Haaga-Helia
ammattikorkeakoulu Oy

Paikallisesta konesalista julkiseen pilveen

Jussi Lemmetyinen

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2021

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Jussi Lemmetyinen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Paikallisesta konesalista julkiseen pilveen

Sivu- ja liitesivumäärä
51 + 0

Opinnäytetyön tavoitteena on tuoda esille yleisimpiä haasteita, kun siirrytään perinteisistä konesaleista julkiseen pilveen. Työ esittelee tunnettuja arkkitehtuurisia ratkaisumalleja ja tapoja ratkaista haasteita pilviympäristössä. Opinnäytetyön konkreettisena lopputuloksena on raportti julkisen pilven tuomista hyödyistä ja haasteista, niin liiketoiminnallisesti kuin teknisesti. Opinnäytetyön tietoperusta pohjautuu AWS:n virallisiin dokumentaatioihin, alan kirjallisuuteen ja verkkojulkaisuihin.

Opinnäytetyö keskittyy päällimmäisenä Amazon Web Services (AWS) -pilveen. AWS on Amazon verkkokaupan tytäryhtiö ja sen tarjoamat palvelut voidaan jakaa tiedonkäsittelyyn, verkkoyhteyksiin ja tietoturvan perustaviin palveluihin, sekä suureen määrään automaatiota. AWS:ää voidaan ajatella massiivisena työkalupakkina, jotka voivat tehdä valikoiman infrastruktuuritehtäviä. AWS tarjoaa pilvipalvelualustoja ja -sovellusliittymiä yksityishenkilöille, yrityksille ja hallituksille.

AWS:ää ei ole mahdollista oppia kokonaisuudessaan yhdellä opinnäytetyöllä, sillä se ei ole yksinkertaisesti mahdollista jo sen takia, että AWS muuttuu jatkuvasti. Muuttuvuus on kuitenkin käsite, minkä oppii omaksumaan.

Tämä opinnäytetyö on suunnattu opintojen loppuvaiheen opiskelijoille, teknologiarooleissa oleville, kuten teknologiajohtajille, arkkitehdeille, kehittäjille ja operatiivisen ryhmän jäsenille.

Opinnäytetyön tavoitteena on tuoda esille AWS:n käytänteitä ja strategioita, joita käytetään pilviarkkitehtuureja suunniteltaessa. Tarkoitus ei kuitenkaan ole käydä läpi kaikkia julkisen pilven tuomia etuja tai haittoja, vaan yleisimpiä teknisiä ominaisuuksia, jotka hyödyntävät suurinta osaa yrityksistä. Opinnäytetyö ei myöskään sisällä seikkaperäistä pilvipalveluiden käyttöopasta, eikä tarkkoja toteutuksen yksityiskohtia tai arkkitehtonisia malleja. Työ on toteutettu vuoden 2021 kevään aikana.

Asiasanat
Pilvipalvelut, Amazon Web Services, AWS, AWS-kehys, arkkitehtuuri, pilvipalvelumallit, pilvipalvelualusta

Sisällys

1 Johdanto

Pilvipalveluiden vaikutuksia teollisuudelle ja loppukäyttäjille on vaikea yliarvioida. Esimerkiksi pilviverkoissa toimivien ohjelmistojen läsnäolo on muuttanut monia jokapäiväisen elämän osa-alueita (Glass, 2020). Hyödyntämällä pilvipalveluja startup- ja suuryritykset voivat optimoida kustannuksia ja lisätä tarjontaansa ostamatta tai hallinnoimatta kaikkia laitteita ja ohjelmistoja.

Ohjelmistokehittäjillä on valtuudet käynnistää maailmanlaajuisesti saatavilla olevia sovelluksia ja verkkopalveluja hetkessä. Tutkijat voivat jakaa ja analysoida tietoja sellaisessa mittakaavassa, joka on aiemmin ollut mahdollista vain hyvin rahoitetuille hankkeille. Internet käyttäjät voivat nopeasti käyttää ohjelmistoja ja tallennustilaa, jotta he voivat luoda, jakaa ja tallentaa digitaalista mediaa määrinä, jotka ylittävät reilusti heidän henkilökohtaisten laitteidensa laskentakapasiteetin.

Pilvipalveluiden kasvavasta läsnäolosta huolimatta, sen yksityiskohdat pysyvät monille hämärän peitossa. Mikä pilvi tarkalleen on, miten sitä käytetään, ja mitkä ovat sen etuja yrityksille, kehittäjille tai opiskelijoille? Pilvilaskentapalveluita ja -malleja on monenlaisia. Niistä kolme tärkeintä, sekä tunnetuinta ovat ohjelmistopalveluna (SaaS), infrastruktuuri palveluna (IaaS) ja alusta palveluna (PaaS).

Tämä opinnäytetyö antaa yleiskatsauksen pilvipalveluihin liittyvistä aspekteista, sen historiasta, toimintamalleista, tarjonnasta ja riskeistä. Oheinen laadullinen tutkimustyyppinen opinnäytetyö keskittyy aiheeseen ennen kaikkea julkisia pilvipalveluita tuottavan AWS:n (Amazon Web Services) kannalta. Samoja lainalaisuuksia on kuitenkin mahdollista hyödyntää muissakin ympäristöissä, sillä monet hyvät käytänteet eivät rajoitu vain pilvimaailmaan.

Työ lähti liikkeelle opiskelijan kasvavasta ammatillisesta kiinnostuksesta aiheeseen. Opiskelija on työskennellyt ammattinsa takia teeman parissa ja on täten päässyt tutustumaan aiheeseen työmaailmassa. Pilvipalveluita ja niiden merkitystä tuodaan harvakseltaan esille tietojenkäsittelyn tutkinnon opiskelun aikana. Opiskelijan mielestä pilvipalvelut mielletään usein ohjelmointiin, liiketoimintaan tai digitaalisiin palveluihin liittymättömänä aiheena ja täten aihetta käsitelläänkin Haaga-Helian opinnoissa vain ICT-infrastruktuurin kannalta. Pilvipalveluihin liittyy kuitenkin paljon muutakin teemoja, kuin infrastruktuuri tai ohjelmointi.

Opiskelija oli toteuttamassa työelämän projektia syksyn aikana vuonna 2020, jossa siirrettiin perinteiseen konesalimalliin suunniteltu web-sovellus julkiseen pilvipalveluun. Oheinen projekti toimi lopullisena päätöksentekijänä opinnäytetyön aihetta valittaessa, sillä projektin aikana opiskelija havaitsi, että on olemassa useita tapoja luoda ja tehdä turvallisia ja skaalautuvia asioita pilvessä.

Opinnäytetyön toisen kappaleen aikana tutkitaan eroavaisuuksia perinteisten ja pilvipalveluiden välillä tutkimalla sekä tietoturvallisuutta että taloudellisia aspekteja. Kappaleen loppuun mentäessä lukijalla on käsitys siitä, kuinka pilvi voi yleisellä tasolla auttaa yritystä ja mitä aspekteja täytyy ottaa huomioon, kun aloitetaan pilven käyttö omissa projekteissa.

Kolmannessa kappaleessa tutustutaan tarkemmin hyvin suunniteltuun AWS-kehykseen. Kappaleessa tuodaan esille hyvään arkkitehtuurin liittyviä ylätason aspekteja, sekä muutamia käytännön haasteita, joita kohdattiin lopussa esiteltävässä case-esimerkissä, sekä esimerkkitapaus huonosti hoidetusta tietoturvasta.

Opinnäytetyön tutkivan teoriaosuuden jälkeen opiskelija esittelee konkreettisen case-esimerkin. Case-esimerkissä esitellään Haaga-Helian opiskelijoiden toteuttama Softalaprojekti-kurssin aikana tekemä sovellus, joka on opinnäytetyön tekijän opastuksella viety julkiseen pilveen teoriakappaleissa esiteltyjä käytäntöjä hyödyntäen.

Toiminnallisessa case-esimerkissä nousee esille tyypillisiä haasteita, joita pilviympäristöissä työskennellessä tulee vastaan. Case-esimerkissä käytetty kolmivaiheinen arkkitehtuuri toimii monelle sovellukselle hyvänä ponnahduslautana esimerkiksi mikropalveluarkkitehtuuriin, sillä suurimalla osalla web-sovelluksista on hyvin samanlaisia perustarpeita. Case-esimerkki toteutettiin keväällä kolmannen ja neljännen periodin aikana vuonna 2021.

Opinnäytetyössä ei puhuta ”parhaista käytänteistä”, sillä termin ongelmana on se, että se saattaa saada lukijat ajattelemaan, että tiettyyn ongelmaan on vain yksi ratkaisu, riippumatta asiayhteydestä. Opiskelija mieluummin kuvailee käytäntöjä ja malleja, sekä panee merkille, milloin ne ovat hyödyllisiä ja mitkä ovat niiden rajoitukset. Opiskelija kuvaa joitain näistä vaihtoehdoista tehokkaammiksi tai sopivammiksi, mutta pyrkii olemaan avoin vaihtoehdoille. Lähteiden merkintään ja kokoamiseen on käytetty RefWorks työkalua.

2 Pilvipalvelut yritysmaailmassa

Tietoa on kaikkialla ja tiedonsaannista on tullut yksi modernin historian suurimmista tapahtumista. Elämme kaikkialla sijaitsevien pilvipalveluiden aikakaudella. Pilvipalvelut ja pilviverkot tarjoavat ketteryyttä, alhaisempia kustannuksia ja paremman pääsyn resursseihin maailmanlaajuisesti.

Pilvipohjaisten datan sosiaalisia etuja löydetään jatkuvasti. Kun yhä monimutkaisemmat sovellukset eivät enää rajoitu yhteen fyysiseen sijaintiin, kasvu voi olla eksponentiaalista. Pilvipohjaisella datalla viitataan tietoon, joka on tallennettuna kolmannen osapuolen pilvipalvelujen hallinnoimassa datakeskuksessa (Castgna, Lelii,).

Tässä kappaleessa käsitellään pilvipalveluiden yleisiä käsitteitä, sekä niihin liittyvää historiaa ja taloudellisia aspekteja, jotta myöhemmin opinnäytetyössä voidaan syventyä tarkemmin asioihin AWS:n näkökulmasta. Lisäksi käsitellään erilaisia kokonaisarkkitehtuurillisia vaihtoehtoja, joita on hyvä miettiä pilvipalveluihin siirryttäessä. Kappaleessa esiteltäviä jaetun vastuun sekä pilvipalvelumalleja hyödynnetään vahvasti case-esimerkin aikana.

2.1 Pilvipalveluiden historia

Sanaa pilvi on alun perin käytetty metaforana Internetille, standardoitua pilvimäistä muotoa taas käytettiin verkon merkitsemiseen vuokaavioissa ja esityksissä (CHM, 2021). Pilvipalvelulla (cloud service) - taas viitataan sen nykyisessä kontekstissa yleensä laajaan valikoimaan palveluita, jotka tarjotaan yrityksille ja asiakkaille Internetin välityksellä (Citrix, a). Nämä palvelut ovat suunniteltu tarjoamaan pääsyn sovelluksiin ja resursseihin ilman sisäistä infrastruktuuria tai laitteistoa (Citrix, b). Esimerkiksi sähköpostin tarkistaminen tai asiakirjojen tekeminen yhteistyössä – monet käyttävät pilvipalveluja koko työpäivän ajan riippumatta siitä, ovatko he siitä tietoisia (Citrix,).

Pilviverkoilla (cloud networking) taas viitataan joidenkin tai kaikkien verkkoresurssien ja palveluiden – virtuaalisten reitittimien, kaistanleveyden, virtuaalisten palomuurien tai verkonhallintaohjelmistojen – ylläpitämistä tai käyttöä pilvestä, joko julkisesta, yksityisestä tai hybridistä (Citrix,).

Termi pilvilaskenta (cloud computing) on mahdollisesti alun pitäen mainittu Compaqin sisäisessä dokumentaatiossa vuonna 1996 (Regalado, 2011). Todennäköisesti termi sai kuitenkin modernin kontekstinsa vasta kymmenen vuotta myöhemmin Googlen CEO Eric Schmidtin toimesta: ”Se alkaa siitä, että datapalvelujen arkkitehtuurin tulisi olla palvelimilla. Kutsumme sitä pilvipalveluksi – niiden pitäisi olla ”pilvessä” jossain” (Schmidt,

2006). Nykypäivänä Pilvilaskennalla tarkoitetaan tietotekniikkaresurssien on-demand-toimitusta Internetissä porrastetusti hinnaston mukaisesti (AWS, f). On-demandilla taas viitataan palveluun, jossa kuluttajan ei tarvitse tehdä ennakkoon taloudellisia panostuksia, vaan maksu suoritetaan kun palvelua tai toimintoa käytetään (Techopedia,).

AWS lanseerasi 25.elokuuta 2006 Elastic Compute Cloudin (EC2). EC2 on keskeinen osa Amazon.comin pilvipalvelualustaa, sillä se sallii käyttäjien vuokrata virtuaalitietokoneita omien tietokoneohjelmiensa ajamiseksi (Cubrilovic, 2006). Samana vuonna Google osti yrityksen nimeltä Upstartle ja ennen kaikkea heidän web-pohjaisen tekstinkäsittely ohjelmistonsa, Writelyn (Hinchcliffe, 2006). Pian tämän jälkeen Google julkisti Google Docs Servicen. Vuonna 2007 binge-watchingista eli maratonkatselusta tuli asia, kun Netflix-niminen yritys aloitti videoiden suoratoistosivuston (Anderson, 2007).

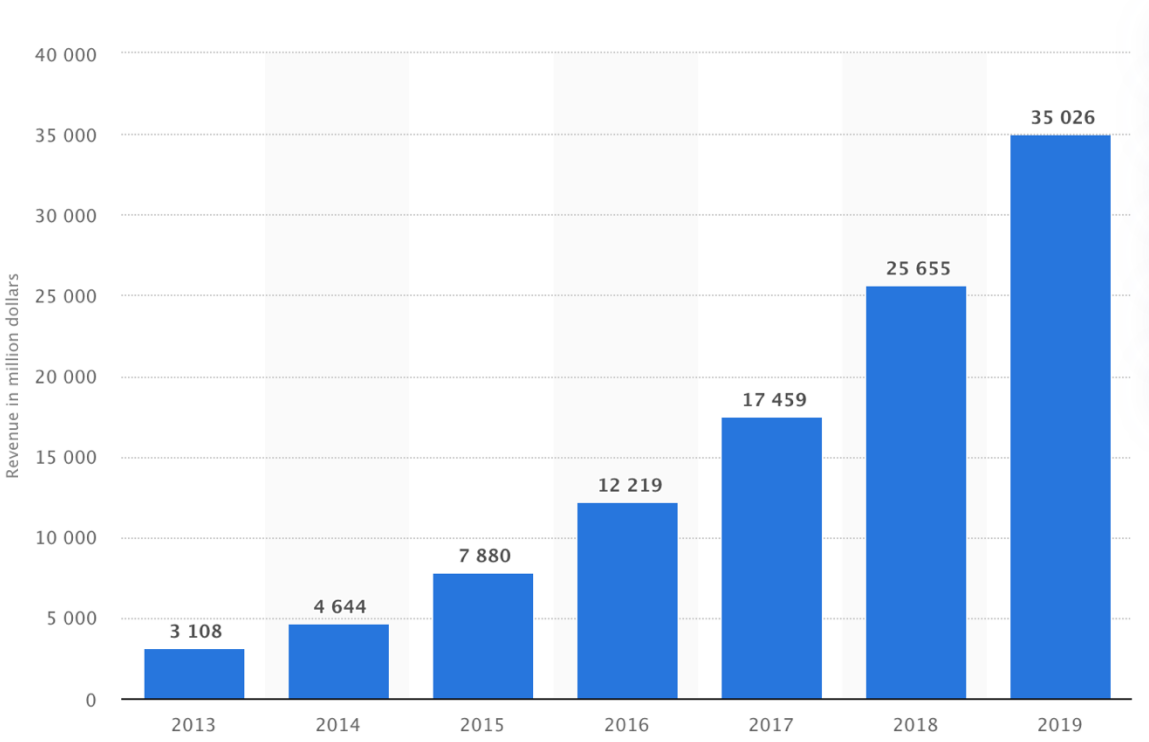
Puolustavat mestarit ovat vanhojen laitteiden ja ohjelmistojen toimittajia: HP, IBM, Oracle, Microsoft ja VMware. Haastajia ovat pilvipalvelualustat, kuten Amazon Web Services (Furrier, 2015). On arvioitu, että yli yhdeksänkymmentä maailman sadasta suuresta pankista, kymmenen suurinta vakuutusyhtiötä, suurin osa kahdestakymmenestäviidestä vähittäiskauppiaasta ja suurin osa maailman suurimmista lentoyhtiöistä luottavat edelleen IBM:n keskusyksiköihin (Wilkins, 2019).

Amazon Web Service (AWS) pilvi-infrastruktuuri palveluna julkaistiin alun perin verkkokauppa Amazon.comin sivuliiketoimintana. AWS:n juuret juontavat 2000-luvun alkuun, kun Amazon oli hyvin erilainen yritys kuin nykypäivänä – verkkokauppa yritys, joka kamppaili skaalautuvuus ongelmien kanssa. (Miller, 2016)

Kyseiset ongelmat pakottivat Amazonin rakentamaan luotettavat ja vankat sisäiset järjestelmät selviytymään kokemastaan hyperkasvusta (Miller, 2016). Hyperkasvulla viitataan nopeaan laajenemisvaiheeseen, jonka yritykset kokevat, kun ne kasvavat – erityisesti kun yrityksen vuotuinen kasvuvauhti on vähintään 40 % (Karnes, 2020).

Pilvipalvelut kasvavat jatkuvasti ja esimerkiksi Amazon saa suurimman osan rahoistaan tytäryhtiö AWS:ltä. AWS - maailman suurin pilvipalveluyritys, joka on kaksi kertaa suurempi kuin seuraavaksi suurin - liikevaihdoksi on arvioitu 46 miljardia dollaria (Raeste, 2020). Kuitenkin on arvioitu, että yritysten datasta vain 4–6 prosenttia on siirretty pilveen (Raeste, 2020). Pilvipalveluiden tarjoajilla on siis 94—96 prosenttia markkinoista hyödyntämättä.

Vuonna 2019 AWS raportoi 37 %:n vuotuisen kasvun ja se oli 12 % Amazonin liikevaihdosta (Sparks, 2020). Kuvassa 1 on AWS:n vuosituottojen kehitys vuosien 2013 ja 2019 välillä.



Kuva 1. AWS:n vuosituotto vuosina 2013–2019. (Statista Research Department, 2021)

2.2 Pääomakulut ja liikekulut

Hyppääminen pilvipalveluihin tekemättä etukäteen suunniteltua työtä on hieman kuin sopimuksen allekirjoittamien lukematta hienoa tekstiä. Se, mikä näyttää suoraviivaiselta, voi nopeasti muuttua odotettua monimutkaisemmaksi ja kalliimmaksi. Onnistuneen pilvitransformaation suorittamiseksi on oleellista ymmärtää, miten kulut muuttuvat pilveen siirryttäessä.

Sijoittamalla omaan konesaliin investointeja yhtiö sijoittaa pääomaa mahdolliseen tulevaisuuden tarpeeseen. Toisaalta käyttämällä pilvipalvelua, joka tarjoaa on-demand palvelua, raha menee jatkuviin käyttökustannuksiin tai liikekuluihin.

Pääomakulut ja liikekulut ovat molemmat liiketoiminnan kuluja ja ne liittyvät siihen, miten rahat maksetaan yritykseltä (Software Advisory Service,). Ne eivät ole ainutlaatuisia IT-alalle; ne koskevat yrityskuluja yleisesti. Yritysten IT-tarpeiden käsittelyyn on tarjolla monia lähestymistapoja, joten on oleellista miettiä, miten oheiset kulut tulisi ottaa huomioon arvioitaessa ratkaisuja (Vanderweide, 2019).

Pääomakustannukset määritellään liiketoiminnan kuluiksi, jotka aiheutuvat pitkäaikaisten hyötyjen luomisesta tulevaisuudessa, kuten kiinteän omaisuuden, rakennuksen tai laitteiden ostamisesta. Esimerkkejä IT-tuotteista ovat kokonaiset järjestelmät, palvelimet, tulostimet, skannerit tai ilmastointilaitteet ja generaattorit. Tuotteet ostetaan kerran ja niistä pyritään saamaan hyötyä useiden vuosien ajan. Tällaisten tuotteiden huoltoa pidetään myös pääomakustannuksena, sillä se pidentää niiden käyttöikää ja hyödyllisyyttä. (Vanderweide, 2019)

Liikekulut tai toimintakulut ovat käyttökustannuksia, jotka menevät päivittäisen liiketoiminnan ylläpitämiseen, kuten palvelut ja kulutustarvikkeet, jotka kuluvat loppuun ja maksetaan käytön mukaan (Vanderweide, 2019). Esimerkkejä tästä ovat tulostinpatruunat, paperi, sähkö ja vuosittaiset palvelut, kuten verkkosivustojen ylläpito tai verkkotunnusten uusiminen. Nämä ovat välttämättömiä kuluja yrityksen menestykselle, mutta niitä ei pidetä merkittävinä pitkäaikaisina sijoituksina, kuten pääomakustannuksia (Vanderweide, 2019).

Keskeisimpiä syitä miksi organisaatiot joko käyttävät tai suunnittelevat pilvi-It-infrastruktuuria, on rahan säästäminen (Chism, Veksler, 2017). Toiminnan kokonaiskustannus eli total cost of ownership (TCO) -analyysimenetelmiä voidaan käyttää perinteisen palvelinkeskuksen omistamisesta aiheutuvien kustannusten vertaamiseksi verrattuna AWS:n vastaaviin palveluihin (Chism, Veksler, 2017). TCO-ajattelulla pyritään hahmottamaan hankinnan elinkaaren aikaisia kustannuksia (Logistiikan Maailma,). Yksinkertaistettuna se tarkoittaa sitä, että tarvittavien hankintojen osalta pohditaan, mitä kustannuksia hankittava tavara tai palvelu aiheuttaa (Logistiikan Maailma,).

Organisaatioita, jotka harkitsevat siirtymistä pilveen, ohjaa usein rahan säästämisen lisäksi yrityksen tarve tulla ketterämmäksi ja innovatiivisemmaksi, mutta perinteinen pääomakustannusten rahoitusmalli vaikeuttaa uusien ideoiden testaamista. AWS – kuten myös muut julkiset pilvipalvelut mahdollistavat uusien palveluiden kokeilemisen ilman suuria etukäteen aiheutuneita kustannuksia. Julkista pilveä hyödyntämällä voidaan palauttaa pääomakustannuksia takaisin yleisrahastoon ja sijoittaa sitä toimintaan, joka palvelee paremmin organisaatiota (Chism, Veksler, 2017).

