TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JONI OLLIKAINEN

YRITYSTIETOJÄRJESTELMÄN TUOTANTOYMPÄRISTÖN SUUNNITTELU JA PYSTYTYS AMAZON WEB SERVICES -PILVIPALVELUUN

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 5. toukokuuta 2015

TIIVISTELMÄ

JONI OLLIKAINEN: YRITYSTIETOJÄRJESTELMÄN TUOTANTOYMPÄRISTÖN SUUNNITTELU JA PYSTYTYS AMAZON WEB SERVICES –PILVIPALVELUUN
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua
Kesäkuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Pervasive Systems
Tarkastaja: professori Tommi Mikkonen

Avainsanat: pilvipalvelut, Amazon Web Services, infrastructure as a service, yritystietojärjestelmät, mikropalveluarkkitehtuuri, jatkuva toimitus

Yritystietojärjestelmien vaatimuksien täyttämiseksi etsitään jatkuvasti tehokkaampia menetelmiä. Keskeisinä haasteina ovat muiden muassa vaihtelevaan kuormitukseen vastaaminen ja tuotantoonvientiprosessin nopeus versiopäivityksissä. Perinteinen konesali vastaa huonosti vaihtelevaan kuormitukseen. Lisäksi laiteviat vaikuttavat suoraan tietojärjestelmän käyttämiin resursseihin. Usein tuotantoonvientiprosessia hidastaa käyttöpalveluntarjoajan tekemät etukäteen sovitun aikataulun mukaiset päivitykset. Usein liiketoimintaa parantavia ominaisuuksia haluttaisiin viedä tuotantoon heti, kun ne ovat valmiita.

Tässä diplomityössä tutkittiin pilvipalveluiden mahdollisuutta ratkaista nämä ongelmat. Pilvipalveluista erityistarkastelussa oli Amazon Web Services. Pilvipalveluita tarkasteltiin virtuaalisen infrastruktuurin ominaisuuksien kautta. Pilvipalveluiden käyttöä pohdiskeltiin myös aiemmin toteutettujen pilvipohjaisten järjestelmien pohjalta syntyneiden suunnitteluperiaatteiden kautta. Yritystietojärjestelmien asettamia haasteita käytiin läpi koko ohjelmistokehitysprosessin läpi. Suunnitteluvaiheen osalta keskityttiin mikropalveluarkkitehtuuriin, joka on eräs pilviyhteensopiva ohjelmistoarkkitehtuuri. Mikropalveluarkkitehtuuria tarkasteltiin myös ohjelmistokehityksen näkökulmasta. Jatkuvan toimituksen prosessia tutkittiin pilvipalveluiden mahdollistamana nopeana prosessina saada uusia ominaisuksia lyhyellä syklillä tuotantoon.

Käytännön esimerkit osoittivat pilvipalveluista saatavan hyödyn olevan merkittävä sekä järjestelmällä, joka on alusta asti suunniteltu pilviyhteensopivaksi, että järjestelmällä, joka myöhemmässä vaiheessa elinkaarta viedään pilveen. Tuotantoympäristön lisäksi myös testi- ja kehitysympäristöjen vienti pilveen tuovat etuja. Pilvipalveluiden käyttö myös luo uudenlaisia haasteita ohjelmistokehitykselle. Tässä työssä tarkasteltujen tapausten osalta hyödyt olivat selvästi haasteita suurempia. Kahden tapauksen perusteella ei voida kuitenkaan väittää, että tilanne olisi kaikissa tapauksissa näin. Jokaisen järjestelmän kohdalla hyödyt ja haitat tulisi arvioida tapaus kohtaisesti. Pilvipalveluiden tarjonnan osalta Amazon Web Services –pilvipalvelun todettiin pystyvän täyttämään yritystietojärjestelmien asettamat vaatimukset.

ABSTRACT

JONI OLLIKAINEN: DESIGNING AND IMPLEMENTING PRODUCTION ENVIRONMENT OF ENTERPRISE SOFTWARE IN AMAZON WEB SERVICES CLOUD
Tampere University of Technology
Master of Science Thesis, 45 pages
June 2016
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Professor Tommi Mikkonen

Keywords: cloud computing, Amazon Web Services, infrastructure as a service, enterprise software, microservices, continuous delivery

Effective methods to fulfill the requirements of enterprise software are constantly searched. The speed of the delivery process and dealing with varying load are central challenges. Traditional dedicated server hosting responds poorly to varying loads. Also hardware faults affect directly to the resources the system is using. The delivery process is often slowed down by previously agreed scheduled updates done by the hosting service provider. Often there is an ambition to deliver business improving updates to production as soon as they are ready.

In this thesis the ability of cloud computing to solve the above issues was examined. Amazon Web Services was the selected cloud computing service in deeper analysis. Cloud computing was analyzed via the properties of virtual infrastructure. The use of cloud computing was reflected via cloud design patterns risen from previously implemented cloud based systems. The requirements of enterprise system were considered through the whole process of software production. The design phase was covered by a cloud-native architecture called microservices. The microservices architecture was also reflected from the software development point of view. The process of continuous delivery was examined as a cloud computing enabling fast process to deliver new features to production.

Practical studies showed that the advantages gained form the cloud computing were remarkably higher than the disadvantages. The practical study consists two software systems, of which one was built as cloud-native system from the beginning. The other one was an older system which was not cloud-native but was redesigned towards it. In addition of production environment also testing and developing environments was discovered to benefit from cloud migration. The use of cloud computing also creates new kinds of challenges. Cases covered by this thesis gained remarkably more advantages than disadvantages from cloud computing. From the results of two cases it cannot be claimed that results would be same in all cases. Advantages and disadvantages should be considered case by case. In the case of cloud computing service providers Amazon Web Services was discovered to meet the requirements of enterprise software.

ALKUSANAT

Tämä diplomityö tehtiin Gofore Oy:lle selvitystyönä yritystietojärjestelmien käytöstä pilvipalveluissa. Tarkoituksena oli saada lisätietoa erityisesti vanhojen tietojärjestelmien siirtämisestä pilvipalveluihin. Haluan kiittää Gofore Oy:tä mahdollisuudesta tehdä tätä diplomityötä työtehtävieni ohessa sekä hyödyntää mielenkiintoisia projekteja työn tutkimuskohteina.

Haluan kiittää Professori Tommi Mikkosta kannustavasta palautteesta sekä asiantuntevasta ohjauksesta erityisesti tieteellisen kirjoittamisen osalta. Haluan myös antaa kiitokset Gofore Oy:n Tapio Rautoselle asiantuntevasta ohjauksesta diplomityön asiasisällön kanssa. Kiitokset myös Fonectan Tero Nevalaiselle haastattelusta.

Tampereella, 22.5.2016

Joni Ollikainen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

API	Application Programming Interface.
Autoskaalaus	Usealle instanssille jaetun palvelun tai järjestelmän instanssien lukumäärän vähentäminen tai kasvattaminen automaattisesti kuormituksen mukaan.
AWS	Amazon Web Services.
AWS-alue	Maantieteellinen alue, jossa käytettävät Amazon Web Services –pilviresurssit fyysisesti sijaitsevat.
DevOps	Kulttuuri, jossa ohjelmistokehittäjien ja järjestelmäasiantuntijoiden roolit ja tehtävät yhdistyvät samoille henkilölle.
End-to-end –testi	Koko järjestelmän läpäisevä testi, jonka läpäisemiseksi järjestelmän täytyy toimia käyttöliittymätasolta tietokantaan.
IaaS	Infrastruktuuri palveluna, pilvi-infrastruktuuri. Englanninkielinen termi: Infrastructure-As-A-Service.
Instanssi	Tietyn mallin tai kuvauksen mukainen erillinen ilmentymä, joita voi olla useita samanlaisia. Pilvipalveluissa esimerkiksi virtuaaliset palvelinkoneet ovat instansseja.
IP	Internet Protocol.
Jatkuva integraatio	Ohjelmistokehityksessä käytetty prosessi, jossa lähdekoodiin tulevia muutoksia integroidaan muihin muutoksiin automaattisesti. Englanninkielinen termi: continuous integration.
Jatkuva toimitus	Ohjelmistokehityksessä käytetty prosessi, jossa uusista ominaisuuksista tuotetaan automaattisesti uusia julkaisukelpoisia koontiversioita. Englanninkielinen termi: continuous delivery.
JSON	JavaScript Object Notation.
Koontiversio	Lähdekoodista käännetty tai muuten tehty suoritettava versio sovelluksesta. Englanninkielinen termi: software build.
Kuormantasaaja	Usealle instanssille jaetun palvelun edustalla toimiva liikennettä ohjaava palvelu, joka tasaa kuormaa instanssien kesken.
Lähdekoodirepositorio	Ohjelmistokehityksessä käytetty lähdekoodin keskitetty tallennuspaikka, johon kehittäjät tallentavat tuottamansa ohjelmakoodin sekä koodiin tehdyt muutokset.
PaaS	Alusta palveluna. Englanninkielinen termi: Platform-As-A-Service.

Refaktorointi	Ohjelmakoodin, konfiguraation tai tietokantaskeeman rakenteen muuttamista siten, että sen toiminta ei ulkoisesti muutu.
SaaS	Sovellus palveluna. Englanninkielinen termi: Software-As-A-Service.
Sisällönjakeluverkko	Maantieteellisesti hajautettu palvelinverkosto, jonka tarkoituksena on jakaa resursseja verkkopalvelun varsinaisen Web-palvelimen sijasta lähempänä loppukäyttäjää. Englanninkielinentermi: content delivery network.
Sovelluspalvelin	Palvelinohjelmisto ja ohjelmistokehys ohjelmakoodin suorittamiseksi, joka tarjoaa ominaisuuksiaan ohjelmoijan käytettäväksi.
VPN	Virtuaalinen erillisverkko. Tapa luoda näennäisesti yksityinen verkko julkisen verkon yli kahdelle tai useammalle sisäverkolle. Englanninkielinen termi: Virtual Private Network.

.

1 JOHDANTO

Perinteinen ratkaisu yritystietojärjestelmän tuotantoympäristöksi on käyttöpalvelun tarjoajalta ostettu palvelinkapasiteetti. Perinteisen konesalipalvelun ongelmina ovat kiinteä kapasiteetti, korkeat kulut sekä tuotantoon viennin hitaus. Kapasiteetti ei skaalaudu ylöspäin kuormituspiikkien aikaan, eikä hinta laske silloin kun kuormitus on vähäistä. Lisäksi useat käyttöpalveluntarjoajat haluavat tarjota tuotantoonviennin ja palvelinten ylläpidon palveluna siten, että ohjelmiston kehitystiimillä ei ole pääsyä tuotantopalvelimille. Tuotantoon vienti tapahtuu tällöin manuaalisesti, tehdyn tilauksen pohjalta. Liiketoimintaa parantavat ominaisuudet saattavat joutua odottamaan jopa viikkoja päivitysikkunaa ennen kuin ne saadaan päivitettyä tuotantoon. Perinteisessä konesalipalvelussa tyypillisesti tehdään sopimus fyysisistä palvelinkoneista. Tällöin palvelinkoneessa ilmenevä laitevika vaikuttaa suoraan palvelun käyttöön.

Tässä diplomityössä selvitetään yritystietojärjestelmän tuotanto-, testi-, ja kehitysympäristöjen suunnittelua ja pystytystä Amazon Web Services –pilvipalveluun. Tarkastelussa on myös pilvipalveluista hyötyvät tuotantoonvientitavat sekä pilvipalveluiden kyky toipua vikatilanteista. 2010-luvulla pilvipalveluiden tarjonta on kasvanut ja niiden tarjoama palvelun taso on kehittynyt riittävästi pystyäkseen täyttämään yritystietojärjestelmien tarpeet. Pilvipalvelut pystyvät ratkaisemaan edellä mainitut perinteisten konesalien ongelmat. Pilvipalveluiden hyödyntäminen onnistuu parhaiten siihen sopivalla ohjelmistoarkkitehtuurilla. Tässä työssä esitellään mikropalveluarkkitehtuuri, jonka mukaiseksi järjestelmän voi alusta asti suunnitella tai muuttaa silloin, kun järjestelmää ollaan siirtämässä pilveen. Pilviyhteensopiva ohjelmistoarkkitehtuuri ei kuitenkaan ole edellytys pilvipalveluiden käytölle.

Pilvipalveluiden käyttö luo myös uusia haasteita, joita tässä työssä vertaillaan saavutettaviin hyötyihin sekä teoreettisesti, että kahden tutkimuskohteen kautta. Näistä toinen on jo vuosia Amazonin pilvessä toiminut tietojärjestelmä, jota arvioidaan tehtyjen päätösten, saavutettujen hyötyjen ja koettujen ongelmien näkökulmista. Tämä järjestelmä on kasvanut pilviyhteensopivaksi ja mikropalveluarkkitehtuurin mukaiseksi vuosien varrella. Toisena tutkimuskohteena kokeillaan vanhan tietojärjestelmän tuotanto- ja kehitysympäristöjen siirtämistä Amazonin pilveen. Kehitysympäristön siirto tehtiin tämän työn puitteissa. Tälle järjestelmälle tehtiin myös suunnitelma ohjelmistoarkkitehtuurin muuntamisesta pilviyhteensopivaksi. Järjestelmän arkkitehtuuria ei vielä muutettu, mutta tämän suunnitelman avulla järjestelmälle suunniteltu pitkä elinkaari voidaan tulevaisuudessa toteuttaa helpommin pilvipalveluiden avulla.

Luvussa 2 käydään läpi pilvipalveluiden ja yritystietojärjestelmien ominaisuuksia. Pilvipalveluista esitellään kolme palvelumallia sekä yleisellä tasolla pilvipalveluiden hyötyjä ja haittoja. Yritystietojärjestelmiä tarkastellaan niiden asettamien vaatimusten kautta sekä käydään läpi arkkitehtuurillisia ratkaisuja yritystietojärjestelmien toteuttamiseksi. Pilviyhteensopivana ohjelmistoarkkitehtuurina esitellään mikropalveluarkkitehtuuri, jota vertaillaan vanhempaan palvelupohjaiseen arkkitehtuuriin. Luvun lopussa luodaan katsaus jatkuvan toimituksen prosessiin, tuotantoonvientitapoihin sekä näiden käyttöön pilvipalveluissa. Luvussa 3 esitellään Amazon Web Services –pilvipalvelun tarjoamia palveluita. Tarkasteluun on pyritty valitsemaan sellaiset palvelut, jotka ratkaisevat keskeisimpiä yritystietojärjestelmän haasteita. Lopussa käydään lyhyesti läpi muita yritystietojärjestelmien kanssa käytettyjä palveluita. Luvussa 4 käydään läpi kaksi tutkimuskohdetta, joissa yritystietojärjestelmiä käytetään pilvipalveluissa. Luku 5 on yhteenveto, jossa tehdään loppupäätelmä yritystietojärjestelmien käytöstä pilvipalveluissa.

2 PILVIPALVELUT JA YRITYSTIETOJÄRJESTELMÄT

Tässä luvussa käydään läpi pilvipalveluiden ja yritystietojärjestelmien määritelmät. Pilvipalveluista käydään läpi kolme palvelumallia, vertaillaan pilvi-infrastruktuurin ja perinteisten konesalipalveluiden eroja sekä käydään läpi pilvipalveluiden keskeisiä ominaisuuksia. Yritystietojärjestelmät määritellään niiden asettamien vaatimusten kautta sekä tarkastellaan yritystietojärjestelmissä käytettyjä ohjelmistoarkkitehtuureita. Pilviyhteensopivana ohjelmistoarkkitehtuurina esitellään mikropalveluarkkitehtuuri sekä muutamia pilvipohjaisen järjestelmän suunnitteluperiaatteita, joita erityisesti mikropalveluiden kanssa suositellaan käytettäväksi. Viimeisenä luodaan katsaus jatkuvan toimituksen prosessiin, tuotantoonvientitapoihin sekä näiden käyttöön pilvipalveluissa.

2.1 Pilvipalvelut

Alikohdassa 2.1.1 käydään läpi pilvipalveluiden määritelmä sekä siihen liittyvät peruskäsitteet. Alikohdassa 2.1.2 esitellään pilvipalveluissa käytetyt kolme palvelumallia. Alikohdassa 2.1.3 tarkastellaan pilvipalveluun siirtymisen etuja erilaisissa tilanteissa.

2.1.1 Pilvipalveluiden peruskäsitteitä

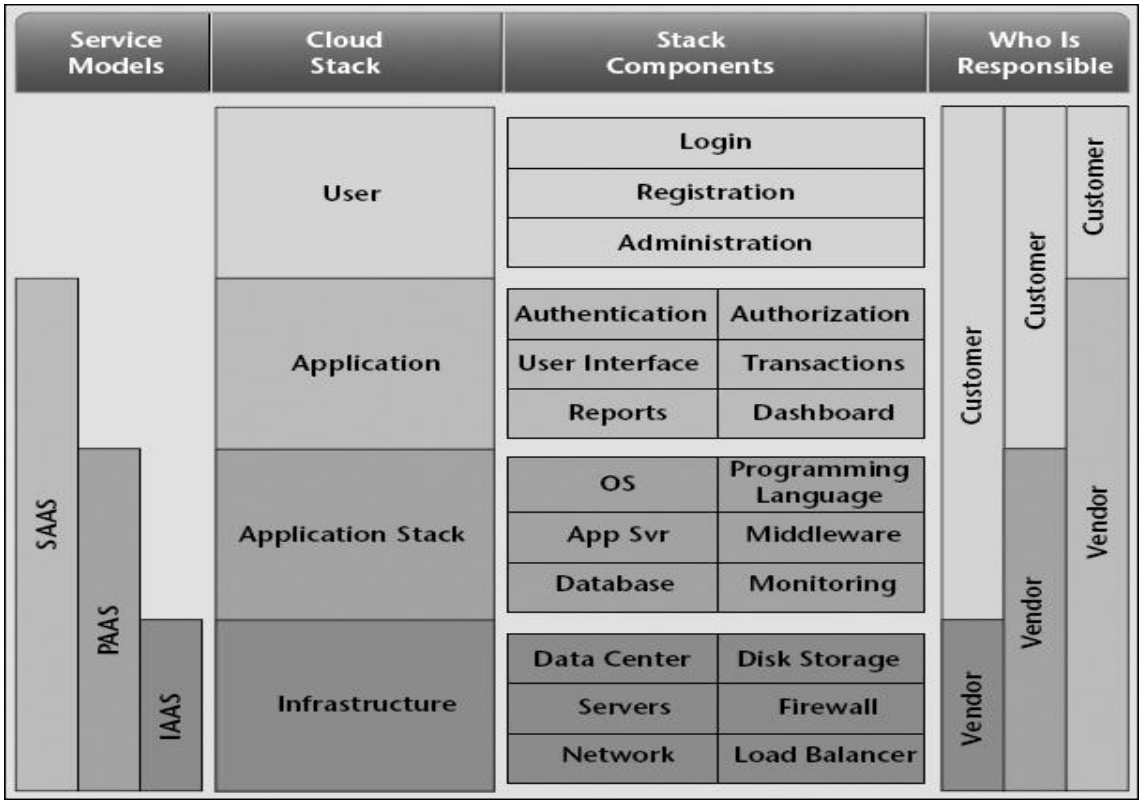
Yhdysvaltain kansallinen tekniikanstandardointivirasto NIST on tehnyt vuonna 2011 oman määritelmänsä pilvipalveluista. Se pyrkii luomaan perustason määritelmän pilvipalveluista, jonka avulla asiasta voidaan helpommin käydä keskustelua. Sen mukaan pilvipalvelun tulisi koostua verkon yli saatavilla olevista konfiguroitavista resursseista. Resurssien tulisi olla näennäisesti rajaton varanto palvelimia, verkkoyhteyksiä, tallennuskapasiteettia, sovelluksia ja palveluita, joiden käyttöönotto onnistuu nopeasti ilman palveluntarjoajan avustusta. Pilvipalvelun resurssit muistuttavat tämän vuoksi sähköverkkoa tai vesijohtoverkostoa, joista käyttäjä saa tarpeensa mukaan vettä tai sähköä. Tämä määritelmä toimii tässä työssä pilvipalveluiden määritelmänä. [1]

NIST listaa määritelmässä viisi keskeistä ominaisuutta, jotka pilvipalvelun tulisi toteuttaa. Ensimmäisenä ominaisuutena mainitaan, että pilvipalvelusta pitää pystyä varaamaan resursseja käyttötarpeen mukainen määrä ilman palveluntarjoajan avustusta. Toisena ominaisuutena pilvipalvelun täytyy olla julkisen Internetin yli saatavilla useiden rajapintojen kautta. Pilvipalveluun pääsy ei siis saa olla rajattu esimerkiksi pelkästään työpöytäsovellukseen. Kolmantena ominaisuutena pilvipalvelun resurssien tulee näkyä käyttäjälle virtuaalisena varantona, josta käyttäjä saa vain tarvitsemansa määrän resursseja sen sijaan, että käyttäjä joutuisi varaamaan kokonaisia fyysisisiä palvelinkoneita. Virtuaalinen varanto on käyttäjien kesken yhteinen ja sama fyysinen palvelinkone voi palvella

useampaa käyttäjää yhtäaikaisesti. Neljäntenä ominaisuutena pilvipalvelun resurssien varaaminen ja vapauttaminen tulee onnistua nopeasti käyttäjän tarpeiden mukaan. Resursseja varatessa niiden tulisi näkyä käyttäjälle näennäisesti loppumattomana. Viidentenä ominaisuutena pilvipalvelun resurssien käyttöä täytyy pystyä monitoroimaan niin, että resurssien käyttömäärät näkyvät sekä palveluntarjoajalle että palvelun käyttäjälle. Monitoroinnin tulee olla resurssin tyypin mukaista, esimerkiksi tallennuspalvelussa levytilan käyttö tai laskentayksikössä suoritinkäyttö. Pilvipalvelussa tulee olla käyttöperusteinen laskutus, joka pohjautuu resurssien käytön monitorointiin. Usein pilvipalveluissa resurssien käytöstä maksetaan jokaiselta alkavalta tunnilta tai minuutilta. [1]

2.1.2 Pilvipalveluiden kolme palvelumallia

Pilvipalvelut jaetaan useimmiten kolmeen palvelumalliin infrastruktuuri palveluna (IaaS, Infrastructure as a Service, pilvi-infrastruktuuri), alusta palveluna (PaaS, Platform as a Service, pilvialusta) ja sovellus palveluna (SaaS, Software as a Service). Niillä rajataan, mikä osa laskentaresursseista on käyttäjän hallinnoitavissa ja mikä tarjotaan valmiina palveluna. Palvelumallien nimet viittaavat mihin kerrokseen asti laskentaresurssit ovat pilvipalvelun tarjoamia. Kuvassa 1 on havainnollistettu palvelumallien erot.



