisuihin on päädytty. Tämän jälkeen esitellään palvelun arkkitehtuuri ja siihen tehdyt muutokset. Seuraavaksi dokumentoidaan vaiheittainen siirto Azureen ja tähän liittyvät askeleet. Lopuksi käsitellään opinnäytetyöstä saavutettavat hyödyt ja mahdollinen jatkokehittäminen.

2 Tietoperustat

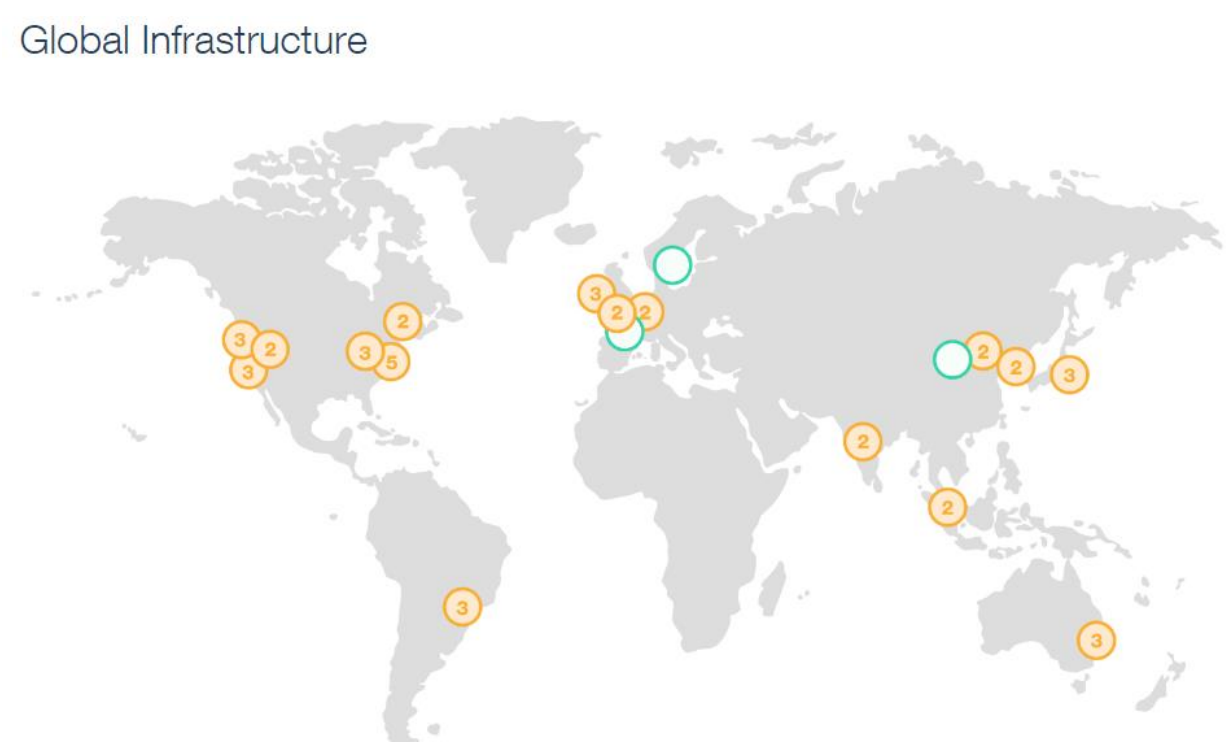
2.1 Alusta

Palvelualustaksi työhön on valittu Microsoft Azure. Azure on vuonna 2010 julkaistu Microsoftin julkisen pilvipalvelualusta, joka sisältää n. 70 eri tuotekokonaisuutta virtuaalipalvelimista varastotilaan ja hajautettuun sisällönjakeluun. (Microsoft Azure: Cloud Computing Platform & Services 2017.) Työssä käytetään pääasiassa virtuaalipalvelimia, App Service -sovelluspalvelua ja Azure SQL tietokantoja. Muita vaihtoehtoja alustaksi olisi ollut esimerkiksi Amazon Web Service (AWS) ja Google Cloud Platform (GCP).

Amazon lanseerasi AWS-palvelun vuonna 2006, huomattavasti ennen kilpailijoitaan. Tämän seurauksena AWS mielletään usein luotettavimpana palveluna, kattaen suurimman valikoiman palveluita usein myös halvimpaan hintaan. AWS rakentuu 42:sta Availability Zone nimisestä alueesta, jotka jakautuvat yhteen tai useampaan konesaliin. Nämä alueet ovat jaettu 16 maantieteelliseen sijaintiin ympäri maailmaa. AWS aloitti tarjontansa puhtaasti IaaS palveluna, ja palvelun laskennallisen kapasiteetin arvioidaan olevan enemmän kuin kilpailijoillaan yhteensä. (Butler 2017, 1.) Tähän perustuen puhtaasti IaaS palveluita varten AWS tuntuu olevan paras vaihtoehto. Yleisesti ottaen AWS mielletään pilvipalveluista turvallisimpana ja helpoimpana valintana. Suurimpana negatiivisena puolena esiintyy alun oppimiskäyrä, johon vaikuttaa palvelun laajuus, jatkuva kehittyminen ja epäintuitiivinen nimeämispolitiikka.

Microsoftin Azure on AWS:n jälkeen merkittävin pilvipalvelualusta. Vuonna 2010 julkaistu palvelu on viime vuosina kasvanut suurempaa vauhtia kuin muut kilpailijansa. (Panettieri 2016.)

Azurella on kokonaisuudessaan 34 konesalia, joista jokainen eri maantieteellisessä sijainnissa. Azure on maantieteellisesti hieman laajemmin levinnyt kuin AWS, joka kuitenkin kattaa suuremman määrän konesaleja. Tämä esiintyy vahvimmin USA:n keskiosissa, Kiinan itäpuolella ja Intiassa, joissa Azurella on AWS:ää vahvempi vaikutuspiiri. (Azure Regions 2017.) (Regions and Availability Zones 2017.)



Kuva 2. AWS konesalien levinneisyys. Oranssi pallo kattaa aina vähintään 2 konesalia, vihreät suunnitteilla olevia. (Regions and Availability Zones 2017.)

Kuva 3. Azure konesalien levinneisyys. Vihreät pallot ovat konesaleja, siniset suunnitteilla olevia. (Azure Regions 2017.)

Palvelutarjonnassa AWS ja Azure ovat miltei identtiset, joskin Azure on tarjonnassaan keskittynyt hieman enemmän Microsoft tuotteisiin, kun AWS kattaa tasapuolisesti myös avoimen lähdekoodin tuotteita. Azuren suurimpana etuna kilpailijoihinsa onkin integroitavuus ja tuki Microsoftin tuotteiden ja palveluiden kanssa, kuten esimerkiksi Office 365, joka käyttää suoraan Azure Active Directoryä käyttäjäkantana. (Butler 2016, 2.)

Samoin kuin Azure, Googlen GCP aloitti alun perin lähinnä PaaS -palveluihin keskittyneenä pilvipalveluna. Vuonna 2011 aloittanut palvelu on sittemmin lisännyt IaaS tarjontaansa merkittävästi, ja keskittynyt varsinkin konttiteknologian tukemiseen. Siinä missä Azure tukee erinomaisesti Microsoftin palveluita, myös GCP tuo etuja Google SaaS palveluiden käyttöön. GCP on AWS:ää ja Azurea molempia jäljessä sekä palvelutarjonnassa että konesalien määrässä. Google on kuitenkin suorittanut merkittäviä investointeja palveluun viime vuosina ja palvelun odotetaan kasvavan. (Butler 2016, 3.)

Yrityksessä päädyttiin ottamaan Microsoft Azure käyttöön IaaS -palvelualustaksi jo n. puoli vuotta ennen opinnäytetyön aloittamista. Koska lisäkokemus Azuren tarjonnasta ja käytöstä olivat merkittävä osa opinnäytetyön tavoitteita, valittiin Azure alustaksi työtä varten testaamatta muita vaihtoehtoja.

Päämääräinen syy Azuren alkuperäiseen valintaan pilvipalveluiden välillä pohjautui laajempaan koko Yritystä koskevaan Microsoftin tuoteperheeseen siirtymiseen. Yrityksessä otettiin käyttöön Microsoftin Office 365 ja Dynamics CRM palvelut, joiden lisäksi haettiin läheisempää yhteistyötä Microsoftin kumppanina. IaaS -tarjontaa varten yrityksessä oltiin aikaisemmin testattu AWS:ää, mutta tässäkin Azure todettiin käyttäjäystävällisemmäksi. Pienemmistä konesalipalveluista haluttiin siirtyä kattavampaan pilvipalveluratkaisuun, joka takaisi vakaamman palvelin-infrastruktuurin, kattavamman tuotekirjon ja paremmat hallintamahdollisuudet. Tärkeänä pidettiin myös hyvää tukea Microsoft-tuotteille, monia PaaS -palveluita, kuten Azure SQL tietokannat, sekä integroitavuutta muihin SaaS palveluihin kuten PowerBI, Dynamics CRM ja Salesforce.