2.3 Erilaisia pilviratkaisuja

Tässä kappaleessa tutustutaan erilaisiin potentiaalisiin ja vaihtoehtoisiin pilviratkaisuihin. Pilviratkaisuja miettiessä, on oleellista tutustua ja löytää organisaatiolle itselleen parhaiten sopiva malli. Perinteisesti yritykset ovat luottaneet palvelinkeskusten sisäisiin malleihin, jotka vaativat valtavan määrän pääomakustannuksia, kun ostetaan tilaa, laitteita, ohjelmistoja, sekä työvoimaa kaiken ylläpitämiseksi (Vanderweide, 2019).

Jos yritys ei käytä perinteisiä suurkoneita, on todennäköistä, että käytössä on oma yksityinen pilvi omissa palvelinkeskuksissa. Suurkoneita kutsutaan myös nimellä Mainframe – ”pääkehikko” (Vance, 2005). Tunnetuin ja suurin suurkoneiden valmistaja on IBM (Nummela, 2010). Suurkoneet ovat lähtökohtaisesti suuritehoisia tietokoneita, joissa on huomattavan paljon muistia sekä prosessoreita, ja ne käsittelevät miljardeja yksinkertaisia laskutoimituksia ja tapahtumia reaaliajassa (IBM, 2020).

Pilviin liittyvää infrastruktuuria on kolme yleistä tyyppiä: Julkinen, yksityinen ja hybridipilvi, mikä yhdistää julkisen, että yksityisen pilven elementit.

Yksityinen pilvi määritellään laskentapalveluiksi, joita tarjotaan joko Internetin yli tai yksityisen sisäisen verkon kautta ja vain valituille käyttäjille suuren yleisön sijasta (Microsoft Azure, a). Yksityiset pilvet tarjoavat lähtökohtaisesti korkeamman tason turvallisuutta ja yksityisyyttä sekä yrityksen palomuurien että sisäisen isännöinnin avulla (Microsoft Azure,). Täten yritetään varmistua siitä, että toiminnot ja arkaluontoiset tiedot eivät ole kolmansien osapuolten käytettävissä (Microsoft Azure,).

Eräs haittapuoli tämän tyyppisessä ratkaisussa on se, että yrityksen IT-osasto on vastuussa yksityisen pilven hallinnan kustannuksista ja vastuullisuudesta (Microsoft Azure,). Täten yksityiset pilvet vaativat samat henkilöstö-, hallinta- ja ylläpitokulut kuin perinteisen palvelinkeskuksen omistus (Microsoft Azure,). Tunnettuja yksityisiä pilviä ovat esimerkiksi OpenStack, Apache CloudStack, Eucalyptus ja OpenNebula (Priyam, 2018).

Yksityiset pilvet voidaan myös yhdistää julkisten pilvien kanssa hybridipilven luomiseksi (Microsoft Azure,). Hybridipilvellä viitataan sekoitettuun tietojenkäsittely-, tallennus- ja palveluympäristöön, joka koostuu paikallisesta infrastruktuurista, yksityisistä pilvipalveluista, julkisesta pilvestä – kuten Amazon Web Services (AWS) tai Microsoft Azure – ja jota voidaan orkestroida eri alustojen välillä (NetApp,).

Hybridipilven haittapuolena on korkeammat kustannukset kuin julkisella pilvellä, sillä korkeammat alkumaksut sekä laitteiden kustannuksien takaisin maksamisen nostaa hintaa (NetApp,). Lisäksi mahdollisena haittapuolena on vastuu omasta datakeskuksesta, IT-laitteistosta ja yritysohjelmiston – sekä oman turvallisuuden ja vaatimustenmukaisen – ylläpidon ja toiminnan (NetApp,).

Julkinen pilvi määritellään tietojenkäsittelypalveluksi, joita kolmannet osapuolet tarjoavat julkisen Internetin välityksellä esimerkiksi yksityishenkilöille, yrityksille ja hallituksille. Palvelut voivat olla joko ilmaisia tai niitä voidaan myydä tilauksesta, jolloin asiakkaat voivat

maksaa vain kulutuksesta käyttämistään suorittimien jaksoista, tallennustilasta tai kaistanleveydestä (Microsoft Azure, b).

Esimerkkejä julkisien pilvipalvelujen tarjoajista ovat AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud (SoftLayer) ja Alibaba Cloud (Priyam, 2018). Pilvitransformaatiota suunniteltaessa on syytä ottaa huomioon, että julkisten pilvipalvelujen palvelutarjonta vaihtelee tietyissä määrin. Esimerkiksi Oraclen lisensointeja voidaan tehdä vain valituissa pilvipalveluissa kuten AWS ja Azure.

Datakeskusten fyysinen sijainti on myös eräs tekijä palveluntarjoajaa valittaessa. Lähimmästä datakeskuksesta on yleensä mahdollista saada matalin viive. Lähtökohtaisesti mitä lyhyempi etäisyys pilven ja loppukäyttäjän välillä on, sitä pienempi etäisyys. Jokaisessa maassa tai unionissa on erilaiset vaatimustenmukaisuusnormit ja -säännöt käyttäjätietojen suojaamiseksi. Jotkut maat saattavat kieltää siirrot alueidensa ulkopuolelle. Tällaisten sääntöjen rikkominen voi johtaa oikeudenkäynteihin ja sitä kautta kriittiseen taloudelliseen ja maineelliseen vahinkoon organisaatiolle. (Perry, 2019)

Virtualisoinnin ja Internetin yleistymisen myötä pilvipalvelut ja julkiset pilvet ovat korvanneet monet paikalliset datakeskukset ja tekevät niin jatkossakin (Wilkins, 2019). Onkin siis oleellista selvittää sovelluksen pääomakustannukset julkisessa pilvessä oman datakeskuksen sijaan. Julkisessa pilvessä pääomakustannukset luokitellaan toisinaan vuokraukseksi omistamisen sijaan (Wilkins, 2019). Kustannuksiin liittyviä aiheita on kuvattu tarkemmin kappaleessa 2.2.

Toisin kuin yksityiset pilvet, julkiset pilvet voivat säästää yrityksiä kalliilta alku kustannuksilta, jotka aiheutuvat paikan päällä tapahtuvan laitteisto- ja sovellusinfrastruktuurin ostamisesta, hallinnasta ja ylläpidosta. Syynä tähän on se, että pilvipalveluiden tarjoaja on vastuussa järjestelmän hallinnasta ja ylläpidosta. Julkisia pilviä voidaan myös ottaa nopeammin käyttöön kuin paikallisia infrastruktuureja melkein loputtomasti skaalautuvalla alustalla. (Microsoft Azure,)

Potentiaalisesti jokainen yrityksen työntekijä voi käyttää samaa sovellusta mistä tahansa toimistosta tai sivuliikkeestä valitsemallaan laitteella, kunhan hän voi käyttää Internetiä. Vaikka julkisista pilviympäristöistä on otettu esiin tietoturvaongelmia, kun ne on toteutettu oikein, julkinen pilvi voi olla yhtä turvallinen tai turvallisempi kuin tehokkaimmin hallinnoitu yksityinen pilvipalvelutoteutus, jos palveluntarjoaja käyttää asianmukaisia suojausmenetelmiä, kuten tunkeutumisenhavaitsemis- ja estojärjestelmiä. (Microsoft Azure,)

Pilvipalveluiden käyttö voi siis lisätä tietoturvan tasoa, mutta on myös tapauksia, joissa se on laskenut tietoturvan tasoa. Tietoturvan laskun syynä on kuitenkin hyvin usein syynä esimerkiksi vähiten oikeuden periaatteen noudattamatta jättäminen. Vähiten oikeuden periaatetta käsitellään tarkemmin kappaleessa 3.4.

Julkisen pilven haittapuolina voidaan pitää mahdollista vaihtoehtojen puutetta. Julkisilla pilvipalvelujen tarjoajilla on yleensä ”yksi koko sopii kaikille” tyyppinen lähestymistapa vakiovaihtoehdoilla. Jos yrityksellä on jokin ainutlaatuinen tarve, se ei välttämättä pysty täyttämään näitä vaatimuksia. Lisäksi kun IT-infra ulkoistetaan julkiseen pilveen, menettää yritys osan kontrollista. Julkisista pilvimalleista puuttuu yleensä myös asiakastuki – tarkemmin sanottuna se on erillinen sopimus asiakkaalle. Asiakastuki on tärkeä asia muistaa, sillä palveluntarjoajien tuen määrä ja laatu vaihtelee.

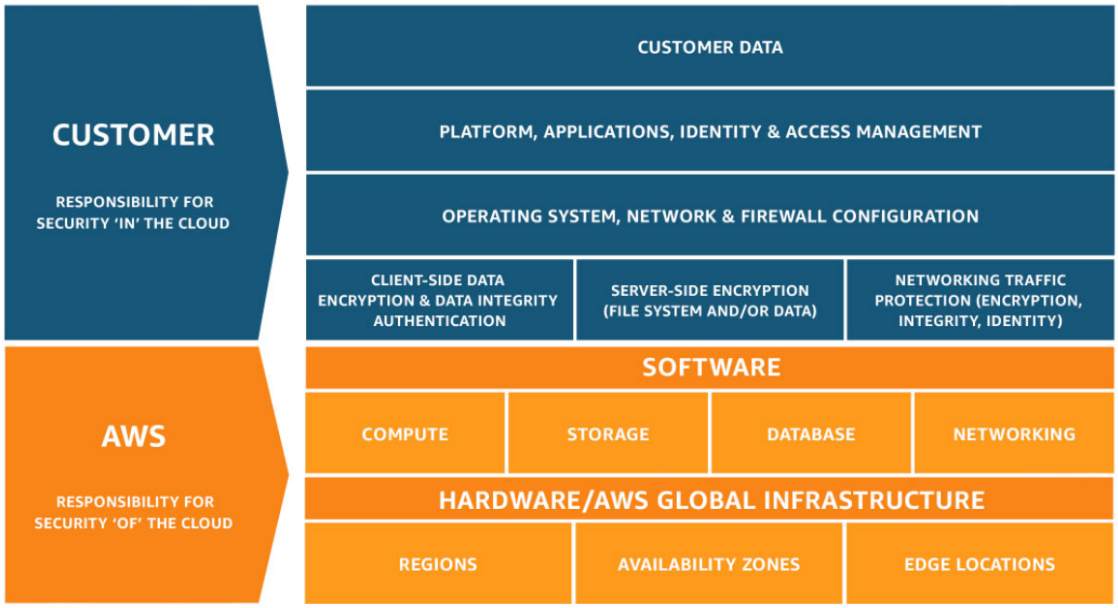
Jos organisaatiolla on kuitenkin hyvin erikoistuneita tietoturva-, sääntely- tai infrastruktuuritarpeita tai organisaatio haluaa maksimaalisen hallinnan pilviympäristössä, voi yksityinen pilvi tai yksityisen pilven kaltainen palvelu sopia paremmin.

2.4 Pilvipalvelumallit

Pilveen siirryttäessä joudutaan usein valitsemaan eri pilvipalvelumallien väliltä. Helpoin tapa ymmärtää karkearakeisia eroja Iaas. PaaS- ja Saas-pilvipalvelumallien välillä on käydä läpi, mitä osia pilvipalvelun tarjoaja hallitsee ja mitä loppukäyttäjä.

Perinteisessä paikallisessa konesalissa tai palvelinkeskuksessa asiakas omistaa ja on vastuussa kokonaisuudesta. Vastuussa voi myös olla asiakkaan puolesta palveluntarjoaja, joka huolehtii ylläpidosta. Asiakas tai palveluntarjoaja vastaa tietoturvasta koko käyttöympäristössä, mukaan lukien sovellukset, fyysiset palvelimet ja myös fyysisen rakennuksen turvallisuudesta (CloudPassage,).

Siirryttäessä julkiseen pilveen kuten AWS:ään, osa vastuista siirtyy pilvipalveluiden tarjoajalle. Jaetun mallin on tarkoitus vähentää asiakkaan operatiivista taakkaa. AWS operoi, hallitsee ja ohjaa komponentteja isäntäkäyttöjärjestelmästä ja virtuaalisointikerroksesta aina palvelujen fyysiseen turvallisuuteen asti, joissa palvelu toimii (AWS, d). Kuvassa 2 on visuaalisesti kuvattu jaetun mallin vastuualueet IaaS-mallia noudattaen.



Kuva 2. Jaetun vastuun malli (AWS,)

Kuva 2 vastaa IaaS-tyylistä eli infrastruktuuria palveluna tai jaetun vastuun mallia EC2:lle. EC2-instanssilla tarkoitetaan palvelinta, joka virtualisoi osan laitteistosta, jotta yritykset voivat käyttää omia käyttöjärjestelmiään ja sovelluksiaan.

IaaS-tyylissä pilvipalvelun tarjoaja on vastuussa vain fyysisen infrastruktuurin ja tietoturvan hallinnasta fyysisellä tasolla. Pilvipalvelun käyttäjä on vastuussa muista osa-alueista (Priyam, 2018). IaaS-mallissa käyttäjä selvittää, kuinka paljon palvelimia, tallennustilaa ja tietokantatehoa hän tarvitsee etukäteen (Miller, 2015).

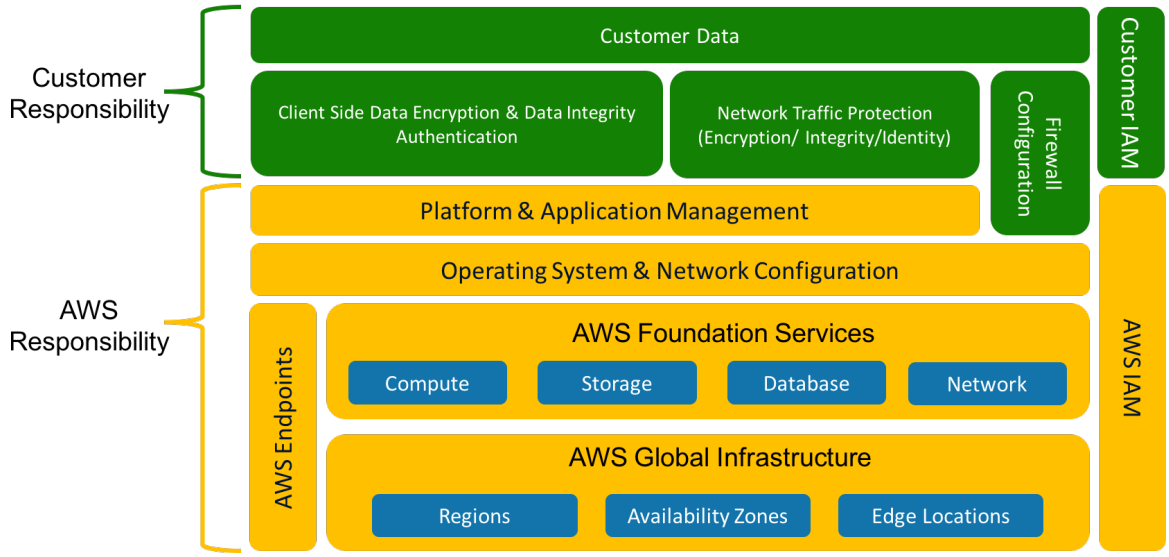
Toisin sanoen, jos loppukäyttäjä eli esimerkiksi organisaatio pystyy ennakoimaan tulevan kuorman etukäteen, voi loppukäyttäjä skaalata ja kutistaa resursseja tarpeen mukaan, mikä vähentää korkeiden, ennakkoon maksettavien investointien tai tarpeettoman ”omistetun” infrastruktuurin tarvetta, erityisesti ”piikikkään” työmäärän tapauksessa (IBM Cloud Education, 2020b).

IaaS-mallista tuli suosittu laskentamalli 2010-luvun alkupuolella ja se on edelleen hyvin yleinen abstraktiomalli monenlaisille työmäärille. Kuitenkin uusien tekniikoiden, kuten konttien ja palvelimettomien tulon, sekä niihin liittyvien mikropalvelusovelluksien nousun myötä voidaan todeta, että vaikka IaaS toimii usein edelleenkin perustana, sille löytyy myös vaihtoehtoisia toteutustapoja. (IBM Cloud Education, 2020)

Konteilla viitataan suoritettaviin ohjelmistoyksikköihin, johon sovelluskoodi on pakattu kirjastojensa ja riippuvuuksiensa tavoin, jotta sitä voidaan käyttää missä tahansa, käyttäjän

työpöydällä, perinteissä konesalissa tai pilvessä. Konteille ominaista on se, että ne ovat pieniä, nopeita ja siirreltäviä, sillä toisin kun perinteisissä virtuaalikoneissa, konteissa ei tarvitse olla vieraskäyttöjärjestelmää kaikissa tapauksissa – ne voivat yksinkertaisesti hyödyntää isäntäkäyttöjärjestelmän ominaisuuksia ja resursseja. (IBM Cloud Education, 2020a)

Konttitekniikan avulla virtualisoidaan käyttöjärjestelmä, jotta yritykset voivat käyttää erillisiä sovelluksia, joilla on erilaiset ja usein yhteen sopimattomat ohjelmistoriippuvuudet (CloudHealth, 2019). Kuvassa 3 on esitelty jaetun vastuun malli konttipalveluiden osalta AWS:ssä.



Kuva 3. Jaetun vastuun malli: Konttipalvelut (Scott, 2017)

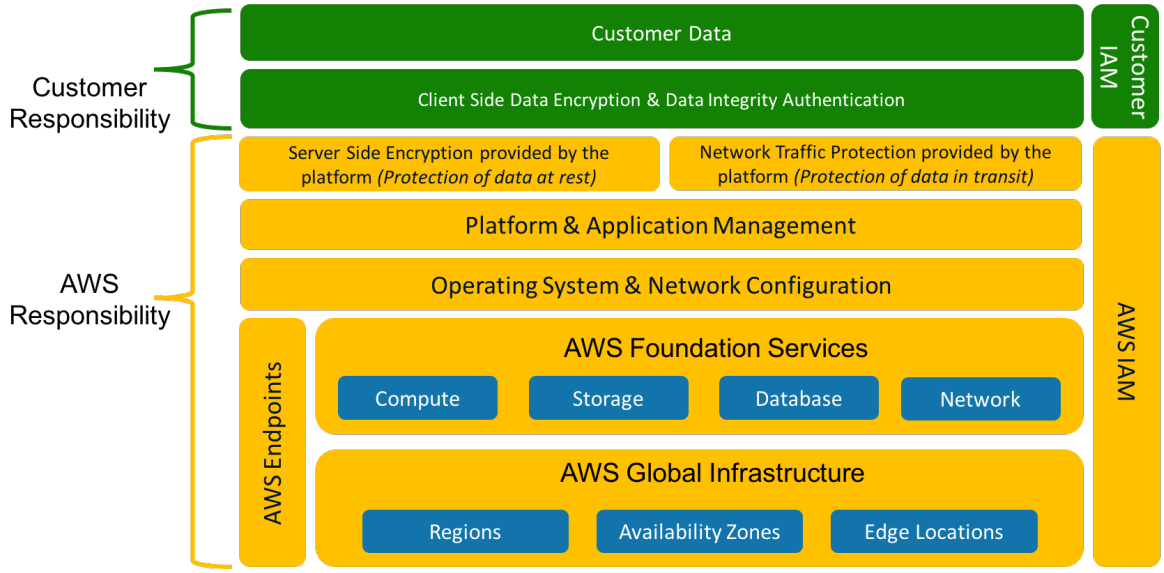
Konttipalvelut toimivat samalla infrastruktuurilla kuin EC2-instanssit, mutta käyttäjä ei hallitse käyttöjärjestelmää tai käyttöympäristöä. Esimerkkejä tällaisista palveluista AWS:ssä ovat Relational Database Service (RDS), Elastic Beanstalk ja Elastic Map Reduce (EMR). Pilvipalveluiden kannalta konttipalvelut ovat Amazonin PaaS (platform-as-a-service) -tarjontaa.

Konttipalveluiden turvallisuusmalli siirtää enemmän vastuuta AWS:n vastuulle. Esimerkiksi jos loppukäyttäjä käyttää Oraclen tietokantaa RDS:än avulla, AWS on vastuussa Oracle-tietokantaohjelmiston ajan tasalla pitämisestä sekä taustalla olevan käyttöjärjestelmän ja EC2-instanssin tietokannoista. Asiakas taas on vastuussa osasta palomuurien määrityksistä ja tietojen salauksesta. Konttipalveluiden turvallisuusmalli vastaa SaaS-pilvipalvelumallia.

AWS tarjoaa myös abstrakteja palveluita, kuten SQS tai S3. S3 on objektien tallennuspalvelu, joka on skaalautuva tietojen saatavuuden, suojauksen ja suorituskyvyn osalta. SQS (Amazon Simple Queue Service) on taas täysin hallittu viestijonopalvelu, jonka avulla voidaan irrottaa ja skaalata mikropalveluita, hajautettuja järjestelmiä ja palvelimettomia sovelluksia (AWS, b).

Palvelimeton sovellus tai tietojenkäsittely on pilvilaskennan malli, jossa pilvipalveluiden tarjoaja jakaa koneresursseja tarvittaessa, huolehtien palvelimesta asiakkaiden puolesta. Palvelimeton tietojenkäsittely ei pidä resursseja haihtuvassa muistissa; tietojenkäsittely tapahtuu pikemminkin lyhyinä purskeina tulosten pysyessä varastoinnissa. Kun sovellusta ei käytetä, sovellukselle ei ole varattu laskentaresursseja. Hinnoittelu perustuu sovelluksen käyttämään todelliseen resurssimäärään (Miller, 2015).

Teknisesti mikään sovellus ei voi olla palvelimeton. Sovelluksen perustana on oltava jonkinlainen laitteisto, mutta pilvipalvelun tarjoaja antaa kehittäjille mahdollisuuden automatisoida ohjelmointi siihen pisteeseen, että AWS huolehtii kaikesta palvelinten käyttöönottoon, tallennukseen ja tietokantaan liittyvistä haasteista. Kuvassa 4 esitellään vaihtoehtoinen abstraktien palvelujen jaetun vastuunmalli.