Kuva 1. *Pilvipalveluiden kolme palvelumallia [2]*

Infrastruktuuri palveluna on matalimman abstraktiotason palvelumalli, jossa palveluntarjoaja tarjoaa laitteistoresursseja, kuten erilaisia virtuaalikoneita, tallennustilaa ja niiden

välisiä konfiguroitavia verkkoyhteyksiä. Virtuaalinen palvelinkone on yleisin käyttäjälle näkyvä infrastruktuuriresurssi. Käyttäjälle näytetään yleensä virtuaalikoneesta jotain perustietoja, kuten suorittimen ja muistin kapasiteetti, joiden perusteella käyttäjä valitsee sopivan virtuaalikoneen. Virtuaalikoneille on usein valittavissa käyttöjärjestelmä, joka asennettaan automaattisesti. Ohjelmien asennus, ohjelmien suoritusympäristöt ja verkon konfigurointi ovat käyttäjän vastuulla. Käyttäjällä on toisaalta myös vapaus tehdä niiden suhteen enemmän valintoja. Merkittävimpiä IaaS-palveluntarjoajia ovat Amazon (Amazon Web Services[1]), Microsoft (Azure[2]) sekä Google (Google Compute Cloud[3]). [3]

Alusta palveluna on seuraava taso. Siinä on kaikki samat palvelut kuin IaaS-mallissa, mutta niiden lisäksi myös käyttöjärjestelmä ja ohjelmien suoritusympäristö tarjotaan valmiina palveluna. Sovelluskehityksen kannalta tämä tarkoittaa sitä, että PaaS-resurssia varatessa valitaan myös käytettävä ohjelmointikieli. PaaS-palvelumallissa käyttäjä ei välttämättä edes tiedä minkä käyttöjärjestelmän päällä sovellusalustaa ajetaan. Käyttäjälle näytetään usein sovellusalustan laitteistosta suorittimen ja muistin kapasiteetti, kuten IaaS-palveluidenkin tapauksessa. Tunnettuja PaaS-palveluntarjoajia ovat muun muassa Salesforce (Heroku[4]), IBM (Bluemix[5]), Google (Google App Engine[6]) ja Amazon (Amazon Web Services). Amazonilla on IaaS-palveluiden lisäksi tarjolla valmis PaaS-alusta, AWS Elastic Beanstalk[7], joka toimii Amazonin omista IaaS-palveluista valmiiksi koottuna sovellusalustana. Myös muut PaaS-palveluntarjoajat voivat käyttää pohjana omia tai muiden tarjoamia IaaS-palveluita.

Ohjelmisto palveluna on abstraktiotasoltaan korkein pilvipalveluiden palvelumalleista. Siinä asiakas valitsee itselleen sopivan verkkopalvelun, jota käyttää ja hallinnoi. SaaS-verkkopalvelun käyttäjä maksaa tavallisesti käytön mukaan määräytyvää kuukausimaksua. SaaS-palvelun kapasiteettia on voitava kasvattaa ja vähentää ilman palveluntarjoajan apua ja tämä on keskeisin tekijä, joka erottaa SaaS palvelun muista tilattavista verkkopalveluista. Palvelun käyttäjän ei tarvitse tietää tai olla vastuussa sovellusalustasta, käyttöjärjestelmästä tai laitteistosta. Esimerkiksi Microsoft Office 365[8] ja Google Drive[9] ovat SaaS mallin mukaisia palveluita. Niissä käyttäjille tarjotaan selainpohjaisia toimistosovelluksia, tallennustilaa sekä asiakirjojen jakamista käyttäjien kesken. Sovellukset toimivat useimmilla selaimilla, mobiilialustoilla ja joiltain osin myös työpöytäsovelluksina. Selaimella käytettynä sovellukset eivät vaadi asennusta. Verrattuna sovelluksiin, jotka

[1] Amazon Web Services: https://aws.amazon.com/
[2] Microsoft Azure: https://azure.microsoft.com
[3] Google Compute Cloud: https://cloud.google.com/compute/
[4] Heroku: https://www.heroku.com/
[5] IBM Bluemix: http://www.ibm.com/cloud-computing/bluemix/
[6] Google App Engine: https://cloud.google.com/appengine
[7] AWS Elastic Beanstalk: https://aws.amazon.com/elasticbeanstalk/
[8] Microsoft Office 365: https://products.office.com/
[9] Google Drive: https://www.google.com/intl/fi/drive/

ovat vain työpöytäsovelluksia SaaS-palvelun käyttäjän ei tarvitse asentaa sovelluksia, ostaa ja päivittää uusia versioita, huolehtia varmuuskopioinnista tai tallennuspalvelinten ylläpidosta. [2]

Ohjelmistokehityksen näkökulmasta IaaS- ja PaaS-palvelut ovat käyttökelpoisia palveluita kehitettävän ohjelmiston alustaksi. SaaS-palvelut tarjoavat jo valmiin ohjelmiston, joten niitä käytetään ohjelmistokehityksessä lähinnä apuohjelmina. IaaS- tai PaaS-palvelua hyödyntävän ohjelmistokehityksen lopputuotteena saattaa olla SaaS-palvelu. Tässä työssä käsitellään pilvipalveluita nimenomaan ohjelmistokehityksen näkökulmasta, joten jatkossa pilvipalveluilla tarkoitetaan lähinnä IaaS- ja PaaS-palveluita.

2.1.3 Milloin siirtyä käyttämään pilvipalveluita?

Kolme yleisintä syytä siirtyä käyttämään pilvipalveluita ovat halu nopeuttaa ohjelmistokehitysprosessia, pienentää kuluja ja parantaa vikasietoisuutta. Verkkopalveluita tarjoavilla yrityksillä pilvipalveluihin siirtymisen taustalla on usein tarve nopeuttaa liiketoiminnan kehitystä viemällä uusia ominaisuuksia nopeammin tuotantoon. Uusien ominaisuuksien nopea julkaiseminen ja mahdollisuus palata nopeasti aiempaan versioon mahdollistavat hallitun riskinoton. Jos hitaammassa päivitystahdissa havaitaan tuotantoon julkaistun ominaisuuden hidastavan liiketoimintaan, ehtii se aiheuttamana enemmän haittaa ennen korjauspäivitystä. Tästä syystä hitaalla päivitystahdilla etenevä ohjelmistokehitys ei ole niin suotuisa uusien innovaatioiden kokeiluun. Pilvipalveluita hyödyntävät verkkopalvelut saattavat julkaista tuotantoon jopa satoja muutoksia päivässä. [2][4][5]

Pilvi-infrastruktuurin käyttöönoton jälkeen käyttäjän on voitava itsepalveluna lisätä tai vähentää käytössään olevia resursseja milloin tahansa täysin ilman palveluntarjoajan avustusta. Perinteisen konesalipalvelun tapauksessa palvelimista luopuminen tai uusien varaaminen vie nopeimmillaankin päiviä. Esimerkiksi käyttäjä ei yleensä voi saada kaksinkertaista palvelinkapasiteettia yllättävän muutaman tunnin kuormituspiikin ajaksi. Perinteisen konesalin kanssa toimiessa kapasiteettia on varattava sen verran, että oletettu kuormitus ei ylitä palvelimien kapasiteettia. Tällöin täydestä kapasiteetista myös maksetaan jatkuvasti, vaikka tarve ei ole jatkuva. Pilvipalveluissa sen sijaan resursseista maksetaan vain käytön mukaan. Pilvipalveluita voidaan myös hyödyntää lisäkapasiteettina, jos järjestelmää ei haluta siirtää kokonaan pilveen. Tällöin konesalissa on tietty peruskapasiteetti ja pilvipalvelua käytetään vain tarvittaessa. Tällainen pilvipalveluiden käyttö on nimeltään cloudbursting. [4][6]

Perinteisessä konesalissa pidettävä järjestelmä voi hyödyntää pilvipalveluita myös tallennukseen, varmuuskopiointiin tai suurten datamäärien analysointiin. Esimerkiksi tietojärjestelmän keräämän analytiikkadatan tallennuksessa voidaan hyödyntää skaalautuvaa pilvitallennusta. Data voidaan myös prosessoida pilvessä, jolloin prosessointiin voidaan käyttää kapasiteettia ja rahaa tarpeen mukaan. Tällöin esimerkiksi kuukausittain ajettava

raskas laskentaprosessi on helppoa ja kustannustehokasta suorittaa pilvessä, koska laskentaresursseista ei tarvitse maksaa koko kuukauden ajalta. Edellisen esimerkin lisäksi pilvitallennusta voi hyödyntää kaikenlaisten varmuuskopioiden säilömiseen.

Omassa tai vuokratussa konesalissa resursseja käsitellään fyysisten koneiden tasolla. Varattuun fyysiseen resurssiin mahdollisesti kohdistuvat laiteviat, sähkökatkot, verkkoyhteyden katkokset ja konfiguraatiovirheet aiheuttavat katkoksen resurssin käytössä. Pilvipalvelussa resurssit ovat virtuaalisia ja sen vuoksi edellä mainittujen häiriötilanteiden kohtaaminen on käyttäjän kannalta helpompaa. Virtuaalisia resursseja voidaan siirtää koneelta toiselle tai ne voivat olla kahdennettuja useampaan konesaliin, jolloin suoraan fyysiseen laitteistoon kohdistuvien häiriöiden vaikutus käyttäjälle on vähäistä. Fyysisen laitevian vuoksi toimimaton virtuaalikone voidaan terminoida ja luoda uudestaan samalla konfiguraatiolla hyvinkin lyhyessä ajassa toisin, kuin konesalista vuokrattu palvelin.

Pilvipalveluihin siirryttäessä yhteistä kaikissa palvelumalleissa on laitteiston ylläpidosta koituvien kustannusten muuttuminen. Oman konesalin ylläpito vaatii muun muassa laitteiston hankinnan, asentamisen, konfiguroinnin, jäähdytyksen, sähkön, turvallisen tilan, vanhan laitteiston poiston, varmuuskopioinnin sekä verkkoyhteyden. Yrityksen liiketoiminnan kannalta on oleellista tarkastella, halutaanko itse huolehtia näiden kulujen hoitamisesta kustannustehokkaasti. Tämä pilvipalvelun etu saavutetaan myös silloin, jos fyysisiä palvelinkoneita vuokrataan jonkun muun ylläpitämästä konesalista, jolloin suurin osa edellä luetelluista vastuista siirtyy konesalin ylläpitäjälle. [4]

Pilvipalveluita voidaan hyödyntää myös testaus- ja kehistysympäristöinä. Testiajoja varten palvelinta ei välttämättä tarvitse pitää jatkuvasti päällä, joten pilvipalvelun käytöllä voidaan säästä kuluja. Samoin kehitysympäristöjä ja esimerkiksi jatkuvan integraation palvelinta voidaan pitää pilvessä. Laskentatehoa ja palvelininstanssien määrää voidaan skaalata molempiin suuntiin aina tarpeen mukaan. Lisäksi testaus- ja kehitysympäristöt voidaan pilvipalvelussa ajastaa sammumaan yöksi, viikonlopuksi ja lomien ajaksi, jos niille ei ole tarvetta. Pilvipalveluiden käytönmukainen laskutus tuottaa kulusäästöjä kaikissa edellä mainituissa tapauksissa.

2.2 Yritystietojärjestelmät

Alikohdassa 2.2.1 käsitellään yritystietojärjestelmän luonnetta ja sen aiheuttamia haasteita. Alikohdassa 2.2.2 käydään läpi yritystietojärjestelmissä yleisesti käytettyjä ohjelmistoarkkitehtuureita.

2.2.1 Yritystietojärjestelmän luonne

Yritystietojärjestelmä eroaa luonteeltaan pienemmistä sovelluksista useista eri näkökulmista. Yritystietojärjestelmän on tarkoitus ratkaista ja digitalisoida monimutkaisia liike-

toiminnan asettamia vaatimuksia. Tyypillisesti sen tulee skaalautua suurelle käyttäjämäärälle ja käyttäjätileihin liittyvä tietoturvan taso on vaatimuksiltaan tiukka. Käyttäjärooleja on paljon, integraatioita on paljon ja järjestelmä toimii usein yhteydessä muihin tietojärjestelmiin. Tim Walter esittää, että tietojärjestelmää ei pitäisi kutsua sen ominaisuuksien perusteella yritystietojärjestelmäksi vaan nimitys tulisi antaa vasta siinä vaiheessa, kun järjestelmä oikeasti pystyy ratkaisemaan liiketoiminnan haasteita [7]. Toisaalta voidaan ajatella, että ilman riittäviä ominaisuuksia tietojärjestelmä ei tähän pysty.

Yritystietojärjestelmä voi tarkoitukseltaan painottua enemmän asiakkaiden rajapinnaksi, kuten verkkokauppa tai yrityksen sisäiseen käyttöön, kuten toiminnanohjausjärjestelmä. Sama järjestelmä voi toteuttaa näistä molempia, mutta on hyvä huomata, että molemmat käyttötarkoitukset luovat omat vaatimuksensa järjestelmälle. Asiakkaiden käytössä olevaa järjestelmää saatetaan käyttää ympäri maailmaan, jolloin palvelun replikointi useisiin maantieteellisiin sijainteihin on perusteltua. Tällöin myös järjestelmän kuormituksessa voi olla suuremmat vaihtelut. Yrityksen sisäinen järjestelmä voi myös kansainvälisen yrityksen tapauksessa vaatia replikointia useampaan maantieteelliseen sijaintiin. Tyypillisesti yrityksen sisäisen järjestelmän suurimpia haasteita ovat lukuisat integraatiot muihin järjestelmiin. [8]

2.2.2 Yritystietojärjestelmien arkkitehtuurilliset ratkaisut

Ensimmäiset yritysten liiketoimintaa tukevat tietojärjestelmät olivat pääteyhteydellä toimivia asiakastietojärjestelmiä. Tällöin ohjelmakoodin suoritus tapahtui kokonaan palvelimella, mikä oli ensimmäinen rajoittava tekijä yhtäaikaisten käyttäjien määrässä. Seuraavaksi tuli laskentakykyä käyttäjän koneeseen ja tämä toi mukanaan asiakas-palvelinmallin. Käyttäjän kone suoritti ohjelmaa, teki pyyntöjä palvelimelle ja sai vastauksia jollain aikaviiveellä. Tämä mahdollisti yhtäaikaisten käyttäjien määrän kasvaessa kuormantasauksen ja jonotuksen useammalle palvelimelle. Seuraavana kehitysaskeleena tuli kolmikerrosarkkitehtuuri, jossa asiakas-palvelin -mallin palvelin jaettiin vielä kahtia käyttöliittymää palvelevaksi sekä toimintalogiikkaa suorittavaksi kerrokseksi. Kerroksia alettiin jakaa useaan osaan ja näin päästiin n-kerrosarkkitehtuurin. [9][10]

Palvelukeskeinen arkkitehtuuri (Service Oriented Architecture, SOA) on 2000-luvulla keksitty tapa toteuttaa ohjelmistoarkkitehtuuri, joka pohjautuu palveluihin. Palveluihin jaettu järjestelmä on usein myös jaettu kerroksiin. Yhdellä kerroksella voi olla yksi tai useampi palvelu eikä järjestelmän kerrosjako ole välttämättä niin selkeä kuin kerrosarkkitehtuurissa. Jokainen itsenäinen palvelu toimii rajapinnan takana ja sitä käyttävien palveluiden ei tarvitse tietää sen sisäisestä toteutuksesta mitään. Ideaalitilanteessa sisäistä toteutusta voidaan muuttaa tai palvelu voidaan vaihtaa toiseen vaihtoehtoiseen toteutukseen ilman että palvelun kutsujien tarvitsee muuttaa toimintaansa. Tämä mahdollistaa myös paljon suurempien järjestelmien toteuttamisen hallitusti. Kerrosarkkitehtuurissa ilman palveluita järjestelmän kasvaessa kerrokset kasvavat ja muutosten hallinta vaikeutuu. [10]

Palveluarkkitehtuurissa usein käytetään älykästä viestintävälikerrosta (messaging middleware), joka tietää mitä osia järjestelmässä on ja mistä ne löytyvät. Viestintävälikerroksen tehtäviin voi kuulua myös viestien muunnos, viestien rikastaminen ja eri protokollein yhteensovitus. Viestintävälikerros toteutetaan usein palveluväylän (Enterprise Service Bus, ESB) avulla. Viestintävälikerroksen kanssa kommunikoivilla järjestelmän osilla ei tyypillisesti ole omaa älykästä viestinvälitystä vaan ne luottavat siihen, että viestintävälikerros toimittaa viestin oikealle vastaanottajalle. Älykkääseen viestinvälittäjään nojaavaa palvelujoukkoa kutsutaan orkestraatioksi. Älykäs viestinvälittäjä toimii orkesterin kapellimestarin tapaan ohjaten kaikkien toimintaa. Viestintävälikerros mahdollistaa erilaisten palveluiden integroinnin osaksi järjestelmää, koska se pystyy muuntamaan viestit järjestelmään yhteensopiviksi. Integraatioiden näkökulmasta tämä on SOA-arkkitehtuurin suurin vahvuus. [10][11]

2.3 Pilviyhteensopivan ohjelmistoarkkitehtuurin hyödyntäminen

Alikohdassa 2.3.1 esitellään mikropalveluarkkitehtuuri pilviyhteensopivana ohjelmistoarkkitehtuurina. Alikohdassa 2.3.2 käydään läpi mikropalveluista ja pilvipalveluista aiheutuvia haasteita. Alikohdassa 2.3.3 tarkastellaan pilvipalveluiden käyttöön tarkoitettuja suunnitteluperiaatteita, jotka nojaavat vahvasti mikropalveluarkkitehtuurin käyttöön.

2.3.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuri muistuttaa monessa mielessä palvelukeskeistä arkkitehtuuria. Molemmissa arkkitehtuureissa tarkasteltavat komponentit ovat palveluita ja molemmat tuottavat hajautettuja järjestelmiä. Molemmissa palvelut pyritään tekemään ainakin jossain määrin itsenäisiksi moduuleiksi, joita pysty kehittämään itsenäisesti. Eroavaisuutena palvelupohjaisessa arkkitehtuurissa palvelut pyrkivät jakamaan mahdollisimman paljon tietoa keskenään toistensa hyödynnettäväksi. Palvelu voi yksittäisen operaation suorituksen aikana kutsua useita kertoja toista palvelua. Mikropalveluissa sen sijaan mikropalveluiden on tarkoitus jakaa mahdollisimman vähän tietoa keskenään ja pyrkiä suorittamaan operaatiot itsenäisesti. Tällöin mikropalveluiden vastuualueet täytyy olla selkeästi rajattu. Tietojärjestelmässä useampi palvelu saattaa tarvita jonkun operaation osana samaa toimintalogiikkaa, joka on kuitenkin niin pieni, että siitä ei kannata tehdä omaa palvelua. SOA-järjestelmässä on tapana tehdä tälle osalle operaatiota yhteinen toteutus, jota sen tarvitsijat kutsuvat. Tämä kuitenkin lisää palveluiden välistä riippuvuutta ja kasvattaa regressiovirheiden mahdollisuutta muutosten yhteydessä. Mikropalveluarkkitehtuurissa näiden operaatioiden yhteinen osa joko kopioidaan jokaiseen mikropalveluun tai kaikki operaatiot yhdistetään yhdeksi mikropalveluksi. Näin riippuvuudet joko katkaistaan tai ne tuodaan selkeästi yhdeksi moduuliksi. Valinta riippuu siitä, kuinka isoksi mikropalvelu kasvaisi, jos kaikki operaatiot toteutettaisiin yhdessä mikropalvelussa. [10][12]

Mikropalveluiden välinen integraatio on yksi keskeisimpiä eroja verrattuna palvelukeskeiseen arkkitehtuuriin. Mikropalvelut kommunikoivat keskenään ilman keskitettyä älykästä viestinvälittäjää. Viestinvälityksen logiikka tulee toteuttaa mikropalveluihin. Ilman älykästä viestinvälittäjää toimivaa palvelujoukkoa kutsutaan koreografiaksi. Mikropalveluiden välisen kommunikoinnin tulisi tapahtua mekanismeilla, jotka rajoittavat mahdollisimman vähän toteutustekniikkaa. Tämä jättää eniten vapauksia muuttaa yksittäisten mikropalveluiden sisäistä toteutusta ilman, että se vaikuttaa muihin. Viestien tulisi olla sisällöltään korkeamman abstraktiotason pyyntöjä, jotka eivät ota kantaa mikropalvelun toteutuksen yksityiskohtiin. Mikropalveluarkkitehtuurissa viestintää voi toteuttaa synkronisesti tai asynkronisesti. Mikropalveluarkkitehtuurissa tyypillisesti valitaan yksi viestintäprotokolla, jolla kaikkia mikropalveluita kutsutaan. Tämä on selkeä ero palvelupohjaisiin järjestelmiin, jossa viestintävälikerroksen mahdollistamana integroidaan useita viestintäprotokollia samaan järjestelmään. [10][12][13]

Mikropalvelu muodostuu yhdestä tai useammasta käyttöjärjestelmän prosessista, mutta samassa prosessissa ei ole useampaa mikropalvelua. Usein mikropalvelut sijaitsevat eri palvelimilla. Jokaista mikropalvelua voidaan skaalata kuormituksen mukaan käynnistämällä useita rinnakkaisia instansseja. Useissa pilvijärjestelmissä skaalaus toteutetaan automaattiseksi. Vikasietoisuuden näkökulmasta yksittäisen mikropalvelun skaalaus voidaan toteuttaa useisiin konesaleihin, jolloin paikalliset katkokset eivät lamaannuta edes yksittäistä mikropalvelua. Mikropalveluarkkitehtuurissa skaalaus voidaan toteuttaa vain skaalausta tarvitseville palveluille, jolloin kustannuksia saadaan optimoitua. Monoliittisen järjestelmän tapauksessa puolestaan koko järjestelmää pitäisi skaalata ja tällöin skaalaus koskisi myös niitä järjestelmän osia, jotka eivät ole skaalauksen tarpeessa. [12]