2.2 IaaS vs PaaS

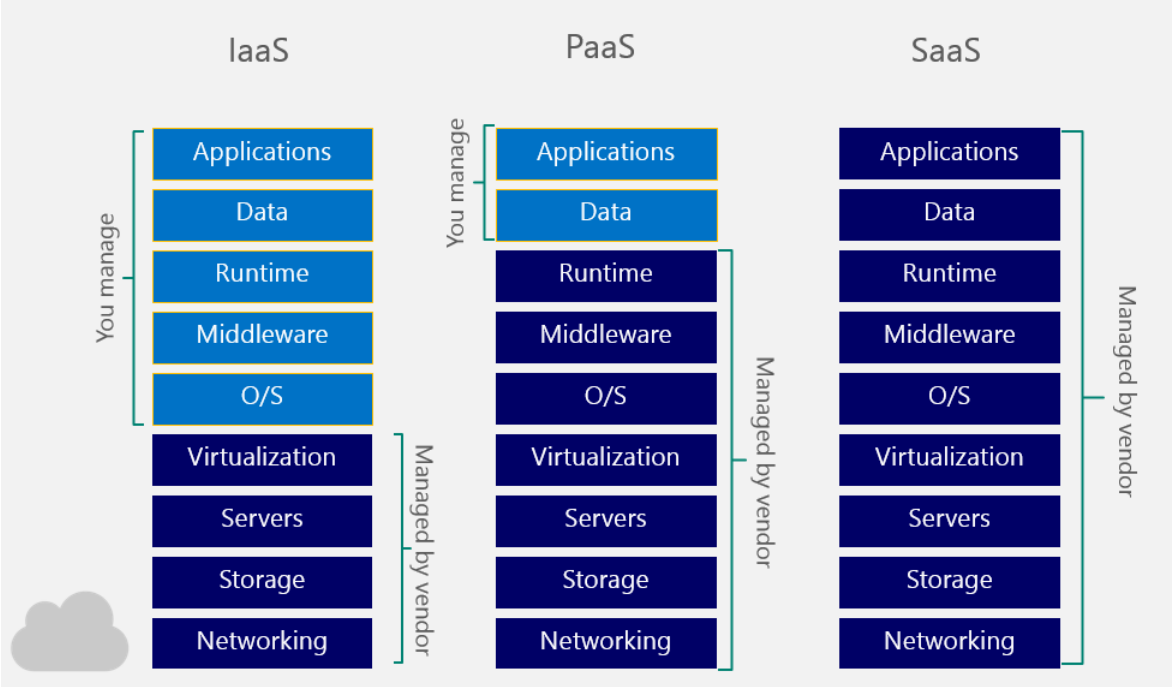
Infrastructure as a Service, eli infrastruktuuri palveluna, tarkoittaa palveluntarjoajan laitteistojen ostamista yrityksen käyttöön palvelumallina. Käytännössä tämä esiintyy yleensä palveluntarjoajan virtualisoituna konesalina, jota voidaan hallinnoida verkon yli. (Eronen 2016.) IaaS -palveluiden tarkoitus on häivyttää fyysiset laitteet ja niihin liittyvä hallinta, ylläpito ja säilytys. IaaS -palveluilla voidaan vähentää laitekustannuksia, laitteiden

vaatimia yritystiloja ja ylläpitoon käytettävää aikaa. Tämän lisäksi voidaan mahdollistaa infrastruktuurin helpompi hallinta ja parempi skaalautuvuus.

Pilvipalvelumallissa muodostetaan ympäristö, joka on joko jaettu useiden asiakkaiden kesken tai rajattu vain yhdelle asiakkaalle kerrallaan. Tätä jaottelua kutsutaan yksityiseksi ja julkiseksi pilvipalveluksi. (Tamara 2017.) IaaS -palveluiden negatiivisina puolina esiintyy hallinnan heikkeneminen laitteistotasolla. Yrityksen laitteistolliset vaatimukset täytyy sovittaa palvelun mukaan. Mahdolliset käyttökatkot ovat myös täysin palveluntarjoajasta riippuvaisia. Usein kuitenkin palvelut määrittävät palvelusopimuksessa luvatun käytettävyystason, esimerkiksi Azuressa tämä on monessa tuotteessa 99.95%.

Platform as a Service, eli sovellusalusta palveluna, on palveluntarjoajalta vuokrattava alusta sovellusten, verkkosivujen tai ohjelmistojen isännöimiseen ja ajamiseen. Tyypillinen esimerkki tästä on Wordpress-sivuston isännöinti webbihotellissa. IaaS -malliin verrattuna PaaS -palveluissa häivytetään käyttöjärjestelmätaso palveluntarjoajan ylläpidettäväksi, samaten myös verkkoyhteydet ja alla oleva tietoturva kuuluvat palveluntarjoajan vastuisiin. (Eronen 2016.) Tämän seurauksena PaaS -alustat saadaan käyttöön hyvin nopeasti eikä palvelussa tarvitse suorittaa käyttöjärjestelmätason asennuksia. Itse hallinnoidaan puhtaasti sovellustasoa.

PaaS -palvelut mahdollistavat nopeat ja helppokäyttöiset ympäristöt, jotka eivät vaadi ylläpitoa. Hallittavuus palveluissa on kokonaisuudessaan rajoitetumpaa, kun pääsy käyttöjärjestelmätasolle menetetään. Yleensä palveluntarjoajat mahdollistavat eri tapoja hallita alustaa kuten SFTP, SSH tai selainpohjainen ratkaisu. PaaS -palvelut mahdollistavat hyvin tehokkaan skaalautuvuuden.

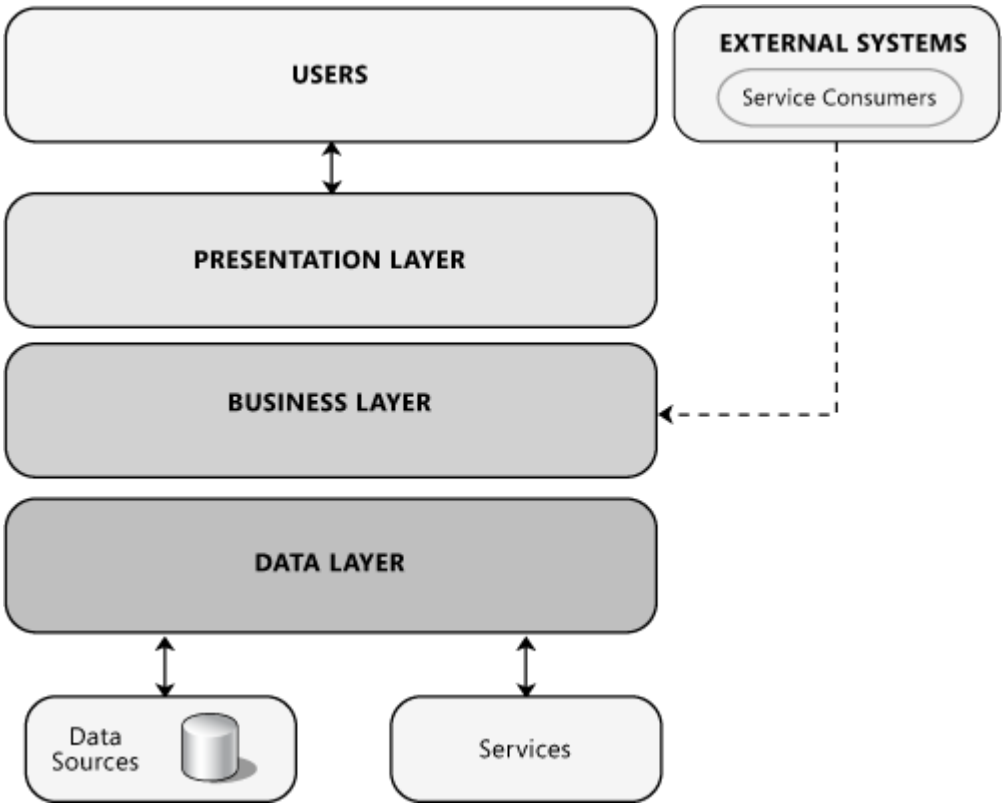


Kuva 4. IaaS, PaaS ja SaaS -palveluiden sisältöjen eroavaisuudet. (Tamara 2017.)

Molemmat PaaS ja IaaS luovat mahdollisuuksia kustannustehokkuuteen, kunhan palvelun käyttö ja hinnoittelu on toteutettu järkevästi. Kustannuksia voidaan säästää sekä puhtaasti rahallisesti että ylläpitoon ja asennuksiin liittyvissä työmäärissä. Molemmat tarjoavat elastisuutta resurssien käyttöön ja skaalautuvuuteen. (Tamara 2017.) Usein ratkaisu parhaasta palvelusta riippuu vaadittavasta hallinnan määrästä sekä yrityksen IT-resursseista. Vaatimukset pitää pystyä suhteuttamaan tarjontaan. Suurten pilvipalveluiden takia palveluiden hinnat ovat riittävän alhaalla ja tuotekirjo sen verran kattava, että miltei jokainen yritys pystyy hyödyntämään palveluita jossain määrin.

2.3 Arkkitehtuuri

Kolmitasoarkkitehtuurilla tarkoitetaan sovelluksen jakamista loogisiin kokonaisuuksiin, joita kutsutaan tasoiksi. Erottelu on puhtaasti loogisella tasolla, ja eri tasot voivat sijaita eri palvelimilla tai verkoissa. Klassisesti tasot jaetaan seuraavasti: esityskerros (presentation), logiikkakerros (business) ja datakerros (data). (Microsoft MSDN tietovarasto Chapter 5.)



Kuva 5. Kolmitasoarkkitehtuurin tyypillinen malli. (Microsoft MSDN tietovarasto Chapter 5.)

Esityskerros vastaa käyttöliittymästä ja datan esityksestä käyttäjälle. Esityskerros yleisimmin esiintyy sovelluksen käyttöliittymänä tai verkkosivuna. Käyttöliittymään on usein rakennettu ominaisuuksia ja rajapintoja, joilla voidaan ottaa yhteyttä muihin tasoihin, mitä kautta sovellusta voidaan hallita. (Microsoft MSDN tietovarasto Chapter 5.)

Logiikkakerros on klassisesti vastuussa sovelluksen ydintoiminnoista ja liiketoiminnan toimintalogiikasta. Logiikkakerros voi jakautua useampaan komponenttiin kuten erilaiset rajapinnat, toimintalogiikan hallinta ja työnkulun hallinta. Logiikkakerros on myös vastuussa kommunikaatiosta esitys- ja datakerrosten välillä, sekä yleensä hallinnoi yhteyksiä muihin järjestelmiin. (Microsoft MSDN tietovarasto Chapter 7.)

Datakerros pitää sisällään sovelluksessa käytettävän datan ja sen käsittelyyn ja hakemiseen liittyvän logiikan. Kerros rakentuu yleensä rajapinnasta, jota kutsutaan logiikkakerroksesta, ja tietokannasta. Kerroksen vastuulla on datan saatavuus ja eheys. (Microsoft MSDN tietovarasto Chapter 8.)