Kuva 4. Jaetun vastuun malli: Abstraktit palvelut (Scott, 2017)

Kuvan 4 tyylisessä mallissa pilvipalveluiden tarjoaja on vastuussa fyysisen kerroksen, virtualisointikerroksen, verkkotason, tallennustilan, käyttöjärjestelmän ja ohjelmistojen turvallisuudesta (Priyam, 2018). Käyttäjien tai kuluttajien on määriteltävä vain käyttäjätason käyttöoikeus ja salaus, jos niitä voidaan käyttää kyseisessä palvelussa. Abstraktien palveluiden jaetun vastuun malli vastaa SaaS-pilvipalvelumallia.

Myöhemmin esiteltävässä case-esimerkissä hyödynnetään SQS:ää sekä abstraktien palveluiden jaettua mallia sovelluksien osalta. Turvallisuuden varmistamiseksi hyödynnetään virtuaalista yksityistä pilveä, jolloin projekti käyttää tältä osin konttipalveluiden jaetun vastuunmallia eli PaaS tyylistä pilvipalvelumallia.

Edellisten mallien lisäksi on hyvä mainita myös niin sanotut jaetut ohjaimet, joita sovelletaan sekä infrastruktuurikerroksessa että asiakaskerroksissa, mutta täysin erilisissä yhteyksissä tai näkökulmissa. Jaetussa ohjauksessa AWS tarjoaa vaatimukset infrastruktuurille ja asiakkaan on tarjottava oma ohjausratkaisunsa AWS-palvelujensa käytön aikana. Esimerkiksi päivitysten hallinta – AWS kouluttaa AWS:n työntekijöitä, mutta on asiakkaan vastuulla kouluttaa omat työntekijänsä. (AWS,)

IaaS, PaaS ja SaaS tarjoavat siis jokainen progressiivisen abstraktiokerroksen. IaaS tiivistää fyysisen laskennan, verkon, tallennustilan ja tekniikan, joita tarvitaan resurssien virtualisointiin. PaaS menee hivenen pidemmälle ja tiivistää käyttöjärjestelmän, väliohjelmiston ja ajonaikaisen hallinnan. SaaS tarjoaa koko loppukäyttäjäsovelluksen palveluna, jolloin käytännössä loppukäyttäjä on vastuussa vain ohjelmiston käyttämisestä. Oikea pilvipalvelumalli riippuu aina kyseisen organisaation infrastruktuurista ja sovelluksista.

2.5 Pilvimuunnosvaiheet

Kaikkea tietoa ei aina ole mahdollista laittaa pilveen ja tietyissä tapauksissa lait voivat estää tiedon sijoittamisen maan rajojen ulkopuolelle, eikä pilvipalvelut yleensä anna takuuta tiedon sijainnista. Tiedon sijainnilla tarkoitetaan tässä yhteydessä sitä, missä maantieteellisessä sijainnissa organisaation omistamaa tai sen käyttöön tuottamaa tietoa käsitellään ICT-palvelussa tiedon elinkaaren eri vaiheissa (Lausuntopalvelu.fi, 2018). Suomesta katsottuna maantieteelliset alueet ovat yleensä Suomi, EU/ETA-alue sekä muu maailma (Lausuntopalvelu.fi, 2018).

Myös ohjelmistojen lisensointi saattaa olla edullisempaa fyysisesti varatulla palvelimella. Lisenssin siirtämiseen saattaa liittyä myös kustannuksia. Esimerkiksi Oracle-ohjelmien lisensointia varten ”valtuutetussa pilviympäristössä” asiakkaiden on laskettava jokainen virtuaalinen ydin vastaavaksi fyysisen ytimen kanssa (Oracle, 2017).

Ottaen huomioon edelliset asiat, on oleellista suunnitella pilvimuunnosprosessit siten, että ne auttavat organisaatioita siirtämään tietonsa hallitusti pilviympäristöihin (Whitaker, 2020). Kyseinen prosessi voidaan skaalata liiketoiminnan tarpeiden mukaan. Esimerkiksi voidaan siirtää vain tiettyjä sovelluksia ja ohjelmistoja, tiettyjä työpöytiä, tietoja tai koko infrastruktuuri (Whitaker, 2020).

Pilvimuunnos suoritetaan tyypillisesti kolmessa vaiheessa – sovelluksen, verkon ja suojauksen muunnoksena (Whitaker, 2020). Sovelluksen muunnoksen tarkoituksena on varmistaa, että olemassa olevat sovellukset voidaan siirtää uuteen pilviympäristöön. Tämän tavoitteen saavuttamiseksi on useita tapoja, joista jokainen riippuu työn määrästä, joka tarvitaan siirretyn sovelluksen ja sen uuden ympäristön yhteensopivuuden aikaansaamiseksi.

Joissakin tapauksissa sovelluksen siirtäminen ei vaadi paljon työtä. Tällaista siirtymää kutsutaan ”nosta ja siirrä” tyyppiseksi, koska tällöin tarvitsee vain ottaa sovellus ja siirtää se pilveen ilman muutoksia (Whitaker, 2020). Kaikkia sovelluksia ei voida kuitenkaan yksinkertaisesti siirtää ympäristöstä toiseen, jolloin tarvitaan osittaista uudelleen rakentamista. Näissä tapauksissa joudutaan ehkä muokkaamaan sovelluspinon joitain komponentteja, kuten käyttöliittymää, yhteensopivuuden varmistamiseksi uuden pilviympäristön kanssa.

Toisinaan osittainen uudelleenrakentaminen ei riitä yhteensopivuuden varmistamiseksi. Tällöin on tarve refaktoroinnille, joka tunnetaan myös nimellä sovelluksen uudelleen suunnitteluna. Tämän tyyppinen prosessi vaatii suuren osan sovelluksen korjaamista – käyttöliittymästä väliohjelmistoihin, tietojentallentamiseen ja käsittelyyn (Whitaker, 2020).

Verkonmuunnoksen tarkoituksena on siirtää verkkokomponentit pilveen. Perinteisessä ympäristössä verkkojen käyttämiseksi käyttäjien on luotava yhteys yritysverkkoon, joka on tyypillisesti rajattu kehäpohjaiseen palomuuriin. Tämän yhteyden muodostamiseksi käyttäjät tarvitsevat virtuaalista erillisverkkoa – VPN:ää (Virtual Private Network). Nykyään tätä hidasta prosessia pidetään vanhentuneena (Whitaker, 2020). Pilvipalvelut tarjoavat yleensä nopeamman ja yksinkertaisemman yhteyden, mikä johtaa parempaan käyttökokemukseen (Whitaker, 2020).

Suojauksen muutoksen tarkoituksena on varmistaa, että pilvipalveluiden siirtoprosessit ottavat huomioon turvallisuuteen liittyvät haasteet siirtymäkautta ennen, aikana ja sen jälkeen. Eri ympäristöissä vaaditaan usein erilaisia lähestymistapoja turvallisuuteen. Perinteisen paikan päällä toimineet tietoturvakäytännöt eivät yleensä käänny, kun työmäärät siirtyvät pilveen. (Whitaker, 2020)

Tietoturvan varmistamiseksi siirron aikana täytyy siis käyttää tiettyjä käytäntöjä, kuten tietojen salausta levossa olevalle datalle. Tietotekniikassa levossa olevalla datalla tarkoitetaan dataa, joka on fyysisesti tietokoneiden tietovarastossa eli pysyvään muistiin tallennettuja tietoja (TechTarget Contributor, 2010). Lepotiedot sisältävät sekä jäsenneltyä että jäsentämättömiä tietoja (TechTarget Contributor, 2010).

Liikkuva data on yleensä myös syytä suojata, oli sitten kyseessä yksityinen tai julkinen pilvipalvelu. Liikkuvalla datalla tarkoitetaan dataa, joka on aktiivisesti siirtymässä paikasta toiseen, kuten Internetin tai yksityisen verkon kautta (Lord, 2016). Organisaation tulisi kartoittaa tiedonkulku organisaation, pilvipalvelun ja kaikkien asiakkaiden välillä.

Vaikka suurin osa tämän luvun vaiheista on ollut korkean tason asioita, on ennen lopullista päätöstä välttämätöntä selvittää miten tiedot siirtyvät pilven sisällä, sisään ja ulos pilvestä (Archer et al., 2016).

2.6 Jäsennelty ja ei-jäsennelty tieto

Jos päädytään edellisessä kappaleessa esiteltyyn sovelluksen uudelleen suunnitteluun, on tällöin myös hyvä miettiä, millä tavoin uudelleen suunnittelun jälkeen tietoa kannattaisi parhaiten säilöä, jotta parhaimmalla mahdollisella tapaa voidaan noudattaa ja hyödyntää myöhemmin opinnäytetyössä esiteltävän hyvin suunnitellun kehyksen käytänteitä.

Jäsennelty eli strukturoidut tiedot luokitellaan useimmiten kvantitatiivisiksi tiedoiksi. Yleensä kyseessä on tietoa, jotka sopivat siististi relaatiotietokantojen ja laskentataulukoiden kiinteisiin kenttiin ja sarakkeisiin. Esimerkkejä jäsennellyistä tiedoista ovat nimet, päivämäärät, osoitteet, luottokorttien numerot, osaketiedot ja maantieteellinen sijainti. (Pickell, 2018)

Jäsenneltyjen tietojen hallintaan käytettyä ohjelmointikieltä kutsutaan strukturoiduksi kyselykieleksi, joka tunnetaan myös nimellä Structured Query Language - SQL. Kyseisen kielen on alun perin kehittänyt IBM 1970-luvun alussa ja se on erityisen hyödyllinen suhteiden käsittelyyn tietokannoissa. (Pickell, 2018)

Rakenteettomat eli strukturoimattomat tiedot luokitellaan taas useimmiten kvalitatiivisiksi tiedoiksi, eikä niitä voida käsitellä ja analysoida tavanomaisilla datatyökaluilla ja -menetelmillä. Esimerkkejä strukturoimattomista tiedoista ovat teksti, -video ja äänitiedostot, sosiaalisen median viestit, sekä satelliitti- ja valvontakuvat. (Pickell, 2018)

Strukturoimattomia tietoja on lähtökohtaisesti vaikeampi purkaa, sillä niillä ei ole ennalta määriteltyä tietomallia, joten niitä ei voi järjestää relaatiotietokantoihin. Sen sijaan ei-relaatiotietokannat eli NoSQL-tietokannat soveltuvat parhaiten strukturoimattoman tiedon hallintaan. (Pickell, 2018)

Ei-relaatiotietokantoja on ollut olemassa 1960-luvun lopulta lähtien, mutta nimi ”NoSQL” yleistyi käyttöön vasta 2000-luvun alussa. Yksi tärkeimmistä hetkistä NoSQL:n kannalta

tapahtui vuonna 2007, jolloin Amazon esitteli oman hajautetun NoSQL-järjestelmän DynamoDB:n sisäiseen käyttöön. Amazon oli yksi suurimmista yrityksistä, joka alkoi tallentamaan suuren osan tärkeistä yritystiedoistaan ei-relaatiotietokantaan. (Leavitt, 2010)

2.7 Yritysarkkitehtuuri

AWS:ssä ja julkisessa pilvessä voi hyötyä suuresti, kun myös yritysarkkitehtuuria päivitetään ketterämpään suuntaan. Paikallisissa ympäristöissä yrityksillä on usein teknologia-arkkitehtuurin keskusryhmä, joka toimii peittona muille tuote- tai ominaisuusryhmille varmistaakseen, että he noudattavat parhaita käytäntöjä. Teknologia-arkkitehtuuriryhmään kuuluu tyypillisesti joukko rooleja, kuten: tekninen arkkitehti (infrastruktuuri), ratkaisuarkkitehti (ohjelmisto), data-arkkitehti, verkkoarkkitehti ja suojausarkkitehti.

Usein nämä ryhmät käyttävät esimerkiksi TOGAF:ia (The Open Group Architecture Framework) osana yrityksen arkkitehtuurikykyä. Vaikuttaisi siltä, että IT-maailmassa olisi räjähdys arkkitehtonista toimintaa, mutta tosiasiassa laajamittaisten ohjelmistojärjestelmien suunnittelu on tieteenala, joka on ollut käytössä jo pitkään, keskustietokoneista lähtien. (Reselman, 2020)

Open Group julkaisi ensimmäisen version TOGAF:ista vuonna 1995. Nykyinen TOGAF-versio on tämän kirjoituksen hetkellä 9.2. TOGAF määrittely on valtava ja se on jaettu seitsemään osaan ja se sisältää 52 lukua (Reselman, 2020). Lisäksi julkaisussa on liitteitä, jotka kuvaavat termejä, määritelmiä ja lyhenteitä (Reselman, 2020).

TOGAF on suunniteltu erittäin suurille yrityksille, jotka haluavat toteuttaa erittäin suuria yritysarkkitehtuureja. TOGAF:n mukaan rakennetut järjestelmät vaikuttavat yleensä tuhansiin työntekijöihin, eikä ole epätavallista, että yritysarkkitehtuurilla on budjetti, joka alkaa miljoonista dollareista (Reselman, 2020). Näin ollen yksi TOGAF:in keskeinen rajoittava tekijä on hinta ja se saattaa olla este esimerkiksi AWS:n kanssa.

Eräs TOGAF:n rajoituksista on, että se on suunniteltu hierarkkisille ja osastollisille yrityksille. Jos yrityksellä on tasainen hallintahierarkia ja itsenäinen ajatteleva työvoima, kuten pienemmissä yrityksissä, jotka hyödyntävät avoimen lähdekoodin ratkaisuja, ei TOGAF todennäköisesti ole sopiva ratkaisu. (Reselman, 2020)

Toisin kuin TOGAF, AWS jakaa mieluummin valmiudet ryhmiin sen sijaan, että olisi keskitetty tiimi. Kun päätetään jakaa päätöksentekovaltuuksia, on siinäkin riskinsä, kuten varmistuminen siitä, miten tiimit täyttävät sisäiset standardit. Täten jokaisella tiimillä täytyy olla asiantuntijoita, jotka varmistavat, että tiimi pitää tason yllä, joka heidän on täytettävä.

AWS pyrkii myös hyödyntämään automaattisia tarkastus mekanismeja mahdollisimman pitkälle, jotta voidaan varmistua siitä, että määriteltyjä standardeja noudatetaan. (Lester et al., b)

TOGAF:in prosessit ja hinta tekevät siis julkisen pilven hyödyntämisestä haasteellista, sillä pilvessä toimimisesta saa parhaimmat hyödyt, kun organisaatio liikkuu ketterästi ja tekee nopeaa päätöksentekoa.

2.8 Arkkitehtuurin tarkastusprosessi

Arkkitehtuurin tarkistamisprosessia voidaan hyödyntää otettaessa käyttöön hyvin suunnitellun kehyksen käytänteitä. Tarkastamisprosessi on lähtökohtaisesti rakentava keskustelu arkkitehtuuripäätöksistä eikä auditointimekanismi. Arkkitehtuurien tarkistus on tehtävä johdonmukaisella, syyttä vapaalla lähestymistavalla, joka kannustaa sukeltamaan syvälle. Sen on tarkoitus olla kevyt prosessi, joka kestää tunteja, ei päiviä. Arkkitehtuurin tarkistamisen päällimmäisenä tarkoituksena on tunnistaa kriittiset ongelmat, joita on ehkä käsiteltävä tai alueita, joita voitaisiin parantaa. Tarkastelun tulos on joukko toimia, joiden pitäisi parantaa asiakkaan kokemusta työmäärän käytöstä. (Rodney et al., b)

Työmäärällä (workload) tarkoitetaan tässä opinnäytetyössä ja AWS:ssä joukkoa komponentteja, jotka yhdessä tuottavat liiketoiminnalle arvoa. Esimerkkejä työmääristä ovat markkinointisivustot, sähköisen kaupankäynnin verkkosivustot, mobiilisovelluksen taustat ja analyyttiset alustat (Rodney et al., c). Työmäärät vaihtelevat arkkitehtonisen monimutkaisuuden mukaan staattisista verkkosivustoista arkkitehtuureihin, joissa on useita tietovarastoja ja monia komponentteja (Rodney et al.,). Lisäksi työmäärät voivat koostua resurssien alijoukosta yhdessä AWS-tilissä tai olla kokoelma useita resursseja, jotka kattavat useita resursseja useilta AWS-tileiltä. Pienellä yrityksellä voi olla vain muutama työmäärä, kun taas suurella yrityksellä voi olla tuhansia (Rodney et al.,).

Virstanpylväillä tarkoitetaan tärkeimpiä muutoksia arkkitehtuurissa sen kehittyessä koko tuotteen elinkaaren ajan (suunnittelu, testaus, käyttöönotto ja tuotanto). Virstanpylväät voivat keskittyä tuoteominaisuuksien julkaisemiseen, suuriin virhekorjauksiin tai arkkitehtuurin muuttamiseen (Rodney et al., a).

Arkkitehtuurin arviointeja tulisi soveltaa tuotteen elinkaaren tärkeimpiin virstanpylväisiin jo suunnitteluvaiheessa yksisuuntaisten ovien välttämiseksi. Yksisuuntaisilla ovilla tarkoitetaan päätöksiä, joita on vaikea tai mahdotonta peruuttaa ja ne vaativat enemmän tarkkai-

vasuutta ennen niiden tekemistä (Rodney et al.,). Tuotantoon siirtymisen jälkeen, työmäärät jatkavat kehittymistä samalla kun lisätään uusia ominaisuuksia ja muutetaan tekniikan toteutusta (Rodney et al.,).

2.9 Palvelutasosopimukset pilvessä

Yrityksiä saattaa houkutella pilvipalveluiden edut, kuten nopea käyttöönotto, joustava skaalautuvuus ja alhaiset käynnistyskustannukset. Yrityksillä on kuitenkin oikeus olla samanaikaisesti huolissaan pilvipalveluun liittyvistä riskeistä, kuten tietojen sijainnista, palvelutasosta ja suojausinfrastruktuuriin liittyvistä riskeistä. Gartner kuvaa pilvipalvelua ”laskentatyylinä, jossa skaalautuvat ja joustavat IT-valmiudet tarjotaan palveluna Internet-tekniikkaa käyttäville kuluttajille” (Trappler, 2010).

Avainsana tässä määritelmässä on ”palvelu”, koska pilvipalvelusopimuksen monia ulottuvuuksia säätelee sopimus, jossa on nimenomaisesti ilmoitettava odotetut palvelutasosopimukset (Service-level agreement) (Trappler, 2010). Tältä osin sopimuksen on oltava selvä odotusten suhteen, kuten käytettävyysajan ja seisokkien, sekä vasteajan, kun asiat menevät pieleen infrastruktuurin ja turvallisuuden osalta (Trappler, 2010).

Sopimukset säännöllisesti heijastavat odotusta 99,9 prosentin käyttöajasta. Vaikka tämä kuulostaa hyvältä pinnalta (tasolta), täytyy yrityksen silti kysyä itseltään, täyttääkö tämä käyttöajan määrä yrityksen erityistarpeita. Sopimusneuvottelijoiden on ymmärrettävä seisokkien erityisparametrit.

Esimerkiksi suunnitellaanko mahdolliset seisokit aikaan, joka parhaiten sopii yrityksen liiketoiminnan tarpeisiin. Kattaako sovitut seisokkien parametrit olosuhteet, joita asiakas tai myyjä ei voi valvoa. Entä suunnittelun huollon aiheuttamat seisokit. Tällaiset kysymykset korostavat, että on tärkeää nähdä vaivaa termien huolelliseen ja mahdollisimman täydelliseen määrittelemiseen SLA:ssa.

3 Hyvin suunniteltu AWS-kehys

Hyvin suunniteltu AWS-kehys (Well-Architected Framework) on AWS:n käyttämä termi, joka pyrkii auttamaan yrityksiä ymmärtämään etuja ja riskejä päätöksissä, joita tehdään samalla kun rakennetaan työmäärää AWS:ään .

Käyttämällä kehystä opitaan suunnittelemaan yritykselleen itselleen sopivimmat toimintatavat ja arkkitehtuurit siihen, miten luodaan luotettavat, turvalliset ja kustannustehokkaat kuormitukset pilvessä. Kehys tarjoaa tavan mitata toimintaa ja arkkitehtuuria johdonmukaisesti parhaiden käytäntöjen mukaisesti ja tunnistaa parannettavat alueet (Lester et al., a).

Tässä kappaleessa keskitytään tutkimaan kehyksen viittä pilaria: Operatiiviseen huippuosaamista, turvallisuutta, luotettavuutta, suorituskyvyn tehokkuutta ja kustannusten optimointia. Pohjimmiltaan kaikki hyvin suunnitellut käytännöt perustuvat näiden perustavien pilarien ympärille.

Kehys tarjoaa lisäksi yhdenmukaisen lähestymistavan järjestelmien arviointiin moderneista pilvipohjaisista järjestelmistä ja niihin liittyvistä ominaisuuksista (Lester et al.,). Kappaleessa esitellään muutama käytännön esimerkki teorian tukemiseksi.

3.1 Operatiivinen huippuosaaminen

Operatiivisella huippuosaamisella tarkoitetaan kykyä tukea kehitystä ja samalla hoitaa tehokkaasti kuormitusta, saada tietoa eri toiminnoista sekä parantaa jatkuvasti tukiprosesseja ja menettelyjä liiketoiminnan arvon tuottamiseksi.

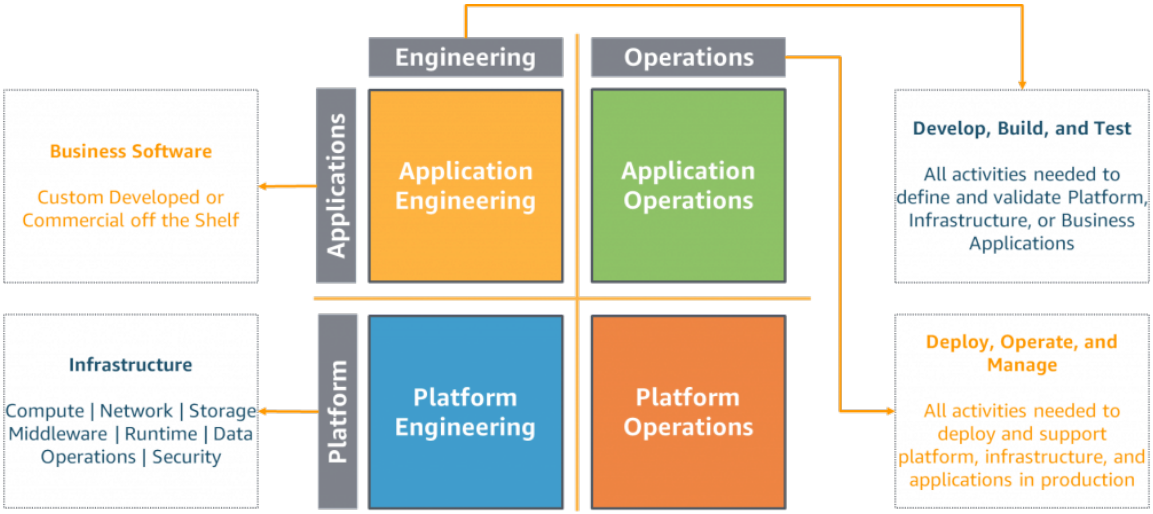
Operatiivinen huippuosaaminen on haastavaa saavuttaa ympäristöissä, joissa operatiivinen toiminto on eristetty toiminto ja erillään sen tukemista liiketoiminta- ja kehitysryhmistä. AWS uskoo, että hyvin suunnitellut työmäärät, jotka on suunniteltu operatiivisessa mielessä lisää liiketoiminnan onnistumisen todennäköisyyttä.