Yksi suuri lähdekoodirepositorio koko järjestelmälle voi aiheuttaa riskejä. Ominaisuuksien välille syntyy helpommin riippuvuuksia ja siten muutokset aiheuttavat helpommin regressiovirheitä. Mikropalvelun tapauksessa jokaisen mikropalvelun lähdekoodi on omassa repositoriossaan. Yhden mikropalvelun sisäiseen toimintaan tehtävät muutokset eivät vaikuta muihin. Koodin kääntäminen, automaattisten testien ajaminen sekä sovelluksen tuotantoon vienti ovat nopeampia ja kevyempiä tehdä vain tarvittaville mikropalveluille koko järjestelmän sijaan. Riippuvuuksista aiheutuva regressiovirheiden riski ja kehityksen nopeus ovat keskeisiä syitä sille, miksi mikropalveluarkkitehtuurissa palvelut halutaan pitää pieninä. Mikropalveluarkkitehtuurissa erilaiset mikropalvelut voidaan toteuttaa eri teknologiapinoilla, jolloin kuhunkin ongelmaan voidaan valita sopivin toteutustekniikka. Uusi teknologioiden kokeilu on vähemmän riskialtista yksi mikropalvelu kerrallaan. Myös tietokantoja voi olla tarpeen mukaan erilaisia. Jotkut palvelut saattavat hyötyä enemmän dokumenttitietokannasta ja toiset relaatiotietokannasta. Mikropalveluarkkitehtuurin mukaisessa järjestelmässä ei tyypillisesti nojauduta yhteen suureen tietokantaan. [12][13]

Mikropalveluarkkitehtuurin eräs tavoitteista on korkea vikasietoisuus. Esimerkiksi autoskaalausryhmissä olevat mikropalvelut pystyvät toipumaan automaattisesti yksittäisen instanssin vikaantumisesta. Vikasietoisuus pyritään huomioimaan suunnittelussa, mutta sitä voidaan myös testata aiheuttamalla vikatilanteita. Vikatilanteita aiheuttamalla pystytään varmistumaan jo varhaisessa vaiheessa järjestelmän vikasietoisuus. Aiheutetut virhetilanteet luovat lisää informaatiota järjestelmän toipumiskyvyistä ja nostavat esille mahdollisia ongelmakohtia. Ajoissa esille nousevat ongelmakohdat pystytään korjaamaan varhain ja järjestelmän vikasietoisuus paranee. Elokuvien ja TV-sarjojen suoratoistopalvelu Netflix on yksi varhaisimpia mikropalveluarkkitehtuurin omaksujia. Netflix on kehittänyt joukon työkaluja vikatilanteiden aiheuttamiseksi, joka tunnetaan nimellä Simian Army[10]. Siinä on erilaisten vikatilanteiden aiheuttamiseen erikoistuneita työkaluja, joita Netflix ajaa testiympäristön lisäksi myös tuotantoympäristössä. [14]

Suuren järjestelmän jakaminen mikropalveluihin myös tehostaa ohjelmistokehitysprosessia, sillä ohjelmistokehitys on tehokkaimmillaan tiimin ollessa sopivan pieni. Mikropalveluiden kehitykseen sopivaksi tiimin kooksi on esitetty muutamasta kehittäjästä kahteentoista kehittäjään. Vuonna 1968 Melvin Conway esitti ajatuksen, jota sittemmin on kutsuttu Conwayn laiksi. Sen mukaan järjestelmiä suunnittelevat organisaatiot tuottavat järjestelmiä, jotka ovat rakenteeltaan samanlaisia, kuin organisaation kommunikointirakenne. Näiden perusteella mikropalveluiden kehittäminen sopivan pienissä tiimeissä on optimaalista, jos tavoitteena on suurikokoinen järjestelmä, jossa keskinäiset riippuvuudet halutaan minimoida. [15]

2.3.2 Pilvipalveluiden ja mikropalveluarkkitehtuurin haasteita

Pilvipalvelun käyttö yritystietojärjestelmän tuotantoympäristönä tuo esille haasteita yleisellä tasolla. Pilvipalvelussa kulut eivät ole kiinteät, koska resursseista maksetaan käytön mukaan. Ohjelmointi- tai konfiguraatiovirheistä johtuva liiallinen resurssien ylöspäin skaalaus voi hallitsemattomana aiheuttaa yllättäviä kuluja. Myös palvelunestohyökkäykset voivat aiheuttaa vastaavan ongelman, jos sellaisiin ei ole oikealla tavalla varauduttu. Toinen kulurakenteeseen vaikuttava asia on palvelumallin valinta. IaaS-palvelumallissa infrastruktuuri pystytään rakentamaan vapaammin, mutta sen suunnittelu ja ylläpito vaativat enemmän työtä. PaaS-palvelumalli sen sijaan tarjoaa nämä valmiiksi tehtynä, mutta siinä ei pääse juurikaan vaikuttamaan infrastruktuurin toteutukseen. [16]

Mikropalveluarkkitehtuurilla on myös vaikutusta ohjelmiston kehitysprosessiin. Mikropalveluiden kehittäminen pienissä tiimeissä voi tuottaa omia haasteitaan. Erityisen ongelmallinen tilanne voi olla, jos kaikki kehittäjät ovat tottuneet toisenlaiseen ohjelmistokehitykseen. Eräässä projektissa on tullut esille tilanne, jossa kehittäjät siiloutuivat omiin mikropalveluihin. Tämän seurauksena esimerkiksi yhden mikropalvelun kehitystehtävien valmistuttua sen kehittäjät eivät lähteneet auttamaan muiden työtaakkaa vaan alkoivat

[10] Netflix/The Simian Army: https://github.com/Netflix/SimianArmy

kehittää ylimääräisiä hienosteluita omaan mikropalveluunsa. Koko järjestelmän kehitystä täytyy seurata eri tavalla silloin, kun sen kehitys koostuu mikropalveluita kehittävistä pienistä tiimeistä. [16]

Mikropalveluarkkitehtuurin mukainen ohjelmistokehitys vaatii kehittäjiltä laaja-alaisempaa osaamista. Esimerkiksi monoliittisen järjestelmän tietokannan ylläpitäjänä pystyy toimimaan kyseiseen tietokantaan erikoistunut ylläpitäjä, mutta mikropalveluissa tietokantoja tyypillisesti on useita ja ne voivat olla erilaisia. IaaS-palvelumallia käytettäessä pienestä kehitystiimistä täytyy löytyä osaamista sekä frontend- että backend-kehitykseen, tietokantojen hallintaan, järjestelmän ylläpitoon sekä pilvipalveluiden käyttöön. Ohjelmistokehittäjän ja järjestelmäasiantuntijan roolien yhdistyminen edustaa niin kutsuttua DevOps-kulttuuria. Sen tarkoituksena on jakaa vastuu kokonaisuudesta kaikille osapuolille sen sijaan, että vedettäisiin rajoja siitä mikä on kenenkin vastuulla. Tämä tuo kuitenkin haasteita osaamisen suhteen, koska aiemmissa ohjelmistokehitysmalleissa näin laajalle osaamiselle ei ole ollut niin suurta tarvetta. [16][17] [18]

Mikropalveluarkkitehtuurin mukaisessa järjestelmän suunnittelussa aiempien suunnitteluvirheiden korjaaminen voi olla erityisen haastavaa. Järjestelmän jakaminen mikropalveluiksi on parasta tehdä joko kerralla oikein tai vaiheittain edeten. Vaiheittainen eteneminen ensin muutamaan palveluun ja myöhemmin useampiin pienempiin on yleensä suositeltavampaa, koska se on vähemmän riskialtista. Väärin ositetun järjestelmän refaktorointi on todella työlästä ja altista regressiovirheille. Erilaisilla teknologiapinoilla toteutettuja mikropalveluita ei välttämättä pysty suoraan yhdistämään. Tällaisessa tilanteessa olisi helpompi aloittaa uudestaan alusta, mutta se ei välttämättä tule kyseeseen, jos projektiin on jo käytetty riittävästi työaikaa. Ongelman välttämiseksi järjestelmän pilkkomisvaiheita voidaan lykätä niin kauan, kunnes ollaan riittävän varmoja järjestelmän osien halutusta toimintatarkoituksesta. [12][16][17]

Mikropalveluarkkitehtuuri aiheuttaa haasteita kompleksisuutensa takia. Verrattuna suuremmista osista koostuvaan palvelukeskeiseen arkkitehtuuriin tai monoliittiseen järjestelmään mikropalveluiksi hajautettu järjestelmä sisältää itsessään enemmän liikkuvia osia. Verkkoyhteyden yli tapahtuva viestintä on altis kaikenlaisille tietoliikenteen häiriöille. Hajauttamisen myötä monet ominaisuudet toimivat luonnostaan asynkronisesti. Useampaa palvelua vaativa synkroninen toimenpide on haastavampaa toteuttaa mikropalveluiden kanssa verrattuna esimerkiksi monoliittiseen järjestelmään. Erityisen haasteen luo tällaisen toimenpiteen toteuttaminen transaktiona, sillä mikropalveluarkkitehtuurissa se tarkoittaa hajautettua transaktiota. Hajautetun transaktion hallinta itsenäisten osien kesken voi olla vaativaa ja etenkin hajautetun transaktion poikkeustilanteista toipuminen on hankalaa. [12][16]

Mikropalveluiksi hajautetun järjestelmän testaaminen luo omat haasteensa. Yksikkötestien tai yksittäisten mikropalveluiden testaaminen ei eroa kovinkaan paljoa monoliittisen

järjestelmän testauksesta, mutta koko järjestelmän läpäisevät end-to-end-testit ovat todellinen haaste. Tällaiset testit käyttävät useampaa mikropalvelua. Hyvän testikattavuuden saavuttaminen siten, että samoja asioita ei testata moneen kertaan on äärimmäisen vaikeaa. End-to-end-testit ovat raskaita suorittaa ja nopean kehitysprosessin kannalta on oleellista, että niiden aikana ei tehdä samoja asioita moneen kertaa. Jokaisella mikropalvelulla oma lähdekoodirepositorio ja oma versiointi, joten joudutaan tekemään valintoja, mikä versio kustakin palvelusta valitaan. Mikropalveluiden versioinnit täytyy kuitenkin pitää itsenäisinä, jotta ne eivät tule riippuvaisiksi toisten tietyistä versioista. Eräs hyvä käytäntö on vaatia automaattisten testien läpäisy siten, että muutoksia on vain yhdessä mikropalvelussa kerrallaan. Näin testaus tulee tehtyä kerrallaan pienempään joukkoon muutoksia, jolloin mahdollisten virheiden korjaus on helpompaa. Lisäksi muutosten toimivuus tulee testattua erillään muista muutoksista, mikä osaltaan antaa varmuutta mikropalveluiden keskinäisestä riippumattomuudesta. [12]

Koko järjestelmän läpäisevät testit ovat mikropalveluiden tapauksessa alttiita satunnaisille verkon ja pilvi-infrastruktuurin ongelmille. Tästä johtuvat satunnaisesti epäonnistuvat testit voivat helposti vähentää testien uskottavuutta kehittäjien keskuudessa ja luoda mielikuvan, että testien satunnainen epäonnistuminen on normaalia. Tällöin saatetaan helposti vain käynnistää testit uudestaan etsimättä varsinaisia ongelmia. Testien uskottavuuden ylläpito on tärkeää, jos halutaan varmistaa nopea virheiden korjaus. Nopea virheiden korjaus kehitysversioissa on oleellista nopean päivityssyklin kannalta. Satunnaisesti epäonnistuva testi täytyy heti poistaa tai korjata, jotta testaus saadaan pidettyä uskottavana. Koko järjestelmän läpäisevät testit käyttävät useaa mikropalvelua, jotka voivat olla kukin oman tiimin vastuulla. Tiimien ei kannata tehdä omia koko järjestelmän läpäiseviä testejä toisistaan riippumatta, koska tällöin päädytään helposti tekemään raskaita testejä, joissa testataan paljon samoja asioita. Jokaisella tiimillä täytyy olla ymmärrys yhteisten testien kokonaisuudesta ja myös niistä testeistä, joita tiimi ei ole itse tehnyt. Testien ylläpidosta ja epäonnistumisiin reagoinnista täytyy olla jaettu vastuu, jotta testit saadaan mahdollisimman nopeasti taas onnistumaan. [12][16]

Mikropalveluarkkitehtuurin pohjautuvan järjestelmän monitorointi on haastavaa. Kun monoliittisessa järjestelmässä seurataan yhden instanssin tilaa, mikropalveluiden tapauksessa seurattavia mikropalveluita ja niiden instansseja on lukuisia. Monitorointidataa kertyy lokitiedostoista, resurssien metriikoista ja mahdollisesti myös palveluiden vastausviiveiden kyselyistä, joten kokonaisuutta on jotenkin hallittava järkevän monitoroinnin toteuttamiseksi. Ratkaisuna tähän on monitorointidatan aggregointi keskitettyyn monitorointipalveluun. Aggregoinnin helpottamiseksi instanssien monitoroinnille on syytä valita tietty käytäntö, jota jokaisen mikropalvelun toteutuksessa noudatetaan. Yhtenevässä formaatissa oleva monitorointidata on helpointa koota yhteen. Lokitiedostojen seurantaa voidaan helpottaa lisäämällä jokaiseen toimintoketjuun tunniste, joka kulkee sen toimintoketjun kaikissa mikropalvelukutsuissa. Tällöin lokista tunnistetun virhetilanteen selvityksessä pystytään helposti löytämään muut saman toimintoketjun lokitiedot. [12]

2.3.3 Pilvipohjaisen järjestelmän suunnitteluperiaatteita

Useammat tahot ovat määritelleet pilvipohjaisen tai pilvinatiivin (cloud native) järjestelmän suunnitteluperiaatteita (patterns). Nämä suunnitteluperiaatteet ovat hyväksi havaittuja tapoja ratkaista tiettyjä pilvipohjaisten järjestelmien haasteita. Tässä esitellään muutamia ominaisuuksia, jotka esiintyvät useissa määritelmissä. Kaikissa määritelmissä yhdistävänä tekijänä nousee tilattomuus sekä vikasietoisuus. Yhdestäkään pilvipohjaisen järjestelmän resurssista ei saisi olla niin riippuvainen, että siitä ei voitaisi luopua. Pilvipohjainen järjestelmä tulisi suunnitella hajautetuksi järjestelmäksi, jossa yksittäisen järjestelmän osan toimimattomuus ei lamauta koko järjestelmää. [20][21][22]

Entinen Microsoftin työntekijä Bill Baker on esittänyt vertauskuvan, jonka mukaan pilvipohjaisessa järjestelmässä resurssit ovat kuin karjaa ja konesalin koneet kuin lemmikkejä. Lemmikkiä pidetään ainutlaatuisena, se nimetään, sitä hoidetaan, jos se sairastuu eikä sitä mielellään korvata toisella vastaavalla. Karja sen sijaan nähdään tuotantovälineenä ja sen arvo on täysin siinä mitä se tuottaa. Sairastuessa tai vanhentuessa se lopetetaan ja korvataan uudella. Tämä analogia näkyy pilvipohjaisen järjestelmän suunnitteluperiaatteissa useaan kertaan. Esimerkiksi virtuaalikoneiden konfiguraatioista tulisi olla kopiot erillisessä tietovarastossa. Näin uuden instanssin käynnistäminen ja vanhan terminointi onnistuvat helpoiten, kun kaikki tärkeä instanssilla oleva data on myös jossain muualla tallessa. Lisäksi useiden instanssien käyttämän konfiguraation muokkaaminen, versiointi ja periyttäminen helpottuvat. [19][20][21][22]

Pilvipohjaisen järjestelmän viestinvälityksessä tulisi käyttää jonkinlaista välityspalvelinta, joka tietää onko palvelu toiminnassa vai ei. Välityspalvelimen tehtävänä on ilmoittaa pyynnön tekijälle nopeasti, jos palvelu ei ole saatavissa. Näin palvelupyynnön tekijän ei tarvitse odottaa aikakatkaisuun asti saadakseen tietää palvelun olevan tavoittamattomissa. Välityspalvelimen täytyy seurata palvelun tilaa ja määrittää palvelu toimimattomaksi esimerkiksi silloin, kun riittävän moni peräkkäinen palvelun käyttäjän pyyntö on epäonnistunut. Välityspalvelin voi myös tehdä omia tarkistuksia palveluun ja päätellä niiden perusteella palvelun tilan. Välityspalvelimen tehtävää voi hoitaa kuormantasaaja, joka ohjaa liikennettä ja tasaa kuormaa useiden palveluinstanssien kesken. Vaihtoehtona kuormantasaajalle on viestijono, josta rinnakkaiset palveluinstanssit lukevat pyyntöjä. Viestijonon tapauksessa viestijonon maksimipituus toimii rajana, jonka täytyttyä pyynnön tekijä saa heti tiedon ilman aikakatkaisun odottelua. Monimutkaisessa pilvi-infrastruktuurissa tilapäiset verkkokatkokset tai yksittäisen viestin epäonnistunut käsittely ovat tyypillisiä tilanteita ja ne täytyy pystyä käsittelemään hallitusti. Tästä syystä palvelupyynnön tekijällä tulisi olla uudellenyritysmekanismi, jonka avulla se yrittää tehdä palvelupyyntöä uudestaan tietyn ajan jälkeen, jos aiempi yritys epäonnistuu. [20][21]

Palveluiden tilaa tulisi monitoroida, jotta nopeat automaattiset toimenpiteet häiriöstä toipumiselle olisivat mahdollisia. Tilan tarkistus voidaan toteuttaa saman rajapinnan kautta

mistä palvelua muutenkin käytetään esimerkiksi tekemällä yksinkertainen todellisen palvelukutsun kaltainen operaatio, joka onnistuakseen vaatii koko palvelun toimivuutta. Toimimaton palvelu pitäisi sulkea automattisesti ja käynnistää uudelleen. Tämä voi tarkoittaa joko sovellusprosessin tai koko instanssin terminointia ja uudelleenkäynnistystä. Kuormantasaajan tulisi heti häiriön ilmetessä lopettaa uusien pyyntöjen ohjaaminen vialliselle instanssille. Instansseja tai sovellusprosesseja voidaan myös satunnaisesti terminoida ja korvata uudella, vaikka häiriötä ei havaittaisi. Tällä tavoin voidaan varmistaa, että yksikään instanssi ei saavuta liian pitkää elinikää. [20][21]

Toistuvasti haettava data tulisi säilöä välimuisteihin, joihin pääsy on nopeampaa kuin alkuperäiseen lähteeseen. Välimuistia voidaan käyttää staattisen tai generoitavan datan säilömiseen. Staattisen datan tapauksessa hyöty näkyy siirtoviiveen pienenemissä, siirtokapasiteetin kasvamisessa sekä tietovaraston kuormituksen pienenemisessä. Generoitava data puolestaan saadaan haettua välimuistista pienemmällä prosessointiviiveellä, koska samaa datan generointia ei tarvitse tehdä uudestaan. Asiakas-palvelin –mallin näkökulmasta tarkasteltuna välimuisti voi sijaita asiakkaalla, palvelimelle tai näiden välissä. Asiakkaalla oleva välimuisti voi olla esimerkiksi selaimen tai sovellusohjelman välimuisti. Asiakkaan ja palvelimen välissä oleva välimuisti voi olla esimerkiksi sisällönjakoverkko (CDN, Content Distribution Network), joka sijaitsee maantieteellisesti loppukäyttäjää lähempänä. Välimuisti voi myös olla pilvipohjaisen järjestelmän sisäinen. [12][20][21]

Hajautetun pilvipohjaisen tietojärjestelmän jokaista osaa varten tulisi olla jonkinlainen käyttäjien autentikointi ja auktorisointi. Autentikointi varmistaa, että käyttäjä on se, kuka tämä väittää olevansa. Auktorisointi puolestaan kertoo mitä oikeuksia autentikoidulla käyttäjällä on. Käyttäjän kannalta ei kuitenkaan ole mielekästä, että jokaiseen tietojärjestelmän osaan tehtäisiin autentikointi erikseen. Tätä varten tulisi olla keskitetty identiteetin tarjoaja, johon käyttäjä autentikoituu. Tämän jälkeen järjestelmän osien tulee hoitaa vain kyseisen käyttäjän auktorisointi. Identiteetin tarjoaja voi olla järjestelmän sisään toteutettu osa tai se voi olla ulkoinen identiteetin tarjoaja, kuten Google Identity Platform[11]. Identiteetin tarjoajalle keskitetty autentikointi yksinkertaistaa käyttäjien oikeuksiin liittyvää ylläpitoa. [12][20]

2.4 Jatkuva toimitus ja tuotantoonvienti

Yksi keskeinen syy pilvipalveluiden käytölle on halu saada uusia ominaisuuksia nopeammin tuotantoon. Tämä puolestaan vaatii tehokkaita prosesseja koontiversion tuotantokelpoisuuden varmistamiseksi. Jatkuvan toimituksen prosessi on yksi vaihtoehto tämän ratkaisemiseksi. Sen käyttö ei edellytä pilvipalveluiden käyttöä, mutta mikropalveluarkkitehtuuria käytettäessä jatkuvan toimituksen hyödyntäminen on mielekästä. Kuten alikohdassa 2.3.1 tuotiin esille, yksittäiset mikropalvelut ovat nopeita kehittää ja testata ja siksi mikropalveluarkkitehtuuri pystyy hyötymään nopeasta toimitusprosessista. Lisäksi usein

[11] Google Identity Platform: https://developers.google.com/identity/

ja epäsäännöllisesti tehtävät tuotantopäivitykset eivät sovi sellaiseen perinteiseen konesalipalveluun, jossa päivitykset tehdään käyttöpalvelun tarjoajan toimesta säännöllisten huoltoikkunoiden aikaan.