Sovelluksen jakaminen loogisiin tasoihin helpottaa hahmottamaan komponenttien tehtäväkokonaisuuksia. Koska tasot ovat omia kokonaisuuksiaan, voidaan niitä kehittää eteenpäin itsenäisesti, muuttaen muissa tasoissa vain viitaukset muokattuun tasoon.

Tämän lisäksi tasoja on helppo käyttää uudelleen ja siirtää ympäristöstä toiseen. Sovellus on myös skaalautuvampi, sillä jokaista tasoa on mahdollista skaalata itsenäisesti, jolloin pullonkaulojen muodostuminen on helpompi ehkäistä. (Microsoft MSDN tietovarasto Chapter 5.)

2.4 Azure Web App

Azure Web App on osa Microsoft Azuren App Service palvelua. Web App on PaaS -sovellusalusta, jolla voidaan isännöidä verkkosivuja ja sovelluksia. Azuressa Web App:in pohjalla olevat resurssit toteutetaan jaetuilla tai dedikoiduilla virtuaalikoneilla palvelutasosta riippuen. Palvelutasosta huolimatta ajettava sovellus on eristetty jokaista asiakasta varten omaan virtuaalikoneeseen. Tällä hetkellä Web App:it tukevat seuraavia kieliä ja viitekehyksiä: ASP.NET, Node.js, Java, PHP, Python ja HTML. Azure App Service palvelu tarjoaa monia PaaS -alustasta muodostuvia etuja sekä käyttää hyväkseen Microsoftin globaalia konesalia infrastruktuuria. (Web Apps Overview 2017.)

Azure App Service palvelu on erinomaisesti skaalautuva, Web App:eja voidaan skaalata sekä horisontaalisesti että vertikaalisesti. Tämä tarkoittaa, että alustan resursseja voidaan laskea ja nostaa dynaamisesti, ja alustasta voidaan luoda uusia instansseja tarpeen mukaan. Web App:eja voidaan skaalata manuaalisesti käyttöliittymän kautta tai automaattisesti, esimerkiksi alustan laskentatehon mukaan. (Web Apps Overview 2017.)

Koodin ja sovellusten käyttöönoton alustassa voi suorittaa monilla eri tavoin. Web App:iin voidaan ottaa suora FTP- tai FTPS-yhteys, sitä voidaan hallinnoida powershellin tai Azure-CLI:n kautta ja se tukee tiedostojen latausta suoraan yleisistä versionhallintapalveluista kuten Githubista tai Bitbucketista. (Web Apps Overview 2017.) Asetukset voi määrittää niin, että Web App hakee versionhallinnasta aina uusimman version muutoksen jälkeen.

Web App:it antavat myös mahdollisuuden ajaa sovelluksesta rinnakkaisia instansseja niin kutsutuilla Deployment Slot:eilla. Jokaisella instanssilla on oma nimi, joka jaetaan Azuren nimipalvelimille. Näihin instansseihin voidaan tuoda esimerkiksi testausta varten uusi versio sovelluksesta ennen tuotantoon siirtymistä. Kun versio on todettu toimivaksi, voidaan testi-instanssi vaihtaa tuotantoon nappia painamalla. (Set up staging environments in Azure App Service 2017.)

Microsoft takaa palvelusopimuksessa 99.95% tavoitettavuuden App Service palvelulle (SLA for App Service 2017). Tämän lisäksi on mahdollista hyödyntää Microsoftin laajaa

konesaliverkostoa ja isännöidä sovelluksia ympäri maailmaa. App Service on myös rakennettu integroitumaan suureen määrään muita julkisia palveluita kuten Office 365, Facebook tai Salesforce. (Web Apps Overview 2017.)

App Service -palvelun pohjalla olevat virtuaalikoneet ovat lähtökohtaisesti Windows pohjaisia ja palvelu isännöidään IIS-verkkopalvelimella. Viime vuoden aikana Microsoft on julkaissut myös Linux-pohjaisen Web App -palvelun, joka on tämän työn aikana kuitenkin vielä Preview -tilassa, eikä sen käyttöä suositella tuotantoon.

3 Kuhina-sovelluksen siirto Azureen

Kuhina-sovelluksen siirto arkkitehtuuriin kattaa palvelun alkuperäisen arkkitehtuurin kuvauksen, Azurea varten kehitetyn arkkitehtuurin, jokaisen komponentin yksittäisen siirron Azureen sekä kuvauksen lopullisesta kokonaisuudesta ja siihen liittyvistä toimenpiteistä.

3.1 Arkkitehtuuri

Kuhina-palvelussa käytetään kolmitasoarkkitehtuuria, jossa komponentit jaetaan kolmeen loogiseen kokonaisuuteen. Nämä kokonaisuudet jaotellaan seuraavasti:

1. Esityskerros (Front End).
2. Logiikkakerros (Middleware).
3. Datakerros (Back end).

Front end:iin kuuluvat AngularJS-ohjelmistokehys, Node.js-ajoympäristö sekä ulkoinen autentikaatiopalvelu. Front end:in tehtävänä on muodostaa käyttäjärajapinta sekä autentikaatiopalvelun avulla huolehtia sisäänkirjautumisista.

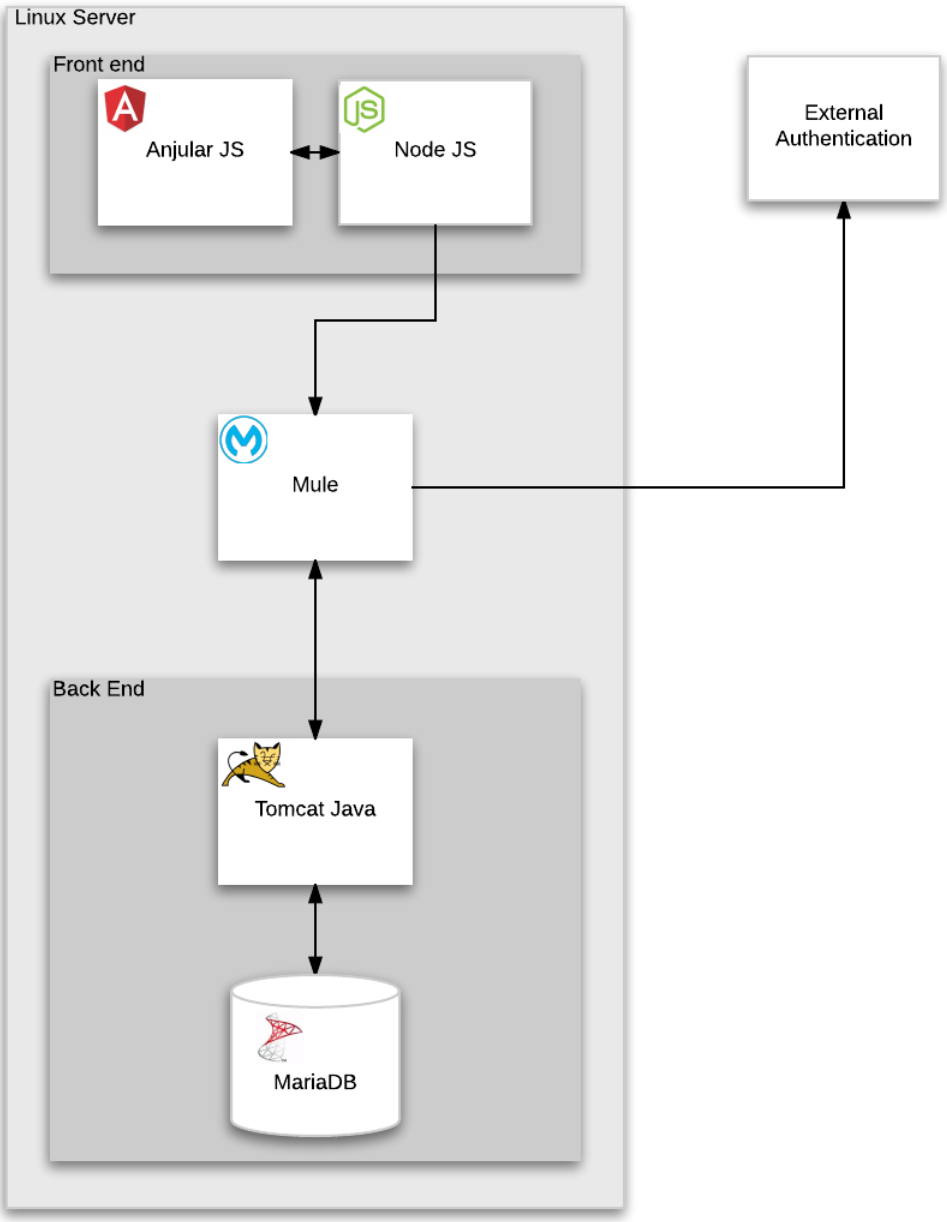
Middleware koostuu yksinomaan Mule ESB-integraatioalusta. Mulen on tarkoitus luoda API-rajapinnat Front end:in ja Back end:in välille, sekä hallinnoida integraatioita muihin järjestelmiin.

Back end sisältää Apache Tomcat-palvelimen sekä MariaDB tietokannan. Back end:in tarkoitus on hallinnoida, hakea ja tallentaa dataa.

Tämän lisäksi taustalla pyörii myös Apache-verkkopalvelin käänteisenä välityspalvelimena, tarkoituksena ohjata kaikki saapuva liikenne Node.js:lle. Linux-koneelle on myös asennettu Postfix-sähköpostipalvelin, jonka tehtävänä on hallinnoida palvelun lähtevää sähköpostia. Näitä kahta ei ole kuvattu arkkitehtuurissa, sillä ne suorittavat vain tukitoimia.

3.1.1 Aikaisempi arkkitehtuuri

Alla oleva kuva esittelee Kuhina-palvelun alkuperäisen arkkitehtuurin. Kuvassa kaikki palvelun komponentit, paitsi ulkoinen autentikaatiopalvelu, ovat osa yhtä Linux-palvelinta.