Pilari myös kertoo, että kehittäjien ja operatiivisten ryhmien tulisi sovelluksia luodessaan etsiä tietoja liiketoiminnasta ja asiakkaista, jotka tukisivat tehokkaasti tuotannon kuormitusta. Tämä tarkoittaa ymmärrystä riskeistä, mahdollisista muutoksista sekä tavoitteista, jotka ohjaavat päivittäistä liiketoimintaa.

“No matter how it looks at first, it’s always a people problem” (Weinberg, 1986)

Opiskelija arvostaa huumoria ja nokkeluutta, jolla Weinberg lähestyi ohjelmistokehitystä, sillä organisaatiot, ryhmät ja tiimin jäsenet ovat aina merkittävässä osassa operatiivisen huippuosaamisen pilaria. Riippumatta siitä mitä menetelmiä käytetään – viime kädessä kaikki suorittavat samoja toimintoja.

Merkittävin yksittäinen ero on siinä, miten organisoidaan vastuita ja tiimejä. Yksilöiden ja tiimien välisten suhteiden ymmärtäminen on välttämätöntä työn sujuvuuden kannalta (Carlson, 2020). Kuvassa 5 on havainnollistettu organisaatio yksinkertaistetusta näkökulmasta, joka rajoittuu sekä sovellusten että infrastruktuurin suunniteluun ja toimintaan.



Kuva 5. Yksinkertaistettu kaavio organisaation osa-alueista (Carlson, 2020)

Käyttämällä kuvan 5 muunnelmia, voivat vastuut ulottua useampaan neljännekseen, jolloin myös käydään läpi tiimien, hallinnon ja päätöksenteon välisiä suhteita. Hyvin määritellyt vastuualueet vähentävät ristiriitaisia ja tarpeettomia ponnisteluja. Tällöin liiketoiminnan tulokset on helpompi saavuttaa, kun liiketoiminta- kehitys- ja operatiivisten ryhmien välillä on vahva linjaus ja suhteet. (Carlson, 2020)

Yrityksessä tiimit saattavat käyttää erilaisia toimintamalleja kykyjensä ja tarpeidensa mukaan ja yrityksellä on myös mahdollista menestyä minkä tahansa toimintamallin kanssa. Joillakin toimintamalleilla on kuitenkin etuja tai ne sopivat paremmin yksittäisille tiimeille. Kartoittamalla mitä kukakin tekee tiimissä, on mahdollista saada merkittävä käsitys siitä, miten heidän tekemisensä vaikuttavat työhön tai sen etenemiseen. Tämän tyyppiset oivallukset voivat johtaa parannusmahdollisuuksien tunnistamiseen sekä mahdollisuuksiin hyödyntää tiimein nykyisiä valmiuksia ja innovaatiota entisestään. (Carlson, 2020)

Operatiivisen huippuosaamisen pilari sisältää siis sen, miten organisaatio tukee yrityksen liiketoiminnallisia tavoitteita ja kykyä hoitaa kuormitusta tehokkaasti.

3.2 Operatiiviset suunnitteluperiaatteet

Työmäärän toimintaan voi liittyä riskejä. Yrityksen on suositeltavaa ymmärtää nämä riskit ja tehtävä tietoinen päätös tuotannon aloittamisesta. AWS:n hyvin suunnitellun kehyksen viiden operatiivisen suunnitteluperiaatteiden avulla pystytään ennakoimaan mahdollisia riskejä paremmin (Carlson et al., 2020).

Ensimmäiseksi suoritetaan operatiivinen toiminnallisuus koodina. Pilvessä on mahdollista hyödyntää samoja tekniikoita kuin muussakin sovelluskoodissa koko ympäristössä. Kaikki työ (sovellukset, infrastruktuurit, jne.) kannattaa määritellä koodiksi ja niitä täytyy voida päivittää koodilla.

Tällaista tapaa kutsutaan myös infrastruktuuria koodina (Infrastructure as Code). Kyseessä on lähestymistapa infrastruktuurin automatisointiin, joka perustuu ohjelmistokehityksen käytäntöihin. Siinä korostetaan johdonmukaisia, toistettavia rutiineja järjestelmien ja niiden kokoonpanojen hallintaa ja muuttamista varten (Morris, 2020).

Jokaisella infrastruktuurityökalulla on eri nimi lähdekoodilleen, mutta tässä opinnäytetyössä kirjoittaja viittaa niihin tekstissään yleisessä mielessä infrastruktuurikoodina tai infrastruktuurin määritelmänä. Infrastruktuuri koodina tarkoittaa määritelmän mukaan infrastruktuurin määrittelemistä tekstipohjaisissa tiedostoissa (Morris, 2020). Käyttäjät voivat lukea, muokata ja analysoida määrityksiä haluamallaan työkalullaan (Morris, 2020).

Järjestelmänvalvojat ovat käyttäneet komentosarjoja infrastruktuurin hallintatehtävien automatisoimiseksi vuosikymmenien ajan. Yleiskäyttöiset komentosarjakielet kuten Bash, Perl, PowerShell, Ruby ja Python ovat edelleen osa monen infrastruktuuritiimin työkalupakettia (Morris, 2020)

Pino-suuntautuneet työkalut, kuten AWS CloudFormation käyttää deklaratiivista toimialakohtaista kieltä – DSL (Domain-specific language). Deklaratiiviset kielet ovat erikoistuneita tietylle sovellusalueelle ja ne yksinkertaistavat infrastruktuurikoodia erottamalla infrastruktuurin määritelmän sen toteuttamisesta. (Morris, 2020)

Pinolla (stack) tarkoitetaan tässä yhteydessä kokoelmaa AWS-resursseja, joita voi hallita yhtenä yksikkönä. Toisin sanoen on mahdollista luoda, päivittää tai poistaa resurssikokoelmia, luomalla, päivittämällä tai poistamalla pinoja. Kaikki pinoon liittyvät resurssit määritetään CloudFormation -mallin avulla. Esimerkiksi pino voi sisältää kaikki verkkosovelluk-

sen suorittamiseen tarvittavat resurssit, kuten verkkopalvelimen, tietokannan ja verkkosäännöt. Kun kyseistä verkkosovellusta ei enää tarvita, voidaan yksinkertaistesti poistaa pino, jolloin kaikki siihen liittyvät resurssitkin poistetaan. (AWS, g)

CloudFormation varmistaa, että kaikki pinon resurssit luodaan tai poistetaan tarvittaessa. CloudFormation käsittelee pinoresursseja yhtenä yksikkönä, jolloin kaikki resurssit on joko luotava tai poistettava, jotta pino voidaan luoda tai poistaa. Jos resurssia ei voida luoda, CloudFormation palauttaa pinon takaisin ja poistaa kaikki luodut resurssit. Jos resurssia ei voida poistaa, kaikki jäljellä olevat resurssit säilytetään, kunnes pino voidaan taas poistaa. (AWS,)

AWS CloudFormation on siis malli, joka on ilmoitus tai listaus AWS:n resursseista. Malli tallennetaan tekstitiedostona, jonka muoto on JavaScript Object Notation (JSON) - tai YAML-standardin (YAML Ain't Markup Language) mukainen. YAML ja JSON ovat pohjimmiltaan tekstitiedostoja, niintä voidaan luoda ja muokata missä tahansa tekstieditorissa ja niitä voidaan hallita lähdekoodijärjestelmässä sovelluksen muun lähdekoodin kanssa.

Kuvassa 6 on esimerkki JSON muodossa olevasta AWS:n resurssista. Oheisen tiedon avulla AWS CloudFormation osaa tehdä tarvittavat toimenpiteet Amazon S3 bucketin luomiseksi. AWS CloudFormationin avulla voidaan siis luoda ja varata AWS-infrastruktuurin käyttöönottoja ennakoivasti ja toistuvasti.

```
{
   "Resources" : {
      "HelloWorldBucket" : {
         "Type" : "AWS::S3::Bucket"
      }
   }
}
```

Kuva 6. Esimerkki HelloWorldBucket nimisestä S3 Bucket resurssin luonnista

Kuvassa 7 on esitelty samanlainen HelloWorldBucket niminen S3 Bucket resurssi luotuna YAML-muodossa.

```
Resources:
  HelloWorldBucket:
    Type: AWS::S3::Bucket
```

Kuva 7. HelloWorldBucket S3 bucket luotuna YAML muodossa

Viime aikoina on noussut trendi uusista infrastruktuurityökaluista, jotka käyttävät olemassa olevia yleiskäyttöisiä ohjelmointikieliä infrastruktuurin määrittelemiseksi, kuten AWS CDK. Eräs yleiskäyttöisen ohjelmointikielen, kuten JavaScriptin, Pythonin, Rubyn tai TypeScriptin käytön suurimmista eduista on työkalujen ekosysteemit. Kehittimet tukevat näitä kieliä hyvin ja niillä on tehokkaat tuottavuusominaisuudet, kuten syntaksin korostus ja koodin korjaus. Testaaminen on myös hyödyllinen osa yleiskäyttöisen ohjelmointikielen ekosysteemiä. (Morris, 2020)

AWS Cloud Development Kit (AWS CDK) on avoimen lähdekoodin ohjelmistokehitysprojekti, joka määrittää pilvi-infrastruktuurin koodina ja provisioi sen AWS CloudFormationin kautta. CDK tarjoaa korkean tason objektisuuntautuneen abstraktion AWS-resurssien määrittelemiseksi käyttämällä nykyaikaisia ohjelmointikieliä. Kuvassa 7 on esitelty tapa luoda samanlainen S3 bucket kuin kuvassa 8 käyttämällä TypeScript ohjelmointikieltä ja AWS CDK:ta.

```
export class MyS3ConstructStack extends core.Stack {
  constructor(scope: core.App, id: string, props?: core.StackProps)
{
    super(scope, id, props);

    const bucket = new s3.Bucket(this, "MyVpc", {
      name: " HelloWorldBucket"
    });
  }
}
```

Kuva 8. CDK:n tapa esittää HelloWorldBucket nimisen S3 Bucketin luominen

Operatiivisia toimintoja voidaan myös komentaa ja automatisoida käynnistämällä niiden suorittamisia vastauksina tapahtumiin. Tapahtumalla voidaan tarkoittaa toimintaa, tapahtumaa tai tilan muutoksesta tehtyä havaintoa. Tapahtumat voivat olla suunnitelmallisia tai suunnittelemattomia ja ne voivat syntyä sisäisestä tai ulkoisesta työmäärästä.

Esimerkiksi vähittäiskauppasivuston mainostaminen on suunniteltu tapahtuma, joka syntyy työmäärän ulkopuolella. Toisaalta komponenttivika on suunnittelematon tapahtuma, joka syntyjään johtuu sisäisestä työmäärästä. Tapahtumia, jotka edellyttävät vastausta, kutsutaan vaaratilanteiksi. Suorittamalla operatiivista toimintaa samalla tavoin kuin sovelluskoodia, rajoitetaan inhimillisiä virheitä ja tehdään johdonmukaisia vastauksia tapahtumiin. (Carlson et al., 2020)

Toisena suunnitteluperiaatteena on järkevää tehdä usein, pieniä, palautuvia muutoksia eli toimia toistettavuusperiaatteen mukaisesti. Työmäärät kannattaa suunnitella niin, että komponentteja voidaan päivittää säännöllisesti hyödyllisten muutosten virran lisäämiseksi. Muutoksia tehdään pienin askelin, jotka voidaan kumota, jos ne eivät auta tunnistamaan ja ratkaisemaan esille tuotuja asioita pilviympäristössä (vaikuttamatta asiakkaisiin mahdollisuuksien mukaan). (Carlson et al., 2020)

Toistettavuusperiaatteen pohjalta käyttäjän tulisi pystyä toistamaan kaikki tekemänsä asiat infrastruktuurilleen. Toimintoja on helpompi toistaa komentosarjojen ja kokoonpanohallintatyökalujen avulla kuin tehdä se käsin. (Morris, 2020)

Oletetaan esimerkiksi, että palvelimella on kiintolevy, joka täytyy osioida eli jakaa loogisiin osiin kertaluontoisena tehtävänä. Komentosarjan kirjoittaminen ja testaaminen vaatii paljon enemmän työtä kuin vain kirjautua sisään ja fdisk-komennon suorittaminen, joten käyttäjä tekee sen käsin. Ongelma tulee myöhemmin, kun jonkun toisen täytyy osioida levy. Hän tulee samaan johtopäätökseen kuin aikaisempi käyttäjä ja tekee työn käsin sen sijaan, että kirjoittaisi komentosarjan. Ensimmäinen käyttäjä teki erikokoisen osion eri tiedostojärjestelmällä kuin toinen käyttäjä. Näin mahdollistetaan ja kasvatetaan kokoonpanon ajautumista, jotka entisestään heikentävät kykyä automatisoida luotettavasti. (Morris, 2020)

Kolmantena suunnitteluperiaatteena on suositeltua tarkentaa toimintamenettelyjä usein: Kun käytetään toimintamenetelmiä, on järkevää etsiä mahdollisuuksia parantaa niitä. Kun työmääriä kehitetään, on suositeltua kehittää myös proseduureja. Säännöllisten retrospektiivien määrittely toimintamenettelyjen tarkistamiseksi ja vahvistamiseksi, jotta kaikki menettelyt ovat tehokkaita ja että tiimit tuntevat ne. (Carlson et al., 2020)

Neljäntenä suunnitteluperiaatteena on epäonnistumisien ennakointi: Suoritetaan ”pre-mortem” harjoituksia potentiaalisten vikalähteiden tunnistamiseksi, jotta niitä voidaan poistaa tai lieventää. ”pre-mortem” on tilanne, jossa tiimi kuvittelee projektin epäonnistumisen ennen kuin niin on vielä tapahtunut. Vikatilanteiden testaus vahvistaa ymmärrystä niiden vaikutuksista ja vastausmenettelyillä varmistutaan, että ne ovat tehokkaita ja että tiimit tuntevat, miten niitä suoritetaan. (Carlson et al., 2020)

Viides suunnitteluperiaate on oppiminen kaikista toimintahäiriöistä: On oleellista tehdä parannuksia kaikista opeista, joita on saatu operatiivisista tapahtumista ja epäonnistumisista. Oppi täytyy jakaa kaikissa tiimeissä ja koko organisaatiossa. (Carlson et al., 2020)

3.3 Tietokantamigraatiot

Case-esimerkki projektin alku vaiheella kohdattiin tilanne, jossa projektitiimillä ei ollut tarkkaa tietoa tietokantaskeemasta. Tilanne aiheutti haasteita varsinkin silloin kun ohjelmiston kehityksen aikana tuli vastaan tilanteita, joissa tietokantaskeemaa täytyi muuttaa. Tilanteen ratkaisemiseksi hyödynnettiin edellisessä kappaleessa läpikäytyä mentaliteettia – Kaikki kannattaa määritellä koodiksi ja niitä täytyy voida päivittää koodilla. Tässä tapauksessa myös tietokantamigraatiot on syytä voida päivittää koodilla.

Tietokantamigraatiot toteutetaan yleensä niin, että tietokannassa pidetään yllä tietokantataulujen muutos- ja muokkauskomentoja sisältäviä versiokohtaisia tiedostoja (Helsingin Yliopisto, 2019). Myös projektissa käytetty Sequelize kirjasto toimii näin.

Tietokantamigraatioiden keskeinen ajatus liittyy versionhallintaan. Tietokantamigraatiot seuraavat muutoksia tietokantamalliin (ja joskus myös tietoihin (Amanda, 2018). Oheiset muutokset heijastuvat tyypillisesti erillisinä komentotiedostoina ja tällä tavoin skeemamuutokset näkyvät lähdekoodina, joka voidaan ottaa talteen millä tahansa versionhallintaohjelmistolla (Amanda, 2018). Kuvassa 9 esimerkki tietokantamigraatio tiedostosta Sequelize kirjastoa hyödyntäen.

```
export const up: Migration = async ({ context: sequelize }) => {
  await sequelize.getQueryInterface().createTable('Colors', {
    id: {
      type: DataTypes.UUID,
      allowNull: false,
      autoIncrement: false,
      primaryKey: true,
    },
    color: {
      type: DataTypes.STRING
    },
    createdAt: {
      allowNull: false,
      type: DataTypes.DATE
    },
    updatedAt: {
      allowNull: false,
      type: DataTypes.DATE
    }
  });
};

export const down: Migration = async ({ context: sequelize }) => {
  await sequelize.getQueryInterface().dropTable('Colors');
};
```

Kuva 9. Esimerkki case-esimerkin tietokanta migraatiosta

3.4 Turvallisuus

19. heinäkuuta 2019 amerikkalaisen pankin Capital Onen arkaluontoisia tietoja päätyi ulkopuoliselle taholle. Yli 106 miljoona ihmistä kärsi tavalla tai toisella. 140 000 sosiaaliturvatunnusta, 80 000 pankkitilinumeroa ja 1 000 000 sosiaalivakuutusnumeroa (Capital One, 2019). Capital One:lla oli noin tapahtuman aikana noin 700 erilaista S3 buckettia, joiden sisällön tunkeutuja kopioi ja latasi itselleen (Morrison, 2019). Capital One joutui maksamaan 80 miljoonaa dollaria Yhdysvaltain sääntelyviranomaisille (Barrett, 2020). Capital One -hakkerointi oli yksi suurimmista tietoturvaloukkauksista, joka on koskaan kohdannut rahoituspalveluyritystä (Barrett, 2020).

Oletusasetuksilla Amazon S3 bucketit ovat yksityisiä, ja niitä voivat käyttää vain henkilöt, joille on nimenomaisesti myönnetty pääsy niiden sisältöön (Scroxton, 2020). Niiden jatkuva altistuminen tietoturvaloukkauksille viittaa siihen, että johdonmukainen viestintä pilvipalvelujen suojauskäytännön, toteutuksen ja kokoonpanon ympärillä ei onnistuta saavuttaman monien IT-ammattilaisen toimesta (Scroxton, 2020).

Capital Onen tapauksessa tunkeutuja pääsi sisään palvelimelle (EC2-instanssi) palvelimen palomuurien aukon kautta (Morrison, 2019). Sisäänpääsyn syynä oli siis joko väärin asetettu suojausryhmä, ACL (Access Control List), oma mukautettu verkkosovellusten palomuuri tai palvelinpuolen pyyntöjen väärentäminen (Morrison, 2019).

AWS:n suojausryhmät toimivat instanssitason palomuurina, jolla on säännöt, jotka suodattavat liikennettä instanssiin ja sieltä pois. Ne toimivat protokolla- ja porttitasolla, rajoittamalla lähdeliikennettä IP- ja suojausryhmätasolla. Tämän avulla on mahdollista myöntää pääsy instansseille käyttämällä määritettyjä protokollia ja porttinumeroita, avaamalla pääsyn vain yhdestä IP-osoitteesta (x.x.x.x/32), mistä päin maailmaa tahansa (0.0.0.0/0) tai toisen, ennalta määritetyn suojausryhmän kautta. (Scott, 2017). Jokainen suojausryhmä – joka toimii siis palomuurin tavoin – sisältää joukon sääntöjä, jotka suodattavat liikenteen, joka tulee EC2-instanssille ja sieltä pois (Scott, 2019).

Internet-protokollan osoite eli IP-osoite on numeerinen tunniste, joka on osoitettu jokaiselle laitteelle, joka on kytketty tietoverkkoon ja käyttää Internet-protokollaa viestintään (Postel, 1980). IP-osoitteet ovat pohjimmiltaan siis tunnisteita, joiden avulla tietoja voidaan lähettää verkon laitteiden välillä: ne sisältävät sijaintitietoja ja tekevät laitteista tiedonsiirtoyhteyksiä (kaspersky, 2021). Internet tarvitsee tavan erottaa tietokoneet, reitittimet ja verkkosivut (kaspersky, 2021). IP-osoitteet tarjoavat tavan tehdä niin ja muodostavat olennaisen osan Internetin toiminnasta.

IP-osoite on pisteillä erotettu numerosarja ja ne ilmaistaan neljän numeron joukkoina – esimerkkiosoite voi olla 192.158.1.38. Jokainen joukko voi vaihdella välillä 0 - 255. Täten koko IP-osoitealue vaihtelee normaalisti välillä 0.0.0.0 - 255.255.255.255. (kaspersky, 2021)

Suojausryhmien kanssa täytyy laittaa täytäntöön vähiten oikeuksien sääntö, kun on kyseessä sääntöjen suunnittelusta ja toteuttamisesta suojausryhmässä (Scott, 2019). Tavoitteena on sallia vain tarvittavat käyttöoikeudet, eikä antaa liian sallittua pääsyä EC2-instanssille, koska se saattaa johtaa mahdollisiin tietoturvaloukkauksiin ja haavoittuvuuksiin (Scott, 2019).

Turvallisuuden varmistamiseksi virtuaalisella yksityisellä pilvi (VPC) -aliverkkotasolla voidaan ottaa erikseen käyttöön NACL-luettelot (Network Access Control Lists). Virtuaalisella yksityisellä pilvellä tarkoitetaan potentiaalisesti turvallista, eristettyä yksityistä pilveä, jota isännöidään julkisessa pilvessä (Cloudflare,).

NACL on samanlainen kuin suojausryhmät, koska se koostuu säännöistä, mutta se valvoo liikennettä aliverkon tasolla. On myös hyvä huomata, että turvallisuusryhmät ovat tilallisia, kun taas NACL:t ovat tilattomia. Tämä tarkoittaa sitä, että NACL:lle on määriteltävä säännöt sekä saapuvalle että lähtevälle liikenteelle (Scott, 2017).

Verkkosuunnittelun huolellisuus ja hallinta ovat perusteita sille, miten organisaatio tarjoaa eristämisen ja rajat resursseille työmäärän sisällä. Monet työmäärän resurssit toimivat VPC:ssä ja perivät sille asetetut suojausominaisuudet, joten on kriittistä, että suunnittelua tuetaan automaation tukemilla tarkastus- ja suojausmekanismeilla. Vastaavasti työkuormille, jotka toimivat VPC:n ulkopuolella – esimerkiksi palvelimettomat sovellukset – parhaita käytänteitä sovelletaan hyödyntämällä kappaleessa 2.4 esiteltyä abstraktien palvelujen jaetun vastuun mallia.

