Tampereen yliopisto

Pieti Lehtinen

MONITOROINTITYÖKALUJEN KÄYTTÖ INTEGRAATIOALUSTOJEN VALVONTAAN

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Kesäkuu 2023

TIIVISTELMÄ

Pieti Lehtinen: Monitorointityökalujen käyttö integraatioalustojen valvontaan
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Kesäkuu 2023

Tämän pro gradu -tutkielman tarkoituksena on esitellä ja vertailla erilaisia monitorointiratkaisuja integraatioalustojen automaattiseen valvontaan ja niiden kykyä avustaa integraatioratkaisuja ylläpitäviä tahoja ongelmatilanteiden seurantaan ja ratkaisuun. Monitorointi tarkoittaa tässä yhteydessä tapaa seurata automaattisesti integraatioiden ja ajoympäristöjen toimintakykyä ja helpottaa virhetilanteiden selvitystä. Yleisellä tasolla esitellään myös, mitä tarkoitetaan järjestelmäintegraatiolla, sekä esitellä markkinoilla Suomessa toimivia integraatiosovelluksia, joita myös integraatioalustoiksi kutsutaan.

Systemaattisen integroinnin ja integraatioalustojen käyttö ja kriittisyys jatkaa kasvuaan yritysten ja muiden toimijoiden jatkuvien digitalisaatiopyrkimysten myötä. Kun ICT-ekosysteemeihin tuodaan uusia järjestelmiä, pyritään samalla rakentamaan moderni ja skaalautuva IT-arkkitehtuuri, jolla voidaan palvella nykyistä toimintaa, mutta myös mahdollistetaan tulevaisuuden kehityshankkeet.

Kriittisyyden myötä myös integraatioalustojen valvontaan voidaan panostaa erilaisin monitorointiratkaisuin. Tämän tutkielman pohjana toimii erään yrityksen uuden toiminnanohjausjärjestelmän käyttöönotto. Uuden järjestelmän käyttöönoton myötä syntyi tarve ottaa käyttöön uusi integraatioalusta, Dell Boomi ja tarve toteuttaa myös keskitetty integraatio- ja palvelinmonitorointiratkaisu. Yrityksen tarpeita kartoitettiin haastatteluiden avulla, joiden pohjalta rakennettiin vaatimuslista mahdolliselle monitorointityökalulle. Tutkielmassa tutustuttiin muutamaan potentiaaliseen työkaluun, joiden toiminnallisuuksia ja ominaisuuksia verrattiin yrityksen esittämiin tarpeisiin. Tätä vertailua hyväksikäyttäen pystytään rakentamaan arkkitehtuuri, jossa integraatioalustan monitorointia ja ylläpitoa tuetaan erillisellä monitorointiratkaisulla. Lisäksi lopputulemana löytyi kyseisen yrityksen tarpeisiin sopivin työkalu sekä ensiaskeleet käyttöönottoa varten.

Tutkielman tuloksista voidaan päätellä, että suurin osa markkinoilla olevista monitorointityökaluista vastaavat esimerkkiyrityksen tarpeita, mutta Splunk-alusta näytti näistä vaihtoehdoista sopivimmalta. Sen tekninen arkkitehtuuri sopivat olemassa olevaan IT-arkkitehtuuriin ja sen avulla yritys pystyy seuraamaan toiminnalleen kriittistä tilausliikennettä. Lisäksi yritys voi käyttää Splunk-alustaa palvelimiensa hyvinvoinnin automaattiseen seurantaan, sekä tarvittaessa hälyttämään virhetilanteista oikeita tahoja.

Avainsanat: Monitorointi, integraatioalusta, IPaaS, monitorointityökalu, integraatio

1 Johdanto

Lyhykäisyydessään järjestelmäintegraatio tarkoittaa kahden tai useamman IT-järjestelmän välisen viestinnän toteuttamista. Tämä voidaan toteuttaa käytännössä käyttämällä pistemäisiä end-to-end-ratkaisuja, joissa esimerkiksi lähettävään järjestelmään on ohjelmoitu jo suoraan viestin lähettäminen vastaanottavaan järjestelmään. Keskitetty integraatioalusta on ohjelmisto, johon nämä järjestelmän sisäiset viestintämetodit on tuotu, jotta niiden hallinta ja seuranta on helpompaa ja järjestelmällisempää. Lisäksi voidaan toteuttaa jonkinlaista hybridimallia näiden ääripäiden välillä.

Integration Platform as a Service (iPaaS, integraatioalusta) -tuotteet tarjoavat mahdollisuuden hankkia ohjelmisto, jonka avulla voidaan mm. kehittää integraatioprosesseja käyttäen uudelleenkäytettäviä komponentteja. Lisäksi tuotteiden ominaisuuksien jatkokehitys ja usein myös ajoympäristö (esimerkiksi julkipilvessä) sisältyy suoraan ohjelmiston lisenssihintaan, jolloin yritysten ei tarvitse tehdä kehitystyötä itsenäisesti tai alihankkimalla. Eräs esimerkki tälläisestä iPaaS-tuotteesta on Dell Boomi, jonka tässä opinnäytetyössä esitelty asiakasyritys on valinnut omaksi integraatioalustakseen.