Jatkuva toimitus ohjelmistokehityksessä tarkoittaa uusien koodimuutosten viemistä valmiiksi julkaisukuntoon usein ja lyhyellä syklillä. Se koostuu useista vaiheista, joiden aikana koodin toimivuus varmistetaan alkaen yksinkertaisista nopeasti testattavista asioista. Jatkuvan toimituksen prosessissa käydään ensin läpi jatkuvan integraation prosessi. Siinä uudet muutokset koodissa integroidaan muihin kehityshaarassa oleviin muutoksiin. Jatkuvan integraation palvelu käynnistyy automaattisesti, kun se havaitsee lähdekoodirepositoriossa muutoksen. Tyypillisesti se tekee koodille ensin käännöksen ja automaattisia testejä. Näin saadaan jo varhaisessa vaiheessa tieto mahdollisista ongelmista muiden muutosten kanssa. Käännöksessä tai testeissä havaitut ongelmat on helpompi korjata ennen, kuin niiden päälle ehtii kasaantua enemmän muutoksia. Onnistuneiden testiajojen jälkeen koodista luodaan koontiversio (build). Jatkuva toimitus on tämän pohjalta kehitetty pidempi prosessi, joka jatkaa koontiversion käsittelyä tätä pidemmälle. [12][23]

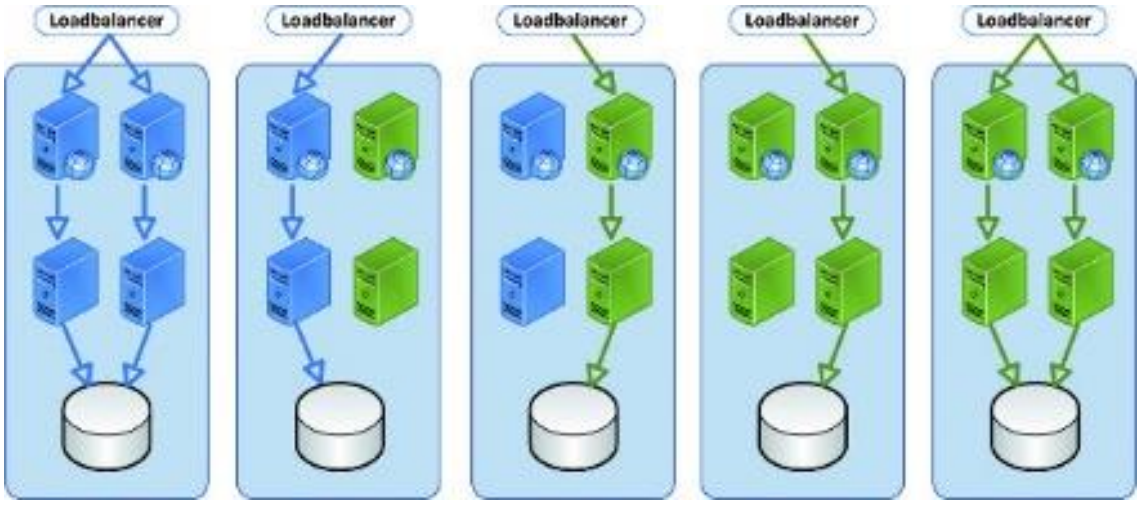
Jatkuvan toimituksen prosessi jatkuu tämän jälkeen yleensä käynnistäen koontiversion tuotannon kaltaiseen testiympäristöön, jossa sille saatetaan ajaa lisää automaattisia testejä. Testiympäristössä tehdään usein myös manuaalista testausta. Tämän lisäksi koontiversiolle saatetaan tehdä suorituskykytestausta. Prosessin aikana jokainen vaihe halutaan tehdä vain kerran. Kun jatkuvan integraation vaihe on saatu tehtyä, halutaan kaikki myöhemmät vaiheet tehdä samalla koontiversiolla. Jokaisen koontiversion kulkemista prosessin läpi on pystyttävä seuraamana, jotta voidaan varmistua koontiversion käyneen läpi prosessin jokaisen vaiheen. Prosessin päämääränä on tuottaa julkaisukelpoisia koontiversioita nopeasti ja usein. Ideaalitilanteessa prosessin läpi käynyt koontiversio voidaan päivittää tuotantoon yhtä nappia painamalla. [12]

Jatkuvan toimituksen prosessi kuvataan usein koontiversioputkena (build pipeline). Koontiversioputki kuvastaa prosessin vaiheet peräkkäisinä tai rinnakkaisina osina reaaliajassa, jolloin siitä näkee missä vaiheessa prosessi on menossa. Sen on tarkoitus myös auttaa järjestämään prosessin vaiheet järkevästi. Esimerkiksi automaattisten testien tapauksessa nopeasti suoritettavat yksikkötestit kannattaa ajaa omana vaiheena. Hitaampiin ja kattavampiin testeihin siirrytään vain silloin, jos nopeat testit menevät läpi. Näin tieto testien epäonnistumisesta saadaan nopeammin, kun ei tarvitse odottaa kaikkien testien päättymistä. Oikeanlaiset jatkuvan toimituksen työkalut mahdollistavat prosessin kuvaamisen koontiversioputkena. [12]

Näiden vaiheiden jälkeen koontiversio on julkaisuvalmis. On syytä huomata, että jokaista koontiversiota ei välttämättä haluta viedä tuotantoon. Valmiit koontiversiot vain mahdollistavat nopean tuotantoonviennin heti kun se halutaan tehdä. Myös tuotantoonvientiprosessi voidaan tehdä monella tavalla. Tietyt uudet tavat ovat nousseet pilvipalveluiden mahdollistamina suosituiksi. Tuotantoonviennissä kiinnostavia asioita ovat päivityksen

katkottomuus ja uuden koontiversion toimivuus tuotannossa. Uusien ominaisuuksien vieminen tuotantoon nopealla syklillä on siltä osin riskialtista, että kaikki ominaisuudet eivät saata toimia tuotantoympäristössä ja oikeilla loppukäyttäjillä niin kuin on haluttu. Tämän ongelman hallitsemiseksi esitellään kaksi tuotantoonvientiprosessia.

Blue/green deployment on tuotantoonvientitapa, jossa uutta versiota varten pystytetään toinen tuotantoympäristö tai nykyisestä tuotantokapasiteetista puolet varataan tuotantopäivitystä varten. Varsinainen päivitys tehdään ohjaamalla liikenne nykyisestä versiosta uuteen. Vanha versio jätetään rinnalle joksikin aikaa, kunnes saadaan varmuus, että uusi versio toimii luotettavasti. Ongelmien ilmetessä liikenne voidaan ohjata takaisin vanhaan versioon. Kuvassa 2 on kuvattu blue/green deploymentin vaiheet siten, että tuotantokapasiteetista puolet on varattu päivitystä varten. Tässä mallissa ongelmaksi saattaa nousta tuotannon tietokantaskeeman versiot. Tietokantaskeeman muutokset eivät välttämättä sovi yhteen vanhan version kanssa. Yksi vaihtoehto tämän ratkaisemiseksi on refaktoroida etukäteen tietokantaskeema tukemaan uutta ja vanhaa versiota. Kun uusi versio todetaan toimivaksi, voidaan tietokantaskeemasta poistaa tuki vanhalle versiolle. Päivityksen aikana tietokantaan voi kuitenkin kertyä kelvotonta dataa, joka ei enää toimi oikein, kun tietokannasta poistetaan tuki vanhalle versiolle. [12][24]



Kuva 2. *Blue/green deployment [25]*

Canary release on blue/green deploymentista pidemmälle kehitetty tuotantoonvientitapa. Siinä uuteen tuotantoympäristöön ohjataan käyttäjiä vähän kerrallaan. Tällöin siis osa käyttäjistä käyttää samaan aikaan vanhaa versiota ja osa uutta. Uuden version toimintaa monitoroidaan ja käyttäjämäärää lisätään vähitellen. Esimerkiksi yhteisöpalvelu Facebook julkaisee tällä tavalla suuret päivitykset ensin vain työntekijöidensä käyttöön ja sitten yksi maa tai alue kerrallaan. Uudelle versiolle voidaan asettaa suoritusvaatimuksia, joiden toteutumista monitoroidaan. Suoritusvaatimuksina voivat olla esimerkiksi jonkin

toiminnon viive, virheiden määrä lokissa tietyllä ajanjaksolla tai verkkokaupan tapauksessa ostoksen tehneen käyttäjän keskimäärin sivulla käyttämä aika ennen ostoksen tekoa. Suoritusvaatimuksia monitoroidessa kahta versiota pidetään usein pidempään rinnakkain kuin blue/green deploymentissa, jossa ainoastaan varmistetaan, että uusi koontiversio ja tuotantoympäristö toimivat. [12][26]

Jatkuvan toimituksen prosessi eroaa huomattavasti hitaammasta ja aikataulutetusta toimituksesta, jossa koontiversio tehdään harvoin, suuri joukko ominaisuuksia integroidaan ja testataan kerrallaan ja tuotantopäivityksiä tehdään harvoin. Jatkuva toimitus mahdollistaa tuotannon päivittämisen tuntien aikataulussa tai jopa sitäkin nopeammin. Esimerkiksi Amazonin verkkokauppa kertoo tehneensä vuoden 2011 toukokuussa tuotantopäivityksiä mikropalveluarkkitehtuuriin perustuvaan tietojärjestelmäänsä arkipäivinä keskimäärin 11,6 sekunnin välein. Tämä lukema kertoo mikropalveluiden rinnakkaisesta kehittämisestä ja niiden itsenäisestä tuotantoonviennistä. Liiketoimintaa parantavien uusien ominaisuuksien seisottaminen viikkoja odottamassa tuotantopäivitystä ei ole kannattavaa. Myös virheiden korjaukset saadaan tämän prosessin ansiosta nopeammin tuotantoon. Nopea toimitus, nopea virheiden korjaus ja mahdollisuus palata nopeasti edelliseen versioon luovat mahdollisuuden kokeilla uusia innovaatioita tuotannossa. [12][27]

3 AMAZON WEB SERVICES

Tässä luvussa esitellään Amazon Web Services –pilvipalvelu yleisellä tasolla sekä käydään yksityiskohtaisemmin läpi yritystietojärjestelmien kannalta oleellisia palveluita. Tällä hetkellä Amazon Web Services tarjoaa yhteensä yli 50 palvelua. Tässä työssä ei käsitellä niistä kaikkia.

3.1 Keskeisimmät Amazon Web Services –palvelut

Alikohdassa 3.1.1 esitellään Amazon Web Services –pilvipalvelun perustietoja sekä peruskäsitteitä. Alikohdassa 3.1.2 käydään läpi yleisiä Amazonin palveluiden pohjalla olevia infrastruktuuri- ja tietoturvakerroksia. Alikohdassa 3.1.3 esitellään virtuaalikoneet ja niihin keskeisesti liittyvät palvelut. Alikohdassa 3.1.4 käsitellään Amazonin tarjoamat tietokantapalvelut. Alikohdassa 3.1.5 käydään läpi Amazonin tietovarasto- ja sisällönjakopalveluita.

3.1.1 Peruskäsitteitä Amazon Web Services –palveluista

Amazon Web Services on vuonna 2006 perustettu pilvi-infrastruktuuria ja pilvialustoja tarjoava pilvipalvelu, jonka omistaa Amazon.com Incorporated [26]. AWS on markkinaosuudeltaan maailman suurin pilvi-infrastruktuuria tarjoava palvelu. Vuoden 2015 toisella neljänneksellä neljän maailman suurimman IaaS-palveluntarjoajan (Amazon, Microsoft, IBM, Google) yhteenlaskettu markkinaosuus maailman IaaS-markkinoilla oli 54% ja Amazonin 29% [3]. Palvelun alkuperäinen sekä nykyinen ydinalue on virtuaalisten palvelinresurssien tarjoaminen helpommin ja halvemmalla kuin perinteiset konesalipalvelut. AWS:n palveluiden käytön voi aloittaa kuka tahansa luottokortin omaava henkilö tai yritys. Resurssit saa käyttöön heti käyttäjätilin luomisen jälkeen. Käyttäjätilin luominen ja resurssien varaaminen tapahtuvat täysin ilman palveluntarjoajan avustusta. [28]

Fyysisesti Amazonin konesaleja sijaitsee kymmenellä AWS-alueella (region). Nämä sijoittuvat Eurooppaan, Aasian ja Tyynenmeren alueelle sekä Pohjois- ja Etelä-Amerikkaan. Jokaisella AWS-alueella on vähintään kaksi saatavuusaluetta (availability zone), jotka ovat toisistaan riippumattomia, fyysisesti erillisiä konesaleja, joilla on erilliset verkkoyhteydet ja sähkönsyöttö. Tällä pyritään varmistamaan, että käyttäjät voivat hajauttamalla varmistaa palveluidensa saatavuuden jokaisella AWS-alueella sähkökatkoista tai tietoliikennehäiriöistä huolimatta. Liiketoimintakriittisen yritystietojärjestelmän kannalta tämän on merkittävä ominaisuus. [29]

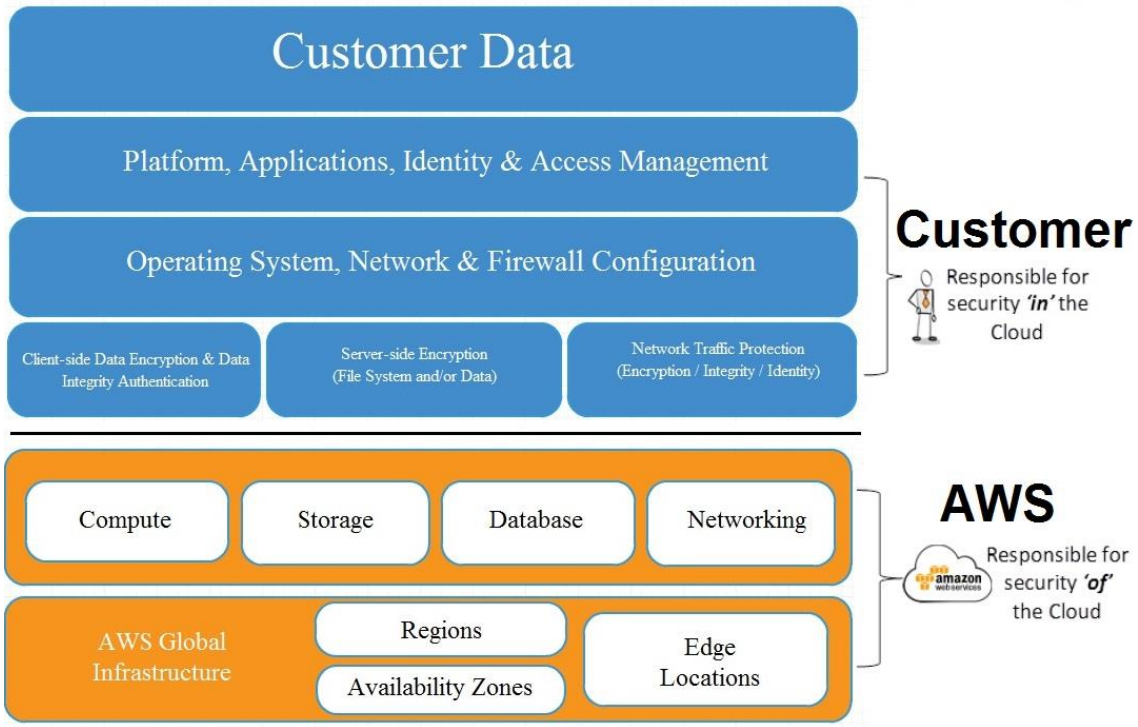
Amazon Web Services tarjoaa erilaisia resursseja tyypistä riippuen joko määrätyn kokoisina instansseina tai jatkuvana varantona. Instansseja ovat esimerkiksi EC2-virtuaalikoneet (Elastic Computing Cloud)[12], joita varatessa käyttäjä valitsee instanssityypin, jonka mukaan määrittyvät muun muassa koneen keskusmuistin määrä sekä suoritinteho. Jatkuvaksi varannoksi voidaan luokitella esimerkiksi S3-objektivarasto (Simple Storage Service)[13], joka kasvaa jatkuvasti käyttäjän tarpeiden mukaan ilman tietyn kokoisia resurssivarauksia. Kaikkia Amazonin pilviresursseja voidaan käyttää sekä selainpohjaisen AWS-konsolin kautta, että ohjelmallisen rajapinnan (API) kautta. Amazon tarjoaa valmiit kirjastot useille ohjelmointikielille sekä komentoriviskriptejä, joiden avulla rajapintaa voidaan käyttää. Ohjelmallisen rajapinnan avulla virtuaalisen infrastruktuurin pystyy automatisoimaan. Tarvittavien resurssien määrän kasvaessa käsin tehtävä hallinnointi muuttuu vaikeaksi ja virhealttiiksi. Tällöin resurssien ohjelmallinen hallinnointi on helpompaa ja turvallisempaa.

Amazon Web Services määrittelee käyttäjilleen selkeästi palveluntarjoajan ja palvelun käyttäjän vastuualueet niin sanotussa jaetun vastuun mallissa (shared responsibility model). Malli on havainnollistettu kuvassa 3. Amazon vastaa pilvipalveluiden toimivuudesta ja tietoturvasta laitteiston tasolla. Käyttäjän vastuulla on huolehtia pilviresurssien varaamisesta, konfiguroinnista, tietoturvasta ohjelmallisella tasolla. Esimerkiksi EC2-virtuaalikoneilla käyttöjärjestelmän ja ohjelmien tietoturvapäivitykset sekä tietoliikenneverkon ja palomuurien konfiguraatiot ovat käyttäjän vastuulla. Verrattaessa tätä alikohdassa 2.1.2 esitettyihin pilvipalveluiden palvelumalleihin huomataan yhtäläisyys IaaS-palvelumallin määritelmän kanssa. [30]

3.1.2 Pilvi-infrastruktuurin alimmat kerrokset

Tietoturvan ja hallinnoinnin parantamiseksi pilviresurssit voidaan ryhmitellä usealla eri tavalla. Tässä työssä tietoturvaan liittyvöt ryhmittelyt määritellään pilvi-infrastruktuurin alimmiksi kerroksiksi, koska kaikki Amazonin pilviresurssit toimivat näiden ryhmien puitteissa. Myös pilvi-infrastruktuurin suunnittelu on luontevaa aloittaa näistä ryhmittelyistä. Suuren yritystietojärjestelmän tapauksessa valtava määrä resursseja on mielekästä ryhmitellä sekä hallinnoinnin että tietoturvan kannalta.

[12] Amazon EC2: https://aws.amazon.com/ec2/
[13] Amazon S3: https://aws.amazon.com/s3/



Kuva 3. *Amazon Web Services –pilvipalvelun jaetun vastuun malli [30]*

Kaikki AWS:n instanssit toimivat käyttöoikeusryhmissä (security group), joihin voidaan määrittää erilaisia pääsyoikeuksia. Käyttöoikeusryhmä toimii virtuaalisena palomuurina sallien yhteydet tietystä portista tietyllä protokollalla. Oletuksena kaikki lähtevä liikenne on sallittu ja kaikki saapuva liikenne estetty. Saapuvia yhteyksiä voi sallia portti- ja protokollakohtaisesti tietyistä IP-osoitteista, aliverkoista tai kaikkialta. Käyttöoikeusryhmiin ei pysty määrittämään kieltäviä sääntöjä. Lähtevään liikenteeseen ulkoapäin tulevat vastaukset ovat automaattisesti sallittua eli kyseessä on tilallinen palomuuri. Käyttöoikeusryhmä on tosiasiassa joukko asetuksia, jotka rajaavat jokaisen instanssin verkkoliikennettä erikseen. Sitä ei siis pidä ajatella ryhmän yhteisenä ulkokuorena, jonka sisäpuolella instanssit ovat. Säännöt rajaavat myös samassa käyttöoikeusryhmässä olevien instanssien keskinäistä liikennettä. [31]

AWS:n virtuaalipilvi (virtual private cloud)[14] on keino luoda virtuaalisesti rajattuja AWS-pilven osia. Virtuaalipilvelle määritellään luomisvaiheessa IP-aliverkko sekä oletusturvallisuusryhmä. AWS-instanssit kuuluvat aina johonkin virtuaalipilveen. Mikäli sellaista ei ole erikseen määritelty, instanssit kuuluvat oletusvirtuaalipilveen. Virtuaalipilven IP-osoiteavaruuden voi jakaa useampaa IP-aliverkkoon. Yleensä suuri yritystietojärjestelmä vaatii useita IP-aliverkkoja, sillä saatavuuden varmistamiseksi se on tarpeen saada toimimaan useammalla saatavuusalueella ja yksi AWS:n IP-aliverkko voi toimia vain yhdellä

[14] Amazon Virtual Private Cloud: https://aws.amazon.com/vpc/

saatavuusalueella. Joskus voi olla myös tarpeen rajata järjestelmän eri osia IP-aliverkoilla tai virtuaalipilvillä tietoturvasyistä. [32]

Virtuaalipilven IP-aliverkoille voidaan myös asettaa virtuaalinen palomuuri ACLs (Network Access Control Lists). Käyttöoikeusryhmistä poiketen tämä ei vaikuta IP-aliverkon sisällä instanssien väliseen kommunikointiin. ACLs mahdollistaa sekä sallivien että kieltävien sääntöjen luomisen portti ja protokolla kohtaisesti. Säännöt numeroidaan siten, että tärkein on pienimmällä numerolla. Yhteyden salliminen päätellään säännöistä numerojärjestyksessä. Jos pienemmän numeron sääntö kieltää yhteyden eivät suuremman numeron säännöt voi kieltoa kumota. ACLs on tilaton palomuuri eli siinä lähtevään liikenteeseen tulevat vastaukset täytyy erikseen sallia. Erona käyttöoikeusryhmiin ACLs:n säännöt koskevat aina kaikkia sen aliverkossa olevia instansseja ilman poikkeuksia. Käyttöoikeusryhmien ja ACLs:n yhtäaikainen käyttö on mahdollista. [33]

Jokaisella virtuaalipilvellä on oma oletus käyttöoikeusryhmä sekä oletus-ACLs, joihin kaikki siellä toimivat instanssit kuuluvat oletuksena. Käyttöoikeusryhmät ovat kuitenkin instanssikohtaisia, joten oletusryhmän lisäksi jokainen instanssi voidaan määrittää kuuluman yhteen tai useampaan virtuaalipilvestä riippumattomaan käyttöoikeusryhmään. Virtuaalipilven sisälle voidaan esimerkiksi määrittää käyttöoikeusryhmien avulla tietokantainstansseja, joihin saa yhteyden vain kyseisen virtuaalipilven sisällä olevilta EC2-virtuaalikoneilta. Tällöin EC2-instansseilla sekä tietokantainstansseilla on eri käyttöoikeusryhmät, vaikka ne ovat samassa virtuaalipilvessä.