Verkkokerroksia suunnitellessa komponentit kuten EC2-instanssit, RDS-tietokantaklusterit ja Lambda-toiminnot, jotka jakavat samat saavutettavuusvaatimukset, voidaan ryhmitellä aliverkkojen muodostamiksi tasoiksi. Esimerkiksi RDS-tietokantaklusteri VPC:ssä, jossa ei tarvita Internet-yhteyttä, tulisi sijoittaa aliverkkoihin ilman reittiä sisään tai ulos Internetiin. Case-esimerkissä verkkokerrokset ja aliverkot suunniteltiin niin, että ainoastaan tarvittavilla palveluilla oli yhteys Internettiin.

Capital Onen tapauksessa palomuurin tai reitin sulkeminen oli ilmeisen helppo sulkea, kuten virallisessa tiedotteessa todettiin (Capital One, 2019). Todellinen ongelma oli palvelimelle annettu IAM-rooli. Kyseinen IAM-rooli salli pääsyn Capital Onen kaikkiin yli 700 S3 buckettiin (Morrison, 2019).

IAM on peräisin sanoista Identity and Access Management. Sillä on kriittinen rooli pilvipalveluiden tietoturvan toteuttamisessa (Priyam, 2018). AWS:ssä IAM tarjotaan globaalina palveluna. Tässä yhteydessä globaali tarkoittaa, että IAM:n käyttöalue on globaali; se on määritelty kerran ja sitä voidaan käyttää kaikilla AWS-alueilla ja se ei ole aluekohtainen palvelu (Priyam, 2018). Lisäksi AWS:ssä IAM tulee ilmaiseksi; siihen ei liity kustannuksia.

Verizonin vuoden 2019 DBIR:n – Data Breach Investigations Report (Tietovuotoloukkauksia koskeva selvitys) – mukaan etuoikeuksien väärinkäyttö aiheuttaa lähes 70 % tietovuodoista ja 29 % kaikista rikkomuksista liittyy varastettuihin käyttöoikeuksiin (Verizon,).

AWS:ssä vähiten oikeuksien käsite tarkoittaa, että käyttäjille annetaan mahdollisimman vähän käyttöoikeuksia ja vastuuta heidän tehtävien suorittamiseen. Vähiten etuoikeuksia voidaan kutsua myös roolipohjaiseksi käyttöoikeudeksi tai tietotarpeeksi. Ne kuuluvat AWS:n identiteetin ja käyttöoikeuksien hallinnan (IAM) käytäntöihin.

Järjestelmän määrittäminen vähäisimpien oikeuksien ja tietotarpeen periaatteiden perusteella eivät ole uusia käsitteitä. Etuoikeuksien hallinta, väärinkäyttö sekä identiteetin- ja käyttöoikeuksien hallinnan väärät määritykset olivat yksi suurimmista virheistä Capital Onen tietovuodossa.

Turvallisuuspilari kuvaa siis sitä, miten pilvitekniikoiden avulla voidaan suojata tietoja, järjestelmiä ja omaisuutta tavoilla, jotka parantavat tietoturvallisuutta. Tietoturva ja vaatimustenmukaisuutta kutsutaan jaetun vastuun malliksi AWS:n ja asiakkaan välillä. Jaetun mallin tavoitteena on auttaa organisaatiota vähentämään operatiivista taakkaa ja ne ovat tarkemmin esitelty kappaleessa 2.4.

3.5 Luotettavuus

Luotettavuus-pilarilla kuvataan työmäärän kykyä suorittaa aiottu toiminto oikein ja johdonmukaisesti. Tähän sisältyy kyky käyttää ja testata työmäärää koko sen elinkaaren ajan. Valvomalla keskeisten suorituskykyindikaattorien – key performance indicator (KPI) - työmäärää voidaan käynnistää automaatiotoimenpiteitä, kun tietty kynnysarvo ylittyy.

Suorituskykyindikaattorien pitäisi olla liiketoiminnan arvon mittari, eikä palvelun toiminnan teknisten näkökohtien mittari. Korkean tason suorituskykyindikaattorit saattavat keskittyä liiketoiminnan yleiseen suorituskykyyn, kun taas matalan tason indikaattorit voivat keskittyä osastojen prosesseihin, kuten myyntiin, markkinointiin, henkilöstöhallintoon, tukeen tai muuhun vastaavaan (Klipfolio,).

Esimerkiksi API-pohjainen työmäärä saattaa käyttää yleistä vastauksen viivettä osoituksena kokonaistehokkuudesta ja verkkokauppasivusto voisi käyttää ostojen määrää KPI:na. KPI:n valvominen mahdollistaa automaattisten ilmoituksen ja vikojen seurannan, jotka kiertävät tai korjaavat vian. Kehittyneemmällä automaatiolla on mahdollista ennakoida ja korjata viat ennen niiden esiintymistä. Suorituskykyä ja siihen liittyvää tehokkuutta käsitellään tarkemmin kappaleessa 3.7.

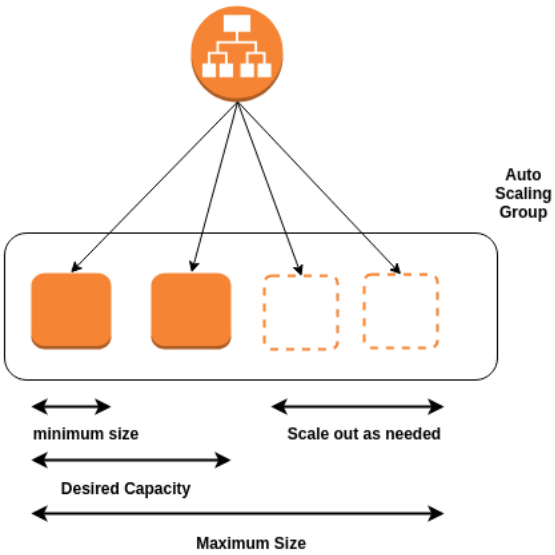
Skaalaaminen vaakasunnassa lisää kokonaiskuormituksen saatavuutta. Yhden suuren resurssin korvaaminen useilla pienillä resursseilla auttaa vähentämään yksittäisen vian vaikutusta kokonaiskuormituksen. Pyyntöjen jakaminen useille pienemille resursseille varmistaa sen, että resursseilla ei ole yhteistä epäonnistumispistettä.

Vaakasuuntainen skaalaus on melkein aina toivottavampaa kuin pystysuuntainen skaalaus, koska siitä ei seuraa resurssivajetta. Sen sijaan, että palvelin olisi hetken offline-tilassa, kun se skaalautuu pystysuunnassa, vaakasuuntainen skaalaus mahdollistaa nykyisen tietojenkäsittelyresurssien pitämisen päällä ja lisätä enemmän jo olemassa olevaan. Kun sovelluksen mittakaava on vaakasuora, on myös mahdollista saada joustavuuden etu.

Automaattinen skaalaus auttaa varmistamaan, että käynnissä on aina oikea määrä esimerkiksi Amazon EC2 -instansseja sovelluksen kuorman käsittelemiseen. Kokoelmaa EC2-instansseista kutsutaan nimellä automaattinen skaalausryhmä. On mahdollista määrittää vähimmäismäärä instansseja jokaisessa automaattisessa skaalausryhmässä, tällöin automaattinen skaalaus varmistaa, että ryhmä ei mene koskaan kyseisen vähimmäismäärän alle. On myös mahdollista määrittää skaalausryhmän enimmäismäärä EC2 instansseja. Skaalauskäytänteiden avulla voidaan instanssien käynnistämiset ja lopettamiset, kun sovelluksen kysyntä kasvaa tai vähenee. (AWS, e)

Yksi tärkeimmistä asioista, jotka on tehtävä automaattista skaalausta käyttäessä, on muistaa skaalata sisäänpäin. Skaalausryhmän laajentaminen maksaa enemmän, joten aina kun on mahdollisuus skaalata sisäänpäin ilman suorituskykyyn ja luotettavuuteen liittyviä menetyksiä, kannattaa niin tehdä.

Esimerkiksi kuvassa 10 olevassa automaattisessa skaalausryhmässä on yhden instanssin vähimmäiskoko, kahden instanssin haluttu kapasiteetti ja enimmäiskoko on neljä. Määritetyt skaalauskäytännöt säätävät instanssien määrää minimi- ja enimmäistapauksissa määritettyjen ehtojen perusteella. (AWS,)



Kuva 10. Esimerkki kuva automaattisesta skaalausryhmästä (Godden-Payne, 2019)

Automaattisen skaalausryhmän rekisteröinti kuormituksen tasaajaan kuvan 10 mukaisesti auttaa luomaan kuormituksella tasapainotetun sovelluksen. Kuormituksen tasapainotus toimii automaattisen skaalausryhmän kanssa niin, että se jakaa saapuvaa liikennettä esimerkiksi terveiden EC2-instanssien välillä (AWS,). Kuormitusta tasapainottamalla nostetaan palvelun luotettavuutta.

Ennen kuin voidaan aloittaa kuormituksen tasapainotus, täytyy lisätä vähintään yksi tai useampi kuuntelija. Kuuntelija on prosessi, joka tarkistaa yhteyspyynnöt määritellyn protokollan ja portin mukaisesti. Kuuntelijalle määritetyt säännöt määräävät, miten kuormituksen tasaaja reitittää pyynnöt rekisteröidyille kohteille. (AWS, c)

AWS:n kuormituksen tasaajan kuuntelijat tarjoavat vain http- tai https-protokollaa, mutta WebSocket ottaa ensimmäiseksi palvelimeen yhteyttä http:n avulla ws:// alkuisella osoitteella. Kun käytössä on https, käytetään wss:// alkuista osoitetta (Demichelis, 2020). Palvelin vastaa selaimelle 101 kytkentäprotokollakoodilla ja kehottaa asiakasta vaihtamaan WebSocket-yhteyteen (Demichelis, 2020). Ero tavallisen http-yhteyden kanssa on se, että WebSocket-yhteyden on tarkoitus pysyä avoimena (Demichelis, 2020).

Jotta pitkäaikaisia yhteyksiä voidaan tukea, täytyy oletus tyhjäkäynnin yhteyden aikakatkaisuasetusta (Idle Connection Timeout) nostaa 60 sekunnista suuremmaksi. Tyhjäkäynnin aikakatkaisu tapahtuu silloin, kun palvelimen ja asiakkaan välillä ei tapahdu mitään. Kyseessä on siis viive, ennen kuin kuormituksen tasauslaite katkaisee yhteyden, mikä puolestaan aiheuttaa asiakkaalle 1006-virheen. (Demichelis, 2020)

Potentiaalisen virhetilanteen välttämiseksi, täytyy sovellustasolle rakentaa uudelleenkytkentälogiikka, joka mahdollisessa yhteyden katkaisemistilanteessa, kytkee yhteyden uudelleen, jotta kuormituksen tasaaja ei katkaise yhteyttä (Demichelis, 2020). Case-esimerkissä hyödynnetty Apollo Server tukee ominaisuutta, jolla tilaus päätepiste voi lähettää keep-alive sanomia avoimille asiakasyhteyksille millisekunteina, jos sellaisia on.

3.6 Arkkitehtuurillisia ratkaisuja

Luotettavuutta miettiessä täytyy usein myös miettiä sovellusarkkitehtuuria. Pilvisovellukset ovat perinteisesti tukeutuneet suuriin koodikokoelmiin, jotka tunnetaan paremmin nimeltä monoliitit, jotka toimivat yhtenä suurena sovelluksena (Lefkowitz, 2019). Sovelluksien kasvaessa monimutkaisemmiksi ja vaatimusten kasvaessa, monoliiteista tulee hankalia. Täten yritykset ovat siirtymässä mikropalveluihin – lukuisiin pieniin sovelluksiin, joista kukin suorittaa yhden toiminnon ja kommunikoi verkon kautta toimiakseen yhdessä (Lefkowitz, 2019).

Skaalautuvia ja luotettavia palveluita suunniteltaessa hyödynnetään usein palvelukeskeistä arkkitehtuuria (SOA) tai mikropalveluarkkitehtuuria. Palvelukeskeinen arkkitehtuuri on käytäntö tehdä ohjelmistokomponenteista uudelleen käytettäviä palvelurajapintojen kautta. Palvelukeskeisessä arkkitehtuurissa käytetään yhteisiä tiedonsiirtostandardeja, jotta ne voidaan nopeasti sisällyttää uusiin kuormituksiin. (Eliot Seth et al., 2020)

Mikropalveluarkkitehtuurissa mennään pidemmälle komponenttien pienentämiseksi ja yksinkertaisemmaksi. Mikropalveluiden avulla voidaan erottaa eri palveluiden vaadittavan saatavuuden mukaan ja siten sijoittaa investoinnit tarkemmin niihin mikropalveluihin, joilla on suurimmat saavutustarpeet. Esimerkiksi tuotesivujen näyttämiseksi Amazon.com sivustolla käytetään satoja mikropalveluja. Vaikka hinnan ja tuotetietojen toimittamiseksi on oltava käytettävissä muutama palvelu, valtaosa sivun sisällöstä voidaan yksinkertaisesti sulkea pois, jos palvelua ei ole saatavilla. Esimerkiksi valokuvia tai arvosteluja ei vaadita siihen kokemukseen, jossa asiakas voi ostaa tuotteen. (Eliot Seth et al., 2020)

Mikropalveluarkkitehtuuria suunnitellessa on otettava huomioon myös kompromisseja. Eräs niistä on se, että kyseessä on hajautettu tietojenkäsittely, joka voi vaikeuttaa käyttäjien viivästymisvaatimusten saavuttamista, virheenkorjausta. Käyttäjien vuorovaikutusten jäljittäminen on vielä monimutkaisempaa. (Eliot Seth et al., 2020)

Edellä esitellyt mikropalvelu ja palvelukeskeinen arkkitehtuuri ei kuitenkaan soveltunut case-esimerkissä tehtyyn projektiin aikataulullisista syistä. Arkkitehtuuri suunniteltiin kuitenkin niin, että sovellusta olisi mahdollista pilkkoa pienempiin osiin ja viedä kohti mikropalveluarkkitehtuuria.

Case-esimerkin sovellusarkkitehtuuria mietittäessä päädyttiin hyödyntämään kolmitasoista arkkitehtuuria. Kolmitasoinen arkkitehtuuri viitaa sovellusarkkitehtuuriin, joka järjestetään kolmeen loogiseen ja fyysiseen eri kerrokseen: esitystaso tai käyttöliittymä, sovellustaso, jossa tietoja käsitellään ja datataso, jossa sovellukseen liittyvät tiedot tallennetaan ja hallitaan (IBM Cloud Education, 2020c).

Kolmitasoisen arkkitehtuurin yksi tärkeimpiä etuja on se, että jokainen taso toimii omalla infrastruktuurillaan, joten erilliset kehitystiimit voivat kehittää kukin tasoja samanaikaisesti ja päivittää tai skaalata tarpeen mukaan vaikuttamatta muihin tasoihin (IBM Cloud Education, 2020).

Sovellustaso, joka tunnetaan myös logiikkatasona tai keskitasona, on sovelluksen sydän. Tässä kerroksessa käsitellään esitystasolta kerätyt tiedot – toisinaan muita tietotason tietoja vastaan – käyttämällä liiketoimintalogiikkaa, eli tiettyä liiketoimintasääntöä. Sovellustaso voi myös lisätä, poistaa tai muokata datatason tietoja (IBM Cloud Education, 2020). Sovellustaso tyypillisesti kommunikoi datatason kanssa API-kutsujen avulla (MDN Web Docs, 2021). Datataso, jota joskus kutsutaan myös tietokantatasoksi, on paikka, johon sovelluksen käsittelemät tiedot tallennetaan ja hallitaan (IBM Cloud Education, 2020).

Esitystaso on sovelluksen käyttöliittymä ja viestintäkerros, jossa loppukäyttäjä on vuorovaikutuksessa sovelluksen kanssa (IBM Cloud Education, 2020). Sen päätarkoitus on näyttää tietoja käyttäjälle ja kerätä tietoja käyttäjältä (IBM Cloud Education, 2020). Toteutetussa projektissa ylätaso toimii verkkoselaimella, mutta se voisi myös olla työpöytäsovellus tai jokin muu graafinen käyttöliittymä. Verkkoesitystasot kehitetään yleensä HTML:n, CSS:n ja JavaScriptin avulla (IBM Cloud Education, 2020). Case-esimerkissä esitellään tarkemmin esitystasoon käytettyjä komponentteja.

3.7 Suorituskyvyn tehokkuus

Suorituskyvyn tehokkuuden pilari sisältää kyvyn käyttää laskentaresursseja tehokkaasti järjestelmän vaatimusten täyttämiseksi. Lisäksi tarkoitus on ylläpitää tehokuutta kysynnän muuttuessa ja tekniikan kehittyessä. (Pullen Eric et al., 2020)

Ensimmäinen suunnitteluperiaate ehdottaa selvittämään palvelut ja resurssit, joita pilvipalveluntarjoaja antaa käyttöön. Monimutkaisten tehtävien delegointi pilvipalveluntarjoajalle, kuten NoSQL-tietokannat, median transkoodaus ja koneoppiminen ovat kaikki teknologioita, jotka vaativat erikoisosaamista. Julkisessa pilvessä näitä tekniikoita tarjotaan palveluina, joita tiimit voivat hyödyntää. Tällöin yrityksen tiimit voivat keskittyä tuotekehitykseen resurssien valmistelun ja hallinnan sijaan. (Pullen Eric et al., 2020)

3.8 Tietokanta-arkkitehtuurin valinta

Optimaalinen tietokantaratkaisu järjestelmälle vaihtelee kolmen halutun ominaisuuden välillä: saatavuuden, johdonmukaisuuden ja osiotoleranssin perusteella (Pullen Eric et al., 2020). Kolmea ominaisuutta kutsutaan myös CAP-lauseeksi, jonka on luonut tietojenkäsittelytieteen tutkija Eric Brewer (Browne, 2009). AWS lisää CAP-lausekkeeseen mukaan myös kestävyyden, skaalautuvuuden ja kyselyominaisuuden vaatimukset.

CAP-lausekkeen lisäksi on myös ACID – atomisuus, johdonmukaisuus, eristäytyminen ja kestävyys. Yhdessä ACID on joukko ohjaavia periaatteita, jotka varmistavat, että tietokantatapahtumat käsitellään luotettavasti. Tietokantatapahtuma on mikä tahansa suoritettu toiminto, kuten uuden tietueen luominen tai tietojen päivittäminen. (Watts, 2020)

Tietokannassa tehdyt muutokset on tehtävä varoen, jotta varmistetaan, että tiedot eivät vioitu. Tietokanta muutokset kannattaa lähtökohtaisesti tehdä tietokantamigraatioita hyödyntäen kuten kappaleessa 3.3. ACID-ominaisuuksien soveltaminen jokaiseen tietokannan muokkaukseen on hyvä tapa ylläpitää tietokannan tarkkuutta ja luotettavuutta. (Watts, 2020)

Käyttämällä yksinkertaista esimerkkiä voidaan antaa todellinen merkityksen kyseisille termeille: Käyttäjä yrittää ostaa kopion Tolstoin Sodasta ja rauhasta verkkokirjakaupasta, jossa on varastossa yksi kappale. Käyttäjä lisää sen ostoskoriin, mutta muistaa, että hän tarvitsee muutamia muita asioita, joten hän selaa sivustoa vähän. Käyttäjän lukiessa rusketustuotteen asiakasarvosteluja, joku muu saapuu paikalle, lisää kopion ostoskoriinsa ja menee suoraan kassalle ja ostaa tuotteen. (Browne, 2009)

Johdonmukainen palvelu toimii kokonaan tai ei ollenkaan. Johdonmukaisuuden sijasta voidaan käyttää myös termiä atominen, jossa muutoksen kohteena olevasta tiedosta on aina käytettävissä jokin ehjistä versioista. Atomisuuden käyttäminen on teknisesti järkevämpää, sillä tarkasti ottaen johdonmukaisuus on ACID:n (atomicity, consistency, isolation, durability) C-arvo sovellettuna tietokantatapahtumien ihanteellisin ominaisuuksin, mikä tarkoittaa, että tietoja ei koskaan säilytetä tavalla, joka rikkoo tiettyjä ennalta asetettuja rajoituksia. (Browne, 2009)

Kirjan ostoesimerkissä käyttäjä voi lisätä kirjan ostoskoriinsa tai epäonnistua. Käyttäjä ostaa sen tai ei, mutta käyttäjä ei voi lisätä tai ostaa puoliksi kirjaa. Varastossa on vain yksi kappale, ja vain yksi henkilö voi sen saada. Jos molemmat asiakkaat voivat jatkaa tilausprosessia loppuun eli suorittaa maksun, ongelmaksi muodostuu varastojen ja järjestelmässä olevien välinen epäjohdonmukaisuus. (Browne, 2009)

Saatavuus tarkoittaa, että palvelu on käytettävissä – eli toimii kuten johdonmukaisuudessa ja jonka aikana palvelu pystyy käsittelemään pyynnöt onnistuneesti (Kleppmann, 2015). Vastauksen katsotaan yleensä onnistuneen, jos se on kelvollinen (ei virhe ja tyydyttää tietokannan turvallisuusominaisuuksia) ja se saapuu asiakkaalle tietyn ajan kuluessa, mikä voidaan määritellä palvelutasosopimuksella (Kleppmann, 2015).

Käyttäjän ostaessa kirjan, hän haluaa saada loogisen vastauksen, eikä selainviestiä siitä, että verkkosivusto ei kykene viestimään (Browne, 2009). Professorit Gilbert ja Lynch CAP-lausekkeen todisteessaan, että saatavuus heikkenee usein, kun käyttäjä juuri eniten sitä tarvitsee – sivustot yleensä menevät alas kiireisinä aikoina juuri siksi, että ne ovat liian kiireisiä (Gilbert, Lynch, 2012). Myöskään palvelusta, joka on käytettävissä, mutta jota ei käytetä, ei ole hyötyä kenellekään.