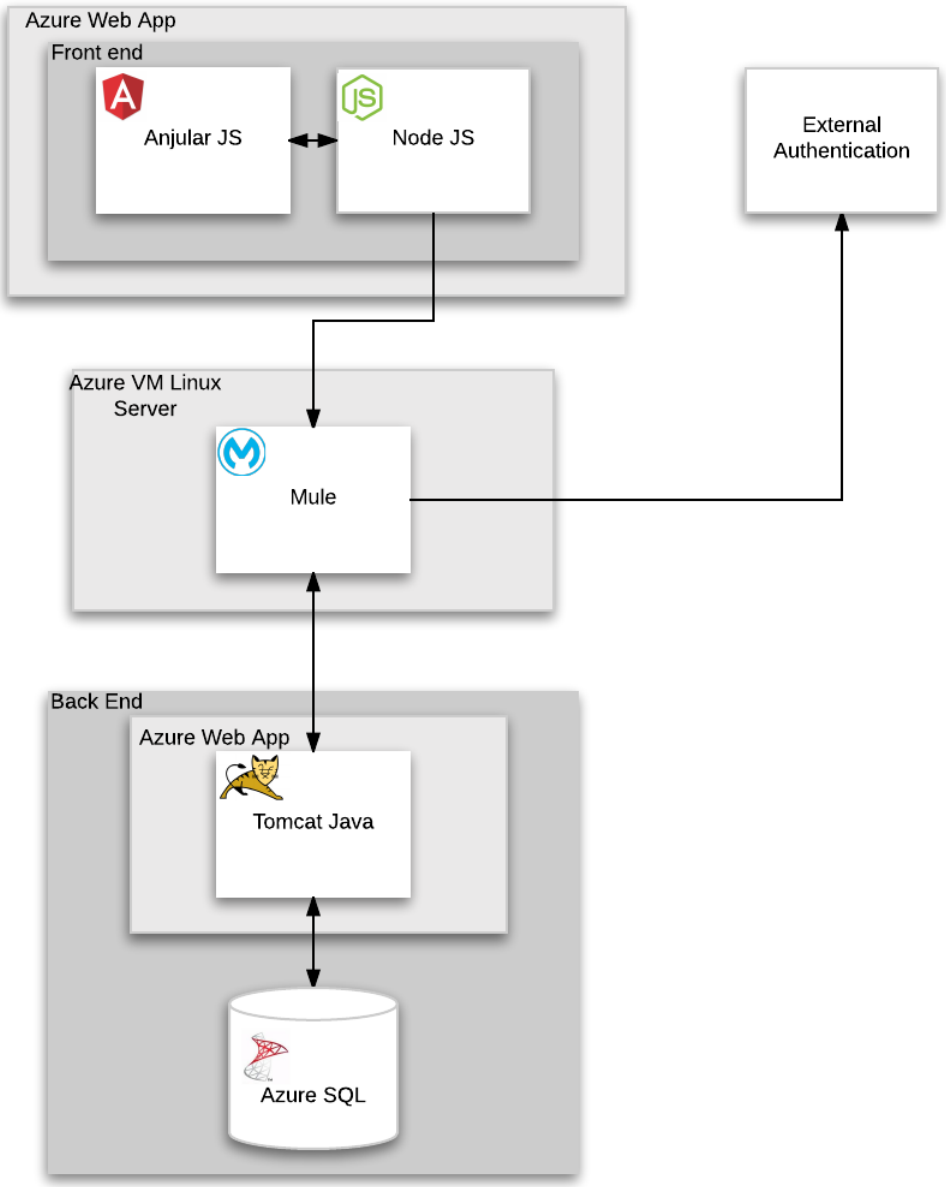
Kuva 6. Alkuperäinen arkkitehtuuri

Koska kaikki komponentit ovat yhden palvelimen varassa, on palvelun skaalautuvuus heikko. Palvelimelle voidaan lisätä tehoa, mutta tämä on manuaalinen toimenpide. Samaten resurssienkäyttöä palvelimen sisällä voi joutua erikseen optimoimaan.

Tietoturvan kannalta on myös kyseenalaista säilyttää tietokanta samalla palvelimella muiden komponenttien kanssa. Jos julkiseen verkkoon yhteydessä oleva verkkopalvelin murretaan, on hyökkääjän helpompi päästä kiinni tietokantaan ja asiakasdataan.

3.1.2 Azure arkkitehtuuri

Seuraavassa kuvassa arkkitehtuuri on muokattu vastaamaan Kuhinaa Azuressa. Aikaisempaan verrattuna, Linux-palvelin sisältää nyt vain Mulen. Front ja Back end ovat siirretty Linux-palvelimen sijaan Azure Web App:ien päälle. Samaten MariaDB on korvattu Azure SQL-tietokannalla. Linux-palvelimena käytetään Azuren Ubuntu pohjaista virtuaalikonetta.



Kuva 7. Arkkitehtuuri Azuressa

Tässä mallissa jokainen yksittäinen komponentti (Angular toimii Node.js:n alla) saa omat resurssinsa, mikä Web App:ien kohdalla mahdollistaa myös horisontaalisen skaalautumisen vertikaalisen ohella. Jokaista komponenttia voidaan tarvittaessa kasvattaa itsekseen, jolloin pystytään välttymään pullonkauloilta. Tämän lisäksi resursseja ei tarvitse jakaa käyttöjärjestelmätasolla erikseen.

3.2 Siirron dokumentaatio

Lähtökohtina palvelun siirtoa varten opinnäytetyössä on:

- Toimiva ympäristö.
- Sovellusten koodi versionhallinnassa Githubissa.
- Arkkitehtuurikuvaus.
- Tarvittavat tietotaustat Azure Web Appeista.

Aluksi tulee luoda ensimmäinen Web App Azureen. Azuressa Web App:eja on mahdollista luoda eri tavoilla. Voidaan luoda tyhjä Web App ja sitten itse konfiguroida siihen tarvittavat kirjastot ja asetukset. Toinen vaihtoehto on luoda template-pohjalta valmis Web App tarvittavilla asetuksilla.

Kuten tietoperusteissa kerrottiin, Azure Web App:it toimivat perinteisesti IIS-verkkopalvelimen päällä. Uutuutena Azuressa on nykyään myös mahdollisuus käyttää Linux-pohjaisia Web App:eja. Tämä olisi Kuhinan kannalta järkevämpää, sillä se on alun perin rakennettu Linux-palvelimen päälle, joten toiminnallisuus varmempaa. Valitettavasti Linux Web App:it ovat kuitenkin vielä tämän työn aikana preview-tilassa, jota ei suositella tuotantokäyttöön, eivätkä ne sisällä kaikkia ominaisuuksia.

Tarjotuista Web App vaihtoehdoista työhön valittiin template-pohjaiset Web App:it. Nämä luovat pisimmälle konfiguroidun alustan, joka on helpointa ottaa käyttöön ja muistuttaa eniten Linux-pohjaista ympäristöä.

3.2.1 Front end

Front end:in kanssa käytetään Microsoftin omaa Node JS Empty Web App -pakettia. Tämä löytyy Azuressa suoraan kaupasta kategorian Web + Mobile alta.

Uutta Web App:ia luodessa täytyy määritellä nimi, Azuren resurssiryhmä sekä App Service Plan, joka määrittää Web App:in resurssit ja hinnan.

Kun Web App on luotu, se tulee heti käyttöön. Azure määrittää palvelulle oman nimen nimipalveluun ja ilmoittaa URL:in, jolla palveluun voidaan ottaa yhteyttä. Selaimella voidaan tarkistaa osoitteesta palvelun olevan käynnissä, joskin saadaan vain tyhjä Web App sivu.

Jotta voimme käyttää aikaisempaa JavaScript-koodia, joka on luotu Linux-pohjaiselle Node.js alustalle, pitää Web App:issa tehdä muutamia muutoksia. Iisnode-moduuli, joka mahdollistaa Node.js:n ajamisen IIS-palvelimessa, toimii osittain eri tavalla kuin tavallinen Node.js. Ensinnäkin tavallisen portin sijaan pitää käyttää process.env.PORT -muuttujaa, johon iisnode dynaamisesti määrittää kuuntelevan portin. Jos käytämme valmista Node.js Web App:ia, tämä muutos on tehty jo valmiiksi. Lähtökohtaisesti Node.js:llä ajettavan javascript-käynnistystiedoston pitää olla nimetty server.js, muutoin iisnode:lle pitää erikseen määrittää tiedoston nimi. (Janczuk 2011.) (Kampschmidt 2014.) Helpointa tässä kohtaa oli nimetä index.js -tiedosto server.js nimiseksi ja päivittää tieto package.json -tiedostoon. Iisnode eroaa muillakin tavoilla tavallisesti Node.js:stä mutta Kuhinan kannalta nämä muutokset riittivät tässä kohtaa.

Seuraava vaihe on sovelluksen käyttöönotto alustalla. Azure tarjoaa useita käyttöönottovaihtoehtoja – tiedostojen siirto FTP tai FTPS-työkaluja käyttäen, suoraan versionhallinnasta tai käyttämällä Web Deploy -ominaisuutta työkaluissa kuten Visual Studio. Koska koodi on valmiina Githubissa, voimme hakea sen Web App:illa suoraan. Tämä onnistuu valikon Deployment Options alta. Valitaan Setup -> Choose source -> Github, jonka jälkeen pitää antaa validit tunnukset Github-järjestelmään. Kun tunnukset on annettu, pystytään selaamaan ja valitsemaan haluttu arkisto sekä haara. Lopuksi sovelluksen käyttöönotto alkaa.

Koska Web App käyttää Node.js:ää, osaa se rakentaa koodin käyttövalmiiksi automaattisesti. Jos jotain ongelmia esiintyy, portaalissa on myös Console-ominaisuus, jolla saadaan komentokehote, millä pystytään ajamaan yksinkertaisia komentoja, kuten npm install, jolla voidaan korjata mahdolliset riippuvuudet. Console on myös helpoin tapa seurata Web App:in lokitiedostoja realiaikaisesti tail-komennolla.