3.1.3 Virtuaalikoneet, autoskaalausryhmät ja kuormantasaus

Amazonin pilvipalveluiden yksi keskeisin resurssi on EC2. EC2 on palvelu, joka tarjoaa virtuaalisia palvelinkoneita. Virtuaalikoneita tarjotaan eri määrällä keskusmuistia ja suoritintehoa. EC2-palvelu tukee Linux-[15], Unix-[16] ja Windows-käyttöjärjestelmiä[17]. Virtuaalikonetta luodessa sille valitaan AMI (Amazon Machine Image), johon on tallennettu malli virtuaalikoneen käyttöjärjestelmästä, siihen asennetuista ohjelmista, käyttöoikeuksista sekä levyosioista. Käyttäjä voi valita valmiiksi tehdyn AMI:n tai luoda sellaisen itse. Samaa AMI:a voidaan käyttää useamman samanlaisen virtuaalikoneen käynnistämiseen. Tämä helpottaa erityisesti horisontaalisen autoskaalauksen toteuttamista samanlaisilla instansseilla. EC2-virtuaalikoneella ajetaan tyypillisesti sovelluspalvelinta tai apuohjelmia. Vaikka Amazon tarjoaa kuormantasaajia ja viestijonoja valmiina palveluina voi käyttäjä halutessaan asentaa ja konfiguroida EC2-koneen toimimaan tällaisena palveluna. EC2-koneella voi periaatteessa tehdä mitä tahansa mitä konesalista vuokrattavalla fyysisellä palvelinkoneellakin voi. [34]

[15] Linux: https://www.linux.com/
[16] Unix: http://www.unix.org/
[17] Windows: http://www.microsoft.com/en-us/windows

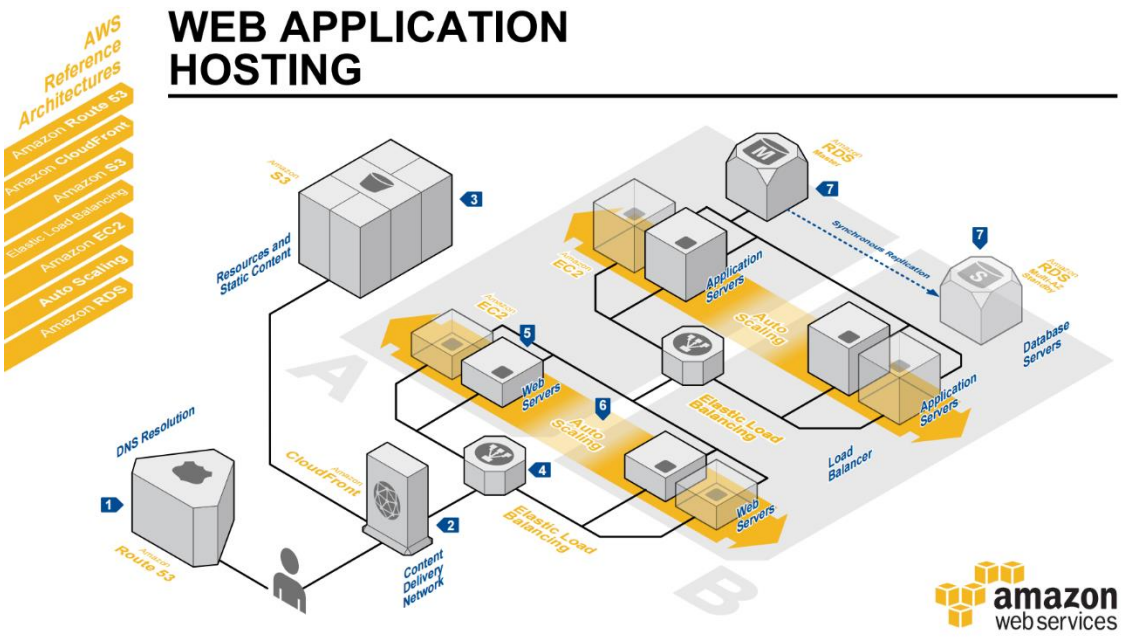
Virtuaalikoneella on sisäinen tallennustila, joka tyhjenee aina kun kone sammutetaan. Uudelleen käynnistettäessä virtuaalikone käynnistyy AMI:n kuvaamaan tilaan. Pysyvää tallennusta varten virtuaalikoneeseen voidaan liittää ulkoisia levyosioita. Levyosiopalvelu on nimeltään EBS (Elastic Block Store)[18]. EBS tarjoaa kolmea erilaista hinnoittelumallia levyosioille. Yleiskäyttöinen SSD-levylle tallennettava EBS maksaa vain tallennettujen gigatavujen verran. Toinen vaihtoehto on nopeampia levyoperaatioita tukeva IOPS-provisioitu SSD-levy, jossa voi valita kuinka monta levyoperaatiota sekunnissa haluaa. Tällöin levyosiolle taataan levyoperaationopeus, joka voi maksimissaan olla 20000 luku- tai kirjoitusoperaatiota sekunnissa. Tässä vaihtoehdossa maksetaan myös jokaisesta miljoonasta levyoperaatiosta. Kolmas vaihtoehto on magneettilevy, jonka tallennustila gigatavua kohden on halvempi. Magneettilevy on kuitenkin levyoperaatioiden suhteen hitain ja myös sitä käytettäessä maksetaan jokaisesta miljoonasta levyoperaatiosta. [35]

Autoskaalausryhmä (ASG, Auto-Scaling Group)[19] on Amazonin palvelu, jolla voidaan käynnistää ja sammuttaa EC2-virtuaalikoneita tiettyjen ehtojen täyttyessä. Autoskaalausryhmään määritellään minimi ja maksimi lukumäärä instansseja. Kaikki instanssit käynnistyvät saman käynnistyskonfiguraation (Launch Configuration) mukaisesti. Siinä määritellään käynnistettävä AMI, EC2-instanssityyppi, EBS-levyosion koko sekä turvallisuusryhmä. Instanssien määrää kasvatetaan haluttujen sääntöjen mukaan esimerkiksi silloin, kun instanssien suoritinkäyttö nousee riittävän korkealle. Toisena esimerkkinä mainittakoon järjestelmä, jossa EC2-instansseille ohjataan viestejä viestijonosta. Tällöin instansseja voidaan lisätä esimerkiksi, kun viestijonossa olevien viestien määrä kasvaa tietyn raja-arvon yli. Molemmissa esimerkeissä instanssien määrää voidaan vähentää kohti minimimäärää, kun suoritinkäyttö tai viestijonon pituus pienenee riittävästi. Autoskaalausryhmä on mahdollista asettaa myös muuttamaan instanssien määrää kellonaikojen tai viikonpäivien mukaan. [36]

Horisontaalisesti skaalattujen, rinnakkaisten EC2-koneiden kuormaa voidaan tasata automaattisesti käyttämällä kuormantasausta (ELB, Elastic Load Balancing)[20]. Kuormantasaaja on instanssi, joka jakelee palveluun tulevia pyyntöjä palvelun toteuttaville EC2-instansseille pyrkien jakamaan kuorman tasaisesti instanssien välillä. Amazonin kuormantasaajaa voidaan käyttää joko vakiolukumääräisen EC2-instanssijoukon kanssa tai autoskaalausryhmän kanssa. Kuormantasaajainstanssille ei valita kapasiteettia sitä luodessa vaan sen kapasiteetti kasvaa joustavasti käytön mukaan. Amazonin kuormantasaaja voi olla, joko julkinen tai sisäinen. Julkisella kuormantasaajalla on julkinen IP-osoite ja

[18] Amazon Elastic Block Store: https://aws.amazon.com/ebs/
[19] Auto Scaling: https://aws.amazon.com/autoscaling/
[20] Elastic Load Balancing: https://aws.amazon.com/elasticloadbalancing/

domain-nimi, joiden avulla siihen voi kohdistaa pyyntöjä julkisesta verkosta. ELB mahdollistaa myös SSL-terminoinnin[21] sekä sisäisten että ulkoisten kuormantasaajien tapauksessa. Kuvassa 3 on esimerkki autoskaalausryhmien ja kuormantasaajien käytöstä Web-sovelluksen infrastruktuurissa. [37]



Kuva 4. *Esimerkki Web-sovelluksen infrastruktuurista Amazon Web Services –pilvipalvelussa [38]*

3.1.4 Tietokannat

Amazon Relational Database Service (RDS)[22] on relaatiotietokantapalvelu, joka toimii tietokantainstanssien avulla. Se tukee MySQL[23], OracleDB[24], PostgreSQL[25], MariaDB[26], Amazon Aurora[27] ja Microsoft SQL Server[28] tietokannanhallintajärjestelmiä. Tietokannan käyttöoikeudet määritellään instanssin asetuksista eikä tietokantainstanssiin voi ottaa SSH-yhteyttä. Eri hintaisia tietokantainstansseja on tarjolla samaan tapaan kuin EC2-instansseja eli suoritintehoa ja keskusmuistia saa tarpeen mukaan. Lisäksi instanssille määritellään erikseen halutun kokoinen EBS-tallennuslevy. RDS-tietokantainstanssit toimivat käyttöoikeusryhmissä samalla tavalla kuin EC2-instanssit. RDS-palvelussa Amazon hoitaa automaattisesti tietoturvapäivitykset sekä tietokannan varmuuskopioinnin. Myös

[21] SSL: https://tools.ietf.org/html/rfc5246
[22] Amazon RDS: https://aws.amazon.com/rds/
[23] MySQL: https://www.mysql.com/
[24] Oracle Database: https://www.oracle.com/database/
[25] PostgreSQL: http://www.postgresql.org/
[26] MariaDB: https://mariadb.org/
[27] Amazon Aurora: https://aws.amazon.com/rds/aurora/
[28] Microsoft SQL Server: https://www.microsoft.com/en-us/server-cloud/products/sql-server/

tietokannanhallintajärjestelmän versiopäivitykset voi asettaa automaattisiksi. Näitä ylläpitotöitä varten määritetään palveluikkuna, jonka tarkoituksena on ajoittaa päivitykset ajankohdalle, jolloin tietokannan käyttö on mahdollisimman vähäistä. [39]

Amazon DynamoDB[29] on puolestaan Amazonin tarjoama NoSQL-tietokantapalvelu[30]. DynamoDB on dokumenttipohjainen tietokanta. Toisin kuin RDS-palvelun tapauksessa, sitä käytettäessä ei varata instansseja vaan se toimii jatkuvana varantona S3-palvelun tapaan. DynamoDB:ssä data tallennetaan tauluihin, jotka sisältävät avain-arvo –pareja. Myös sisäkkäiset avain-arvo –parit ovat tuettuja 32 sisäkkäiseen tasoon asti. DynamoDB voi myös liitännäisten avulla toimia pohjana Titan-graafitietokannalle[31]. [40]

Amazon ElastiCache[32] on tietokantapalvelu muistinvaraisille tietokannoille. Siinä voidaan käyttää Memcached-[33] tai Redis-tietokantaa[34]. Muistinvarainen tietokanta pitää tiedon keskusmuistissa kiintolevyn sijaan. Tämä nopeuttaa tietokannan toimintaa ja säästää levytilaa. Muistinvaraisen tietokannan tiedot eivät säily, jos tietokanta sammutetaan. Tällaisen tietokannan käyttötarkoitus voi olla esimerkiksi HTTP-istuntotietojen tallennus. Autoskaalausryhmässä toimivat Web-palvelimet voivat tallentaa istuntotietonsa yhteiseen paikkaan, jolloin mikä tahansa koneista voi jatkaa toisen aloittamaa istuntoa. [41]

Amazon Redshift[35] on Amazonin tietovarastopalvelu tietokannoille, joiden kokoluokka on petatavuja. Se pohjautuu sarakepohjaiseen tietokannanhallintajärjestelmään. Erona relaatiotietokannoissa tyypillisemmin esiintyvää rivipohjaiseen tiedon tallennukseen sarakepohjainen tallennus mahdollistaa nopeamman yhden sarakkeen tietojen lukemisen suuresta rivimäärästä. Vastaavasti siinä on hitaampaa pienemmästä rivimäärästä jokaisen rivin kaikki sarakkeet. Sarakepohjaisessa tiedon tallennuksessa tietokannanhallintajärjestelmä tallentaa yhden sarakkeen tiedot peräkkäisiin tallennussektoreihin, jolloin niiden peräkkäinen lukeminen nopeutuu. Rivipohjaisessa tietokannassa puolestaan koko rivi tallennetaan peräkkäin, jolloin koko rivin lukeminen on nopeaa. [42]

3.1.5 Tietovarastot ja sisällönjakelu

Amazon Simple Storage Service (S3) on objektivarasto, johon voidaan tallentaa tiedostoja tai tekstimuotoista dataa. S3 koostuu ämpäreistä (bucket), jotka ovat erillisiä tallennusyksiköitä. Ämpärin objekteille voidaan antaa rakenteiset nimet (esimerkiksi /kansio/alikasio/objekti) ja niiden avulla ämpäriin muodostuu näennäinen hakemistorakenne. Kaikkiin S3:n resursseihin voidaan määrittää käyttöoikeuksia. Tallennettuja objekteja on

[29] Amazon DynamoDB: https://aws.amazon.com/dynamodb/
[30] NoSQL: http://nosql-database.org/
[31] Titan: http://titan.thinkaurelius.com/
[32] Amazon ElastiCache: https://aws.amazon.com/elasticache/
[33] Memcached: https://memcached.org/
[34] Redis: http://redis.io/
[35] Amazon Redshift: https://aws.amazon.com/redshift/

mahdollista versioida ja varmuuskopioida. Versiointi auttaa suojaamaan objekteja tahattomalta ylikirjoitukselta tai poistamiselta. Objekteille voidaan myös määrittää elinkaari, jonka mukaisesti ne arkistoidaan tietyn ikäisinä ja poistetaan tietyn ikäisinä. Tietyn AWS-alueen sisällä tallennettu tieto ei poistu sen ulkopuolelle, ellei sitä erikseen halua siirtää. [43]

Amazon Glacier[36] on tiedonarkistointipalvelu. Glacieriin tallennetaan pakattuja tietoarkistoja, jotka voivat olla kooltaan jopa 40 teratavua. Arkiston tallennus ja nouto ovat hitaita operaatioita ja voivat kestää tunteja. Kerran tallennettua arkistoa ei voi muuttaa ilman, että sen ensin noutaa ja sitten tallentaa uudestaan uutena arkistona. Arkistoja voidaan koota holviin (vault). Holvi helpottaa suurien arkistomäärien lajittelua ja käyttöoikeuksien hallintaa. Normaalien käyttöoikeuksien lisäksi holvin pystyy lukitsemaan siten, että sieltä pystytään vain lukemaan tietoa. Holvin lukitukseen määritetään määräaika, jota ennen lukitusta ei voi kukaan avata. Amazon lupaa Glacieriin tallennettujen arkistojen pysyvyydeksi 99.999999999%, eli yksi tallennus sadasta miljardista saattaa kadota. [44]

Amazon CloudFront[37] on sisällönjakopalvelu (CDN). Sen avulla verkkopalvelun sisältö voidaan ladata käyttäjää maantieteellisesti lähimmän päätepisteen kautta, jolloin vasteaika ja siirtonopeus pystytään optimoimaan. Tiedostoja siirretään välimuistiin päätepisteille alkuperäisestä sijainnista sitä mukaa, kun niitä ensimmäistä kertaa käytetään. CloudFront-päätepisteitä on enemmän, kuin AWS-alueita. Esimerkiksi Yhdysvaltojen AWS-alueella toimiva verkkopalvelu voi jakaa sisältöä Suomen käyttäjille Tukholman päätepisteestä ja Uuden-Seelannin käyttäjille Melbournen tai Sydneyn päätepisteestä. Tässä esimerkissä maantieteellinen etäisyys lyhenee huomattavasti. CloudFrontin sisällön alkuperäisenä lähteenä voi olla S3-ämpäri, EC2-instanssi tai kuormantasaaja. CloudFrontia käytettäessä hinta koostuu CloudFrontin lähettämästä datasta, CloudFrontin ja alkuperäisen lähteen välisestä datansiirrosta sekä CloudFrontiin tulevien pyyntöjen lukumäärästä. [45]

3.2 Palveluita pilviyhteensopivan yritystietojärjestelmän vaatimusten ratkaisemiseksi

Alikohdassa 3.2.1 esitellään Amazonin tarjoamia viestinvälitysratkaisuita. Alikohdassa 3.2.2 käydään läpi keinoja Amazonin pilvipalveluiden yhdistämiseksi osaksi yrityksen konesalia tai sisäverkkoa. Alikohdassa 3.2.3 esitellään kolme vaihtoehtoista laskentapalvelua, jotka sopivat tietynlaisiin tarpeisiin paremmin, kuin EC2.

[36] Amazon Glacier: https://aws.amazon.com/glacier/
[37] Amazon CloudFront: https://aws.amazon.com/cloudfront/

3.2.1 Viestinvälitys Amazonin pilvessä

Yritystietojärjestelmät ovat usein suuren kokonsa vuoksi jaettu useisiin osiin joiden välillä tarvitaan viestin välitystä. Viestinvälitys voi pohjautua älykkääseen keskitettyyn viestinvälittäjään tai yksinkertaisiin viestijonoihin. Näistä jälkimmäinen on pilvipalveluissa sekä mikropalveluarkkitehtuurissa suositeltu tapa viestinvälitykseen. Järjestelmän osien välisen kommunikoinnin lisäksi usein on tarpeen välittää viestejä järjestelmän ylläpitäjille tai generoida sähköpostia käyttäjille. Amazon Web Services tarjoaa palveluita kaikkiin näihin tarpeisiin.

AWS-resurssien välisessä kommunikoinnissa voidaan käyttää SQS-viestijonopalvelua (Simple Queue Service)[38]. Viestijonojen käyttö on tarpeen suurien hajautettujen yritystietojärjestelmien asynkronisen viestinvälityksen toteuttamisessa. Viestejä saattaa esimerkiksi tulla purskeina, jolloin viestejä on tarpeen kerätä jonoon odottamaan käsittelyä. Myös viestejä vastaanottava osapuoli saattaa olla väliaikaisesti poissa käytöstä. Tällöin viestit odottavat jonossa kunnes vastaanottava osapuoli alkaa niitä lukea. SQS-viestijonossa viestien maksimi elinikä on 14 vuorokautta. Viestijonopalvelun avulla voi myös tehdä viestien priorisointia. Tällöin eri prioriteeteille tehdään omat viestijonot ja viestin lukija lukee viestejä priorisoidusti. [46].

SQS-viestijono on tarkoitettu käytettäväksi ohjelmallisesti eikä se ole käyttäjäystävällinen tapa lähettää ilmoituksia esimerkiksi järjestelmän valvojalle. Tätä varten Amazon tarjoaa SNS-ilmoitusviestipalvelun (Simple Notification Service)[39]. SNS muodostuu aiheista (topic) joihin julkaistaan ilmoituksia. Ilmoitusten vastaanottajat voivat vastaanottaa ilmoituksia haluamallaan tavalla esimerkiksi mobiilisovelluksella, tekstiviestillä, sähköpostilla tai HTTP-protokolla avulla. SNS:n avulla voi myös lähettää viestejä SQS-viestijonoon.[47].

Amazonin pilvipalveluita on käytetty roskapostin lähettämiseen, joten lähtökohtaisesti useat sähköpostipalvelimen pitävät AWS:n IP-osoitteita mustalla listalla eivätkä välitä sieltä tulevaa postia. Tästä syystä Amazon on luonut oman palvelun sähköpostin lähetykseen. SES (Simple Email Service)[40] suodattaa lähetettävää roskapostia ja sen kautta lähetettyyn postiin suhtaudutaan useimmilla sähköpostipalvelimilla luottavaisesti. SES toimii myös saapuvan postin palvelimena. Sen avulla voi myös luoda statistiikkaa sähköpostiliikenteestä. SES voi myös generoida SNS-ilmoituksia tietyistä sähköposteista. Saapuvaan postin voi myös luoda suodattimia, jolloin esimerkiksi saapuva posti otetaan vastaan vain tiettyjen sähköpostipalvelimien IP-osoitteista. [48].

[38] Amazon Simple Queue Service: https://aws.amazon.com/sqs/
[39] Amazon Simple Notification Service: https://aws.amazon.com/sns/
[40] Amazon Simple Email Service: https://aws.amazon.com/ses/

3.2.2 Pilvi-infrastruktuurin yhdistäminen osaksi yrityksen konesalia

Pilvi-infrastruktuuria hyödyntävän yritystietojärjestelmän voi olla tarpeen integroitua yrityksen oman konesalin palveluihin. Omassa konesalissa saattaa olla käyttäjätietokanta tai joku muu järjestelmä, joka halutaan yhdistää pilvipohjaiseen järjestelmään. Joskus yhdistäminen on tarpeen, koska pilvestä halutaan vain lisätallennustilaa tai sinne säilötään varmuuskopioita. Myös omassa konesalissa toimivan järjestelmän kuormituspiikkien aikana tarvitsema pilvestä käyttöönotettava lisäkapasiteetti tarvitsee yhteyden konesalista pilveen. Käydään seuraavaksi läpi Amazonin tarjoamia palveluita tällaisten yhteyksien muodostamiseen.

Amazonin virtuaalipilven voi yhdistää VPN-yhteydellä (Virtual Private Network)[41] osaksi yrityksen sisäverkkoa. Yhdistämisen voi tehdä Amazonin VPN-palvelun avulla tai käyttäjä voi itse asentaa ja konfiguroida VPN-sovelluksen EC2-instanssille. Amazonin VPN-palvelua käytettäessä molempiin päihin yhteyttä luodaan yhdyskäytävät. Yhteys avataan aina käyttäjän sisäverkon suunnasta eikä sitä ole mahdollista avata Amazonin yhdyskäytävän suunnasta. Samaan yhdyskäytävään on mahdollista sallia yhteyksiä useasta sijainnista. Tällöin esimerkiksi yrityksen eri paikoissa sijaitsevat toimistot voivat käyttää Amazonin VPN-palvelua myös keskinäiseen liikennöintiin. VPN mahdollistaa pilvitallennustilan käytön suojatulla yhteydellä yrityksen omasta sisäverkosta käsin. Lisäksi sen kautta omassa konesalissa sekä pilvessä olevat järjestelmät voivat olla turvallisesti yhteydessä toisiinsa. [49].

AWS:n Storage Gateway[42] tarjoaa oman konesalin tallennustilan laajennusta eri muodoissa. Storage Gateway on iSCSI[43] protokollan avulla käytettävä paikallinen yhdyskäytävä ja välimuisti S3-tallennusosioiden (storage volume) hyödyntämiseksi virtuaalisesti paikallisena tietovarastona. Paikalliseen konesaliin konfiguroidaan AWS Storage Gateway –virtuaalikone. Storage Gateway voidaan konfiguroida kolmella eri tavalla käyttötarkoituksesta riippuen. Kaikissa konfiguraatioissa paikallinen virtuaalikone tarvitsee paikallista tallennustilaa välimuistiksi. Gateway-Cached Volume on konfiguraatio, jossa paikallisen konesalin iSCSI tietovarastoon luodaan AWS Storage Gateway –virtuaalikone. Virtuaalikone tallentaa tietoa Amazon S3-osioihin, mutta pitää tietyn osuuden tiedosta paikallisessa välimuistissa. AWS:n dokumentaatiossa suositellaan, että paikallinen välimuisti olisi kooltaan vähintään 20% kokonaisdatamäärästä. Välimuistissa oleva tieto on luonnollisesti lyhyemmän viiveen takana ja siksi siellä pyritään pitämään useimmin käytetty tieto. Välimuistin lisäksi storage gateway virtuaalikone pitää lähetyspuskuria, johon kertyy uusi tallennettu tieto, jota ei vielä ole tallennettu S3-osiolle. [50].