Dell Boomi rakentuu kahdesta komponentista: julkipilvessä olevasta AtomSphere-ohjauskeskuksesta sekä kahdentyyppisistä ajoympäristöistä: yksittäisistä Atomeista (Atom) ja usean Atomin muodostamista klustereista Molekyyleistä (Molecule). Nämä ajoympäristöt voidaan asentaa joko lokaalisti tai virtuaalikoneisiin, mutta varsinainen kehitys-, ylläpito- ja ohjaustyö tapahtuu aina Boomin Atomspheressä.

Toteutettiin järjestelmien välinen viestintä millä tavalla tahansa, on tuo viestintä yrityksille usein hyvin kriittistä. Kun kyse on esimerkiksi lasku- tai tilausliikenteestä, tulee viestinnän virhetilanteisiin pystyä puuttumaan hyvin lyhyellä vasteajalla. Näitä tilanteita varten yrityksen ylläpitotiimi tarvitsee jonkinlaisen monitorointiratkaisun, jonka avulla voi helposti valvoa liikenteen sujuvuutta ja toimivuutta, sekä hälyttää tarpeellisia osapuolia reaaliaikaisesti.

Integraatioalustoilla on usein sisäänrakennettuja monitorointiratkaisuja, mutta usein ylläpitävien henkilöiden on vaikeaa löytää keskitetysti kaikki tarpeellinen tieto yhdestä paikasta. Myös tietynlaisten virheiden nostaminen esimerkiksi sähköpostilla, tekstiviestillä tai pikaviestimellä on usein vaikeaa tai mahdotonta työkalusta riippuen.

Tätä opinnäytetyötä varten seurattiin integraatiokehitystiimin integraatioalustan käyttöönottoa ja projektin myötä nousseita ongelmatilanteita. Projektin keskeisenä teemana oli vanhasta integraatioalustasta siirtyminen uuteen tuotannonohjausjärjestelmän uudistamisen ohessa. Tällöin myös olemassa olevat prosessit muuttuivat käytännössä koko arvoketjun matkalta, mikä mahdollisti modernimman teknologian, mutta myös modernimman ylläpitoprosessin kehittämisen.

Kohdeyrityksen integraatioasiantuntija ja integraatioarkkitehti ovat huomanneet työskennellessään IPaaS-tuote Dell Boomin kanssa sen sisäänrakennettujen monitorointiratkaisujen olevan puutteellisia tai vaikeaselkoisia, jolloin keskitetylle monitorointisovellukselle on nähtävissä selkeä tarve. Dell Boomi kirjoittaa runsaasti lokia kiintolevylle, jolloin erillinen lokitiedostoihin perustuva monitorointityökalu voi niitä hyödyntää. Tätä varten on tehty selvitystyö siitä, onko olemassa valmista työkalua, joka on todettu potentiaaliseksi työkaluksi tuottamaan tarvittavia ominaisuuksia.

Selvitystyötä varten pidetyssä asiantuntijahaastattelussa nousi esiin puutteita toiminnallisuuksissa nykyisissä ratkaisuissa. Muun muassa ongelmalliseksi koettiin yksittäisten virhetilanteiden selvittelyn tilanne; oliko virhe jo rauennut käyttäjän toimesta vai oliko se edelleen aktiivinen. Vastaavasti joissain tilanteissa Boomin käyttöliittymä ei välttämättä nostanut virhettä onnistuneesti seurantapaneeliin, koska se ei tietyistä teknisistä syistä onnistunut.

Selvitystyön perusteella löytyneiden tarpeiden perusteella voitiin määritellä keskeiset kysymykset, joihin opinnäytetyö voi vastata. Nämä kysymykset ovat: onko integraatioalustan ympärille mahdollista rakentaa ulkoinen monitorointijärjestelmä? Miten sen automatisointi toteutetaan ja kuinka yksinkertaista sen käyttöönotto on? Mitä hyötyä keskitetystä monitorointijärjestelmästä on integraatioalustan käyttäjille ja tuoko se lisäarvoa verrattuna alustan omiin työkaluihin? Vastaavatko alustan ominaisuudet niitä tarpeita, jotka asiakasyritys on tuonut esille tarvekartoituksen aikana?

Opinnäytetyö on tapaustutkimus ja siinä käytetään kvalitatiivisia tutkintametodeja. Työssä käytetään myös tieteellistä kirjallisuutta, konferenssimateriaalia, web-julkaisuja ja asiantuntijalausuntoja, sekä -haastatteluita ja blogeja tuottamaan perusteet sille, miksi integraatiot ovat oleellinen osa IT-arkkitehtuuria ja miten monitorointia tulisi lähestyä.

Opinnäytetyön tarkoitus on selvittää, onko asiakasyrityksen teknisesti mahdollista ottaa käyttöön valittu keskitetty monitorointiratkaisu ja dokumentoida alustava arkkitehtuuri, jonka avulla mahdollinen käyttöönotto voitaisiin toteuttaa.