Käyttöönotto asentaa tiedostot seuraavaan hakemistoon: /site/wwwroot/
Kaikki versionhallinnasta haetut tiedostot tallentuvat tänne. Täältä löytyy myös web.config-niminen tiedosto, joka sisältää iisnode:n asetukset. Tätä ylemmät hakemistot sisältävät logeja ja dataa sovelluksesta, verkkopalvelimesta ja käyttöönotosta. Näihin hakemistoihin pääsee kiinni edeltä mainitulla Console-ominaisuudella tai paremmin FTP/FTPS-

ohjelmilla. FTPS-käyttäjätunnukset voi määritellä kohdassa Deployment Credentials. Yhteystiedot löytyvät Properties-sarakkeen alta.

Nyt Web App on luotu, sovellus on muokattu soveltuvaksi iisnode:lle, koodi on rakennettu ja otettu käyttöön Web App:issa ja pystymme hallitsemaan tiedostoja.

3.2.2 Middleware

Alun perin myös Mule oli tarkoitus saada toimimaan Web App:in päällä. Mule voidaan upottaa Tomcat-palvelimeen ja ajaa palveluna sitä kautta. Prosessi vaatii Tomcat-asennuksen, jonne Mule-asennuksesta kopioidaan tarvittavat kirjastot. Tämän jälkeen Tomcat:in konfiguraatio-tiedostoista muokataan server.xml ja catalina.properties tiedostot käyttämään kopioituja kirjastoja ja lisätään listener-luokka, jotta Tomcat osaa käyttää Mulea. (Mulesoft.)

Prosessi on nopea, mutta Kuhinan kohdalla Muleen olisi pitänyt tehdä koodin puolella suuria muutoksia, jotta tämä olisi toiminut oikein. Tulevaisuuden kannalta olisi ollut järkevää toteuttaa muutokset, jotta Mule oltaisi myös saatu paremmin skaalautuvaksi Web App:iksi. Kuhina kehitystiimin kanssa päätettiin kuitenkin, että tässä kohtaa edetään vielä virtuaalipalvelimelle asennetulla Mulella, sillä se on komponenteista kevein ja sille riittää pitkään pelkkä vertikaalinen skaalautuminen.

Middleware siis toteutetaan virtuaalipalvelimella. Azure kauppa tarjoaa Ubuntu Linux-palvelimia, joista valitaan 16.04 versio. Palvelin lisätään Kuhinan resurssiryhmään ja verkkoon. Kun palvelin on luotu, asennetaan Mule normaalisti Kuhina-asennusdokumentaation mukaisesti. Mulen asentamista ja konfiguroimista Kuhinaa varten ei käsitellä tässä työssä.

3.2.3 Back end

Back end sisältää tietokannan ja Tomcat-palvelimen, joten siirtokin tapahtuu kahdessa osassa.

Koska Tomcatin pitää ottaa yhteyttä tietokantaan, luodaan kanta ensin. Aikaisemmin Kuhinassa ollaan käytetty MariaDB-tietokantaa, mutta Azure ei kuitenkaan natiivisti tue MySQL-tietokantoja. MySQL:ää voi kyllä käyttää Azuressa, mutta niiden tuki on heikkoa

eivätkä ne kunnolla sovellu tuotantoon. Näistä syistä siirrytään käyttämään MySQL:n sijaan Azure SQL-tietokantaa.

Luodaan siis Azure SQL-tietokanta. Tämä löytyy kaupasta Database-kategorian alta. Tietokantaa luodessa pitää määrittää tietokannan nimi, resurssiryhmä, tietokantapalvelin (Tämä on puhtaasti hallinnollinen osa, joka määrittää esimerkiksi käyttäjätunnuksen.), kollaatio sekä tietokannan teho ja hinta. Kun tietokanta on luotu, tulee se näkyviin portaaliin. Overview kohdasta saadaan auki Connection Strings, jossa kerrotaan vaadittavat tiedot tietokantaan yhteyden muodostamista varten.

Seuraavaksi luodaan tietokannalle oma käyttäjä, jotta ei tarvitse käyttää admin-käyttäjää. Kirjaudutaan Azure SQL tietokannan master-kantaan SQL Server Management Studiolla, jonka jälkeen luodaan kirjautumistunnus seuraavalla SQL lauseella:

```
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name = N'<LOGIN_AC-
COUNT>')
  CREATE LOGIN [<LOGIN_ACCOUNT>] WITH PASSWORD=N'<ACCOUNT_PASSWORD>'
GO
```

Tämän jälkeen kirjaudutaan ulos master-kannasta ja kirjaudutaan suoraan luotuun tietokantaan määrittelemällä kannan nimi Connect to database valikkoon. Tämän jälkeen voidaan luoda tietokannan käyttäjä SQL lauseella:

```
CREATE USER [<LOGIN_ACCOUNT>] FOR LOGIN [<LOGIN_ACCOUNT>] WITH
DEFAULT_SCHEMA=[dbo]
ALTER ROLE [db_owner] ADD MEMBER [<LOGIN_ACCOUNT>]
GO
```

Kuhinan tietokantarakenteet ja alustava data ajetaan sisään tietokantaan SQL-lauseilla. Nämä ovat kuitenkin tehty MySQL-kielellä ja vaativat muokkauksia. Yleisesti ottaen MySQL kääntyy MS SQL muotoon helposti. Kuhinaa varten tarvittavat tietokannan rakennemuutokset ovat seuraavat:

- Käytetään määritteen bool sijaan bit.
- MS SQL:ssä user on varattu avainsana eikä se saa olla taulun nimenä, korvataan se userdata määritteellä.
- Funktio now() pitää muuttaa funktioksi getdate().
- Kaikki lainausmerkit tulee muuttaa heittomerkeiksi (” = ’).
- SQL kielessä viittaukset tauluun tehdään database.[tablename] tyylin sijaan [database].dbo.[tablename].

- MS SQL ei tue auto_increment -kenttää vaan sen sijaan tulee käyttää identity(1,1) -funktiota.
- Funktio uuid() korvataan funktiolla newid().

Näillä muutoksilla pystytään luomaan Kuhinan tietokantarakenteet ja alustava data. Tarvittavat SQL lauseet löytyvät versionhallinnasta.

Tämän jälkeen luodaan uusi java-pohjainen Web App. Jälleen helpoimmalla päästään luomalla kaupasta Web + Mobile kategorian alta Apache Tomcat 8 Web App. Tämä luo lähimpänä Linux-palvelinympäristöä vastaan hakemistohierarkian. Web App:illa on käytössä luomishetkellä uusin Tomcat versio.

Määritetään Web App:ille jälleen nimi, laitetaan se samaan resurssiryhmään ja käytetään tässä kohtaa vielä aikaisemmin luotua App Service Plan:iä. Testivaiheessa palvelut voivat jakaa resurssit ja kustannukset, tuotantoon siirryttäessä on syytä määritellä molemmille Web App:eille oma Service Plan. App Service Plan:in voi muuttaa myöhemmin portaalista.

Koodia ei voida hakea suoraan versionhallinnasta, koska käyttöönotto Tomcat-palvelimella vaatii WAR-paketin, eli ensin on rakennettava paketti Maven-ohjelmalla. Koodi pitää sisällään ympäristömuuttujia, jotka ovat määriteltävä pakettia rakentaessa. Koska ympäristö on uusi, ovat muuttujat määriteltävä uudelleen. Tätä varten luodaan uusi environments-hakemisto, jonne kopioidaan jdbc.properties, log4j.xml ja mail.properties. Näistä log4j.xml ei tarvitse muutoksia. Jdbc.properties tiedostoon pitää kopioida tietokannan Connection Strings osiosta palvelimen osoite ja yhteysparametrit. Samaten käyttäjätunnus ja salasana on asetettava oikein ja jdbc.driver on muutettava käyttämään Azuren com.microsoft.sqlserver.jdbc.SQLServerDriver -ajuria. Mail.properties -tiedostoon pitää määrittää Azuren smtp-palvelimen asetukset. Azuressa käytössä on Sendgrid sähköpostipalvelu, jonka tiedot ja tunnukset saadaan portaalista. Lopuksi pom.xml -tiedostoon pitää määrittää uusi profiili, jotta Maven osaa käyttää uusia ympäristömuuttujia. Lopuksi voidaan rakentaa paketti Mavenilla.

Valmis WAR-paketti voidaan siirtää Web App:iin FTP:llä. Määritellään käyttäjätunnus Deployment Credentials kohdassa ja haetaan osoite Properties-valikon alta. Apache Tomcat 8 Wep App luo seuraavan hakemistorakenteen /site/wwwroot/bin/apache-tomcat-8.x.xx/. Tomcat hakemistosta löytyy hakemisto webapps, jonne WAR-paketti siirretään.

Tomcat Web App:issa toimii hyvin pitkälle samalla tavalla kuin Linux-palvelimella. Suurimpina eroina on yhden dynaamisen http portin käyttö, kaikkien muiden kuuntelevien

porttien poisto ja liikenteen rajoittuminen IPv4-protokollalle. /site/wwwroot/ hakemiston alla on web.config -tiedosto, jolla voidaan konfiguroida palvelua tarkemmin.

Kun WAR-paketti on ladattu palvelimelle, voidaan Web App uudelleenkäynnistää portaalista, jolloin Tomcat lataa paketin käyttöön. Nyt Wep App pitäisi olla toiminnassa.

Tässä kohtaa Back End:in koodissa havaittiin ongelmia toimia kunnolla Azuren MS SQL tietokannan kanssa. Asia jäi lopulta Kuhinan kehitystiimin ratkaistavaksi ja ongelma saatiin selvitettyä vasta viikkojen päästä. Lopulta syyksi havaittiin tietokannan pagination-ominaisuus, joka ei toiminut halutulla tavalla MS SQL pohjaisessa tietokannassa. Kun ominaisuus poistettiin käytöstä, saatiin Back end toimimaan.