[41] Virtual Private Network: https://www.ietf.org/rfc/rfc2764.txt
[42] AWS Storage Gateway: https://aws.amazon.com/storagegateway/
[43] iSCSCI: https://www.ietf.org/rfc/rfc3720.txt

Toinen samankaltainen konfiguraatio on Gateway-Stored volumes, jossa paikallinen tietovarasto säilyttää kaiken tiedon ja Storage Gateway –virtuaalikone tekee siitä määrätyin aikavälein inkrementaalisia varmuuskopioita S3-osioille. Tällöin uusissa tallennuksissa viedään vain muuttunut tieto pilvitallennustilaan. Tässä konfiguraatiossa tietoa siis tallennetaan asynkronisesti verrattuna Gateway-Cached Volumen jatkuvaan synkroniseen tallentamiseen. Varmuuskopiot ovat Amazon EBS –osioita, jotka voidaan liittää uuteen paikalliseen Storage Gateway –virtuaalikoneeseen. Varmuuskopioita voi myös liittää EC2-virtuaalikoneille, josta palvelua voidaan tarjota väliaikaisesti esimerkiksi paikallisen laiterikon tai sähkökatkon sattuessa. Kolmas konfiguraatio on Gateway-Virtual Tape Library, joka tarjoaa virtuaalisia nauhoja varmuuskopiointisovelluksen käytettäväksi. Nauhojen tallennustilana käytetään S3-osioita. Virtuaalikoneella on median vaihtaja, joka hoitaa käytettävän virtuaalinauhan vaihtamisen. Virtuaalikone käyttää muiden konfiguraatioiden tapaan välimuistia, johon varmuuskopiointisovelluksen tallentama tieto menee ennen ensin. Välimuistiin kirjoitettu varmuuskopio on pysyvästi tallessa, vaikka virtuaalikone sammuisi välillä. Välimuistista dataa siirretään lähetyspuskuriin, josta se tallennetaan S3-osiolle virtuaalinauhaksi. [50].

3.2.3 Vaihtoehtoisia palveluita pilvilaskentaan

EC2-virtuaalikoneet mahdollistavat kaikenlaisten laskentapalveluiden toteuttamisen, mutta ne vaativat käsin tehtävän asennuksen virtuaalikoneelle. Yleisimpiä tarpeita varten Amazon tarjoaa vaihtoehtoisia palveluita pilvilaskentaan, jolloin käyttäjä säästyy käsin tehtäviltä asennuksilta.

Lambda on yksinkertainen laskentapalvelu lyhyille ohjelmakoodin pätkille. Lambdaan tallennetaan koodia Lambda-funktioiksi, joita ajetaan tietyissä AWS-resurssista syntyvän tapahtuman tai HTTP-kutsun perusteella. Lambdassa käyttäjän ei tarvitse varata instansseja tai laskentakapasiteettia. Lambda funktion suoritus saa kestää korkeintaan 300 sekuntia. Muita rajoituksia funktiolle ovat muun muassa kutsun ja vastauksen kuuden megatavun maksimikoko, 1024 prosessin ja säikeen yhteislukumäärä sekä 512 megatavun käyttö levyltä tilapäistiedostoille. Lambda funktiossa suoritettavassa koodissa voi olla muiden AWS-resurssien käsittelyä API-kutsujen avulla. Lambdan avulla voi esimerkiksi tehdä yksinkertaisen verkkopalvelun, jonka staattinen sisältö ladataan suoran S3-ämpärissä ja RDS-tietokantaan välitetään kyselyitä Lambdan avulla. [51]

Suurten datamäärien analysointiin Amazon tarjoaa Elastic MapReduce (EMR) palvelua. Siinä käyttäjä voi suorittaa suurten datamäärien analysointia Hadoop-[44], Apache Spark-[45] tai Presto-ohjelmistokehystä[46] käyttäen. EMR käyttää laskentaan EC2-instansseja. Las-

[44] Hadoop: http://hadoop.apache.org/
[45] Spark: http://spark.apache.org/
[46] Presto: https://prestodb.io/

kenta ohjelmoidaan MapReduce-ohjelmointimallin mukaisesti. Tuettuja ohjelmointikieliä ovat Java[47], Perl[48], Python[49], Ruby[50], C++[51], PHP[52] ja R[53]. Prosessoitava data voi olla tallennettuna EC2-instanssin levylle, S3-objektivarastoon, Glacier-arkistoon, tai johonkin Amazonin tietokantapalveluista. DynamoDB, Redshift sekä kaikki Amazonin RDS tietokannat ovat tuettuja. [52]

AWS Elastic Beanstalk on PaaS-palvelu, joka käyttää pohjana Amazonin pilvi-infrastruktuuria. Se on helpompi vaihtoehto sovelluksen suorittamiseksi Amazonin pilvessä. PaaS-palvelussa käyttäjän ei tarvitse huolehtia pilvi-infrastruktuurin konfiguroinnista lainkaan. Sitä käytettäessä käyttäjän tarvitsee valita, haluaako käynnistää Web-sovelluksen vai taustasovelluksen sekä ajoympäristön tekniikan. Web-sovellus saa julkisen IP-osoitteen ja taustasovellus pelkän sisäisen IP-osoitteen. Vaihtoehtoisia tekniikoita suoritusympäristölle ovat Node.js[54], PHP, Python, Ruby, Java, Go[55], Tomcat[56] ja IIS[57]. [53]

3.3 Palvelut pilvipohjaisen tietojärjestelmän ylläpitoon ja kehitykseen

Alikohdassa 3.3.1 esitellään Amazonin tarjoamia pilvipohjaisen järjestelmän monitorointia ja valvontaa helpottavia palveluita. Alikohdassa 3.3.2 käydään läpi infrastruktuurin hallintaa helpottavia palveluita. Alikohdassa 3.3.3 käsitellään Amazonin tarjoamia kehitysympäristöpalveluita.

3.3.1 Pilvipohjaisen tietojärjestelmän monitorointi ja valvonta

Amazon CloudWatch[58] on monitorointipalvelu. Sen avulla voi seurata AWS-resurssien tilaa, kerätä tilastotietoa, luoda kuvaajia ja määrittää hälytyksiä tilan muuttuessa tietyn raja-arvon yli. Myös AWS-resursseista kertyvää laskua voi seurata. CloudWatchin keräämää dataa voi tallentaa S3-objektivarastoon. CloudWatchi-hälytyksiä voidaan välittää ilmoituspalvelu SNS:n avulla ylläpitäjille tai järjestelmän muille palveluille. [54].

IAM (Identity and Access Management)[59] on identiteetin- ja pääsynhallintapalvelu. Sen avulla voidaan luoda ja hallinnoida käyttäjiä, ryhmiä ja pääsyoikeuksia pilviresursseihin. Käyttäjät, ryhmät ja pääsyoikeudet ovat samanlaisia, kuin tiedostojen oikeudet UNIX-

[47] Java: https://www.java.com/
[48] Perl: https://www.perl.org/
[49] Python: https://www.python.org/
[50] Ruby: https://www.ruby-lang.org/
[51] C++: http://www.cplusplus.com/
[52] PHP: http://php.net/
[53] R: https://www.r-project.org/
[54] Node.js: https://nodejs.org/
[55] Go: https://golang.org/
[56] Tomcat: http://tomcat.apache.org/
[57] IIS: http://www.iis.net/
[58] Amazon CloudWatch: https://aws.amazon.com/cloudwatch/
[59] AWS Identity and Access Management: https://aws.amazon.com/iam/

pohjaisissa käyttöjärjestelmissä. Käyttäjän autentikointi tapahtuu perustasolla käyttäen käyttäjätunnusta ja salasanaa. Amazon kuitenkin suosittelee käytettäväksi turvallisempaa usean tekijän autentikointia (MFA, Multi-Factor Authentication) etenkin AWS-tilin pääkäyttäjälle. Siinä käyttäjätunnuksen ja salasanan jälkeen vaaditaan autentikointikoodi määritetystä MFA-laitteesta. MFA-laite voi olla älypuhelin, jossa on tätä varten sovellus. Muita vaihtoehtoja ovat Amazon yhteistyökumppaneiden valmistama avaimenperän kokoinen elektroninen koodigeneraattori sekä luottokortin kokoinen koodigeneraattorikortti. [55].

Amazon esittää, että IAM-käyttäjiä voi olla kolmea päätyyppiä: järjestelmän ylläpitäjä, järjestelmän loppukäyttäjä sekä muu järjestelmä. Järjestelmän ylläpitäjät hallinoivat pilviresursseja AWS-konsolin kautta. Heillä on usein laajat käyttöoikeudet. AWS-tilin pääkäyttäjä on järjestelmän ylläpitäjä, jolla on kaikki oikeudet. Loppukäyttäjillä on usein rajatut oikeudet ja heidän pääsy moniin resursseihin tapahtuu käytettävän tietojärjestelmän kautta. Loppukäyttäjillä voi olla myös suora pääsy esimerkiksi rajattuihin tietovaraston resursseihin. Muut järjestelmät ovat tietojärjestelmän osajärjestelmiä tai ulkoisia järjestelmiä, joille annetaan ohjelmallinen pääsy pilviresursseihin. [55].

Amazon CloudTrail[60] on lokitietopalvelu pilviresursseihin kohdistetuista pyynnöistä. Se tallentaa lokiin tiedot käyttäjien tekemistä pyynnöistä ja niiden ajankohdista. Lokitiedosto luodaan käyttäjän määrittämään S3 ämpäriin. CloudTrail voi kirjata yhtä lokia joko kaikilta käytössä olevilta AWS-alueilta tai vain yhdeltä. Sen voi myös asettaa keräämään omaa lokia jokaiselta käytössä olevalta AWS-alueelta. CloudTrail-lokin pohjalta voidaan myös tehdä CloudWatch-monitorointia. Tällöin lokitiedoista saadaan tilastotietoa ja niiden pohjalta voidaan luoda hälytyksiä. [56].

3.3.2 Pilvi-infrastruktuurin hallinta

CloudFormation on pilviresurssien mallinnukseen ja suunniteltuun tarkoitettu palvelu. Siinä tehdyn mallin mukaiset pilviresurssit voidaan käynnistää yhtenä operaationa. Mallia voidaan kopioida ja muuttaa tarpeen mukaan. Malli tallennettaan JSON-muotoisena tekstitiedostona. Mallia voi käsitellä myös graafisen käyttöliittymän kautta, jossa infrastruktuurin muokkaus onnistuu raahaa ja pudota –periaatteella. CloudFormation ei itsessään maksa mitään. Sitä käytettäessä maksetaan vain niistä resursseista, joita sen avulla varataan. Olemassa oleva pilvi-infrastruktuuri voidaan jälkikäteen dokumentoida CloudFormation-malliksi käyttäen CloudFormer-työkalua. Suuren yritystietojärjestelmän tapauksessa infrastruktuuri on hyvä olla dokumentoituna, sillä sen rakentaminen alusta asti uudesta voi olla todella työlästä. Lisäksi muutosten hallinta on helpompaa dokumentoituun ja versioituun infrastruktuuriin. [57]

[60] AWS CloudTrail: https://aws.amazon.com/cloudtrail/

OpsWorks on konfiguraationhallintapalvelu, joka käyttää Chef-konfiguraationhallintatyökalua[61]. Chefiä käytetään kirjoittamalla Ruby-ohjelmointikielellä reseptejä (recipe), joihin määritellään esimerkiksi Web-palvelimen tai tietokannanhallintajärjestelmän asetukset. Uudelle koneelle saadaan reseptin avulla sen sisältämä konfiguraatio automaattisesti. Konfiguraation hallinta vähentää järjestelmän ylläpitäjältä tarvittavaa manuaalista työtä uusien tai olemassa olevien koneiden konfiguraatiossa. OpsWorksissa Chefillä voi hallita myös suoraan EC2-instansseja, EBS-levyosioita ja EC2-instanssien autoskaalausryhmiä. Chefin perustoiminnallisuuksien lisäksi OpsWorks tarjoaa suoran yhteyden muun muassa CloudWatch-monitorointiin. OpsWorksin käyttöoikeuden määritetään IAM-käyttäjille muiden AWS-resurssien tapaan. OpsWorksin käyttö ei itsessään maksa mitään. Sitä käytettäessä maksetaan vain niistä resursseista, joita sen avulla varataan. [58]

AWS tarjoaa tietokannan migraatiopalvelua, jonka avulla olemassa oleva tietokanta on helppo siirtää Amazonin tietokantapalveluun. Siirron voi tehdä joko ulkoisen tietokannan ja RDS:n, ulkoisen ja EC2-instanssilla olevan tai RDS:n ja EC2-instanssilla olevan tietokannan välillä. Tuetut tietokannanhallintajärjestelmät ovat OracleDB, PostgreSQL, Microsoft SQL Server, Amazon Aurora, MariaDB, ja MySQL. Sen avulla on mahdollista siirtää tietokanta siten että lähde ja kohde tietokannat ovat joko samanlaisessa tai erilaisessa tietokannanhallintajärjestelmässä. Tämä palvelu helpottaa olemassa olevan yritystietojärjestelmän siirtäminen Amazonin pilveen. [59]

3.3.3 Kehitysympäristö Amazon Web Services -pilvipalvelussa

Pilvipohjaisen yritystietojärjestelmän kehitysympäristölle luonteva vaihtoehto on samassa pilvipalvelussa tuotanto- ja testiympäristöjen kanssa. Amazon tarjoaa kehitysympäristöä varten valmiina palveluina lähdekoodirepositorion versionhallintaa varten, jatkuvan toimituksen palvelun sekä tuotantoonvientipalvelun. Amazonin tarjoamat kehitysympäristöpalvelut ovat vain yksi vaihtoehto. Valmiiden palveluiden lisäksi EC2-virtuaalikoneille voi asentaa mitä tahansa kehitysympäristöpalveluita tai niitä voi käyttää Amazonin ulkopuolelta. Amazonin tarjoamat kehitysympäristöpalvelut ovat tällä hetkellä saatavilla vain Yhdysvaltojen AWS-alueilla. Tästä syystä niiden täysipainoinen hyödyntäminen vaatii, että koko infrastruktuuri pidetään Yhdysvaltojen AWS-alueilla.

CodeCommit on Amazonin Git-pohjainen[62] yksityinen lähdekoodirepositorio. Repositorio toimii jatkuvana varantona, eli sitä varten ei tarvitse varata instansseja. Pääsynhallinta toimii IAM-käyttäjien oikeuksia hallitsemalla. CodeCommitin repositorioon pääsee selaimella AWS-konsolista, komentoriviltä AWS-työkaluilla tai Gitin avulla. CodeCommit voi generoida SNS-viestejä tai käynnistää AWS Lambda[63] –skriptejä määrättyjen tapahtumien laukaisemina. CodeCommit tallentaa taustalla repositorion S3- ja DynamoDB-

[61] Chef: https://www.chef.io/
[62] Git: https://git-scm.com/
[63] AWS Lambda: https://aws.amazon.com/lambda/

palveluihin, jotka puolestaan huolehtivat korkeasta saavutettavuudesta. CodeCommitin käyttö maksaa yhden dollarin aktiivista käyttäjää kohden kuukaudessa. Käyttäjä saa tehdä tällä perusmaksulla 2000 Git-operaatiota kuukaudessa. Repositorion tallennuskapasiteetti on 10 gigatavua jokaista aktiivista käyttäjää kohden. Näiden yli menevistä osuuksista maksetaan käytön mukaan. 2000 Git-operaatiota kuukaudessa tarkoittaa 20 työpäivän kuukautena 100 repositorioon kohdistuvaa Git-operaatiota työpäivää kohden. Näihin lasketaan vain sellaiset operaation, joissa dataa haetaan repositoriosta tai sitä kirjoitetaan repositorioon. [60]

CodeDeploy on tuotantoonvientipalvelu. Sen avulla hallinnoidaan sovelluksen käynnistämisprosessin vaiheita. Tuotantoonvienti voidaan tehdä joko EC2-instansseille tai omassa konesalissa oleville palvelimille. CodeDeploy voi myös hoitaa CloudFormationin avulla EC2-instanssien käynnistyksen halutulla konfiguraatiolla. CodeDeployn pääasiallinen tehtävä on kuitenkin koontiversion tuotantoonvienti eikä infrastruktuurin hallinta. Koontiversio käsitellään pakettina, jossa voi olla suoritettavien tiedostojen lisäksi staattisia Web-sivuja, asennusskriptejä, konfiguraatiotiedostoja tai muita sovellukseen liittyviä tiedostoja. [61]

CodePipeline on Amazonin tarjoama palvelu jatkuvan toimituksen hallinnointiin ja mallintamiseen koontiversioputkena. CodePipelineen tehdään putkia (pipelines), joiden läpi koodimuutokset kulkevat automaattisesti. CodePipeline ei itsessään tee koontiversiolle mitään, vaan se mallintaa muualla tapahtuvien toimenpiteiden kulkua. Putkeen täytyy valita vähintään lähdekoodin tarjoaja (source provider), tuotantoonviennin tarjoaja (deployment provider) sekä IAM-käyttäjä, jonka oikeuksilla putki toimii. CodePipeline tukee tällä hetkellä lähdekoodin tarjoajaksi vain GitHub-repositorioita[64] sekä S3-objektivarastoa sinne tallennettujen zip-pakettejen muodossa. CodePipeline ei siis tällä hetkellä osaa käyttää suoraan Amazonin omaa CodeCommit-repositoriota. CodePipelineä varten CodeCommittiin voi liittää Lambda-funktion, joka luo zip-paketin S3-objektivarastoon, kun tietty tapahtuma lähdekoodirepositoriossa tapahtuu. Tuotantoonviennin tarjoajaksi täytyy valita AWS CodeDeploy tai AWS Elastic Beanstalk. Näiden lisäksi putkeen voi luoda myös muita vaiheita tai lisätä kahteen edellä mainittuun vaiheeseen lisää toimenpiteitä. Putki voi sisältää useita lähdekoodin tarjoajia ja useita tuotantoonviennin tarjoajia. Lisäksi muita mahdollisia toimenpiteitä ovat ulkoinen koontiversion tuottaminen, ulkoinen testien ajaminen sekä AWS Lambda –funktion käynnistys. Ulkoiseksi koontiversion tuottajaksi valitaan jatkuvan integraation palvelin, joka voi olla Jenkins[65] tai Solano[66]. Testien tarjoajaksi vaihtoehtoina ovat muiden muassa Apica-kuormitustestaus[67], BlazeMeter-

[64] GitHub: https://github.com/
[65] Jenkins: https://jenkins.io/
[66] Solano: https://www.solanolabs.com/
[67] Apica: https://www.apicasystem.com/

kuormitustestaus[68], Ghost Inspector –käyttöliittymätestaus[69], ja Runscope-rajapintatestaus[70]. Putken suoritus etenee seuraavaan vaiheeseen, kun edellinen on suoritettu onnistuneesti. Jokaisessa vaiheessa voi olla sekä rinnakkaisia, että peräkkäisiä toimenpiteitä. Jokainen putki, jota käytetään maksaa yhden dollarin kuukaudessa. Näiden kolmen palvelun avulla Amazon Web Services –palvelussa voi toteuttaa jatkuvan toimituksen sekä automatisoida tuotantoon viennin. [62]

[68] BlazeMeter: https://www.blazemeter.com/
[69] Ghost Inspector: https://ghostinspector.com/
[70] Runscope: https://www.runscope.com/

4 YRITYSTIETOJÄRJESTELMÄT PILVESSÄ

Tässä luvussa käydään läpi kaksi tutkimuskohdetta, joissa yritystietojärjestelmiä käytetään Amazon Web Services –pilvipalvelussa. Ensimmäisenä on haastattelututkimus Fonectan tietojärjestelmästä, joka on suunniteltu alusta alkaen pilviyhteensopivaksi. Toisena on vanhempi SOA-pohjainen järjestelmä Vera, jonka tuotanto- ja kehitysympäristöille tehtiin suunnitelma ja kokeilu Amazonin pilveen siirtymiseksi.

4.1 Fonecta

Tässä kohdassa käydään läpi 2.5.2016 pidetyn puhelinhaastattelun vastauksia. Haastateltavana oli Fonecta Oy:n pilvipalveluiden käytöstä vastaava IT-manager Tero Nevalainen. Myös useita vuosia Fonecta Oy:n pilvipohjaisten järjestelmien kanssa työskennellyt Gofore Oy:n ohjelmistoarkkitehti Tapio Rautonen täydensi jotain vastauksia omalla näkemyksellään. Fonectan tietojärjestelmien infrastruktuurista on Nevalaisen arvion mukaan 80-90% Amazonissa. Microsoftin Azure-pilvipalvelussa on joitain pieniä palveluita ja sen lisäksi joitain palveluita on suomalaisissa konesaleissa. Väestörekisterikeskuksen data on erillisessä ympäristössä Suomessa, koska Suomen lainsäädäntö ei salli henkilötietojen siirtämistä Suomen rajojen ulkopuolelle. Kaikki ympäristöt ovat keskenään VPN-yhteydessä.

Fonectan tietojärjestelmiä on alettu siirtämään Amazonin pilvipalveluun neljä ja puoli vuotta sitten. Sitä ennen ne toimivat perinteisessä konesalissa. Silloin infrastruktuuri oli huomattavasti pienempi kuin nykyisin. Nevalaisen mukaan, pilvipalveluilta haettiin mahdollisuutta nopeampaan kehitystahtiin sekä infrastruktuurin kontrollia omiin käsiin. Siihen aikaan vaihtoehtoisia pilvipalveluntarjoajia ei ollut montaa ja Fonectalla Amazon oli selkeä vaihtoehto. He olivat tehneet Amazonissa pienimuotoisen kokeilun toisessa projektissa ennen tätä. Fonectalla Amazonin pilvipalveluiden käyttö on kehittynyt vuosien myötä. Alussa Amazonin palveluiden tarjonta oli suppeampi ja sen vuoksi jouduttiin tekemään kompromisseja. Vuosien myötä Amazonin julkaisemat uudet palvelut ovat mahdollistaneet enemmän infrastruktuurin automatisointia ja vähentäneet manuaalisen työn tarvetta ylläpidossa. Myös ajan myötä kehittyvät hyvät käytännöt pilvipalveluiden käytössä ovat parantaneet pilvipalveluiden käyttöä.

Nevalainen kertoo Fonectan pilvi-infrastruktuurin olevan nyt neljännessä iteraatiossa. Aluksi Fonectan tietojärjestelmien päivittäminen pilvipalvelussa aiheutti katkoksen palvelu käytössä. Tällöin samoilla EC2-instansseilla pidettiin useita palveluita. Myöhemmin päivittäminen kehitettiin katkottomaksi ja nykyisin heillä on käytössä blue/green-deployment. Fonectan tietojärjestelmiä kehitettäessä käytetään jatkuvan toimituksen prosessia, jossa koontiversiot tuotetaan AMI-levykuvina. Jatkuva integraatio suoritetaan Jenkinsin

avulla. Fonectalla ei ole käytössä Amazonin tarjoamia kehitysympäristöpalveluita. Infrastruktuuriin on kehitetty monitorointia usealle työkalulle. Instanssien resurssimonitorointi on toteutettu New Relicillä[71], jonka piiriin uudet instanssit menevät automaattisesti. Palveluiden tilaa monitoroidaan automaattisilla tilakyselyillä, joita varten palveluihin on toteutettu kuntoisuustarkastussivuja (healthcheck). Amazonin valvontatyökaluista ovat käytössä CloudWatch sekä AWS Trusted Advisor[72]. AWS Trusted Advisor on palvelu, joka ehdottaa parannuksia käytössä olevien pilviresurssien kulujen optimointiin, tietoturvan asetuksiin, vikasietoisuuteen ja suorituskykyyn.