Toisessa luvussa esitellään yleisellä tasolla integraatiosovelluksia ja tapoja tehdä järjestelmäintegrointia. Lisäksi kirjataan mitä etuja hyvin toteutettu integraatioarkkitehtuuri tuo yrityksille, miten arkkitehtuuria voidaan toteutetta ja selvitetään ideologiaa sen taustalla. Kolmannessa luvussa esitellään mitä tarkoitetaan Integration Platform as a Service (IPaaS) -tuotteilla ja selvitetään näiden tuotteiden yleisiä toimintaperiaatteita. Lisäksi tehdään tarkempaa analyysiä tapaustutkimukseen liittyvästä alustasta Dell Boomista.

Neljännessä luvussa esitellään IPaaS-tuotteen käyttöä, sillä tehtävää kehitystä ja ongelmatilanteet, joita Dell Boomin nykyinen toteutus aiheuttaa tarkasteltavassa yrityksessä ja miten niitä voi ulkoisella monitorointityökalulla hoitaa. Viidennessä luvussa esitellään muutama monitorointityökalu, jotka tämän työn tekijä on etukäteen valinnut alalla

yleisimmin käytetyistä vaihtoehdoista. Kuudennessa luvussa esitellään aiempien lukujen esittelemiin tietoihin integraatioista ja monitoroinnista pohjaten tutkimuskysymykset, joihin vastataan seitsemännessä luvussa valitsemalla sopivin monitorointityökalu täyttämään esille tulleet ominaisuustoiveet. Lisäksi tuotetaan keskitetty monitorointisovellus ja dokumentoidaan sen tekeminen. Näin vastataan alkuperäisiin tutkimuskysymyksiin.

Viimeiseksi esitellään johtopäätökset siitä, miten keskitetty monitorointityökalu auttoi esitellyssä ongelmassa, tuodaan esille mahdolliset jatkokehitystarpeet, sekä esitellään opinnäytetyön tekijän omat ajatukset työkalun hyödyllisyydestä ja siitä, miten työkalua voisi jatkokehittää kaupallisiin tarkoituksiin.

2 Integraatiot ja niiden tyypit

Mitä integraatio on ja miksi se on yhä useamman suuryrityksen digihankkeissa omana kokonaisuutenaan, eikä vain pieni sivujuonne suuremmissa hankkeissa? Byrne [2018] kertoo, kuinka jokainen yritys on nykyisin myös sovellustalo. Digitaalinen vallankumous mahdollistaa myös suuret yhtiöt muuttumaan ja muuttamaan toimiaan jatkuvassa turbulenssissa olevien markkinoiden mukana, jolloin uusia prosesseja, tuotteita ja liiketalousmalleja syntyy jatkuvasti.

Vastaavasti perinteinen malli IT:stä omana yksikkönään on muuttunut, kun eri liiketoimintayksiköillä on omanlaisensa tarpeensa, joilla voi saavuttaa etulyöntiasema kilpailijoihinsa. Tällöin jokaisella tiimillä tulee olla pääsy tarvitsemiinsa digitaalisiin työkaluihin, jolloin IT-yksikön vastuulle jää tuottaa näitä palveluita.

Digitaalinen muutos vaatii Byrnen [2018] mukaan erityisesti dataa ja järjestelmiä, ihmisiä sekä prosesseja. Näiden tekijöiden väliin on perinteisesti muodostunut monoliitti, joka hoitaa integraatiot osapuolien välillä. Kehityksen kehityttyä on myös noussut tarve ketterämmälle, älykkäämmille integraatioille; jotta digitaalinen kehitys voi yrityksessä ottaa seuraavan loikan, tulee välineenkin pystyä reagoimaan dataan reaaliaikaisesti.

Yritykset tarvitsevat kattavan digitaalisen strategian, joka mahdollistaa innovatiivisuuden ja ketteryyden. Tämä tarkoittaa myös sitä, että integraatiot tulee toteuttaa siten, että kaikki osapuolet voivat integroitua toiseen osapuoleen ilman ongelmia; uusia liittymiä on pystyttävä luomaan tehokkaasti ja niitä pitää pystyä palvelemaan tilanteesta ja kontekstista huolimatta. [Weir ja Nemec, 2019].

Perinteisesti integraatioita on hoidettu ohjelmallisesti, jolloin kehittäjät ovat joko tehneet omia ohjelmiaan, jotka hoitavat tiedonsiirron tai muuten hyvin pistemäisesti; tieto siirretään pelkästään lähdejärjestelmästä kohdejärjestelmään, vaikka myös kolmas järjestelmä voisi tätä tietoa tarvita. Tämänkaltaista integrointia, esimerkiksi tietokannasta tietokantaan voi kutsua ”pieneksi integroinniksi” [Gulledgen, 2006]. Usein pistemäiset toteutukset (point-to-point) ovatkin tiettyyn tarpeeseen toteutettuja komponentteja tai suoraan ohjelmiin itseensä tehtyjä rajapintoja, kuten esimerkiksi tietokantaproseduuri, joka ottaa yhteyttä toiseen tietokantaan ja tekee datan siirron sinne.