3.3 Kokonaisuus

Tässä kohtaa kaikki komponentit ovat yksittäin siirretty Azureen. Jotta palvelu saadaan toimimaan normaalisti, pitää komponentit saada vielä toimimaan yhdessä. Julkinen palvelu ei voi myöskään toimia Azuren määrittelemällä epämääräisellä nimellä ja verkkotunnuksella, vaan Web App:ille pitää saada käyttöön Kuhinan oma verkkotunnus. Koska kyseessä on julkinen tuotantopalvelu, pitää myös huolehtia tietoturvasta.

3.3.1 Komponenttien toiminta yhdessä

Linux-palvelimessa kaikki komponentit toimivat samalla koneella samassa osoitteessa, kun komponentit ottavat yhteyttä toisiinsa, ne osoittavat localhost-osoitteeseen. Jokaiselle komponentille pitää siis määrittää julkinen osoite, missä palvelu sijaitsee. Optimaalista olisi käyttää vain privaattiosoitteita, mutta Web App:eja ei valitettavasti saa kiinnitettyä Azure VNET-verkkoihin luomatta kokonaan omaa App Service Environment -ympäristöä, joka on huomattavasti tavallisia Web App:eja kalliimpi.

Back end:issä muutokset tietokantayhteyttä varten on jo tehty. Front end:issä muutokset tulee määritellä config-node.js -tiedostoon, joka kertoo, missä Mule sijaitsee. Eli ensin täytyy hakea Linux-palvelimen julkinen osoite. Tämä löytyy helpoiten portaalista virtuaalikoneen tiedoista. Määritellään osoite config-node.js -tiedostoon ja käynnistetään Web App uudelleen. Tämän jälkeen pitää muokata Linux-palvelimella Mulen konfiguraatiotiedostossa onni-override-properties.conf Back end:in osoite kuntoon. Tähän voidaan käyttää Azuren Web App:ille luomaa nimeä, joka voidaan tarkastaa portaalista. Muutosten jälkeen käynnistetään Mule uudelleen.

Nyt liikenteen pitäisi sujuvasti kulkea Front end:in kautta Middleware:lle ja sieltä Back end:iin. Uutta ympäristöä varten täytyy vielä luoda ulkoiseen autentikaatiopalveluun uusi Client, joka määrittää hyväksytyt käyttäjät ja ohjaa liikenteen oikeaan ympäristöön.

3.3.2 Oma verkkotunnus

Jotta palvelun käyttäjät löytävät sivustolle, täytyy palvelun käyttää tunnettua nimeä ja verkkotunnusta. Kuhina-palvelulla tämä on kuhina.io. Tässä vaiheessa ympäristö on testikäytössä, joten palvelulle halutaan määrittää aliverkkotunnus.

Verkkotunnus lisätään vain Front end:in Web App:ille, sillä se sisältää käyttäjärajapinnan ja on ainoa julkiseen verkkoon auki oleva komponentti. Web App:ille verkkotunnuksen lisääminen tapahtuu Custom Domain-osion alta. Sieltä löydämme palvelulle määritetyn ulkoisen IP-osoitteen, joka on määritettävä julkiseen nimipalveluun. Nimipalveluun lisätään siis uusi CNAME-tietue, joka osoittaa halutulla nimellä portaalista löytyvään osoitteeseen.

Kun aliverkkotunnus on yleisesti todennettavissa julkisilla nimipalveluilla, voidaan tunnus ottaa käyttöön Web App:issa. Custom Domains-osion alta valitaa vaihtoehto Add hostname. Kenttään syötetään valittu nimi ja Azure tarkastaa, että nimi on todennettavissa. Tämän jälkeen valittu hostname tallentuu Web App:in käyttöön ja ohjaa suoraan isännöidylle sivustolle.

3.3.3 Tietoturva

Koska Kuhina-palvelu Azuressa toimii julkisen verkon yli, on erityisen tärkeää määrittää mihin kaikkiin rajapintoihin verkosta on mahdollista päästä kiinni. Lähtökohtaisesti käyttäjälle ei pitäisi olla avoinna muuta kuin Front end:in dynaaminen http-portti. Kaikki käyttäjäkohtainen liikenne tulee kulkea tämän portin kautta.

Valitettavasti Web App:it eivät suoraan tue palomuureja mutta IP-osoite rajauksia pystytään tekemään verkkopalvelimen asetuksiin. Eli avataan Web App:in /site/wwwroot/ hakemistossa oleva web.config -tiedosto ja lisätään <system.webServer> elementin sisään seuraava määritys.

```
<system.webServer>
  <security>
     <ipSecurity allowUnlisted="false">
       <add ipAddress="XXX.XXX.XXX.XXX" allowed="true" />
     </ipSecurity>
  </security>
```

```
</system.webServer>
```

Add ipAddress -kenttään laitetaan sallitut IP-osoitteet ja -avaruudet. Tämän jälkeen liikenne Web App:iin sallitaan ainoastaan listatuista osoitteista ja muu liikenne tiputetaan. (Azure Websites Cheat Sheet -- Filtering Traffic by IP 2017.)

Front end puolella määritystä ei tarvitse tehdä, sillä sivuston tulee olla avoin käyttäjille. Back end puolella sallituiksi osoitteiksi määritellään ainoastaan Middleware Linux-palvelimen osoite. Näin Back end:iin päästään kiinni ainoastaan käyttämällä Middleware:n rajapintoja.

Linux-palvelimelle pitää myös määrittää oma palomuuri, joka sallii yhteyden Mulen käyttämään porttiin vain käytetyistä Web App:eista. Web App:ien lähtevän liikenteen osoitteet saadaan Properties-osiosta. Palvelimelle täytyy toki myös määrittää tavallisesti käytetyt palomuurisäännöt myös.

Myös tietokannalle tulee määrittää palomuurisäännöt, jotka sallivat yhteydet vain Back End Web App:ista.

Näillä säännöillä estetään ulkopuolinen pääsy muihin kuin käyttöön tarkoitettuun http-porttiin.

Rajoitusten lisäksi kaikki liikenne on syytä salata käyttäen https-protokollaa http:n sijaan. Tällä estetään liikenteen kuuntelu ja turvataan siirretyn tiedon eheys. Jotta voimme käyttää https-protokollaa, tarvitaan TLS-sertifikaatti. Tätä varten meillä on DigiCert yhtiön myöntämä validi sertifikaatti, jota voimme käyttää.

Sertifikaatti ladataan Web App:eille osiossa SSL certificates. Valmiin sertifikaatin voi ladata suoraan, kunhan se on PFX-formaatissa, sisältäen kaikki sertifikaattiketjun vaaditut sertifikaatit. Kun sertifikaatti on ladattu, voidaan luoda uusi SSL binding, joka sitoo sertifikaatin Web App:in käyttämään nimeen. Tämän jälkeen sertifikaatti pitäisi olla käytössä ja sivuston pitäisi toimia myös https-protokollaa käyttäen.

Koska haluamme varmistaa, että kaikki liikenne kulkee https-protokollaa käyttäen, pitää meidän estää tavallinen http-liikenne. Tätä varten tarvitaan uudelleenohjaussääntö, joka lopullisesti ohjaa liikenteen http-portista https-porttiin. Tämäkin sääntö määritellään Web App:in /site/wwwroot/web.config -tiedostoon <system.webServer> elementin sisään.

```
<system.webServer>
     <rewrite>
       <rules>
         <!-- BEGIN rule TAG FOR HTTPS REDIRECT -->
         <rule name="Force HTTPS" enabled="true">
           <match url="(.*)" ignoreCase="false" />
           <conditions>
             <add input="{HTTPS}" pattern="off" />
           </conditions>
           <action type="Redirect" url="https://{HTTP_HOST}/{R:1}" appendQuer-
yString="true" redirectType="Permanent" />
         </rule>
         <!-- END rule TAG FOR HTTPS REDIRECT -->
       </rules>
     </rewrite>
   </system.webServer>
```
(Azure Websites Cheat Sheet -- Force HTTPS.)