Relaatiotietokannat tukevat luonnollisesti ACID-mallia, kun taas NoSQL tietokannat eivät, joten jos käyttäjät haluavat hyödyntää ACID:n tarjoamaa luotettavuutta heidän yleensä täytyy suorittaa lisäohjelmointia. Myös johdonmukaisuuden osalta tarvitaan manuaalista tukea. Yhdenmukaisuuden puuttuminen mahdollistaa paremman suorituskyvyn ja skaalautuvuuden, mutta saattaa olla ongelma tietyntyyppisille sovelluksille ja liiketoimille. (Leavitt, 2010)

3.9 Kustannusten optimointi

Kustannusten optimointi on jatkuvaa hienosäätöä ja parannusta työmäärän elinkaaren aikana. Tässä kappaleessa käsitellään käytäntöjä, joilla voidaan rakentaa ja käyttää kustan-

nustietoisia työmääriä, jotka saavuttavat liiketoiminnan tulokset pyrkien minimoimaan kustannuksia ja antaen organisaatiolle mahdollisuuden maksimoida sijoitetun pääoman tuottoprosentti.

Kuten muidenkin hyvin suunniteltujen puitteiden kohdalla, joutuu myös kustannuksia optimoidessaan tekemään kompromisseja markkinoiden nopeuden tai kustannusten mukaan. Joissakin tapauksissa on järkevintä optimoida nopeudella – mennä nopeasti markkinoille, toimittaa uusia ominaisuuksia tai yksinkertaisesti noudattaa määräaikaa – sen sijaan, että investoitaisiin ennakkoon kustannusten optimointiin.

Suunnittelupäätöksiä ohjaavat toisinaan pikemminkin kiire kuin data. On helposti kiusaus maksaa liian suuri korvaus ”varmuuden vuoksi” sen sijaan, että käytettäisiin aikaa kaikista kustannustehokkaimman vaihtoehdon löytämiseen. Tämä saattaa johtaa ylivarautumiseen ja ali optimoituihin käyttöönottoihin.

Kustannusten optimointia voidaan tehdä kysyntään perustuva tarjonnan avulla. Organisaation tulisi hyödyntää julkisen pilven joustavuutta resurssien toimittamiseksi vastaamaan muuttuvaan kysyntään. Sovellusliittymien eli API:en ja palveluominaisuuksien hyödyntäminen, jotta voidaan muuttaa arkkitehtuurin pilviresurssien määrää ohjelmallisesti dynaamisesti. Tämän avulla voidaan skaalata arkkitehtuurin komponentteja ja lisätä resurssien määrää automaattisesti kysynnän nousun aikana, jotta voidaan ylläpitää suorituskykyä ja toisaalta kapasiteetin vähentyessä kustannusten pienentämiseksi.

Optimointia voidaan myös tehdä aikaperusteinen tarjonnan avulla. Aikaperusteinen lähestymistapa kohdistaa resurssikapasiteetin kysyntää, joka on ennustettavissa tai ajallisesti tarkasti määriteltävissä. Tämä lähestymistapa ei tyypillisesti ole riippuvainen resurssien käyttöasteista. Aikaperusteinen lähestymistapa varmistaa, että resursseja on käytettävissä juuri sinä ajankohtana, jolloin niitä tarvitaan ja ne voidaan tuoda tarjolle ilman viivästyksiä käynnistysmenettelyjen ja järjestelmän- tai johdonmukaisuustarkastusten vuoksi.

Aikaperusteisen lähestymistavan avulla voidaan tarjota lisäresursseja tai lisätä kapasiteettia kiireisinä aikoina. Työmäärät voidaan suunnitella laajeneviksi määrättyihin aikoihin kuten työajan alkaessa varmistaen siten, että resurssit ovat käytettävissä käyttäjien tai kysynnän saapuessa. Aikaperusteista lähestymistapaa voidaan hyödyntää myös automaattisissa skaalausryhmissä, jotka esiteltiin tarkemmin kappaleessa 3.5. Case-projektissa aikaperusteista lähestymistapaa hyödynnetään niin, että arkisin aamu kahdeksan ja ilta

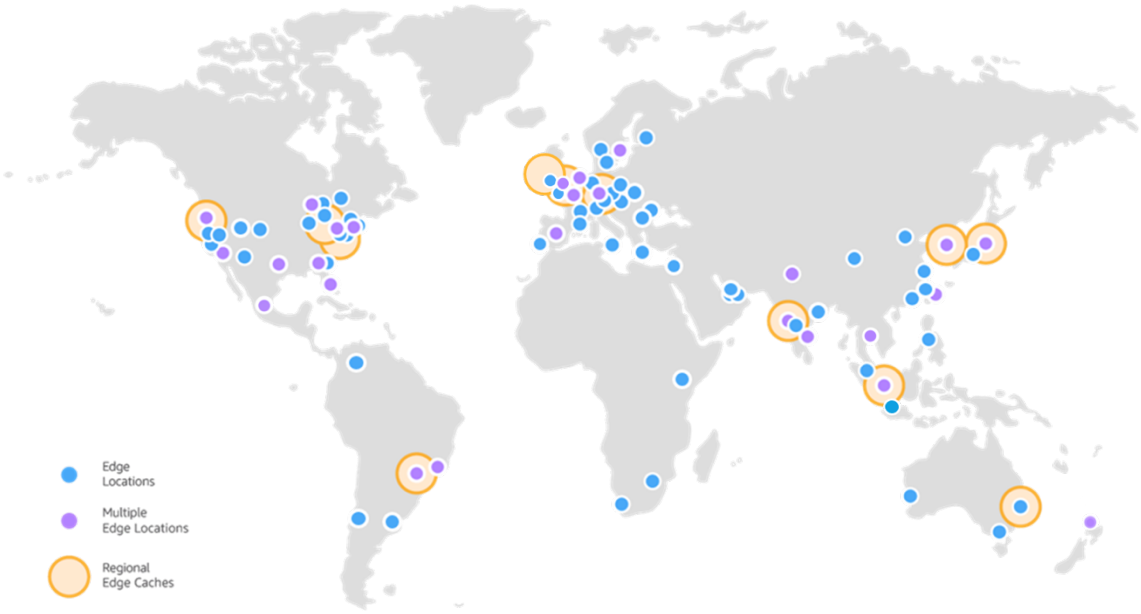
kahdeksan välillä pidetään vähintään kaksi taustapalvelua käynnissä. Muina aikoina palvelussa on vain yksi taustapalvelu käynnissä. Täten palvelu on aina saatavilla toimisto- ja kouluaikoina.

3.10 Sisällönjakeluverkosto

Tiedonsiirto Internetissä riippuu suurelta osin valokaapelien maailmanlaajuisista verkoista, mikä sallii erittäin suuren kaistanleveyden tiedonsiirtoon. Kyseiset kuituverkot lähettävät jo tietoa valonnopeudella, mutta ihmiset ovat kärsimättömiä ja pieniä viiveitä voi olla havaittavissa, kun sisältöä käytetään toiselta puolelta maailmaa. Koska valonnopeus on osoittautumassa vaikeasti ratkaistavaksi haasteeksi, voidaan sisällönjakeluverkostolla parantaa käyttökokemusta verkkosivustoille muilla tavoin. (AWS, 2016)

Mietittäessä parannuksia ja ratkaisuja edellisissä kappaleissa vastaan tulleisiin teemoihin voidaan sisällönjakeluverkostoa (CDN) hyödyntämällä parantaa useampaa hyvän arkkitehtuurin pilaria: suorituskykyä, luotettavuutta ja saatavuutta. Amazon CloudFront on AWS:n tarjoama sisällönjakeluverkko, joka tarjoaa maailmanlaajuisesti hajautetun välityspalvelinverkon, joka tallentaa välimuistiin sisältöä, kuten verkkovideoita, tai muuta suurta mediaa, paikallisille kuluttajille, mikä parantaa pääsynopeutta sisällön lataamiseen (AWS, a).

CloudFrontin avulla mahdollistetaan siis nopeammat loppukäyttäjän vasteajat ja sisältöpyynnöt (AWS,). Sisältötoimitusverkon avulla vähennetään myös tarpeetonta skaalautumista taustapalvelun työmäärissä (AWS,). Kuvassa 11 on kuvattu AWS:n maailmanlaajuinen verkosto.



Kuva 11. AWS:n maailmanlaajuinen verkosto (AWS,)

Kuvan 11 on tarkoitus havainnollistaa sitä, että vaikka esimerkiksi Suomessa ei ole AWS aluetta eli fyysistä sijaintia, jonne AWS resursseja voidaan luoda, on Suomessa kuitenkin AWS:n reuna sijainti (edge location). Mitä suurempi reuna-alueiden määrä, sitä paremmin sisältö jaetaan ympäri maailmaa tai aluetta.

Erä CloudFrontin käyttötarkoituksista on vähentää pyyntöjen määrää, joihin lähtöpalvelimen on vastattava suoraan. CloudFront välimuistin avulla pyritään näyttämään enemmän objekteja reuna-alueista, jotka ovat lähempänä käyttäjiä. Tällä voidaan vähentää lähtöpalvelimen kuormitusta ja vähentää viivettä.

Case-esimerkin projektissa hyödynnettiin myös CloudFront palvelua, mutta siinä törmättiin välimuistin hyödyntämisen osalta haasteisiin. Välimuistin hyödyntäminen on helpointa GET-pyyntöjen osalta, sillä GET kertoo sisällönjakeluverkolle, että sovellus on tekemässä vain datan lukua – ei kirjoittamista. Tämä kertoo CDN:lle, että on hyvä tallentaa tulos välimuistiin, koska se ei odota muuttavansa jotain sovellustasolla. Case-esimerkissä jokainen pyyntö – kirjoitus ja luku – ovat POST pyyntöjä, jolloin sisällönjakeluverkosto ei automaattisesti havaitse palvelimen suuntaan lähteviä lukemisia.

Useimmat GraphQL-työkalut lähettävät HTTP-pyyntöjä POST:n avulla ja käyttävät URL-osoitteen sijaan monimutkaista pyyntörunkoa, joka sisältää kyselyn ja muuttujat. Lisäongelmana on joissakin selaimissa suhteellisen pieni URL-kokorajoitus, mikä tarkoittaa, että ei ole mahdollista sijoittaa koko kyselyä ja muuttujia GET-pyynnön URL-osoitteeseen.

GraphQL on Facebookin ensin sisäisesti kehitetty ja sittemmin suurelle yleisölle avautunut vuonna 2015. Kuten nimestä käy ilmi, GraphQL on API-hakukieli. Sen avulla asiakassovellus voi lähettää kyselyitä saadakseen juuri etsimänsä yhdessä pyynnössä.

CloudFrontia käyttäessä liikenne kulkee Amazonin globaalin verkko selkärangan kautta. Tällä tavoin riippumatta siitä, onko sisältö saatu tallennettua välimuistiin vai ei, sisällöntoimitusta saadaan nopeutettua optimoitujen verkkopolkujen yli. (AWS,)

4 Softalaprojekti

Case-esimerkin projekti toteutettiin yhdessä Haaga-Helian opiskelijoiden kanssa kevään aikana vuonna 2021. Tavoitteena oli siirtää perinteisin tavoin rakennettu web-sovellus pilveen hyödyntämällä teoriaosuudessa läpi käytyjä hyvin suunnittelun AWS-kehyksen käytänteitä. Aikaisemmin sovellusta oli suoritettu joko opiskelijoiden tietokoneelta tai yhdeltä EC2-palvelimelta. EC2-palvelimiin liittyviä aspekteja sekä pilvipalvelumalleja on käsitelty kappaleissa 2.1 ja 2.4.

Kevään softalaprojekti kurssi jatkuu siitä, mihin syksyllä edellinen ryhmä opiskelijoita lopetti. Usein koulun projekteissa pääsee aloittamaan tyhjältä pöydältä. Tämän projektin tavoitteena oli kuitenkin näyttää, miten työelämässä projektia ei useinkaan pääse aloittamaan tyhjästä, vaan täytyy jatkaa jonkun muun tekemän työn pohjalta. Usein projektit voivat olla kestäneet jo vuosien ajan.

Aiheeseen liittyy vahvasti myös dokumentointi ja määrittely, jota tiimin täytyi ylläpitää projektin keston ajan. Sovellus itsessään on projektinhallinta työkalu, joka sisältää mahdollisuuden lisätä eri projekteja, tauluja, sekä tauluihin liittyviä tehtäviä. Taulun tarkoitus on visualisoida tehtävien kulkua eri sarakkeista toiseen. Lisäksi tehtäviä voi merkitä monelle eri käyttäjälle ja niitä on mahdollista värjätä haluamillaan väreillä.

Projektin alussa eräs suurimmista haasteista oli saada kaikille kehittäjille toistettava ja helposti pystytettävä kehitysympäristö. Täten opiskelijoiden kanssa opeteltiin kontti- ja Docker teknologian perusteita. Projektin alussa luotua Docker konfiguraatiota hyödynnettiin sellaisenaan myös AWS:ssä. Kappaleessa 2.4 on kuvailtu konttiteknologioita tarkemmin. Dockerin käyttöönoton jälkeen oli projektilaisten syytä päivittää dokumentointi kehitysympäristön pystyttämisen osalta seuraavaa projektitiimiä varten.

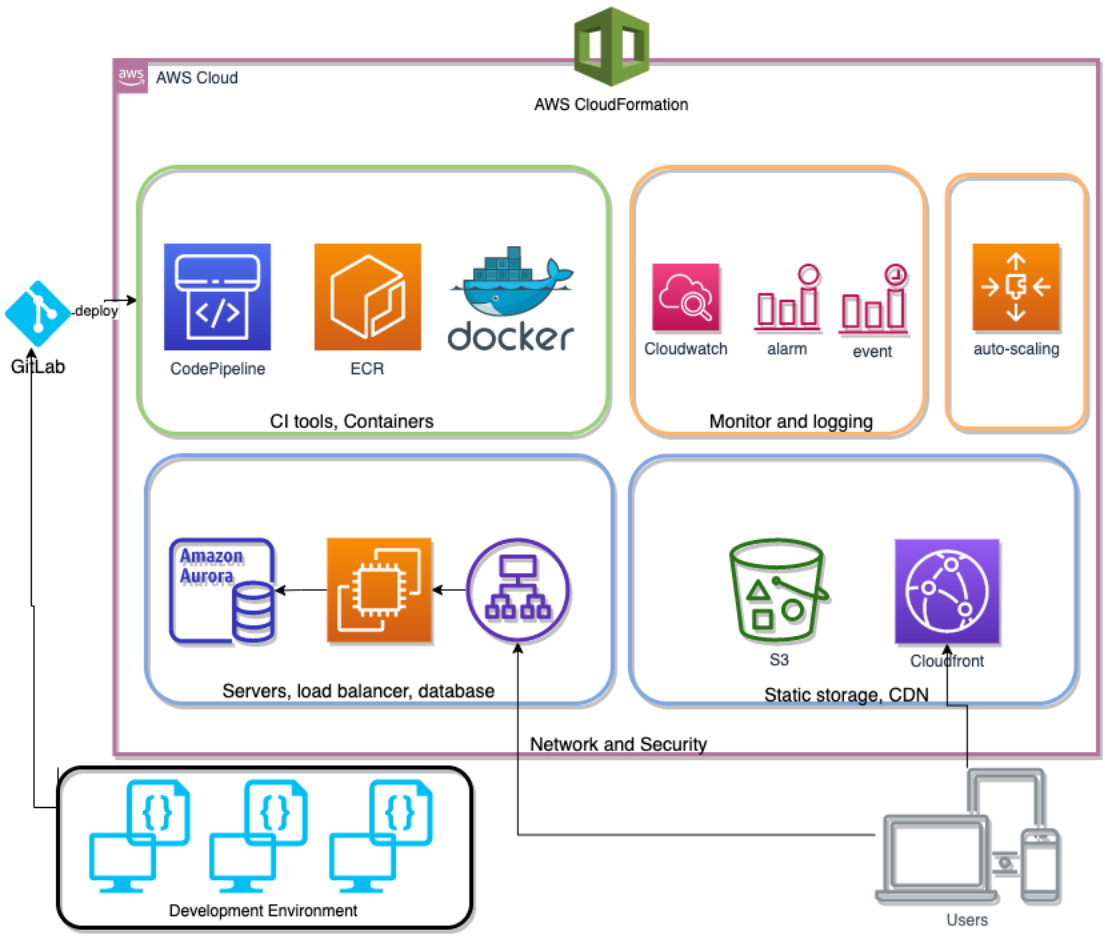
Kappaleessa 3.3 esitellyt tietokantamigraatiot ja ennen kaikkea niiden puuttuminen tuotti myös haasteita. Tietokantamigraatioiden luonti jälkikäteen on työlästä, sillä kukaan aikaisemmasta projektista ei ollut paikalla jakamassa tietoa tietokannan viiteavaimien läpikäynnissä.

Työlään vaiheen jälkeen oli projekti kuitenkin saanut luotettavan teknisen dokumentaation versionhallintaan kaikista tietokantaan liittyvistä muutoksista. Sovelluksen luonteeseen kuuluu lukuisat monesta moneen liitokset eri taulujen välillä, joten migraatiotiedostojen avulla oli huomattavasti helpompi tarkastella taulujen välisiä liitoksia.

4.1 Yleiskuvaus

Projektin aikana pyrittiin hyödyntämään mahdollisimman monipuolisesti AWS:n tarjoamia palveluita. Tarkoitus oli hyödyntää palvelimettomuutta, sekä työkaluja, joihin tyypillisesti törmää AWS:n parissa

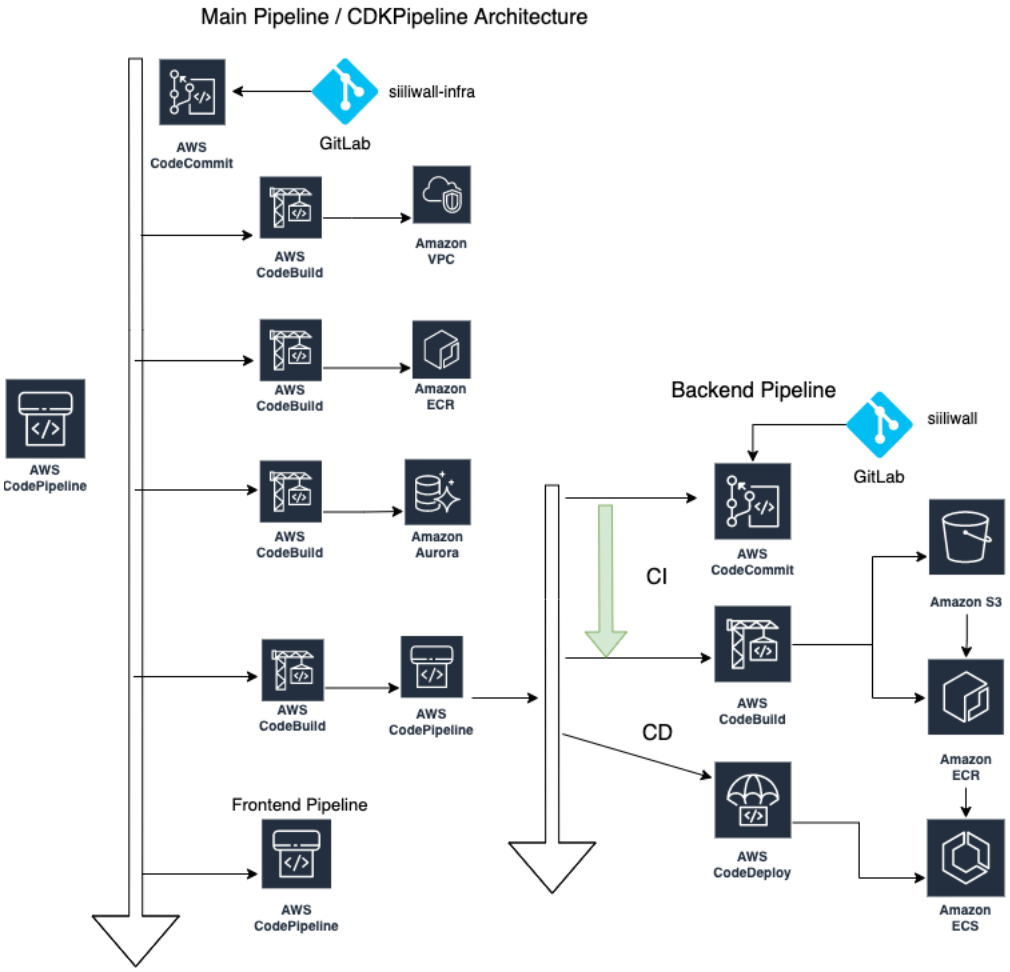
Kuvassa 12 on kuvattuna AWS:n palvelut yleisellä tasolla, joita projektin aikana hyödynnettiin. Versionhallintana käytettiin AWS:n ulkopuolella sijainnutta GitLabia, josta lähdekoodi siirrettiin AWS:ään. Kuvan on tarkoitus ennen kaikkea havainnollistaa yleisiä teemoja, joita infrastruktuurin pystyttämiseen liittyy.



Kuva 12. Yleiskuva käytetyistä projektin aikana palveluista

Kuvan 12 yläreunassa näkyy AWS CloudFormation. Se kuvastaa sitä, että jokainen AWS:n resurssi on dokumentoitu ja luotu CloudFormation mallin mukaisesti. CloudFormationia ei ole kuitenkaan käytetty suoraan, vaan projektissa on päädytty hyödyntämään AWS CDK työkalua ja TypeScript ohjelmointikieltä. Operatiivisissa suunnitteluperiaatteissa kappaleessa 3.2 on käsitelty tarkemmin infrastruktuuria koodina aihetta.

Kuvassa 13 on esitelty projektin pääjulkaisuputki (Main Pipeline) tai toiselta nimeltään CDK putki (CDK Pipeline). Projektin infrastruktuuri muutokset tehdään siiliwall-infra ja sovellustason muutokset siiliwall git-arkistoihin. Molemmat julkaisuputket käynnistyvät jokaisesta versionhallinnan päähaaran muutoksesta. Tämän jälkeen putki tarkistaa mahdolliset muutokset jokaisesta ja jatkaa kunnes kaikki on suoritettu. Julkaisuputket ovat kuvattuna alaspäin suunnatulla nuolella.

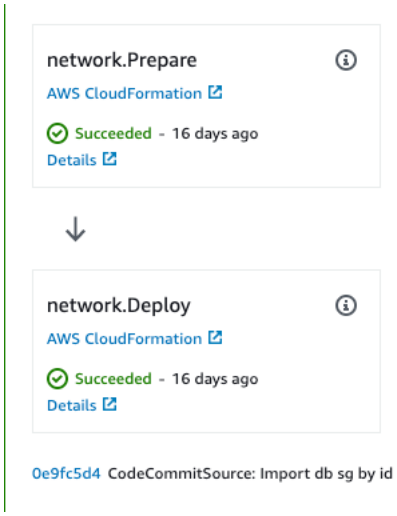


Kuva 13. Infrastruktuurin julkaisuun liittyvä arkkitehtuuri

Kuvan 13 ensimmäinen vaihe on verkon luontivaihe, jossa luodaan VPC, NACL-luettelot ja aliverkot. Verkon luominen hyvin alkuvaiheessa infrastruktuuria on oleellista, sillä osa resursseista on riippuvaisia sen olemassaolosta. Verkkoihin liittyviä aspekteja on käsitelty tarkemmin kappaleessa 3.4 Vähiten oikeuden periaate.