Toinen, varsinkin suurempien yrityksien käytössä yleinen tapa integroida, on käyttää jotain middleware-tuotetta. Middlewarella tarkoitetaan kaikkia sellaisia palveluita, jotka tuottavat rajapintojen välille ”väliohjelmiston”, joka pystyy operoimaan kaikkien siihen liittyvien palveluiden ja rajapintojen kanssa huolimatta siitä, missä ympäristössä ne toimivat. [Bye ja Britton, 2004]

Kolmas selkeä kokonaisuus on näiden yhdistelmä, hybridimalli. Tällöin esimerkiksi voidaan tehdä mikropalveluita, jotka ovat itsenäisiä kokonaisuuksia. Toisin kuin point-to-point-ratkaisuja, näitä palveluita voidaan uudelleen käyttää useassa eri tilanteessa ja

näin ne palvelevat useampaa rajapintaa kahden yksittäisen rajapinnan sijaan. Hybridimallissa voidaan myös toteuttaa sekä point-to-point-liittymiä että middleware-tyypistä yhteistä integraatioalustaa.

Integration as a Service (iPaaS) -tuotteet tarjoavat mahdollisuuden ostaa tuote, joka mahdollistaa integroinnin useiden järjestelmien välillä. Gartner [Gartner, 2022] kuvailee iPaaSeja pilvipohjaisina palveluina, jotka mahdollistavat integraatioprosessien kehittämisen, suorittamisen ja ylläpidon. Näitä tuotteita voidaan käyttää yhdistämään on-premise ja pilvipohjaisia prosesseja, palveluita, applikaatioita ja dataa yrityksen tai useiden yrityksien sisällä.

Nykyaikaisessa IT-arkkitehtuurissa korostuu myös sovellusten tarjoamat rajapinnat, joiden avulla sovelluksen kanssa voi keskustella ohjelmallisesti. Usein nämä rajapinnat ovat myös yhteyspisteitä moderneille integraatioille suorien kantalinkkien vähentyessä. Rajapinnoista puhuessa käytetään termiä API (application programming interface) ja niiden ympärille on muodostunut liiketoimintamalleja. [De, 2017] Esimerkiksi Suomen liikenne- ja viestintävirasto Traficom tarjoaa maksullisen rajapinnan, josta sovelluskehittäjä voi hakea autojen teknisiä tietoja rekisterinumeroa käyttämällä.

APIen avulla sovellukset pystyvät tarjoamaan datansa tai toiminnallisuuksiansa hallitusti ulkoisille kehittäjille, jotka voivat käyttää näitä ominaisuuksia apunaan kehittäessään omia sovelluksiaan. Esimerkiksi De [2017] antaa Google Mapsin API:n paikannustietojen esittämiseen ja Amazonin API:n tuotetietojen hakemiseksi.

Toisaalta kokonaan uuden applikaation rakentaminen API:n varaan on myös mahdollista, De [2017] jatkaa esimerkkiään kuvaamalla sovelluksen, joka käyttää useiden eri matkatoimistojen hintatietoja tarjotakseen kuluttajalle mahdollisuuden etsiä halvin vaihtoehto lomamatkalleen.

Kun integraatioiden välittämä tieto on yhä enenemissä määrin välttämätön osa yrityksen liiketoimintaa, muodostuu näistä iPaaS-tuotteista myös kriittinen osa jokaisen IT-yksikön seuranta- ja ylläpitorutiinia. Tällöin myös niiden monitoroinnin tulee olla helppokäyttöistä, tehokasta ja selkää, jolloin virhetilanteet pystytään havaitsemaan nopeasti ja niihin reagointi tehostuu. Tämän työn tarkoitus onkin selvittää, miten iPaaS-tuotteen monitorointi voitaisiin keskittää tehokkaasti ja lisäarvoa valvontaan tuoden.

2.1 Yleiset integrointimallit

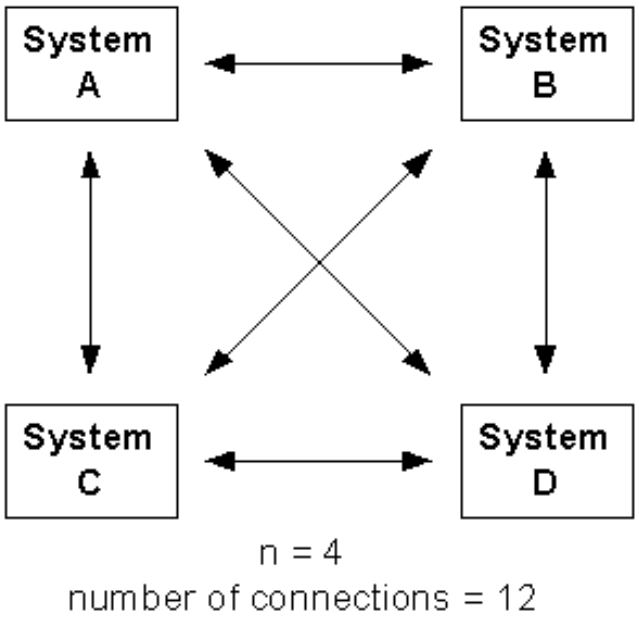
Dataa voidaan siirtää paikoista ja järjestelmistä toiseen monilla eri tavoilla, mutta yleisesti nämä tavat voidaan kuitenkin liitää johonkin yleiseen integrointimalliin. Näitä malleja ovat esimerkiksi Point-to-point, Enterprise Service Bus, Service Oriented Architecture, Microservices ja API architecture, joita tässä kohdassa esitellään.