Nevalainen kuvaa Fonectan tietojärjestelmien ohjelmistoarkkitehtuurin olevan nykyisin pääosin mikropalveluarkkitehtuurin mukainen. Alussa järjestelmät olivat huomattavasti pienempiä ja silloin useita palveluita pidettiin samalla instanssilla. Ajan myötä palveluita on otettu erilleen omiin autoskaalausryhmiin ja näin järjestelmistä on muotoutunut enemmän mikropalveluarkkitehtuuria muistuttavia. Poikkeuksena on edelleen muutama isompi palvelu, joita ei ole pilkottu mikropalveluiksi. Fonectan arkkitehtuurissa mikropalveluiden jakoa on jouduttu ajan myötä muuttamaan hieman, mutta esimerkiksi Newmanin mainitsemia [12] suuria ja työläitä refaktorointeja ei ole tarvinnut tehdä. Muilta osin suunnittelussa ei varsinaisesti yritetty käyttää pilvipohjaisten järjestelmien suunnitteluperiaatteita. Nevalainen ja Rautonen kuitenkin toteavat yhdessä, että useat suunnitteluratkaisut, joihin Fonectan tietojärjestelmien tapauksessa päädyttiin ovat samoja, mitä nykyisin pidetään hyvinä pilvipohjaisen järjestelmän suunnitteluperiaatteina.

Fonectalla on tuotantoympäristön lisäksi myös testi- ja kehitysympäristöt Amazonin pilvessä. Nevalainen arvioi, että tällä hetkellä heillä on käytössä noin 400-450 EC2-instanssia. Pilvipalvelussa käytön mukainen laskutus mahdollistaa kustannusten optimoinnin ja tällaisella määrällä pilviresursseja kustannusten optimoinnista saatava hyöty on merkittävä. Fonectan järjestelmissä palveluita skaalataan automaattisesti kuormituksen mukaan. Tämän lisäksi käytetään Amazonin varattuja instansseja (reserved instances), joita sitoudutaan käyttämään jatkuvasti ja näin instansseille saadaan pienempi tuntihinta. Fonectan kehitysympäristöjen kuluissa säästetään sammuttamalla ne yöksi, viikonlopuksi ja lomien ajaksi. Tiedonlouhinnan osalta Amazonin EMR-instansseina käytetään spot-instansseja, eli resurssit varataan käyttöön vasta sitten, kun spot-hinta on riittävän halpa. Amazon hinnoittelee spot-instanssit kysynnän mukaan, jolloin hinta laskee, kun kyseisten instanssien kysyntä laskee. Fonectalla EMR:n kustannuksia monitoroidaan myös sovelluskohtaisesti. Nevalaisen arvion mukaan Amazonin resurssien kustannuksista yli puolet tulee EC2-instansseista. Seuraavana tulevat RDS- ja Redshift-tietokantapalvelut.

Pilvipalveluista tai mikropalveluarkkitehtuurista koituneet haitat Nevalainen arvioi vähäisiksi. Pilvi-infrastruktuurin vikatilanteiden kanssa he eivät ole ongelmia juurikaan kohdannet. Joitain kertoja käytössä olevan instanssin fyysinen kone on hajonnut, mutta

[71] New Relic: https://newrelic.com
[72] AWS Trusted Advisor: https://aws.amazon.com/premiumsupport/trustedadvisor/

autoskaalausryhmä on korjannut ongelman automattisesti käynnistämällä uuden instanssin hajonneen tilalle. Yhden kerran uutta instanssityyppiä ei ollut heti saatavilla ja sitä jouduttiin odottamaan joitain tunteja. Automaattisen testauksen osalta end-to-end –testejä ei käytetä, mutta muutamia käsin käynnistettäviä end-to-end –testejä on olemassa. Niiden tekeminen ja ylläpito koetaan työlääksi, jonka vuoksi Fonectalla mieluummin panostetaan reagointikykyyn.

Nevalainen visioi muutamia pieniä kehityskohteita tulevaisuuteen, mutta yhtä suurta muutosta ei ole tiedossa. Fonectalla halutaan kehittää parempaa tuotantoonvientiprosessia. Kulujen hallinnassa kehityskohteina ovat turhien resurssien siivous ja käytössä olevien resurssien parempi merkitseminen tarkemman kulujakauman selvittämiseksi. Uusia Amazonin palveluita odotetaan ja niiden hyödyntämistä aiotaan kokeilla.

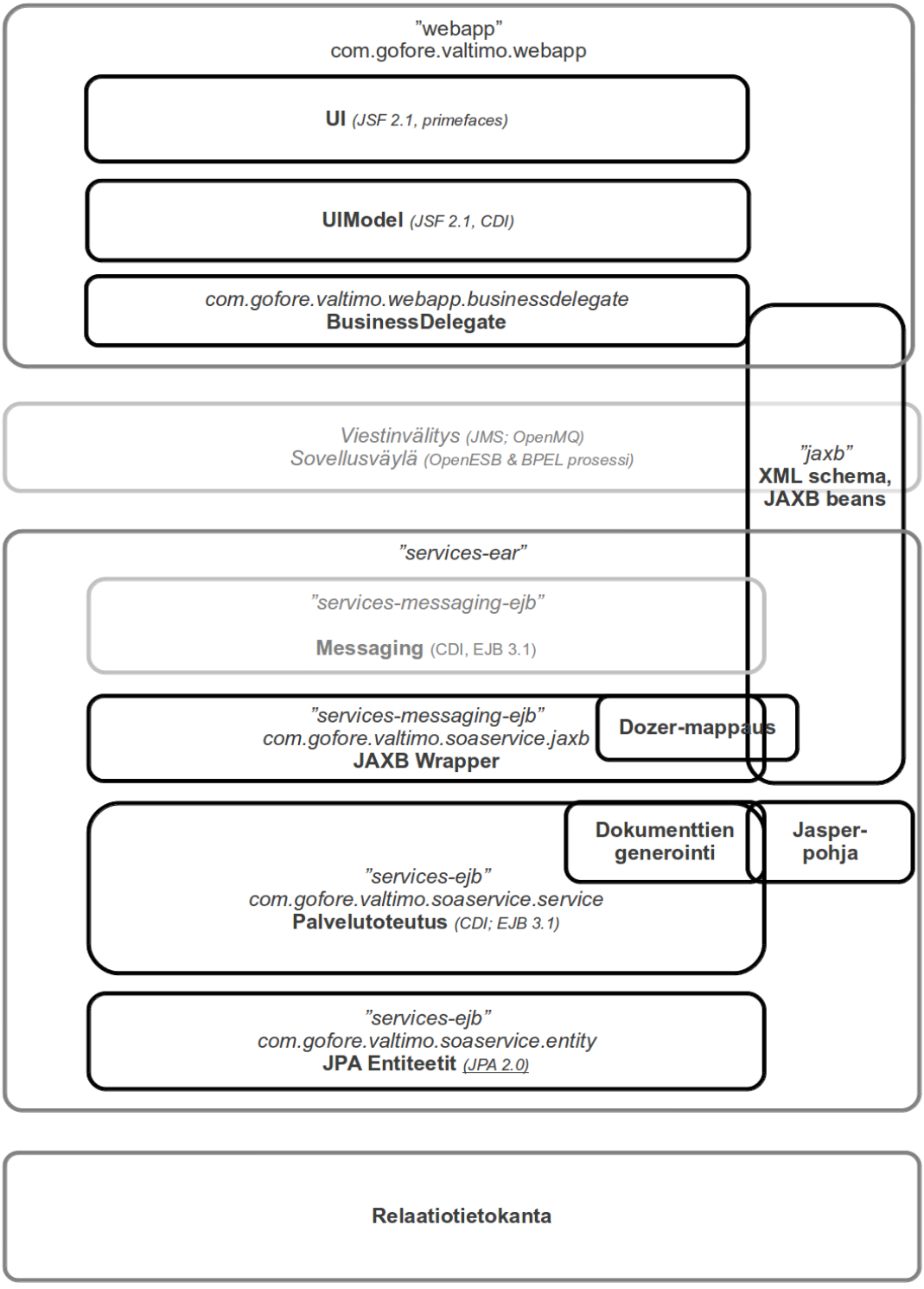
4.2 Valtimo

Valtimo on Gofore Oy:n projekti ja nykyisin jatkuva palvelu, jota on tehty useassa osaprojektissa vuodesta 2010 alkaen ja se jatkuu edelleen. Sen varsinainen kehitysvaihe oli 5/2010 – 8/2014 viidessä osaprojektissa ja siitä eteenpäin se on ollut jatkuva palvelu, johon tehdään ylläpidon lisäksi jatkokehitystä. Valtimossa on keskitytty Vera-tietojärjestelmän suunnitteluun, toteutukseen ja ylläpitoon. Valtimo-projektin asiakas on Sosiaali- ja terveysministeriö. Valtimo projektissa on toteutettu työväline työsuojeluvalvontaan sekä sen suunnitteluun ja seurantaan. Käytännössä se on tietojärjestelmä, jonka avulla voidaan luoda, hallita ja seurata työsuojelutarkastuksia sekä muita siihen liittyviä elementtejä. Veran tuotanto- ja testiympäristöt ovat tällä hetkellä Suomalaisessa konesalissa ja kehitysympäristö Gofore Oy:n sisäverkon palvelimilla. [63][64]

4.2.1 Veran ohjelmistoarkkitehtuuri ja sopivuus pilvipalveluun

Vera-järjestelmä on suunniteltu palvelukeskeisen arkkitehtuurin mukaisesti. Verassa on tietokanta, palvelukerros, palveluväylä ja Web-sovellus. Nykyisessä tuotantoympäristössä kaikki neljä kerrosta sijaitsevat erillisillä palvelimilla. Palvelukerros ja Web-sovellus ovat Java-sovelluksia, joita suoritetaan Glassfish-sovelluspalvelimilla[73]. Palveluväylänä toimii Glassfish OpenESB[74], joka on järjestelmän viestintävälikerros. Tietokanta toimii OracleDB-tietokannanhallintajärjestelmässä. Järjestelmän arkkitehtuuri on kuvattuna kuvassa 5. Nykyinen järjestelmän arkkitehtuuri ei aseta esteitä pilvipalveluun siirtymiselle. Palveluväylä tukee useampaa Web-sovellus ja palvelukerros instanssia, mutta autoskaalauksessa instanssien vähentäminen lennossa ei ole tuettu. [65]

[73] Glassfish: https://glassfish.java.net/
[74] OpenESB: http://open-esb.net/



Kuva 5. *Veran ohjelmistoarkkitehtuurin rakenne [65]*

4.2.2 Veran käynnistäminen Amazon Web Services –pilvipalvelussa

Ensimmäisessä vaiheessa Vera järjestelmä pyrittiin käynnistämään sellaisenaan Amazon Web Services –pilvipalveluun. Tähän kokeiluun valittiin Frankfurtin AWS-alue (eu-central-1). Sieltä varattiin tuotantoympäristön kaltaiset resurssit, eli kolme EC2-instanssia Linux käyttöjärjestelmällä sekä RDS-tietokantainstanssi OracleDB-tietokannanhallintajärjestelmällä. EC2-instansseihin valittiin Ubuntu Server 14.04 –käyttöjärjestelmä[75], koska tuotantoympäristössä käytetty RedHat Enterprise Linux[76] on maksullisen lisenssin alainen myös Amazonissa. EC2-instanssien tyypiksi valittiin t2.micro, jossa on 1 Gt keskusmuistia ja yksi 1,5 Ghz virtuaalisuoritin. Tietokantainstanssiksi puolestaan valittiin RDS db.t2.small, jossa on 2 Gt keskusmuistia ja yksi 1 Ghz virtuaalisuoritin.

Palvelukerroksen ja Web-sovelluksen virtuaalikoneille siirrettiin valmiiksi konfiguroitu Glassfish-sovelluspalvelin sekä suoritettavan ohjelmiston Java-paketit. Palveluväylän virtuaalikoneelle siirrettiin myös valmiiksi konfiguroitu Glassfish OpenESB –palvelinohjelmisto. Tietokannan skeema alustettiin versionhallinnassa olevilla tietokannanalustusskripteillä. Palvelukerroksen, Web-sovelluksen ja palveluväylän virtuaalikoneiden käyttöjärjestelmiin luotiin uusi käyttäjä sovelluspalvelimen suorittamista varten. Tämän lisäksi palvelukerroksen ja Web-sovelluksen sovelluspalvelimien konfiguraation täytyi syöttää palveluväylän IP-osoite. Tietokantaan täytyi populoida testikäyttäjiä, jotta järjestelmän toimivuutta pääsi testaamaan käyttöliittymän kautta. Amazonin virtuaali-infrastruktuuri vaati tässä vaiheessa vain turvallisuusryhmän konfiguroinnin. Turvallisuusryhmässä sallittiin yhteyksien avaaminen virtuaalikoneiden välillä kaikista porteista. Goforen sisäverkon IP-osoitteista sallittiin tietokannan hallintayhteyden, SSH-, HTTP- ja HTTPS-yhteyksien avaaminen.

Näillä järjestelyillä Vera saatiin toimimaan Amazon Web Services –ympäristössä. Järjestelmän toimintaa tarkkaillessa huomattiin, että palvelu- ja Web-sovellusten instansseilla muistin käyttö on hyvin lähellä instanssin muistin maksimimäärää. Yksinkertaista testaamista se ei haittaa, mutta useamman käyttäjän aktiivikäytössä muistia tarvittaisiin enemmän. Instanssien lukumäärää kasvattamalla tämä ei tietenkään korjaudu vaan tarvitaan tehokkaampia EC2-instansseja, sillä pelkkä sovelluksen käynnistäminen Glassfish-sovelluspalvelimelle varasi lähes kokonaan yhden gigatavun keskusmuistin. EC2-instanssit saavat jokaisessa käynnistyksessä uuden sisäverkon IP-osoitteen. Palvelukerroksen sekä Web-sovelluksen instansseille joudutaan siis joka kerta konfiguroimaan palveluväylän IP-osoite, kun palveluväylän instanssi käynnistetään uudestaan. Amazon tarjoaa pysyvää julkista IP-osoitetta EC2-instansseille ElasticIP-palvelun avulla. Amazonin turvallisuus-

[75] Ubuntu: http://www.ubuntu.com/
[76] RedHat Enterprise Linux: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

ryhmän konfiguraatioilla julkisen verkon liikenne voidaan estää, joten sillä pystytään parantamaan palveluväylä ja palvelukerrosinstanssien tietoturvaa käytettäessä ElasticIP-palvelua.

4.2.3 Ohjelmistoarkkitehtuurin muuntaminen pilviyhteensopivaksi

Seuraavaksi esitellään ehdotuksia Veran ohjelmistoarkkitehtuurin muuttamista pilviyhteensopivammaksi. Arkkitehtuuria ei tämän työn puitteissa muutettu, mutta tätä selvitystä voidaan myöhemmin käyttää apuna pohdittaessa järjestelmän jatkokehitystä. Pilviyhteensopivuutta tarkastellaan pilvipohjaisten järjestelmien suunnitteluperiaatteiden pohjalta. Veran arkkitehtuuria pyritään muuntamaan hieman mikropalveluarkkitehtuurin suuntaan. Suunnittelussa todettiin, että ohjelmistoarkkitehtuurista ei kannata yrittää tehdä täysin mikropalveluarkkitehtuurin mukaista.

Pilviyhteensopivan järjestelmän yleisenä periaatteena pidetään tilattomuutta sekä riippumattomuutta yksittäisistä instansseista. Molempien periaatteiden toteuttamiseksi tarpeelliset tilatiedot on tallennettava johonkin muualle, kuin instanssin levylle. Verassa käyttäjien HTTP-istuntotiedot ovat tallennettuna Web-sovelluksen instanssilla. Tämä aiheuttaa alaspäin skaalatessa niille käyttäjille istunnon katkeamisen, joiden Web-sovellusinstanssi terminoidaan. Pilviyhteensopivampi ratkaisu tähän on tallentaa istuntotiedot esimerkiksi muistinvaraiseen tietokantaan, joka on kaikkien Web-sovellusinstanssien käytettävissä. Tällöin mikä tahansa instansseista voi jatkaa toisen aloittamaa istuntoa. Amazonin pilvipalveluista tähän sopii muistinvarainentietokanta ElastiCache.

Riippuvuutta yksittäisestä instanssista saadaan vähennettyä jakamalla tarvittavat resurssit usealle rinnakkaiselle instanssille sen sijaan että kaikki resurssit olisivat yhdellä instanssilla. Resurssien jakaminen voidaan toteuttaa usealle EC2-instanssille, jotka toimivat autoskaalausryhmässä. Autoskaalausryhmälle voidaan ohjata liikennettä joko kuormantasaajan tai SQS-viestijonon kautta. Veran tapauksessa yksi autoskaalausryhmä muodostetaan rinnakkaisista Web-sovellusinstansseista, joihin liikennettä ohjataan kuormantasaajalla. Kuormantasaaja toimii yhteyspisteenä ulkoverkkoon, josta järjestelmän käyttäjät ottavat HTTPS-yhteyden Web-sovellukseen.

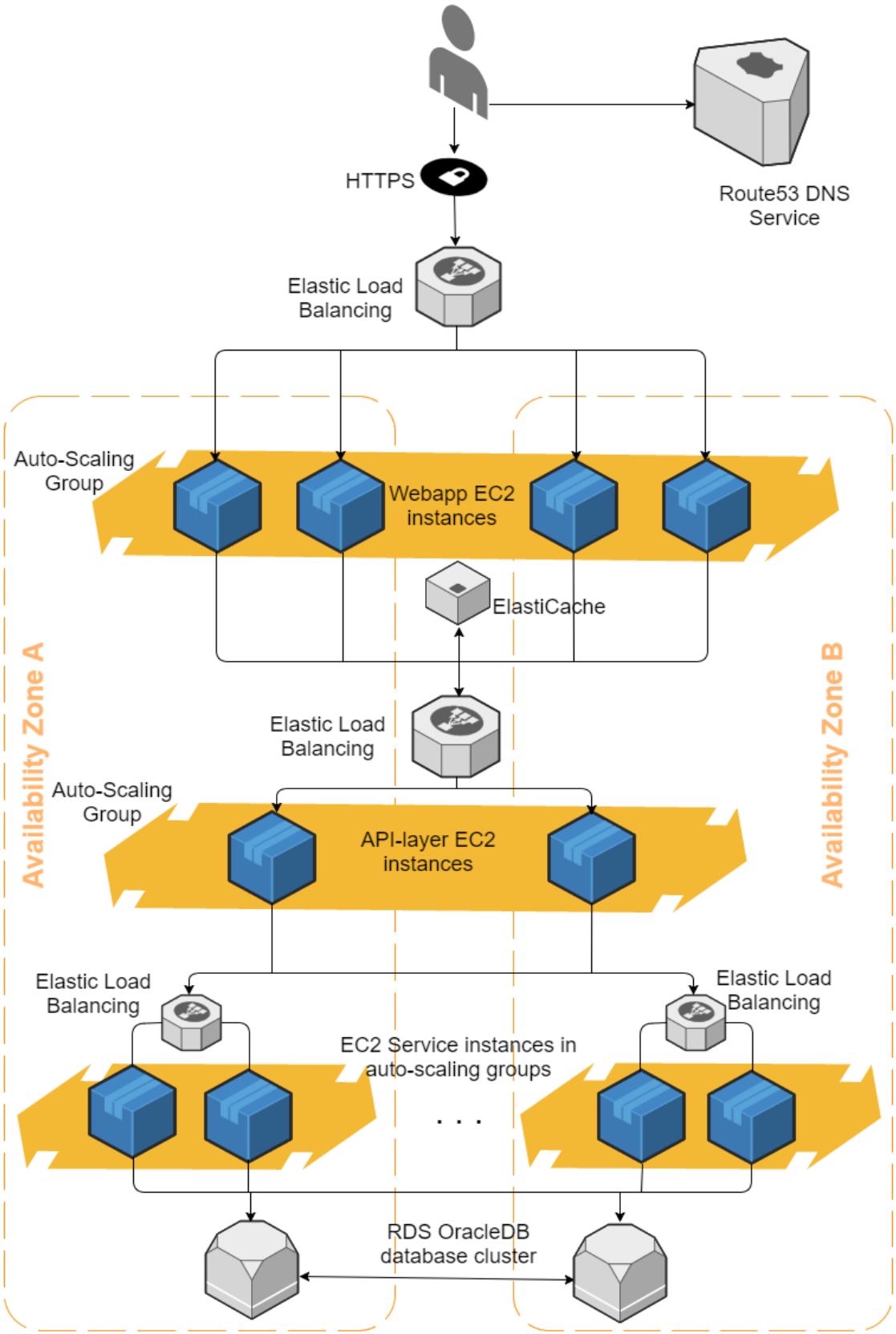
Web-sovellusinstanssin suoritussäie jää odottamaan vastausta palvelukerrokselle lähettämäänsä pyyntöön eivätkä muut Web-sovellusinstanssit osaa käsitellä toisten tekemien pyyntöjen vastauksia. Nykyisessä arkkitehtuurissa palveluväylä huolehtii, että paluuviesti tulee samalle Web-sovellusinstanssille. Pilviyhteensopivassa järjestelmässä ei käytetä älykästä viestinvälittäjää, kuten palveluväylää vaan viestinnän älykkyys toteutetaan lähettäjälle ja vastaanottajalle. ESB-palveluväylän vaihtaminen Amazon SQS –viestijonoon vaatii palveluväylän logiikan toteutusta Web-sovellukseen sekä palvelukerrokselle. Tätä varten Web-sovelluksen täytyy lisätä oma osoitteensa pyyntöön, jotta palvelukerros osaa lähettää vastauksen oikealle Web-sovellusinstanssille.