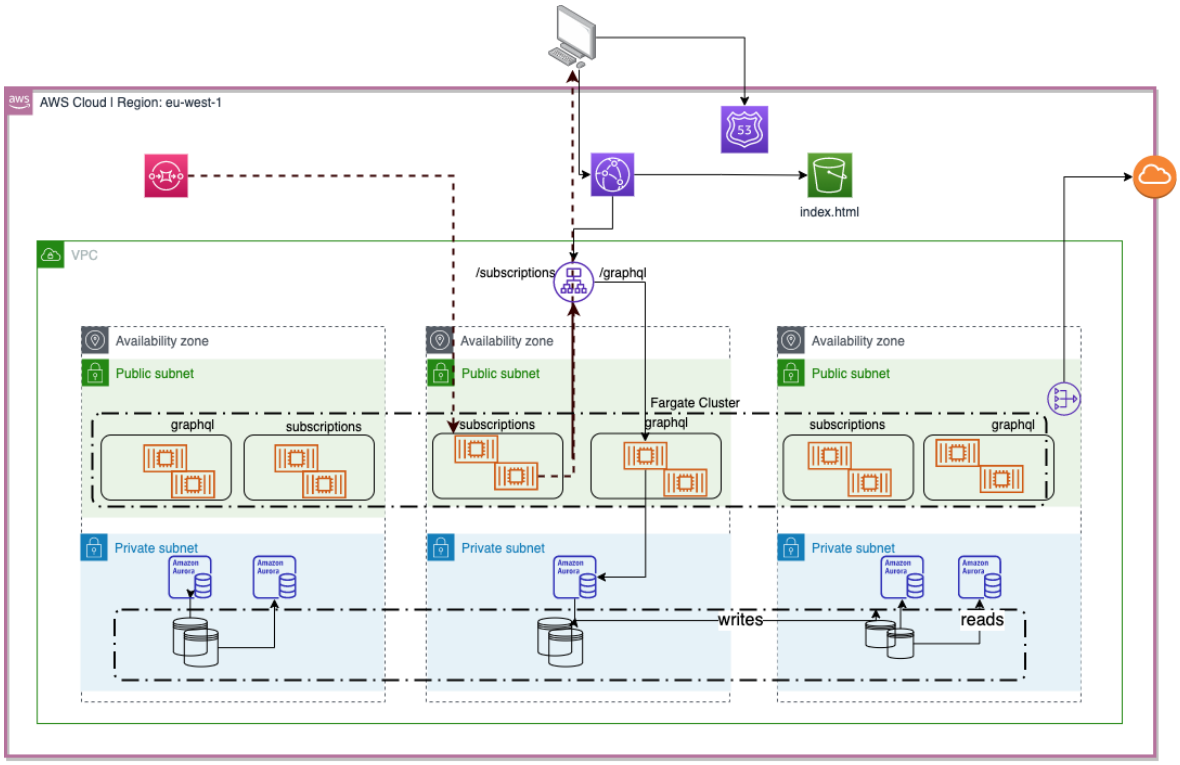
Jatkokehityksenä olisi järkevää siirtää verkkoon tapahtuvat muutokset toiseen git-arkistoon ja tehdä siihen liittyviä muutoksia erillisellä julkaisuputkella, sillä VPC:n ja aliverkkojen IP-avaruuksia ei voida muuttaa enää, jos IP-osoitteita on jo varattu eli jos resursseja kuten tietokanta on jo luotuna. Täten mahdolliset verkkomuutokset jo olemassa olevaan verkkoon ei ole mahdollista.

Seuraavaksi putki luo tyhjän Amazon ECR:n. ECR toimii säilönä sovellustason (backend) julkaisuputken luomille Docker-kuville. Lisäksi tässä vaiheessa luodaan tyhjä AWS CodeCommit säilö myöhemmässä vaiheessa luotavalle sovellustason putkelle. Sekä tyhjä ECR että CodeCommit täytyvät olla etukäteen luotuna, jotta sovellustason putki voi toimia, joten ne ovat järkevää luoda aikaisessa vaiheessa. Kuvassa näkyy myös mikä on käynnistänyt julkaisuvaiheen. Tässä tapauksessa versionhallintaan lisätty muutos on aiheuttanut käynnistyksen. Kuvassa 14 on kuvattuna onnistunut verkonluonti vaihe.



Kuva 14. Onnistunut verkon luontivaihe

Kuvassa 15 on kuvattuna reitti käyttäjän selaimesta kolmivaiheisen arkkitehtuurin näyttö-, sovellus- ja datatason sekä AWS:n hallinnoimien palvelujen läpi.



Kuva 15. Arkkitehtuuri näyttö-, sovellus- ja datatasosta

Kaikki resurssit on luotu Irlannin alueelle, joka on eräs vanhimmista AWS:n alueista ja täten tietyt palvelut tulevat sinne saataville aiemmin kuin esimerkiksi Tukholman alueelle. Opinnäytetyötä kirjoittaessa tietokantana käytettyä Amazon Aurora Server Clusteria ei ollut saatavilla Tukholman alueella.

VPC on jaettu kuvan 15 mukaisesti kuuteen eri aliverkkoon – kolmeen julkiseen ja kolmeen yksityiseen aliverkkoon. Aliverkot ovat jaettu tasaisesti kolmelle eri saatavuusalueelle mahdollisimman hyvän saavutettavuuden parantamiseksi.

Kuvassa 15 näkyvä Fargate Cluster on sovellustason looginen tehtävien tai palveluiden ryhmittelyyn käytettävä klusteri. AWS Fargate on tekniikka, jolla voidaan käyttää Amazon ECS:ää konttien ajamiseksi ilman, että käyttäjän tarvitsee hallinnoida palvelimia tai EC2 -instanssien klustereita. Fargate poistaa tarpeen valita palvelintyyppejä, päättää klustereiden skaalausajankohtaa tai optimoida klusteripakkausta. Fargaten avulla voidaan sovellustasolla hyödyntää kappaleessa 2.4 esiteltyä abstraktin tason jaetun vastuun mallia. Sovelluksen backend eli palvelimen puolella toimivan ohjelmiston muita pääteknologioita olivat Node, Express, Sequelize, Apollo Server.

Projektin tietokanta rakentui hyödyntämällä Amazonin Aurora Serverless Cluster palvelua eli palvelimetonta tietokanta klusteria. Täten myös tietokannan osalta hyödynnetään abst-

raktin tason jaetun vastuun mallia. Palvelimeton tietokanta-moottoritila on suunniteltu epäsäännöllisiin kuormituksiin, jossa tietokannan käyttö saattaa olla raskasta lyhyen aikaa, jota seuraa pitkä kevyt aktiivisuus tai ei lainkaan toimintaa.

Esimerkkejä tällaisista ovat vähittäiskaupan verkkosivustot, joissa on ajoittaisia myyntitapahtumia, tietokannat, jotka tuottavat raportteja tarvittaessa, kehitys- ja testausympäristöt sekä uudet sovellukset, joilla on epävarmat vaatimukset. Tällaisissa tapauksissa kapasiteetin määrittäminen oikein etukäteen ei aina ole mahdollista porvisioidun mallin kanssa ja se saattaa myös johtaa korkeampiin kustannuksiin, jos käytössä on kapasiteettia, joita ei käytetä.

Palvelimettomuudella sekä konttien että tietokannan osalta haetaan tämän projektin tapauksessa rahallisia säästöjä, sillä oletettu käyttö on epätasaista, vähäistä ja se rajoittuu tietyille kellonajoille.

4.2 Käyttöliittymä

Projekti hyödynsi GraphQL:ää, joka on avoimen lähdekoodin tietokysely ja sovellusliittymien käsittelykieli. Sen avulla käyttöliittymä voi määritellä tarvittavan datan rakenteen ja sama tietojen struktuuri palautuu palvelimelta, mikä estää liian suurten tietomäärien palauttamisen, mutta tällä on osaltaan vaikutuksia kyselytulosten tehokkaaseen web-välimuistiin. Välimuistiin liittyviä haasteita käsitettiin tarkemmin teoriaosuuden kappaleessa 3.10.

GraphQL API-hakukielen lisäksi hyödynnettiin Reactia - eli JavaScript-kehystä esitystasolla. JavaScript-kehykset ovat olennainen osa modernia käyttöliittymän web-kehitystä, sillä ne tarjoavat kehittäjille jo kokeiltuja ja testattuja työkaluja skaalautuvien sekä interaktiivisen verkkosovellusten rakentamiseen (MDN Web Docs, 2021).

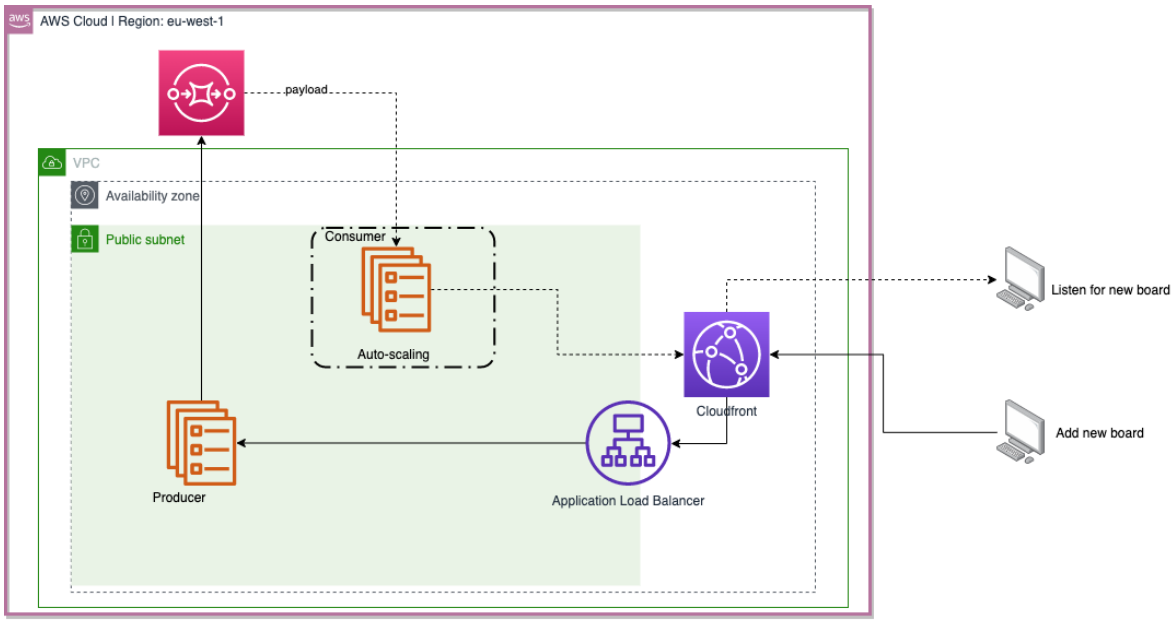
Kuten React virallisessa tunnisteessaan toteaa, React on kirjasto käyttöliittymien rakentamiseen, mutta React itsessään ei ole kehys – eikä sitä ole yksinomaan suunniteltu verkkoselaimelle. Sitä hyödynnetään muiden kirjastojen kanssa, jotta pystytään näyttämään kuvaa tietyissä ympäristöissä. Esimerkiksi React Nativea voidaan käyttää mobiilisovellusten rakentamiseen, React 360:tä voidaan hyödyntää virtuaalitodellisuussovellusten rakentamiseen. (MDN Web Docs, 2021)

Verkossa tai verkkoselaimen kanssa kehittäjät käyttävät Reactia yhdessä ReactDOM:n kanssa. React ja ReactDOM:ista keskustellaan usein samassa yhteydessä kuin muista todellisista web-kehyksistä – ja niitä käytetään samojen ongelmien ratkaisemiseen. Kun

siis Reactia kutsutaan ”kehykseksi” työskentelemme tämän puhekielisen version kanssa. (MDN Web Docs, 2021)

4.3 WebSocket tilaukset

Opinnäytetyössä aiemmin kappaleessa 3.5 esiteltyjen liikenteenjakajan uudelleen kytkennän lisäksi päädyttiin tekemään kuvan 16 mukainen ratkaisu. Oheisella ratkaisulla oli mahdollista luoda täysin tilaton sovellus - eli sovellus, joka ei lue eikä tallenna tietoja tilastaan. Ihanteellisessa ratkaisussa kontti luodaan tyhjästä, tekee tehtävänsä ja katoaa.



Kuva 16. Simple Queue Service arkkitehtuurissa

Arkkitehtuuriratkaisun tavoitteena oli siis saada aikaan sovellus, joka voidaan käynnistää, pysäyttää, poistaa ja tehdä uudelleen mahdollisimman pienellä vaikutuksella käyttäjäkokemukseen kehittäjän tai asiakkaan näkökulmasta.

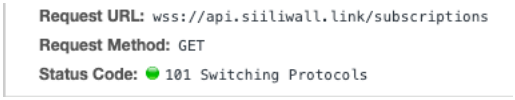
Kuvassa 16 näkyvää Amazon SQS (Simple Queue Service) palvelua käytetään projektissa irrotettujen työmäärien suorittamiseen. Työmäärä termi on kuvattu tarkemmin kappaleen kolme alussa. SQS on hallittu viestijonopalvelu, joka täten täyttää 3.4 kappaleessa esitellyn abstraktin tason jaetun vastuun mallin. Myös SQS itsessään on tarkemmin esitelty kappaleessa 3.4.

Kuvan 16 mukaisesti toisen käyttäjän lisätessä uuden taulun, tuottaja palvelun (producer) kautta tallennetaan tieto SQS:n jonoon. Kuluttaja (consumer) julkaisee uuden taulun tiedot (payload) eteenpäin subscriptions rajapinnan kautta. Tässä skenaariossa ei ole merkitystä

sille, mikä taustapalveluista palauttaa vaan riittää, että selaimella on yhteys auki liikenteenjakajalle.

Kuvassa 16 näkyy lisäksi kaksi eri palvelua tuottaja ja kuluttaja. Molemmat ovat rakentuneet samasta lähdekoodista, mutta palvelut käynnistyvät eri ympäristömuuttujilla. Ratkaisun tavoitteena oli pitää varsinkin kehityksen alkuvaiheessa kaikki lähdekoodit yhdessä paikassa, jotta voidaan tehdä nopeita muutoksia tarvittaessa.

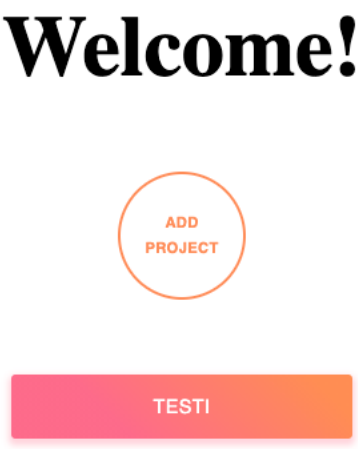
Kuvassa 17 onnistunut WebSocket yhteyden luonti selaimen kehittäjätyökalun näkökulmasta. Sovellustasolla käytössä oleva Apollo Server uudelleen kytkee yhteyden aina kuudenkymmenen sekunnin välein takaisin ja näin varmistaa, että yhteys ei missään vaiheessa katkea pysyvästi selaimelta. Ylimmällä rivillä näkyy kuormantasaajan osoite. Osoitteen ”/subscription” polun perusteella kuormanjakaja osaa jakaa liikennettä oikeille sovellustason taustapalveluille.



Kuva 17. Onnistuneen WebSocket yhteyden luonti selaimen työkalulla katsottuna

4.4 Yhteenveto

Projektin alussa kehitysryhmällä oli haasteita päästä kehityksen alkuun. Kun ympäristöt oli saatu pystyyn ja dokumentaatio ajan tasalle, niin projektiryhmä pääsi työssä paremmin eteenpäin. Pilviteknologioiden opettaminen kurssilaisille osoittautui haasteelliseksi, sillä projekti oli edellisen ryhmän toimesta kasvanut jo sen verran suureksi, että lähdekoodin ja uusien teknologioiden opettelussa kesti tovin. Kuvassa 18 AWS:ään onnistuneesti viety sovelluksen etusivu.



Kuva 18. Palvelun etusivu AWS:n CloudFrontin kautta tarjoiltuna

Jatkokehityksen osalta sovellustasolla olevaa logiikkaa voisi edelleen pilkkoa pienempiin osiin. Esimerkiksi tietokanta yhteyksiä tarvitsevan osuuden ei tarvitsisi olla julkisessa aliverkossa ja se olisi hyvä siirtää yksityiseen aliverkkoon. Tiimin tekemien data- ja sovellustason integraatiotestien ansiosta muutos olisi hyvinkin nopeasti toteutettavissa. Lisäksi aiemmassa kappaleessa esitelty tilausten kuluttaja olisi järkevää siirtää omaan projektiinsa, jotta myös sitä voidaan kehittää itsenäisenä komponenttina.

Näillä kehitysajatuksilla sovellus noudattaisi tehokkaammin mikropalveluarkkitehtuuriin liittyviä suunnittelukäytänteitä. Mikropalveluarkkitehtuurin avulla olisi helpompi toteuttaa hyvän AWS-kehyksen käytänteitä iteratiivisesti komponentti kerrallaan.

5 Pohdinta

Opinnäytetyön aiheen keksiminen oli kirjoittajalle pitkään hyvin haasteellista. Vaikka opiskelija on ollut jo työelämässä useamman vuoden, ei työelämästä löytynyt suoriltaan vetoapua aiheen valintaan. Päällimmäisenä tavoitteena oli löytää aihe, josta olisi aidosti hyötyä ammatilliselle kehittymiselle, eikä aihetta tarvitsisi erikseen tarkastuttaa työnantajalta.

Lopulta aihe oli hyvinkin järkeenkäypä opiskelijan työtausta huomioiden – ”Paikallisesta konesalista julkiseen pilveen”. Opiskelija on tehnyt aiheen parissa töitä useammassa projektissa ja haasteet ovat olleet usein samankaltaisia.

Täysin uutta projektia aloittaessa voidaan suoraan tehdä moderneja ratkaisuja ja suunnitella kaikki yhteensopivaksi pilvessä olevia palveluja hyödyntäen. Usein näin ei kuitenkaan pääse tekemään, vaan jollain tapaa joudutaan sekä siirtämään että suunnittelemaan jo olemassa olevia sovellus-, infrastruktuuri- ja tietoturvaratkaisuja.

Onnistuneeseen pilvitransformaatioon liittyy lukuisten vaihtoehtojen huolellinen puntarointi. Muutostarve nousee usein taloudellisten säästöjen tarpeesta, sen jälkeen palvelujen turvallisuudesta ja luotettavuudesta. Teoriaosuudessa esiin nousseet tulokset osoittautuivat luotettavaksi case-projektin aikana ja niitä hyödynnettiin useampaan otteeseen. Case-projektin aikana nousi myös kiinnostavia haasteita, joihin opiskelija perehtyi tarkemmin teoriaosuudessa.

Case-projektin alussa tutkittiin arkkitehtuurillisia vaihtoehtoja ja lopulta päädyttiin kolmivaiheisen arkkitehtuurin implementointiin mikropalveluarkkitehtuurin sijasta aikataulullisten syiden takia. Projektin alussa kirjoitettiin paljon testejä, jotta sovelluslogiikkaa olisi mahdollista tulevaisuudessa pilkkoa pienempiin osiin rikkomatta toiminnollisuuksia.

Kolmivaiheisen arkkitehtuurin avulla mahdollistettiin joka tapauksessa parempi tietoturva, esimerkiksi hyödyntämällä datatasoa, yksityisiä aliverkkoja, sekä rajattua suojausryhmää, jolla varmistuttiin siitä, että tietokantaan ei voinut yhdistää muualta kuin sovellustasolla oleva sovellus. Tietokantamigraatioiden avulla - joita käytiin kappaleessa 3.3 läpi - varmistuttiin siitä, että itse sovellus ja tietokanta ovat aina ajan tasalla tietokannan tauluista ja muutoksista.

Työn lopputuloksena syntynyt kattava teoriaperusta on tuonut myös päivittäisiä konkreettisia hyötyjä oppilaalle työmaailmassa. Moni aiheista laittoi miettimään, miten asioita voisi

tehdä paremmin ja turvallisemmin. Esimerkiksi järjestelmän ja infrastruktuurin rakentaminen niin, että kuljetaan aina kaksisuuntaisista ovista tekemällä pieniä parannuksia kerrallaan infrastruktuuriin ja sovelluksiin, jotta ei jouduta tilanteisiin, mistä ei pääse takaisin enää.

Taustaoletus huonommasta tietoturvasta liittyy lähes poikkeuksetta ohjeiden vastaiseen toimintaan (Parablu, 2015). Tietoturva kappaleessa esitelty Capital Onen tapaus osoittaa kuitenkin sen, että ihmiset ovat lähes aina tietoturvan heikoin lenkki. Pilvipalveluissa edellä mainitut asiat korostuvat, sillä tietoturva koostuu aina teknologioista, ihmisistä ja prosesseista. Täten organisaation ihmisten tietämyksellä ja kouluttautumisella hyvän arkkitehtuurin pilareihin voi olla valtaisat taloudelliset merkitykset.

Capital Onen tapaus osoittaa myös sen, että pilvipalveluissa identiteetistä on tullut pääsynhallinnan kulmakivi. Perinteisesti on ajateltu, että fyysisellä tietoturvalla estetään pääsy organisaation toimitiloihin, koska toimitilojen sisällä kaikki pääsevät sisäverkkoon ja sitä kautta palveluihin. Suojattavat kohteet on yleensä sijoitettu joko omiin tiloihin tai ulkoistusten yleistyessä palveluntarjoajan tiloihin.

Julkinen pilvi muuttaa tämän kaiken, sillä kaikki tiedot ja järjestelmät viedään sellaiseen paikkaan, johon organisaatiolla itsellään ei ole pääsyä. Samaan aikaan tietoihin kuitenkin halutaan päästä Internetin yli millä tahansa laitteella. Tämän vuoksi tietoturvaa ei ole mahdollista rakentaa IP-osoitteiden varaan, eikä perinteinen palomuureihin perustuva ajattelu toimi.