2.1.1 Point-to-Point-malli

Lyhykäisyydessään Pisteestä pisteeseen (Point-to-point) -integrointi on integraatiomuodoista kaikkein yksinkertaisin. Tämä tyyli oli oikeastaan ensimmäinen, jota integroinnissa käytettiin, sillä se ei vaatinut muuta kuin yhteyden järjestelmien välillä. Sitä käytetään kahden pisteen yhdistämiseen, joko kevyellä transformaatiolla tai vain viestin välitykseen. Se voi kuitenkin johtaa tekniseen velkaan tai spagettikoodiin ja on siten riskialtis valinta väärinkäytettynä. [Guttridge ja Zakheim, 2018] Teknisellä velalla tarkoitetaan tilannetta, missä on tehty teknisiä kompromisseja tai nopeita korjaustoimenpiteitä akuuttiin ongelmatilanteeseen. Tälläisiä tilanteita voi syntyä esimerkiksi tiukkojen aikataulujen takia. Toisaalta käytetty teknologia on saattanut päästä vanhentumaan, jolloin ylläpito on vaikeaa tai ymmärrys ratkaisusta puuttuu. Spagettikoodilla tarkoitetaan lähdekoodia, jota on kehitetty ilman yhtenäistä struktuuria tai punaista lankaa, jolloin sitä on vaikea ymmärtää tai korjata myöhemmin.

Pisteestä pisteeseen arkkitehtuurissa eri järjestelmät integroidaan toisiinsa vain ja ainoastaan suoraan järjestelmästä toiseen. Tämän tyyppisiä toteutuksia tarvitaan, kun järjestelmän tarvitsee tehdä päivitys vain toiseen järjestelmään, jolloin lähettäjiä on yksi ja vastaanottajia on yksi, kuten esimerkiksi HR-tietokannan päivitys ERP-järjestelmän tiedoilla. [Newcomer, 2002]

Tämän mallin ongelmat tulevat kuitenkin hyvin nopeasti selville, jos integroitavia järjestelmiä on useita, jolloin yhteyksien määrä kasvaa kaavalla n(n-1), jossa n merkitsee järjestelmien lukumäärää. Kun jokainen yhteys lasketaan molempiin suuntiin kuvan 1 tavalla, tällöin esimerkiksi pelkästään neljällä järjestelmällä yhteyksien lukumäärä kasvaa 12:ta, kun ne vaihtavat dataa keskenään.



Kuva 1. Point-to-point-malli [Newcomer, 2002].

Lisäksi ongelmaksi tulee riippuvuus kahden järjestelmän välillä, koska rakenteellisesti viestin välitys on tiukasti nivoutunut yhteiseksi. Tällöin esimerkiksi toisen järjestelmän ollessa saavuttamattomissa, voi se aiheuttaa myös toiselle järjestelmälle ongelmia menetettyjen viestien tai muiden ongelmien myötä. [Newcomer, 2002]

Nykyaikaisessa integraatiotyössä on kuitenkin edelleen paikkansa pistemäisille toteutuksille. Varsinkin pilvinatiiveissa applikaatioissa ja pilvialustoissa erilaiset SaaS (Service as a Service) -palvelut tarjoavat valmiita keinoja integroida pistemäisesti eri alustojen välillä. Laajalti käytössä oleva asiakas- ja myyntihallintajärjestelmä Salesforce tarjoaa esimerkiksi valmiita yhteyspisteitä eri ERP:ien rajapintoihin [Salesforce, 2021], jolloin sen käyttöönotto voi olla huomattavasti helpompaa ja tehokkaampaa, kuin rakentaa laajempaa integraatiokokonaisuutta järjestelmien ympärille.

Pistemäiset toteutukset tarjoavat myös usein kevyemmän ja edullisemman ratkaisun, koska ne ovat toteutettavissa räätälimäisesti omiin tarpeisiin, mikäli järjestelmillä on omat rajapintansa. Tällöin tarvittava toiminallisuus voidaan paketoida yhteen pieneen sovellukseen tai integroitavan sovelluksen sisälle, esimerkiksi tietokannan sisällä ajettavaksi proseduuriksi. Tuolloin tarvittava data, kuten esimerkiksi varastosaldo, voidaan viedä varastohallintajärjestelmän tietokannasta suoraan toiminnanohjausjärjestelmään.

2.1.2 Enterprise Service Bus -malli

Jo 1990-luvulta lähtien on tiedetty, että yritykset, jotka pystyvät hallinnoimaan tehokkaasti eri järjestelmiensä välistä tiedonvälitystä saavat siitä kilpailuetua. Eri järjestelmien välinen keskustelu tuntuu kuitenkin mahdottomalta ajatukselta. Edessä on legacy-järjestelmien kirjo, kymmeniä toimittajia ja sadoittain erilaisia datamalleja. [Linthicum, 1999] Linthicumin viittaamilla legacy-järjestelmillä tarkoitetaan vanhtentuneita ohjelmistoja, joiden arkkitehtuuri, toiminnallisuus ja tietoturva eivät enää vastaa nykyisiä tarpeita. Miten järjestelmät voisi saada keskustelemaan, jos ne eivät edes puhu samaa kieltä?