Palvelukerroksen toteutus on koodissa jaettu moduuleihin, jotka voidaan hajauttaa mikropalveluiksi. Hajauttaminen vaatii jokaiselle mikropalvelulle oman autoskaalausryhmän. Verassa Web-sovelluksen lähettämät pyynnöt käsitellään ja ohjataan oikeille palveluille omissa vastaanottajaluokissa. Vastaanottajaluokkien rajapinnoista voidaan muodostaa palveluille yhteinen rajapintakerros, joka toimii abstraktiona peittäen todellisen mikropalvelujaon. Rajapintakerros mahdollistaa muutokset mikropalveluiden jaossa ilman, että Web-sovelluksen tarvitsee vaihtaa pyyntöjen kohdetta. Rajapintakerros toimisi samalla välityspalvelimena, joka pitää kirjaa mikropalveluiden tilasta ja osaa ilmoittaa pyyntöä tekevälle Web-sovellukselle heti, jos joku mikropalveluista on tavoittamattomissa. Myös rajapintakerros kannattaa laittaa kuormantasaajan taakse, autoskaalausryhmään saavutettavuuden varmistamiseksi.

Veran tietokantaskeemaa on hyvin hankala pilkkoa osiin. Tietokantatauluissa on keskinäisiä relaatioita todella paljon, joten tietokantaskeeman osittaminen vaatisi joko viiteeheyksistä luopumista tai saman datan kopiointia useaan paikkaan. Kumpikaan vaihtoehdoista ei ole mielekäs tehdä vain sen takia, että tietokantaskeema saataisiin jaettua. Tietokanta kuitenkin voitaisiin klusteroida useampaan RDS-instanssiin usealla Amazonin saatavuusalueelle, jolloin tietokannan vikasietoisuus ja saavutettavuus kasvaisivat. Myös kaikkien EC2-instanssien autoskaalausryhmät kannattaa samasta syystä asettaa käyttämään useampaa saatavuusaluetta.

Kuvaan 6 tehdystä piirroksesta nähdään näiden ehdotusten mukainen toteutus koko järjestelmän arkkitehtuurista. Arkkitehtuuristen muutosten lisäksi järjestelmä voisi hyödyntää infrastruktuuriin liittyviä tukipalveluita. Amazon Route53[77] –palvelun avulla pystyttäisiin hallinnoimaan järjestelmään liittyviä domain-nimiä Amazonin kautta. CloudWatch-hälytysten ja SNS-ilmoitusten avulla järjestelmän ylläpidosta vastaavalle kehitystiimille saisi välitettyä reaaliaikaiset tiedot vikatilanteista. CloudFormationin käyttö infrastruktuurin dokumentoinnissa helpottaisi sekä muutosten hallintaa, että esimerkiksi tuotannonkaltaisen testiympäristön pystyttämistä.

[77] Amazon Route 53: https://aws.amazon.com/route53/



Kuva 6. *Veran ohjelmistoarkkitehtuuri Amazon Web Services –pilvipalvelun infrastruktuurikomponenteilla kuvattuna*

4.2.4 Veran kehitysympäristön siirtäminen Amazon Web Services pilvipalveluun

Veran kehitysympäristö sijaitsee Gofore Oy:n sisäverkon palvelimilla. Kehitysympäristö sisältää kaksi sovelluspalvelinkonetta, jatkuvan integraation –palvelimen sekä kaksi tietokantapalvelinta. Koontiversioiden tekemisessä käytetään sisäverkon palvelimelta löytyvää artefaktirepositoriota. Lisäksi staattisen koodin analyysiin on käytetty sisäverkon palvelimelta löytyvää SonarQube-palvelinta[78]. Goforen projekteissa on siirrytty vähitellen käyttämään pilvipalveluita kehitysympäristöjen alustana. Vanhoilta sisäverkon palvelimilta loppuu pian takuuaika eikä uusien hankkimiseen ole halukkuutta. Tämän työn puitteissa Veran kehitysympäristöä aloitettiin siirtämään Amazon Web Services –pilvipalveluun.

Verassa kehitystietokanta on kaikkien kehittäjien yhteisesti käyttämä tietokanta. Lisäksi kehitysversioiden sovelluspalvelimet käyttävät samaa kehitystietokantaa. Amazonin pilveen siirryttäessä tietokannalle ovat vaihtoehtoina RDS ja EC2. RDS tarjoaa tietokannan valmiina palveluna, joka helpottaa ylläpitoa. EC2-koneelle voi valita valmiin AMI:n, jossa tietokanta on asennettuna tai sen voi asentaa itse. RDS-palvelussa tietokantainstanssia ei voi sammuttaa, joten se täytyy joko tuhota tai pitää päällä myös niinä aikoina, kun sitä ei käytetä. Kehitysympäristössä tyypillisesti instansseille ei ole tarvetta öisin, viikonloppuisin ja lomien aikaan. EC2-instanssilla ajettava tietokanta on tuntihinnaltaan edullisempi ja sen voi ajastaa sammumaan edellä mainittuina aikoina. Amazonilta löytyy julkinen AMI, jossa on Red Hat Enterprise Linux sekä OracleDB 11g –tietokanta valmiiksi asennettuna. Tämä AMI on kuitenkin vain vanhoihin EC2-instanssityyppeihin sopiva. Amazon on hinnoitellut vanhemmat instanssityypit kalliimmiksi ja lisäksi Red Hat Enterprise Linuxin käytöstä on maksettava. Irlannin AWS-alueella Veran kehitystietokannalle sopivan kapasiteetin RDS db.m4.large –instanssi maksaa 0.193$ tunnissa, EC2 m3.xlarge –instanssi valmiin Red Hat Enterprise Linuxin ja OracleDB 11g –tietokannan kanssa 0.378$ tunnissa ja pelkkä EC2 t4.large –instanssi 0.112$ tunnissa. Näistä päätettiin lähteä kokeilemaan viimeisintä, koska se on halvin ja sen pystyy sammuttamaan silloin, kun sitä ei käytetä. Oraclen tietokanta täytyi asentaa ja konfiguroida manuaalisesti ja siihen kului yksi työpäivä.

Jatkuvan integraation palvelin Jenkins käynnistettiin t2.medium EC2-instanssille. Amazon ei tarjoa valmista palvelua jatkuvalle integraatiolle, joten se täytyy hoitaa EC2-instanssilla manuaalisesti asennettuna. EC2-instanssille asennettiin Jenkins versio 2.1. Siihen saatiin kopioitua vanhat Jenkins-työt Goforen sisäverkossa sijaitsevalta Jenkins versio 1.6 –palvelimelta versioiden eroista huolimatta. Koontiversion tekeminen on aiemmin

[78] SonarQube: http://www.sonarqube.org/

ollut riippuvainen sisäverkon artefaktirepositoriosta, josta koontiversion rakentaja Maven[79] lataa koontiversioon kuuluvia riippuvuuksia. Pilvipalveluun siirryttäessä riippuvuudet on ladattava julkisista repositorioista. Jotta kaikki riippuvuudet saatiin täytettyä, joudutaan käyttämään useampaa julkista repositoriota, kuin aiemmin. Lisäksi integraatiotesteissä käytettävä Oraclen tietokanta-ajuri jouduttiin lataamaan Oracle verkkosivuilta ja lisäämään käsin jatkuvan integraation palvelimelle, koska Oracle ei salli sen jakamisten julkisten repositorioiden kautta. Näiden järjestelyiden jälkeen koontiversion rakentaminen onnistui. Samalle EC2-instanssille asennettiin myös staattisen koodin analyysipalvelu SonarQube sekä sen tulosten pidempiaikaista säilömistä varten PostgreSQL-tietokanta.

Kahden kehityshaaran sovelluspalvelimet käynnistettiin t2.small EC2-instansseille. Niihin valittiin CentOS 7 –käyttöjärjestelmä[80]. Sovelluspalvelin sekä palveluväylä kopioitiin olemassa olevilta kehityspalvelimilta. Sovelluspalvelimien täytyy pystyä yhdistämään kehitystietokantaan. Amazonin instanssit saavat jokaisen käynnistyksen yhteydessä uuden IP-osoitteen, joten konfiguraation yksinkertaistamiseksi EC2-instansseille asetettiin pysyvät julkiset IP-osoitteet ElasticIP-palvelun avulla. Normaalisti instanssien keskinäinen liikennöinti sallitaan turvallisuusryhmän asetuksista sallimalla yhdistäminen turvallisuusryhmän sisältä. Käytettäessä ElasticIP:tä yhteydet kiertävät turvallisuusryhmän ulkopuolelta, joten jokaisen ElasticIP-osoitteen yhteydet täytyi sallia erikseen. Lisäksi saapuvat yhteydet instanssien SSH-porttiin sekä käytettävien palveluiden portteihin täytyi sallia Goforen IP-osoitteista.

Kaikista instansseista luotiin AMI-levykuva, jotta instansseille tehdyt asennukset säilyvät tallessa. Levykuva mahdollistaa myös suurempaan tai pienempään instanssiin siirtymisen, jos valitun instanssin kapasiteetti ei ole sopiva. Tällöin levykuvan pohjalta käynnistetään uusi sopivampi instanssi ja vanha terminoidaan. Amazonin kehitysympäristöpalveluille ei ollut Veran tapauksessa käyttöä. CodeCommit –lähdekoodirepositorio on näistä ainut palvelu, jonka käyttöön voitaisiin joskus siirtyä.

4.3 Johtopäätökset yritystietojärjestelmistä pilvessä

Käytännön esimerkit osoittivat, että Amazon Web Services tarjoaa kattavasti palveluita yritystietojärjestelmien tarpeisiin. Molemmissa tutkimuskohteissa pilveen siirtymisellä saavutettiin haluttuja hyötyjä ja käyttöä aiotaan jatkaa. Fonectan tapauksessa tietojärjestelmä on kasvanut pilviyhteensopivaksi ja siksi se pystyy hyödyntämään enemmän Amazonin palveluita. Veralle tehdystä suunnitelmasta arkkitehtuurin muuntamisesta pilviyhteensopivaksi voidaan nähdä kulujen optimointia ja vikasietoisuuden paranemista. Kokonaisuudessaan nämä kaksi tutkimuskohdetta ovat yksittäistapauksia, eikä niiden pohjalta voida väittää kaikkien yritystietojärjestelmien hyötyvän pilvipalveluista.

[79] Maven: https://maven.apache.org/
[80] CentOS: https://www.centos.org/

5 YHTEENVETO

Yritystietojärjestelmän tuotantoympäristö voi olla perinteisessä konesalissa tai pilvipalvelussa. Konesali voi olla oma tai palvelinkapasiteettia voi ostaa jonkun muun omistamasta konesalista. Molemmissa tapauksissa laiterikko vaikutta konesalia käytettäessä suoraan käytettäviin resursseihin. Pilvipalveluissa virtuaalinen palvelinkone voidaan laiterikon sattuessa siirtää toiselle fyysiselle palvelimelle, jolloin vaikutukset resurssien käyttöön ovat vähäiset. Konesalia käytettäessä resurssien käytöstä tehdään sopimus eikä kapasiteettia pysty muuttamaan kovinkaan nopeasti tarpeen mukaan. Pilvipalveluissa sen sijaan kapasiteettia saa varattua sen verran kuin tarvitsee ja resursseiden kasvattaminen ja vähentäminen onnistuvat milloin tahansa. Nämä ovat edut ovat saatavilla kaikissa pilvipalveluissa.

Pilvipalveluita käytettäessä pilviyhteensopiva ohjelmistoarkkitehtuuri mahdollistaa pilvipalveluiden tehokkaamman hyödyntämisen. Mikropalveluarkkitehtuuri on pilviyhteensopiva ohjelmistoarkkitehtuuri, joka koostuu pienistä, itsenäisistä palveluista. Sen keskeisiä eroja palvelukeskeinen arkkitehtuuriin ovat minimaaliset palveluiden väliset riippuvuudet sekä yksinkertainen viestintäprotokolla, joka ei nojaa keskitettyyn älykkääseen viestinvälittäjään. Mikropalveluarkkitehtuurin vahvuutena on mikropalveluiden kyky toimia itsenäisesti, jolloin yksittäisen mikropalvelun ongelmat rajautuvat. Mikropalveluiden käyttö autoskaalauksessa mahdollistaa kulujen optimoinnin skaalaamalla jokaista mikropalvelua erikseen. Palvelukeskeisen arkkitehtuurin vahvuuksia ovat puolestaan keskitetyn viestinvälittäjän mahdollistamat integraatiot eri protokollia käyttävien järjestelmien kanssa. Mikropalveluarkkitehtuuria käytettäessä saavutetaan merkittävästi enemmän hyötyjä pilvipalveluista. Pilviyhteensopiva ohjelmistoarkkitehtuuri ei ole vaatimus pilvipalveluiden käytölle.

Amazon Web Services tarjoaa monipuolisesti erilaisia IaaS- ja PaaS-palveluita. Palveluista löytyy kaikki tarvittava täyttämään yritystietojärjestelmien vaatimukset. Tämän työn käytännön esimerkit osoittivat, että Amazon Web Services –pilvipalvelun käytöstä saatiin enemmän hyötyjä kuin haittoja sekä tuotanto- että kehitysympäristöjen tapauksissa. Jokainen tietojärjestelmä on oma tapauksensa ja jokaisessa tapauksessa pilvipalveluiden hyödyt ja haitat vaihtelevat. Tämän työn perusteella voidaan kuitenkin sanoa, että pilvipalveluiden käyttöä kannattaa ainakin harkita vaihtoehtona perinteiselle konesalille.

LÄHTEET

[1] P. Mell, T. Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology, 2011, 7 p. Saatavissa (viitattu 9.3.2016): http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[2] M, Kavis, Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), John Wiley & Sons, 2014, 224 p.

[3] C. Preimesberger, Big Four IaaS Providers Now Own Half the Market, 2015. Saatavissa (viitattu 17.4.2016): http://www.eweek.com/cloud/big-four-iaas-providers-now-own-half-the-market.html

[4] A. Velte, T. Velte, R. Elsenpeter, Cloud Computing: A Practical Approach, McGraw-Hill, 2010, 335 p.

[5] M. Stine, Migrating to Cloud-Native Application Architectures, O'Reilly, 2015, 51p.

[6] L. MacVittie, Bursting the Cloud, 2008. Saatavissa (viitattu 26.4.): https://devcentral.f5.com/articles/bursting-the-cloud

[7] T. Walters. It's Time to Rethink "Enterprise" Software - Look for the Answer in the Business Process, not the Application, 2013, 12 p. Saatavissa (viitattu 9.3.2016): http://www.opentext.com/file_source/OpenText/en_US/PDF/DCG_RethinkEnterpriseSoftware_24.09.13.pdf

[8] L. Da Xu, Enterprise Integration and Information Architecture: A Systems Perspective on Industrial Information Integration, John Wiley & Sons, 2015, 446 p.

[9] G. Shroff, Enterprise Cloud Computing: Technology, Architecture, Applications, Cambridge University Press, 2010, 290 p.

[10] M. Richards, Mircroservices vs. Service-Oriented Architecture, O'Reilly Media, 2016, 44p.

[11] R. Sweeney, Achieving Service-Oriented Architecture: Applying an Enterprise Architecture Approach, John Wiley & Sons, 2010, 358 p.

[12] S. Newman, Building Microservices: Designing Fine-grained Systems, O'Reilly Media, 2015, 259 p.

[13] M. Fowler, J. Lewis, Microservices: a definition of this new architectural term, 2014. Saatavissa (viitattu 29.3.2016): http://martinfowler.com/articles/microservices.html

[14] Y. Izrailevsky, A Tseitlin, The Netfix Simian Army, 2011. Saatavissa (viitattu 22.4.2016): http://techblog.netflix.com/2011/07/netflix-simian-army.html

[15] M. Conway, How Do Committees Invent? F.D. Thompsons Publications Inc, 1968. Saatavissa (viitattu 15.5.2016): http://www.melconway.com/Home/Committees_Paper.html

[16] R. Clayton, Failing at microservices, Please avoid our mistakes, 2014. Saatavissa (viitattu 22.4.2016): https://rclayton.silvrback.com/failing-at-microservices

[17] B. Wootton, Microservices – Not a free lunch! 2014. Saatavissa (viitattu 22.4.2016): http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html

[18] R. Wilsenach, DevOpsCulture. Saatavissa (viitattu 23.4.216): http://martin-fowler.com/bliki/DevOpsCulture.html

[19] S. Sharwood, Are your servers PETS or CATTLE?, The Register, 2013. Saatavissa (viitattu 16.3.2016): http://www.theregister.co.uk/2013/03/18/servers_pets_or_cattle_cern/

[20] A. Homer, J. Sharp, L. Brader, M. Narumoto, T. Swanson, Cloud Design Patterns: Prescriptive Arhitecture Guidance for Cloud Appliations, Microsoft Corporation, 2014, 236 p.

[21] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications, Springer, 2014, 393 p.

[22] A. Wiggins, The Twelve-Factor App, 2012. Saatavissa (viitattu 26.3.2016) http://12factor.net

[23] M. Fowler, ContinuousDelivery, 2013. Saatavissa (viitattu 22.4.2016): http://martinfowler.com/bliki/ContinuousDelivery.html

[24] M. Fowler, BlueGreenDeployment, 2010. Saatavissa (viitattu 14.5.2016): http://martinfowler.com/bliki/BlueGreenDeployment.html

[25] C. Neudert, D. Huptas, Einfach nur mehr reicht nicht: Entwicklung und Betrieb skalierbarer Architektur, 2015. Saatavissa (viitattu 22.5.2016): https://jaxenter.de/einfach-nur-mehr-reicht-nicht-entwicklung-und-betrieb-skalierbarer-architektur-12945

[26] D. Sato, CanaryRelease, 2014. Saatavussa (viitattu 15.5.2016): http://martin-fowler.com/bliki/CanaryRelease.html

[27] Velocity 2011: Jon Jenkins, "Velocity Culture", 2011. Saatavissa (viitattu 14.5.2016): https://www.youtube.com/watch?v=dxk8b9rSKOo

[28] Amazon Web Services, About AWS. Saatavissa (viitattu 17.4.2016): https://aws.amazon.com/about-aws/

[29] Amazon Web Services User Guide, Amazon Elastic Compute Cloud, Regions and Availability Zones. Saatavissa (viitattu 17.4.2016): http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

[30] Amazon Web Services, Shared Responsibility Model, Saatavissa (viitattu 19.4.2016): https://aws.amazon.com/compliance/shared-responsibility-model/

[31] Amazon Web Services User Guide, Amazon Virtual Private Cloud, Security Groups for Your VPC. Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html

[32] Amazon Web Services User Guide, Amazon Virtual Private Cloud, What is Amazon VPC? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html

[33] Amazon Web Services User Guide, Amazon Virtual Private Cloud, Network ACLs. Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_ACLs.html

[34] Amazon Web Services User Guide, Amazon Elastic Compute Cloud, What is Amazon EC2? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

[35] Amazon Web Services User Guide, Amazon Elastic Compute Cloud, Amazon Elastic Block Store. Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html

[36] Amazon Web Services User Guide, Auto Scaling, What is Auto Scaling? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/WhatIsAutoScaling.html

[37] Amazon Web Services User Guide, Elastic Load Balancig, What is Elastic Load Balancing? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elastic-load-balancing.html

[38] Amazon Web Services, AWS Reference Architectures. Saatavissa (viitattu 19.4.2016): http://media.amazonWebservices.com/architecture-center/AWS_ac_ra_Web_01.pdf

[39] Amazon Web Services User Guide, Amazon Relational Database Service, What is Amazon Relational Database Service (Amazon RDS)? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

[40] Amazon Web Services User Guide, Amazon DynamoDB, What is Amazon DynamoDB? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[41] Amazon Web Services User Guide, Amazon ElastiCache, What is Amazon ElastiCache? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/WhatIs.html

[42] Amazon Web Services User Guide, Amazon Redshift, What is Amazon Redshift? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html

[43] Amazon Web Services User Guide, Amazon Simple Storage Service, What is Amazon S3? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html

[44] Amazon Web Services User Guide, Amazon Glacier, What is Amazon Glacier? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/amazonglacier/latest/dev/introduction.html

[45] Amazon Web Services User Guide, Amazon CloudFront, What is Amazon CloudFront? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

[46] Amazon Web Services User Guide, Amazon Simple Queue Service, What is Amazon Simple Queue Service? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/Welcome.html

[47] Amazon Web Services User Guide, Amazon Simple Notification Service, What is Amazon Simple Notification Service? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/sns/latest/dg/welcome.html

[48] Amazon Web Services User Guide, Amazon Simple Email Service, What is Amazon SES? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/ses/latest/DeveloperGuide/Welcome.html

[49] Amazon Web Services User Guide, Amazon Virtual Private Cloud, VPN Connections. Saatavissa (viitattu 22.4.2016): http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpn-connections.html

[50] Amazon Web Services User Guide, AWS Storage Gateway, What is AWS Storage Gateway? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/storagegateway/latest/userguide/WhatIsStorageGateway.html

[51] Amazon Web Services User Guide, AWS Lambda, What is AWS Lambda? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[52] Amazon Web Services User Guide, Amazon Elastic MapReduce, What is Amazon EMR? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html

[53] Amazon Web Services User Guide, AWS Elastic Beanstalk, What is AWS Elastic Beanstalk? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html

[54] Amazon Web Services User Guide, Amazon CloudWatch, What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsCloudWatch.html

[55] Amazon Web Services User Guide, AWS Identity and Access Management, What is IAM? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[56] Amazon Web Services User Guide, AWS CloudTrail, What is AWS CloudTrail? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html

[57] Amazon Web Services User Guide, AWS CloudFormation, What is AWS CloudFormation? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

[58] Amazon Web Services User Guide, AWS OpsWorks, What is AWS OpsWorks? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/opsworks/latest/userguide/welcome.html

[59] Amazon Web Services User Guide, AWS Database Migration Service, What is AWS Database Migration Service? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/dms/latest/userguide/Welcome.html

[60] Amazon Web Services User Guide, AWS CodeCommit, What is AWS CodeCommit? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html

[61] Amazon Web Services User Guide, AWS CodeDeploy, What is AWS CodeDeploy? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html

[62] Amazon Web Services User Guide, AWS CodePipeline, What is AWS CodePipeline? Saatavissa (viitattu 19.4.2016): http://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html

[63] Työsuojeluvalvonnan uusi tapa tarkastaa – Valtimo-hanke: Valvonnan parempaa vaikuttavuutta paremmalla tiedonhallinnalla, Sosiaali- ja terveysministeriön julkaisuja 2016:1, 2016. Saatavissa (viitattu 15.5.2016): https://www.julkari.fi/bitstream/handle/10024/130303/STM_2016_1_VALTIOMO-hanke_julkariin.pdf

[64] Valtimo – Projektin tiedot, 2013. Saatavissa (viitattu 15.5.2015): https://extra.gofore.com/confluence/display/Valtimo/Projektin+tiedot

[65] Valtimo – Järjestelmän rakenne, 2011. Saatavissa (viitattu 15.5.2015): https://extra.gofore.com/confluence/pages/viewpage.action?pageId=9799302