Käyttäjän identiteetti on käytännössä ainoa, mitä voidaan kontrolloida. Täten käyttäjän tunnistaminen pilvipalvelussa on äärimmäisen tärkeää, sillä koko tietoturva perustuu käytännössä siihen, että luotetaan käyttäjään, oli hän sitten ylläpitäjä tai loppuasiakas.

Yleisesti ottaen teorian vahva ymmärtäminen auttaa tekemään oikeita ratkaisuja käytännön töissä. Syvempi perehtyminen 2.4 kappaleessa esiteltyihin jaetun vastuun sekä pilvipalvelumalleihin on auttanut opiskelijaa kuvaamaan arkkitehtuurisia ratkaisuja paremmin myös muille ihmisille ja sitä kautta tuomaan paremmin esille pilvipalvelumallien välisiä eroja. Taulukoiden läpi käyminen on järkevää myös organisaatioille, jotta tehdään selväksi kuka vastaa mistäkin.

AWS:n ja julkisen pilven tärkeimmät perusteet ja palvelut käytiin case-projektin aikana läpi. Jokainen aiheista on kuitenkin erittäin laajoja ja projektissa itsessään oli paljon opeteltavaa opiskelijoille, joten syvempi perehtyminen ei tällä kertaa ollut mahdollista. Täten

projektin aikana keskityttiin ennen kaikkea vahvaan ja hyvään perustekemiseen, joihin hyvän arkkitehtuurin pilarit kuitenkin liittyvät. Näin vältettiin suurimmat ongelmat taloudellisesti, turvallisuuden ja saavutettavuuden osalta, sekä luotiin pohjaa opiskelijoiden tulevaisuuden opeille.

Opinnäytetyön kirjoittaneelle case-projekti oli hyvää kertausta ja samalla oivallinen mahdollisuus opettaa AWS:n perusteita muille eli tässä tapauksessa Haaga-Helian oppilaille. Projektin aikana löytyi myös uusia haasteita kuten WebSocket toiminnallisuuden kehittäminen, jota tarkemmin esiteltiin kappaleessa 4.3.

Kuormanjakaja, kontit ja jonopalvelu SQS ovat hyvin usein tärkeitä komponentteja, kun suunnitellaan ratkaisuarkkitehtuureja AWS:n avulla ja jaetaan liiketoiminta logiikkaa pienempiin osiin. Kuormanjakajan, konttien ja SQS:n avulla suunnitellut ja toteutetut WebSocket tilaukset olivat täten hyvin oleellisia asioita henkilökohtaisen oppimisen kannalta. Itsenäisten palveluiden tekeminen ja ohjelmisto logiikan irrottaminen ylipäätänsä on myös oleellista oppia.

Opinnäytetyön käynnistyessä tammikuun alussa teoriaosuus oli hyvin ylätasoista, mutta case-projektin ja kevään edetessä pidemmälle myös teoriaosuuden luonne muuttui ja painopiste siirtyi konkreettisemmalle tasolle. Osasta otsikoista ja kappaleista tuli hyvin mielenkiintoista pohdintaa, sillä ne syntyivät projektin tarpeista, mutta ne olivat hengeltään hyvin teknisiä, verrattuna aiemmin tuotettuun materiaaliin.

Jatkokehityksen kannalta opinnäytetyön aihetta olisi voinut rajata vielä tarkemmin. Pilvitransformaatio aiheena on kuitenkin erittäin laaja, joten jokaista näkökulmaa on mahdotonta käsitellä yhden opinnäytetyön aikana. Aiheen ollessa laaja, menee monen asian taustoittamiseen paljon tilaa ja aikaa.

Eräs tapa rajata opinnäytetyön aiheita olisi voinut olla se, että otsikot ja kappaleet pohjautuisivat kaikki puhtaasti case-projektin aikana syntyneisiin tietotarpeisiin. Tällöin näkökulmat organisaation kannalta olisivat todennäköisesti jääneet syntymättä, sillä kouluprojektien aikana täytyy harvemmin miettiä kovinkaan laajaa perspektiiviä tai vaikutuksia, kun kyseessä on vain yksittäisen sovelluksen vieminen julkiseen pilveen. Case-esimerkin tapauksessa esimerkiksi asiakastiedon fyysisellä sijainnilla ei ole kovinkaan suurta merkitystä, koska sellaista siinä ei varsinaisesti ole. Tästä olisi saattanut koitua myös aikataulullisia haasteita.

Tarkempi rajaus teknisemmän ja ylätason teorian välillä olisi voinut olla toimiva. Esimerkiksi suojausryhmien läpikäynti on hyvin teknistä, mutta toisaalta case-projektin ja minkä tahansa muunkin sovelluksen kannalta erittäin oleellinen asia. Rajaviivan vetäminen tässä kohtaa oli siis haasteellista.

Vaikka työn rajaaminen osoittautui haasteelliseksi, yhteenvetona opinnäytetyö kuitenkin vastaa alkuperäiseen tutkimuskysymykseen - miten siirrytään paikallisesta konesalista julkiseen pilveen - lukuisilta kanteilta. Julkiseen pilveen siirtymisen suurimpia haasteita käsitellään niin ylätasolla kuin teknisesti.

Opinnäytetyö esittelee myös ratkaisuja tapauksiin, jossa julkiseen pilveen siirtyminen ei ole joko osittain tai täysin mahdollista. Teoriaosuuden lainalaisuudet pätevät myös vaihtoehtoisissakin ratkaisuissa ja case-esimerkin kolmivaiheinen arkkitehtuuri toimii esimerkiksi yksityisessä pilvipalvelussa hyvänä vaihtoehtona. Myös WebSockettien kanssa käytetty SQS jonopalvelu on mahdollista korvata lukuisilla eri vaihtoehdoilla paikallisessa konesalissa.

Lähteet

Amanda 2018, Jun 26,-last update, *Database Migration: What It Is and How to Do It*. Available: https://www.cloudbees.com/blog/database-migration/ [2021, Feb 28,].

Anderson, N. 2007, 16.1.-last update, *Netflix offers streaming movies to subscribers*. Available: https://arstechnica.com/uncategorized/2007/01/8627/ [2021, Feb 7,].

Archer, J., Cullinane, D., Puhlmann, N., Boehme, A., Kurtz, P. & Reavis, J. 2016, -04-15-last update, *Security Guidance For Critical Areas Of Focus In Cloud Computing V3.0* [Homepage of Cloud Security Alliance], [Online]. Available: https://web.archive.org/web/20160415105252/https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf.

AWS 2016, *AWS re:Invent 2016: Tuesday Night Live with James Hamilton*.

AWS a, , *Amazon CloudFront Key Features* . Available: https://aws.amazon.com/cloudfront/features/ [2021, Mar 28,].

AWS b, , *Amazon Simple Queue Service*. Available: https://aws.amazon.com/sqs/ [2021, Feb 20,].

AWS c, , *Listeners for your Application Load Balancers*. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-listeners.html [2021, Apr 19,].

AWS d, , *Shared Responsibility Model*. Available: https://aws.amazon.com/compliance/shared-responsibility-model/ [2021, Feb 9,].

AWS e, , *What is Amazon EC2 Auto Scaling?*. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html [2021, -04-01].

AWS f, , *What is Cloud Computing*. Available: https://aws.amazon.com/what-is-cloud-computing/ [2021, Feb 16,].

AWS g, , *Working with stacks*. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html [2021, Apr 19,].

Barrett, D. 2020, *Capital One fined $80 million for 2019 hack of 100 million credit card applications*.

Browne, J. 2009, Jan 11,-last update, *Brewer's CAP Theorem*. Available: https://www.julianbrowne.com/article/brewers-cap-theorem [2021, Feb 13,].

Capital One 2019, Sep 23,-last update, *Information on the Capital One Cyber Incident* . Available: https://www.capitalone.com/facts2019/ [2021, Feb 8,].

Carlson, B. 2020, -07-09T13:46:05-07:00-last update, *What's New in the Well-Architected Operational Excellence Pillar?*. Available: https://aws.amazon.com/blogs/architecture/whats-new-in-the-well-architected-operational-excellence-pillar/ [2021, Feb 7,].

Carlson, B., Steele, J., King, R. & Fitzsimons, P. 2020, July-last update, *Operational Excellence Pillar*. Available: https://d1.awsstatic.com/whitepapers/architecture/AWS-Operational-Excellence-Pillar.pdf?ref=wellarchitected-ws [2021, Feb 9,].

Castgna, R. & Lelii, S. , *What is Cloud Storage and How Does It Work?*. Available: https://searchstorage.techtarget.com/definition/cloud-storage [2021, Feb 16,].

Chism, B. & Veksler, C. 2017, 2.-last update, *Maximizing Value with AWS*. Available: https://d1.awsstatic.com/whitepapers/total-cost-of-operation-benefits-using-aws.pdf [2021, Feb 10,].

CHM 2021, , *Internet History of 1970s | Internet History | Computer History Museum*. Available: https://www.computerhistory.org/internethistory/1970s/ [2021, Apr 19,].

Citrix a, , *What is a Cloud Service?*. Available: https://www.citrix.com/glossary/what-is-a-cloud-service.html [2021, Feb 16,].

Citrix b, , *What is Cloud Networking?*. Available: https://www.citrix.com/fi-fi/glossary/cloud-networking.html [2021, Feb 16,].

Cloudflare , *What Is a Virtual Private Cloud (VPC)?*. Available: https://www.cloudflare.com/learning/cloud/what-is-a-virtual-private-cloud/ [2021, Feb 21,].

CloudHealth 2019, Jul 14,-last update, *How The AWS Shared Responsibility Model Works*. Available: https://www.cloudhealthtech.com/blog/how-aws-shared-responsibility-model-works [2021, Feb 20,].

CloudPassage , *Shared Responsibility Model Explained*. Available: https://cloudsecurityalliance.org/blog/2020/08/26/shared-responsibility-model-explained/ [2021, Feb 13,].

Cubrilovic, N. 2006, 24.8.-last update, *Almost Exclusive: Amazon Readies Utility Computing Service*. Available: https://social.techcrunch.com/2006/08/24/exclusive-amazon-readies-utility-computing-service/ [2021, Feb 7,].

Demichelis, Y. 2020, Sep 16,-last update, *Websocket on AWS with ALB and ECS*. Available: https://techholding.co/blog/aws-websocket-alb-ecs/ [2021, Feb 24,].

Eliot Seth, Hornsby, A., Fitzsimons, P., Lester, R., Miller, R. & Richards, S. 2020, Dec 7,-last update, *Reliability Pillar*. Available: https://d1.awsstatic.com/whitepapers/architecture/AWS-Reliability-Pillar.pdf.

Furrier, J. 2015, -01-30T16:53:55.219Z-last update, *The Story of AWS and Andy Jassy's Trillion Dollar Baby*. Available: https://medium.com/@furrier/original-content-the-story-of-aws-and-andy-jassys-trillion-dollar-baby-4e8a35fd7ed [2021, Feb 7,].

Gilbert, S. & Lynch, N. 2012, February-last update, *Perspectives on the CAP Theorem*. Available: http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf [2021, Mar 15,].

Glass, E. 2020, Nov 2,-last update, *A General Introduction to Cloud Computing*. Available: https://www.digitalocean.com/community/tutorials/a-general-introduction-to-cloud-computing [2021, Feb 14,].

Godden-Payne, C. 2019, Sep 19,-last update, *Autoscaling Fargate tasks, outside of a load balancer*. Available: https://craig.goddenpayne.co.uk/task-autoscaling-fargate/ [2021, Apr 19,].

Helsingin Yliopisto 2019, , *Tietokantamigraatiot*. Available: https://web-palvelinohjelmointi-s19.mooc.fi/ekstra/tietokantamigraatiot [2021, Feb 21,].

Hinchcliffe, D. 2006, 9.3.-last update, *It's official: Google acquires Writely*. Available: https://www.zdnet.com/article/its-official-google-acquires-writely/ [2021, Feb 7,].

IBM 2020, -11-11-last update, *What is a mainframe*. Available: https://www.ibm.com/it-infrastructure/z/education/what-is-a-mainframe [2021, Apr 18,].

IBM Cloud Education 2020a, -12-02-last update, *Containers*. Available: https://www.ibm.com/cloud/learn/containers [2021, Feb 21,].

IBM Cloud Education 2020b, -09-01-last update, *IaaS (Infrastructure-as-a-Service)*. Available: https://www.ibm.com/cloud/learn/iaas [2021, Feb 21,].

IBM Cloud Education 2020c, -10-30-last update, *What is Three-Tier Architecture*. Available: https://www.ibm.com/cloud/learn/three-tier-architecture [2021, Mar 6,].

Karnes, K.C. 2020, 18.8.-last update, *What is Hypergrowth and How Can You Achieve It?*. Available: https://clevertap.com/blog/hypergrowth/ [2021, Feb 7,].

kaspersky 2021, -01-13T17:56:50Z-last update, *What is an IP Address – Definition and Explanation*. Available: https://www.kaspersky.com/resource-center/definitions/what-is-an-ip-address [2021, Apr 17,].

Kleppmann, M. 2015, "A Critique of the CAP Theorem", .

Klipfolio , *What is a KPI?*. Available: https://www.klipfolio.com/resources/articles/what-is-a-key-performance-indicator [2021, Feb 14,].

Lausuntopalvelu.fi 2018, , *Lausuntopalvelu.fi*. Available: https://www.lausuntopalvelu.fi/FI/Proposal/Participation?proposalId=b1e80d52-172f-4f87-ba4d-c75306e51179 [2021, Apr 10,].

Leavitt, N. 2010, Feb-last update, *Will NoSQL Databases Live Up to Their Promise?*. Available: http://www.leavcom.com/pdf/NoSQL.pdf [2021, 11 Apr,].

Lefkowitz, M. 2019, Mar 8,-last update, *With help from AI, microservices divvy up tasks to improve cloud apps*. Available: https://news.cornell.edu/stories/2019/03/help-ai-microservices-divvy-tasks-improve-cloud-apps [2021, Feb 21,].

Lester, R., Carlson, B., Potter, B., Pullen, E., Eliot, S., Besh, N., Steele, J., King, R., Rifkin, E., Ramsay, M., Paddock, S. & Hughes, C. a, , *Introduction*. Available: https://wa.aws.amazon.com/wat.introduction.wa-intro.en.html [2021, Feb 7,].

Lester, R., Carlson, B., Potter, B., Pullen, E., Eliot, S., Besh, N., Steele, J., King, R., Rifkin, E., Ramsay, M., Paddock, S. & Hughes, C. b, , *On Architecture*. Available: https://wa.aws.amazon.com/wat.onarchitecture.wa-onarch.en.html [2021, Feb 7,].

Logistiikan Maailma , *Kokonaiskustannusajattelu – Logistiikan Maailma*. Available: https://www.logistiikanmaailma.fi/osto-ja-myynti/hankintatoimi-ja-ostotoiminta/kokonaiskustannusajattelu/ [2021, Apr 18,].

Lord, N. 2016, -06-13T14:33:30-04:00-last update, *Data Protection: Data In transit vs. Data At Rest*. Available: https://digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest [2021, Apr 17,].

MDN Web Docs 2021, Jan 22,-last update, *Understanding client-side JavaScript frameworks*. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks [2021, Feb 1,].

Microsoft Azure a, , *What is a Private Cloud*. Available: https://azure.microsoft.com/en-us/overview/what-is-a-private-cloud/ [2021, Feb 2,].

Microsoft Azure b, , *What is a Public Cloud*. Available: https://azure.microsoft.com/en-us/overview/what-is-a-public-cloud/ [2021, Feb 7,].

Miller, R. 2016, 2.7.-last update, *How AWS came to be*. Available: https://social.techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/ [2021, Feb 7,].

Miller, R. 2015, Nov 24,-last update, *AWS Lambda Makes Serverless Applications A Reality*. Available: https://social.techcrunch.com/2015/11/24/aws-lamda-makes-serverless-applications-a-reality/ [2021, Feb 20,].

Morris, K. 2020, , *Infrastructure as Code, 2nd Edition*. Available: https://learning.oreilly.com/library/view/infrastructure-as-code/9781098114664/titlepage01.html [2021, Mar 13,].

Morrison, J.C. 2019, Aug 1,-last update, *The Technical Side of the Capital One AWS Security Breach*. Available: https://start.jcolemorrison.com/the-technical-side-of-the-capital-one-aws-security-breach [2021, Feb 8,].

NetApp , *What Is Hybrid Cloud?*. Available: https://www.netapp.com/hybrid-cloud/what-is-hybrid-cloud/ [2021, Feb 7,].

Nummela, T. 2010, *Suurkoneet, yhä täällä.*, Haaga-Helia Ammattikorkeakoulu.

Oracle 2017, , *Licensing Oracle Software in the Cloud Computing Environment*. Available: https://www.oracle.com/us/corporate/pricing/cloud-licensing-070579.pdf [2021, Apr 18,].

Parablu 2015, -10-15T10:58:44+00:00-last update, *Security Considerations when using the Public Cloud*. Available: https://parablu.com/security-considerations-when-using-the-public-cloud/ [2021, Apr 24,].

Perry, Y. 2019, Sep 17,-last update, *AWS Availability Zones: Which Zone is Right for You?*. Available: https://cloud.netapp.com/blog/aws-availability-zones-architecture-how-to-select [2021, Apr 18,].

Pickell, D. 2018, Nov 16,-last update, *Structured vs Unstructured Data – What's the Difference?*. Available: https://learn.g2.com/structured-vs-unstructured-data [2021, Apr 11,].

Postel, J. 1980, Jan-last update, *Internet Protocol*. Available: https://tools.ietf.org/html/rfc791 [2021, Apr 17,].

Priyam, P. 2018, *Cloud Security Automation,* Packt Publishing, O'Reilly.

Pullen Eric, Fitzsimons, P., Lépine, J. & Slasky, R. 2020, Oct 5,-last update, *Performance Efficiency Pillar*. Available: https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#welcome.

Raeste, J. 2020, -12-09T02:00:00.000+02:00-last update, *Verkkokauppa Amazonin kasvu rahoitetaan pilvipalveluilla, joista ei juuri puhuta*. Available: https://www.hs.fi/talous/art-2000007670266.html [2021, Feb 7,].

Regalado, A. 2011, 31.10.-last update, *Who Coined Cloud Computing?*. Available: https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/ [2021, Feb 7,].

Reselman, B. 2020, September 14,-last update, *TOGAF and the history of Enterprise Architecture*. Available: https://www.redhat.com/architect/togaf.

Rodney, L., Carlson, B., Potter, B., Pullen, E., Eliot, S., Besh, N., Steele, J., King, R., Rifkin, E., Ramsay, M., Paddox, S. & Hughes, C. a, , *Milestone*. Available: https://wa.aws.amazon.com/wat.concept.milestone.en.html [2021, Feb 15,].

Rodney, L., Carlson, B., Potter, B., Pullen, E., Eliot, S., Besh, N., Steele, J., King, R., Rifkin, E., Ramsay, M., Paddox, S. & Hughes, C. b, , *The Review Process*. Available: https://wa.aws.amazon.com/wat.thereviewprocess.wa-review.en.html [2021, Feb 15,].

Rodney, L., Carlson, B., Potter, B., Pullen, E., Eliot, S., Besh, N., Steele, J., King, R., Rifkin, E., Ramsay, M., Paddox, S. & Hughes, C. c, , *Workload*. Available: https://wa.aws.amazon.com/wat.concept.workload.en.html [2021, Feb 15,].

Schmidt, E. 2006, 9.8.-last update, *Search Engine Strategies Conference*. Available: https://www.google.com/press/podium/ses2006.html.

Scott, S. 2019, -09-27T10:43:30+00:00-last update, *AWS Security Groups: Instance Level Security*. Available: https://cloudacademy.com/blog/aws-security-groups-instance-level-security/ [2021, Feb 21,].

Scott, S. 2017, -09-07T16:01:36+00:00-last update, *AWS Shared Responsibility Model: Cloud Security*. Available: https://cloudacademy.com/blog/aws-shared-responsibility-model-security/ [2021, Feb 20,].

Scroxton, A. 2020, Jan 20,-last update, *Exposed AWS buckets again implicated in multiple data leaks*. Available: https://www.computerweekly.com/news/252476870/Exposed-AWS-buckets-again-implicated-in-multiple-data-leaks [2021, Feb 13,].

Software Advisory Service , *A Beginners Guide to Capex vs Opex*. Available: https://www.softwareadvisoryservice.com/en/whitepapers/a-beginners-guide-to-capex-vs-opex [2021, Feb 7,].

Sparks, D. 2020, -02-06T09:35:00-05:00-last update, *Amazon's Record 2019 in 7 Metrics*. Available: https://www.fool.com/investing/2020/02/06/amazons-record-2019-in-7-metrics.aspx [2021, Feb 7,].

Statista Research Department 2021, Jan 12,-last update, *Amazon Web Services revenue 2019*. Available: https://www.statista.com/statistics/233725/development-of-amazon-web-services-revenue/ [2021, Feb 7,].

Techopedia , *What is On-Demand Service? - Definition from Techopedia*. Available: http://www.techopedia.com/definition/26711/on-demand-service [2021, Apr 18,].

TechTarget Contributor 2010, Aug 10,-last update, *What is data at rest?*. Available: https://searchstorage.techtarget.com/definition/data-at-rest [2021, Apr 11,].

Trappler, T. 2010, *If It's in the Cloud, Get It on Paper: Cloud Computing Contract Issues*, Educause Brief.

Vance, A. 2005, 20.7.-last update*, IBM preps big iron fiesta*. Available: https://www.theregister.com/2005/07/20/ibm_mainframe_refresh/ [2021, Feb 7,].

Vanderweide, D. 2019, -04-15T19:13:04+00:00-last update*, OpEx vs. CapEx: The Real Cloud Computing Cost Advantage*. Available: https://www.10thmagnitude.com/opex-vs-capex-the-real-cloud-computing-cost-advantage/ [2021, Feb 7,].

Verizon *, 2019 Data Breach Investigations Report*. Available: https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf.

Watts, S. 2020, Apr 24,-last update*, ACID Explained: Atomic, Consistent, Isolated & Durable*. Available: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/.

Weinberg, G. 1986, *The Secrets of Consulting: A Guide to Giving and Getting Advice Successfully,* Dorset House Publishing Company.

Whitaker, J. 2020, Dec 17,-last update*, The Complete Guide to Cloud Transformation*. Available: https://cloud.netapp.com/blog/azure-anf-blg-the-complete-guide-to-cloud-transformation [2021, Apr 10,].

Wilkins, M. 2019, *Learning Amazon Web Services (AWS): A Hands-On Guide to the Fundamentals of AWS Cloud,* Addison-Wesley Professional.