Vastauksena järjestelmien välisiin kommunikaatio- ja tiedonsiirto-ongelmiin kehitettiin ajatusmalli Enterprise Service Bus:sta (ESB). Tämänkaltainen järjestelmä tarjoaisi keskitetyn alustan, jonka palveluihin eri osapuolet voisivat ottaa yhteyttä huolimatta siitä, minkälaisella teknologialla osapuoli on toteuttanut toimintonsa. [Chappell, 2004]

Vaikka ESB-järjestelmillä ei varsinaisesti ole yleisesti tunnustettuja vaatimuksia tai ominaisuuksia, voidaan kuitenkin yleisesti listata tunnusmerkkejä, jotka ovat ominaisia erilaisille ESB-ohjelmistoille. Näitä on esimerkiksi:

- Viestien välitys kahden tai useamman järjestelmän välillä.
- Palvelun viestinvälityksen seuranta ja lokitallennus.

- Datatransformaatiot, viestien rikastaminen ja muunnokset.
- Viestien rakenteiden validointi ja muu viestinnän toimivuuden varmistaminen.

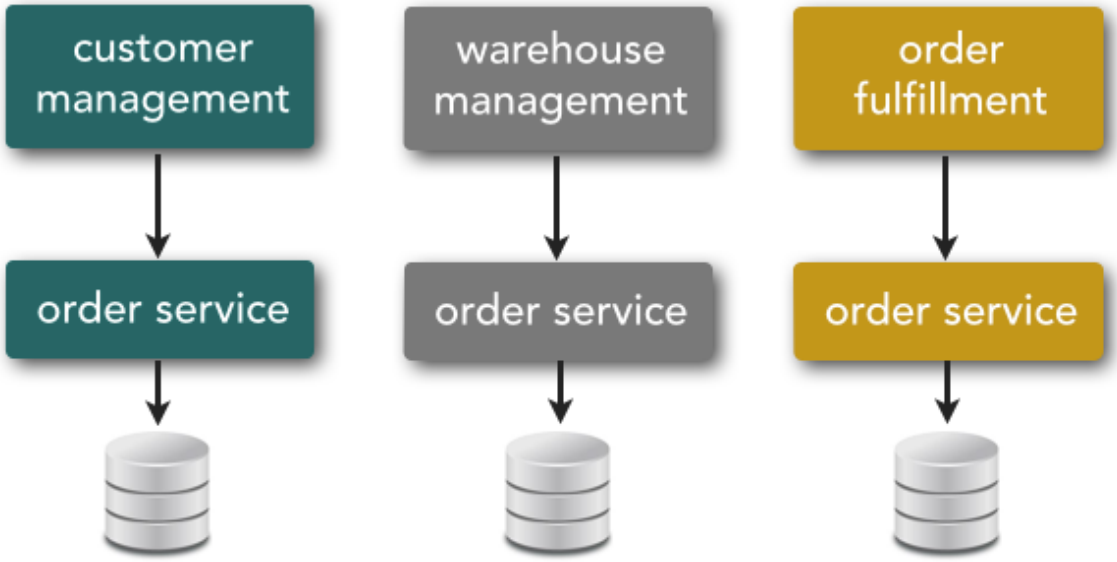
ESB:a toteuttavien järjestelmien eduksi onkin laskettava keskitetty hallinta, jolloin jokaiselle järjestelmälle ei tarvitse toteuttaa pistemäisiä yhteyksiä jokaista siihen liittyvää järjestelmää kohden. [Linthicum, 1999]

2.1.3 Service Oriented Architecture -malli

Service Oriented Architecture (SOA) on ohjelmistoarkkitehtuurinen ajattelutapa, jonka ydinajatuksena ei ole implementoida siihen liittyviä teknologioita, vaan vastata yrityksen tarpeisiin. Sen lähestymistapa on tuottaa järjestelmiä, jotka kykenevät toimimaan autonomisina palveluina ja näin ollen yksinkertaisin tapa toteuttaa SOA:ta onkin www-sovelluspalvelujen (Web Service) käyttö. [Anandamurugan ja Priyaa, 2014]