Koska liikenne Linux palvelimella käyttää myös julkista verkkoa, pitää Mule määrittää käyttämään HTTPS protokollaa myös. Tätä varten palvelimelle pitää luoda tai siirtää validin sertifikaatin sisältävä keystore, jonka jälkeen HTTPS määritetään käyttöön Mulen konfiguraatioon seuraavasti.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:https="http://www.mulesoft.org/schema/mule/https"
xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans" version="EE-
3.6.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-current.xsd
http://www.mulesoft.org/schema/mule/core
http://www.mulesoft.org/schema/mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/http
http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd
http://www.mulesoft.org/schema/mule/https
http://www.mulesoft.org/schema/mule/https/current/mule-https.xsd">
    <http:listener-config name="HTTPS_Listener_Configuration" protocol="HTTPS"
host="0.0.0.0" port="${https.port}" doc:name="HTTP Listener Configuration">
        <tls:context>
            <tls:key-store path="keystore.jks" password="password" keyPass-
word="${keystore.password}"/>
        </tls:context>
    </http:listener-config>
```
(Building an HTTPS service.)

Nyt kaikki liikenne Web App:ien välillä käyttää vain ja ainoastaan https-protokollaa.

4 Yhteenveto

4.1 Tulokset

Kokonaisuudessaan opinnäytetyöprojekti onnistui tavoitteissaan. Koko operaatio suunnittelusta toteutukseen kesti useamman kuukauden, osittain katkonaisen aikataulun ja osittain vaadittavan perehtymisen takia. Lopulta Web App:eja varten tarvittavat muutokset olivat pieniä ja Microsoftin dokumentaatio oli kattavaa. Ongelmakohtia ilmeni muun muassa iisnoden käytössä, Mulen käyttämisessä Wep App:issa ja Back end:in SQL kyselyissä. Näistä iisnoden käyttö ratkesi tutustumalla tarpeeksi dokumentaatioon, Mule päätettiin jättää virtuaalipalvelimelle ja Back end vaatii koodimuutoksia, joissa poistettiin käytöstä pagination-ominaisuus.

Tässä vaiheessa siirrosta ei saada vielä merkittäviä kustannussäästöjä, sillä palvelu vaatii tuotantoon siirtyessään 2 App Service Plan:iä ja Linux-palvelimen, joiden kustannukset tulevat yhdessä olemaan samaa luokkaan kuin alkuperäinen palvelin. Työ on kuitenkin investointi tulevaan sekä jatkokehityksen osalta että antaen kokemusta Web App:ien ja PaaS palveluiden käytöstä koko yritykselle. Jatkossa useammissa projekteissa voidaan käyttää samoja palveluita, jotka tulevat vähentämään ylläpidollista työtä ja palvelinresursseja.

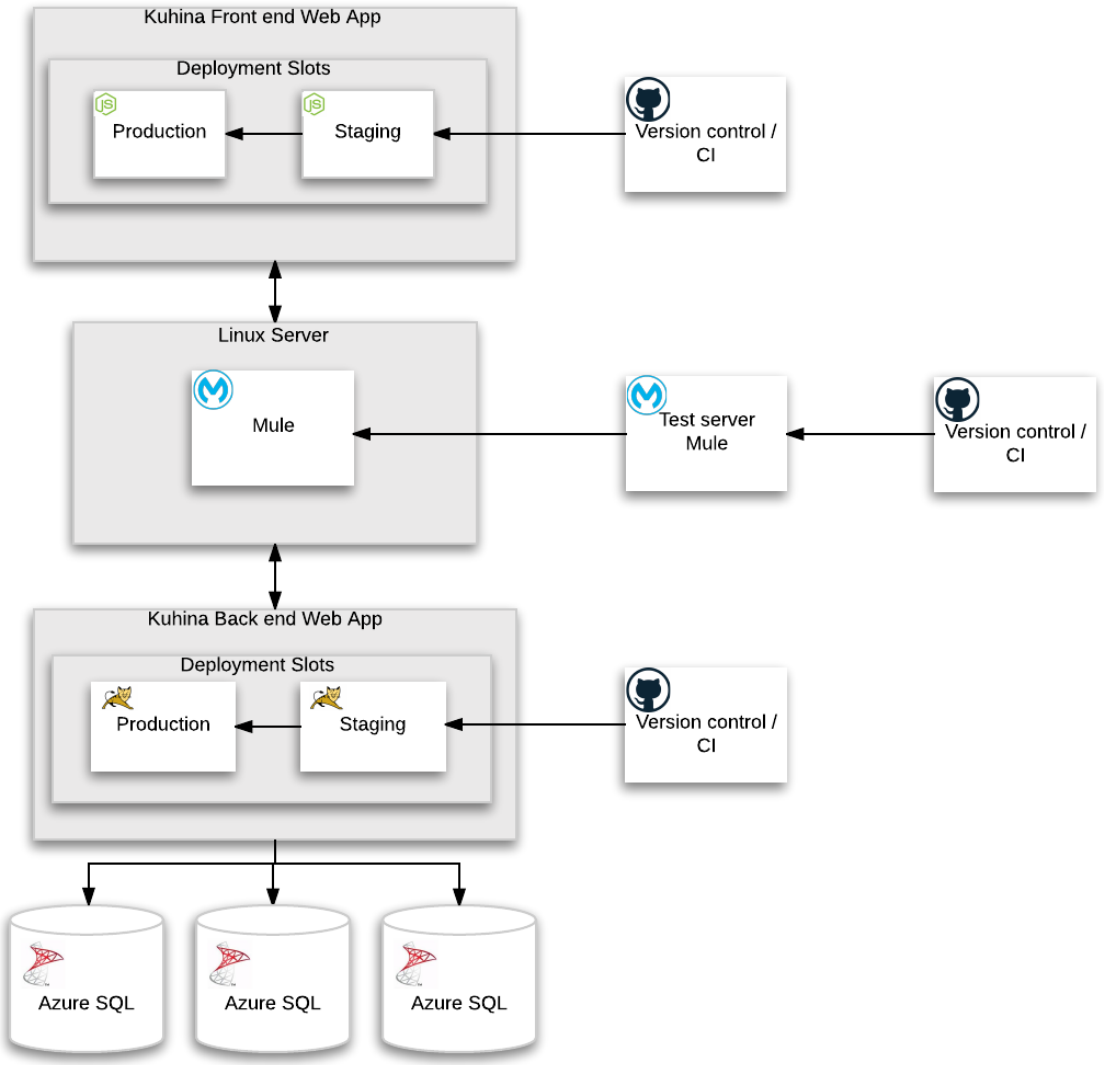
Valitettavasti opinnäytetyön aikana Web App pohjaista Kuhinaa ei keretty testaamaan merkittävissä määrin. Tämän takia esimerkiksi skaalautumisominaisuudet jäivät analysoimatta tarkemmin.

Suurimpana ongelmana Web App:ien kanssa työskentelyssä oli palvelun heikko diagnostiikka ja hankaluus päästä kiinni sovellusten lokitiedostoihin. Console-ominaisuus on heikko ja FTP ei anna avata käytössä olevaa lokitiedostoa. Aivan työn loppuvaiheilla havaittiin kuitenkin Web App:in Advanced Tools-ominaisuus, joka antaa käyttöön selainpohjaiset powershell ja cmd -komentorivityökalut sekä helpommin käytettävät tiedostonsiirto ja -muokkaus mahdollisuudet.

Yleisesti ottaen työ paransi yrityksen ymmärrystä ja käyttövalmiutta Azuren tuotteiden kanssa merkittävästi. Aikaisemmin yrityksessä oli lähinnä käytössä Azuren IaaS palveluita, Azure SQL tietokantoja lukuun ottamatta. Jatkossa on kuitenkin tarkoitus siirtää muitakin sovelluksia käyttämään Web App palvelua.

4.2 Jatkokehitys

Opinnäytetyö antaa paljon mahdollisuuksia jatkokehitykseen. Alunperinkin työn oli tarkoitus olla ensimmäinen askel kohti jaettua Kuhina-ympäristöä. Lopullinen tarkoitus on saada kaikki asiakkaat käyttämään yhtä samaa ympäristöä, jonka komponentteja voidaan automaattisesti skaalata käyttövaatimusten mukaan. Ainoastaan tietokanta on jokaisella asiakkaalla uniikki. Alla oleva kuva kuvaa tämän kaltaista ympäristöä.



Kuva 8. Jatkokehityksen kuvaus.

Kuvassa Kuhinan komponentit muodostavat yhden sovelluskokonaisuuden. Production slot kuvaa tuotannossa olevaa Web App:ia. Tässä ympäristössä asiakas yhdistää Front end:in tuotantosovellukseen, josta liikenne kulkee Linux-palvelimen Mulen kautta Back end:iin ja ottaa yhteyden asiakkaan omaan tietokantaan. Sisältö asiakkaalle tarjotaan mahdollisesti asiakas ID:n tai muun määrityksen perusteella. Uutta asiakasta varten ei

tarvitse perustaa kokonaan uutta ympäristöä vaan pelkästään uusi tietokanta. Tämä vähentää palvelun käyttöönottoon liittyvää työtä.

Koska Azure Web App:it sisältävät mahdollisuuden skaalata resursseja CPU käytön mukaan, voidaan komponenteista todellakin tehdä automaattisesti skaalautuvia. Näin palvelu pysyy toiminnassa liikenteen kasvaessa suurestikin. Kun palveluun voidaan helposti ja nopeasti lisätä uusia käyttäjiä ja sovelluksen resurssit eivät pääse muodostumaan pullonkaulaksi, on palvelu oikeasti skaalautuva ja mahdollistaa suurenkin käyttäjämäärän kasvun.

Azure Web App:it tarjoavat myös mahdollisuuden käyttää eri versioita Web App:in sisällä. Deployment slots-ominaisuus mahdollistaa testiversion ajamisen vierekkäin tuotannon kanssa. Versioita vois myös vaihtaa päittäin. Eli ennen uuden version tuotantoon ajamista, se voidaan ajaa testiversiolle ja testata perusteellisesti. Kun versio on todettu toimivaksi, voidaan nappia painamalla siirtää se tuotantoon, jonka jälkeen tuotantoversio jää vanhaksi testiksi. Nämä versiot toimivat yhtäläisillä resursseilla ja kustannuksilla, joten ominaisuuden käyttäminen ei tuo lisäkustannuksia.

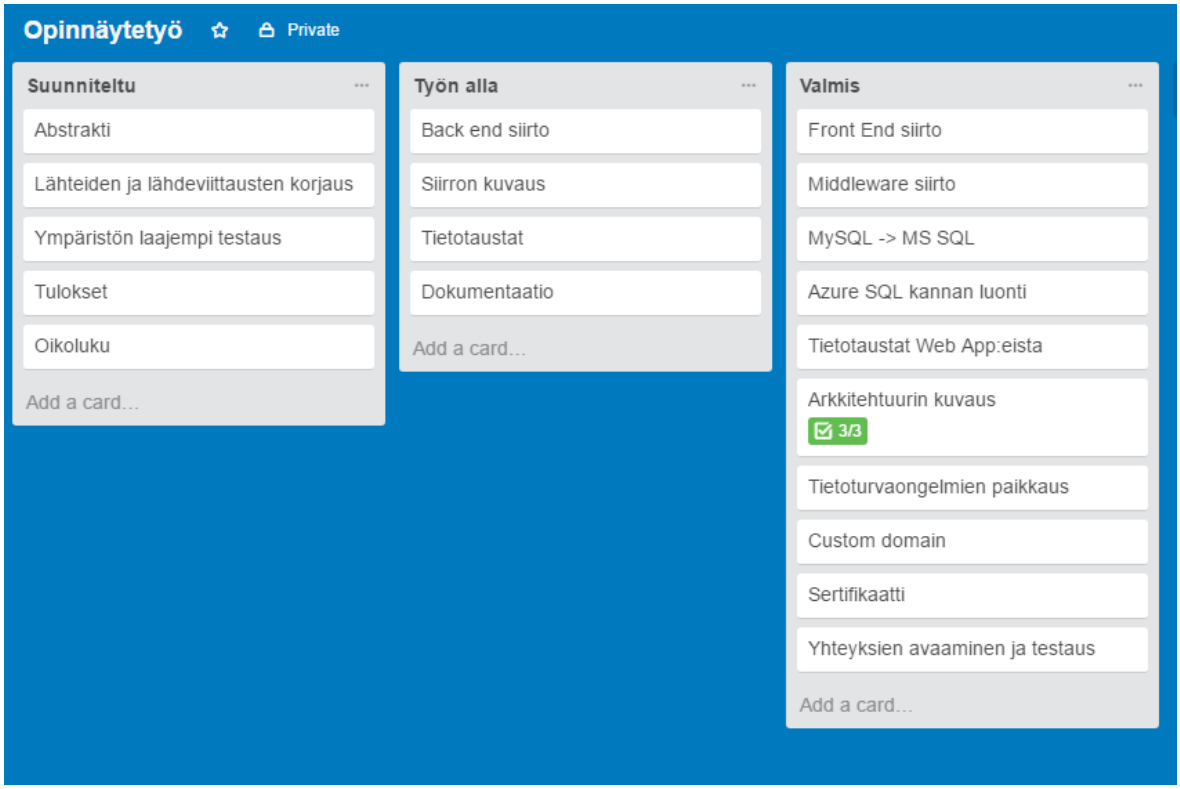
Kuvassa 8 kuvataan, miten uudet versiot haetaan versionhallinnasta suoraan testiversioksi, mitä kautta ne päätyvät testattuina tuotantoon. Tämän operaation voi myös automatisoida, jolloin Web App automaattisesti hakee uuden version aina kun koodin tehdään muutoksia. Lopullisena tarkoituksena on luoda Continuous Integration -putki, joka aina koodimuutosten jälkeen rakentaa Jenkins-palvelimella uuden paketin, ajaa siihen automatisoituja testejä, jonka jälkeen koodi viedään Web App:in testiversiolla, mistä se lopulta ajetaan tuotantoon. Tämä vähentää testaamisen määrää ja inhimillisiä virheitä, joita versionvaihdoksissa saattaa tapahtua.

4.3 Arvio projektista