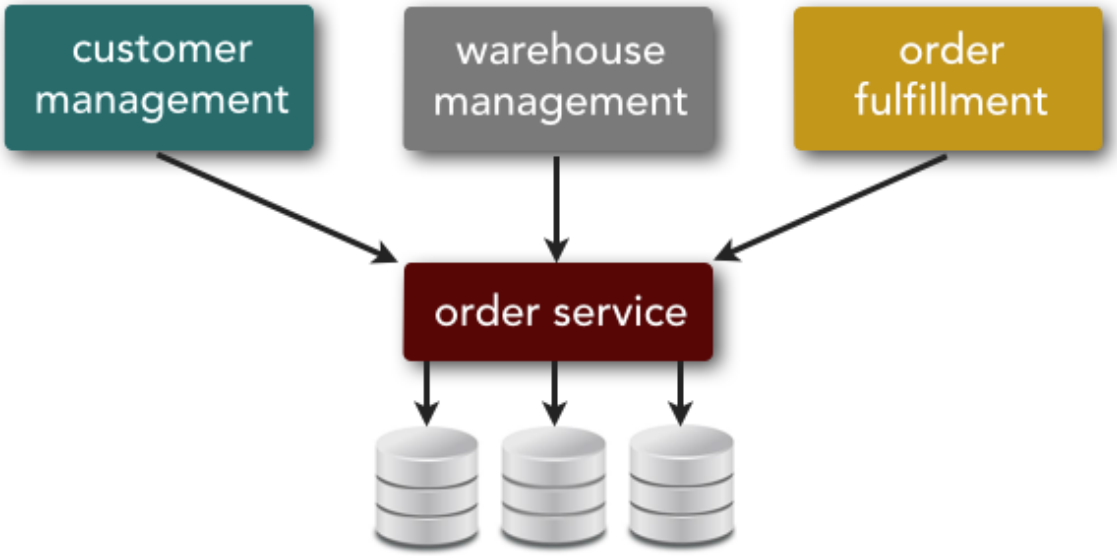
Tarve palvelupohjaiselle arkkitehtuurille juontaa juurensa organisaatioiden usein nopeatempoisiin IT-projekteihin, joissa laajalle arkkitehtuurisuunnittelulle ei jätetä juuri aikaa. Tällöin yksittäiset palvelut mahdollistavat ketterän kehittämisen markkinatilanteiden muutoksien mukaisiksi. Nämä yksittäiset palvelut voidaan ketjuttaa, jolloin pystytään tarjoamaan samat ominaisuudet, kuin yhdestä massiivisesta yksittäisestä monoliittijärjestelmästä. Näin saadaan myös palveluiden verkko, joka mahdollistaa automaattisten järjestelmien viestinnän ilman ihmisten tekemiä muutostarpeita. [Anandamurugan ja Priyaa, 2014]

Richards [2014] määrittelee komponenttien jakamisen yhdeksi SOA:n perusajatukseksi. Tämä yleiskäyttöisten komponenttien jako useiden järjestelmien käytettäväksi onkin käytännössä se ajatus, mikä SOA-arkkitehtuuria toteuttavilla yrityksillä on. Esimerkissään (kuva 2) Richards esittelee perinteisen myyntiyrityksen, jolla on käytössään useita järjestelmiä (customer management, warehouse management ja order fulfillment), jotka ovat tekemisissä myyntitilauspalvelun (order service) kanssa. Tällöin jokaisella näistä palveluista on käytössään oma versio myyntilaus-palvelusta. Jokaisessa putkessa tarvitaan kopio samasta palvelusta, jolloin sama myyntitilaus-palvelu replikoidaan kolmeen kertaan.



Kuva 2. Kuvaus pistemäisestä järjestelmäarkkitehtuurista [Richards, 2014].

Tätä ongelmaa SOA yrittää poistaa käyttämällä yrityksen laajuisia sisäisiä palveluita, jolloin keskitetty myyntitilaus-palvelu kykenee palvelemaan kaikkia myyntitilauksen kanssa tekemisissä olevia järjestelmiä (kuva 2). Tällöin kaikki palvelut voivat käyttää samaa prosessointilogiikkaa, eikä sitä tarvitse ylläpitää useassa eri paikassa.



Kuva 3. Kuvaus keskitetystä tilauspalvelusta [Richards, 2014].

Kuvassa on kuitenkin edelleen kolme tietokantaa ja tämä kuvastaa toista SOA suunnittelutyyliä, ”share-as-much-as-possible”, eli myyntitilauspalveluun on rakennettu tarpeeksi logiikkaa, että se osaa käsitellä jokaista tietokantaa ja käyttävää sovellusta niiden tarvitsemalla tavalla. Tällöin myyntitilauspalvelu palvelee itseasiassa yhden tietokannan sijaan kolmella tietokannalla. [Richards, 2014]