Kokonaisuudessaan työ eteni katkonaisesti vaiheittain. Web App:ien kanssa työskentely oli lopulta melko yksinkertaista, mutta vaati merkittävän panostuksen ymmärtämään kaikki palvelun toimintaperiaatteet. Kokonaisuudessaan muutoksia sovellukseen piti tehdä hyvin pienessä mittakaavassa Web App:ien takia, suuremmat muutokset johtuivat siirtymisestä MS SQL tietokantaan.

Palvelun heikon diagnostiikan takia ongelmanratkonta oli työlästä ja aikaa vievää. Vaikka Microsoftin dokumentaatio palvelusta oli kattavaa, niin dokumentaatio muuttui radikaalisti kesken työn ja miltei kaikki alkuperäiset käyttämäni ohjeet muuttuivat kokonaan.

Suurin ongelma oli kuitenkin ajanhallinta. Käytin työssä projektinhallintaan apuna Trello-palvelua, jossa työ voidaan pilkkoa helpommin hallittaviksi osuuksiksi. Tämä auttoi selventämään kokonaisuuksia mutta täysipäiväinen työ hankaloitti paneutumista varsinkin kirjoitustyöhön. Sain kuitenkin töistä allokoitua jonkin verran työaikaa opinnäytetyötä varten, mikä auttoi huomattavasti.



Kuva 9. Katsaus Trello-taulusta ja projektin edistymisestä.

Opinnäytetyön projektisuunnitelmassa määritelty aikataulu oli melko täsmällinen. Viimeisillä viikoilla jäin hivenen jälkeen suunnitellusta aikataulusta, ja tämän seurauksena en kerennyt lähettämään työtä ohjaajalle tarkastettavaksi ja saamaan palautetta ennen lopetuskokousta.

Ylipäätään näen työn tuovan hyötyä sekä yritykselle että kerryttäneen minulle arvokasta kokemusta nykytekniikan parissa. Pilvipalvelut ja PaaS -ratkaisut kehittyvät ja yleistyvät kovaa vauhtia ja on tärkeää pysyä kartalla niiden tuomista hyödyistä ja mahdollisuuksista.

Lähteet

Amazon Web Serives. 2017. Regions and Availability Zones. Luettavissa: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html. Luettu 8.5.2017.

Butler, Brandon. 2017. AWS vs Azure vs Google: Cloud platforms compared. 1. Luettavissa: https://www.networksasia.net/article/aws-vs-azure-vs-google-cloud-platforms-compared.1488273117. Luettu 9.5.2017.

Butler, Brandon. 2017. AWS vs Azure vs Google: Cloud platforms compared. 2. Luettavissa: https://www.networksasia.net/article/aws-vs-azure-vs-google-cloud-platforms-compared.1488273117/page/0/1. Luettu 9.5.2017.

Butler, Brandon. 2017. AWS vs Azure vs Google: Cloud platforms compared. 3. Luettavissa: https://www.networksasia.net/article/aws-vs-azure-vs-google-cloud-platforms-compared.1488273117/page/0/2. Luettu 9.5.2017.

Eronen, Heidi. 2016. IaaS, PaaS, SaaS? Mikä pilvipalvelu sopii yrityksellesi. Luettavissa: http://blog.planeetta.net/iaas-paas-saas. Luettu 11.5.2017

Janczuk, Tomasz. 2011. Hosting node.js applications in IIS on Windows. Luettavissa: https://tomasz.janczuk.org/2011/08/hosting-Node.js-applications-in-iis-on.html. Luettu 19.4.2017

Kampschmidt, Joe. 2014. Getting started with iisnode: node.js on Windows. Luettavissa: https://www.jokecamp.com/blog/getting-started-with-iisnode/. Luettu 19.4.2017

Microsoft. 2017. Azure Websites Cheat Sheet -- Filtering Traffic by IP. Luettavissa: http://microsoftazurewebsitescheatsheet.info/#filtering-traffic-by-ip. Luettu 21.4.2017

Microsoft. 2017. Azure Websites Cheat Sheet -- Force HTTPS. Luettavissa: http://microsoftazurewebsitescheatsheet.info/#force-https. Luettu 21.4.2017

Microsoft. 2017. Azure Regions. Luettavissa: https://azure.microsoft.com/en-us/regions/. Luettu 8.5.2017.

Microsoft. Chapter 5: Layered Application Guidelines -- MSDN tietovarasto. Luettavissa: https://msdn.microsoft.com/en-us/library/ee658109.aspx. Luettu 13.5.2017.

Microsoft. Chapter 7: Business Layer Guidelines -- MSDN tietovarasto. Luettavissa: https://msdn.microsoft.com/en-us/library/ee658103.aspx Luettu 13.5.2017.

Microsoft. Chapter 8: Data Layer Guidelines -- MSDN tietovarasto. Luettavissa: https://msdn.microsoft.com/en-us/library/ee658127.aspx. Luettu 13.5.2017.

Microsoft. 2017. Microsoft Azure: Cloud Computing Platform & Services -- Microsoft Azure kotisivut. Luettavissa: https://azure.microsoft.com/en-us/. Luettu 8.5.2017.

Microsoft. 2017. Set up staging environments in Azure App Service. Luettavissa: https://docs.microsoft.com/en-us/azure/app-service-web/web-sites-staged-publishing. Luettu 8.5.2017.

Microsoft. 2017. SLA for App Service. Luettavissa: https://azure.microsoft.com/en-us/support/legal/sla/app-service/v1_0/. Luettu 14.5.2017

Microsoft. 2017. Web Apps Overview. Luettavissa: https://docs.microsoft.com/en-us/azure/app-service-web/app-service-web-overview. Luettu 7.5.2017

Mulesoft. Deploying Mule as a Service to Tomcat. Luettavissa: https://docs.mulesoft.com/mule-user-guide/v/3.5/deploying-mule-as-a-service-to-tomcat. Luettu 20.4.2017

Mulesoft. Building an HTTPS Service Luettavissa: https://docs.mulesoft.com/runtime-manager/building-an-https-service Luettu 25.5.2017

Panettieri, Joe. 2016. Cloud Market Share 2016. Luettavissa: https://www.channele2e.com/2016/02/04/cloud-market-share-2016-aws-microsoft-ibm-google/. Luettu: 9.5.2017.

Scott, Tamara. 2017. IaaS vs. PaaS: Which Cloud Model is Right for Your Business? Luettavissa: http://technologyadvice.com/blog/information-technology/iaas-vs-paas/. Luettu 11.5.2017.

Liitteet

Liite 1. Käsitteet.

API: Sovellusrajapinta.
Angular: Avoimen lähdekoodin JavaScript-ohjelmistokehys.
Azure: Microsoftin julkisen pilven palvelu
Back end: Sovelluksen tallennus- ja datakerros. Hallinnoi sovelluksen dataa, sen tallentamista ja hakemista.
Front end: Sovelluksen esityskerros. Hallinnoi sovelluksen käyttöliittymää ja ulkoasua.
Git: Versionhallintaohjelmisto.
Github: Julkinen git-pohjainen versionhallintapalvelu.
IaaS: Infrastruktuuri palveluna
IIS: Windowsille tarkoitettu verkkopalvelin.
iisnode: IIS moduuli, joka mahdollistaa node.js sovellusten isännöimisen IIS-palvelimella.
HTTP: Hypertext transform protokolla
HTTPS: HTTP- ja TLS-protokollan yhdistelmä
Maven: Lähtökohtaisesti Java-koodin rakennukseen käytetty automaatiotyökalu.
Middleware: Sovelluksen logiikkakerros. Hallinnoi sovelluksen toiminnallisuutta ja kommunikaatiota
Mule: Järjestelmien väliseen kommunikaatioon tarkoitettu integraatioalusta.
Node.js: Avoimen lähdekoodin javascript ajoympäristö.
PaaS: Sovellusalusta palveluna
Resurssiryhmä: Azuressa hallinnollinen määrite, jolla ryhmitellään käytettäviä tuotteita ja palveluita.
SSL/TLS: Transport layer security salausprotokolla
Tomcat:
Web App: Microsoft Azuren sovellusten isännöimiseen tarkoitettu PaaS-alusta