2.1.4 Microservices-malli

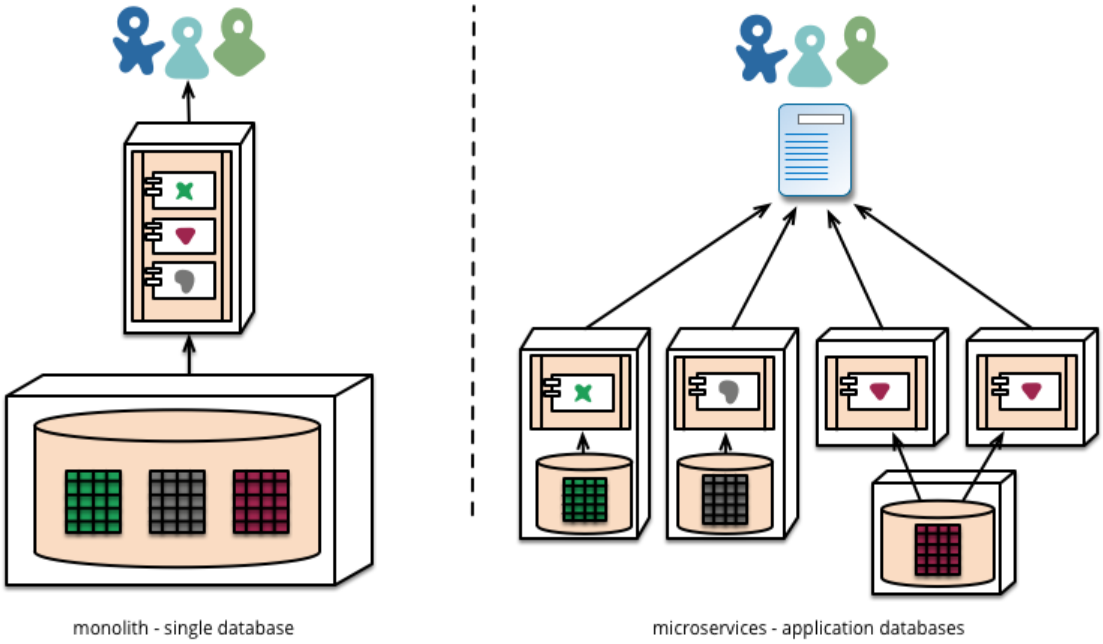
Schneiderin [2020] mukaan yleisesti alalla mikropalvelut (microservices) -termin luojina pidetään James Lewisiä ja Martin Fowleria. Heidän omien sanojensa mukaan termi annettiin yleiselle arkkitehtuuriselle suunnalle, joita eräät ohjelmistoarkkitehdit olivat tutkineet. Vuonna 2012 Lewis esitteli termin ja siihen liittyviä ajatuksia sovellusarkkitehtien konferenssissa. Samoihin aikoihin myös Netflix uursi uraa oman palvelunsa perustana. [Fowler ja Lewis, 2014]

Lewis ja Fowler [2014] kuvaavat mikropalveluarkkitehtuuria usean yksittäisen palvelun kokoelmana, joista jokainen toimii itsenäisenä prosessinaan pystyen kommunikoimaan kevyillä mekanismeilla, kuten esimerkiksi http-protokollalla.

Tällöin sen sijaan, että kaikki palvelut ovat riippuvaisia keskitetystä palvelusta tai paikasta, toimivat ne omina kokonaisuuksinaan mahdollistaen jokaisen palvelun rakentamisen esimerkiksi eri ohjelmointikielillä tai käyttämään erilaista datamallia.

Käytännössä tällöin eri tiimit ja kehittäjät rakentavat itsenäisiä palveluita, jotka vastaavat niiden tarpeita sisäisille ja ulkoisille asiakkaille. Tällöin ei tarvita yhtä yhtenäistä julkaisusuunnitelmaa, vaan jokainen osa voidaan ottaa käyttöön ketterästi sitä mukaan, kun tarvetta ilmaantuu. [Schneider, 2020]

Integraationäkökulmasta Fowlerin ja Lewiksen [2014] mukaan mikropalveluilla rakennetaan useita palveluita, jotka tuottavat ja käyttävät dataa. Loppukäyttäjälle tuleva data haetaankin tällöin useammasta lähteestä, sen sijaan että se tapahtuisi yhden keskitetyn putken läpi, kuten kuvan 4 monoliititarkkitehtuurissa. Tällöin esimerkiksi verkkokaupan tuotteen tiedot tulisivat esimerkiksi yhden ERP (Enterprise Resource Planning) -järjestelmän lävitse. Mikropalveluarkkitehtuurissa voisi esimerkiksi yhden palvelun vastuulla olla tuotteen hintatiedot, toisella master data -tiedot ja kolmannella saatavuus. Tällöin jokainen palvelu palvelisi vain omaa toiminta-aluettaan ja tarjoavat kuitenkin palveluansa myös muille rajapinnoille itsenäisesti.



Kuva 4. Monoliitti ja mikropalvelut [Fowler ja Lewis, 2014].

Toisaalta tässä hajautetussa ja ketjutetussa arkkitehtuurissa on myös mikropalveluiden heikkous. Aiemmin puhutut monoliittiset järjestelmät, joissa kaikki kehitys- ja julkaisutyö oli tarkoin orkestroitua ja ajoitettua samalle palvelimelle, tarkoittivat myös sitä, että kaikki palvelut kykenevät toimimaan keskenään. Mikropalveluiden yhteydessä jokainen mikropalvelumuutos voi aiheuttaa ketjureaktion satoihin muihin mikropalveluihin, jolloin niiden toimivuudessa voi esiintyä yllättäviä ongelmia. [Schneider, 2020]

Mikropalveluita yhdistää muutama keskeinen asia, joiden avulla mikropalveluiden määrittely on mahdollista. Vaikka keskeisiltä osilta mikropalvelut muistuttavat SOA-ajattelua, mikropalveluarkkitehtuuri eroaa kuitenkin mm. vakiintuneiden viestintäprotokollien, middlewaren tarpeen ja kuitenkin pinnan alla keskitetyn versionhallinnan ja julkaisuputkien riippuvuudessa. [Newman, 2021]

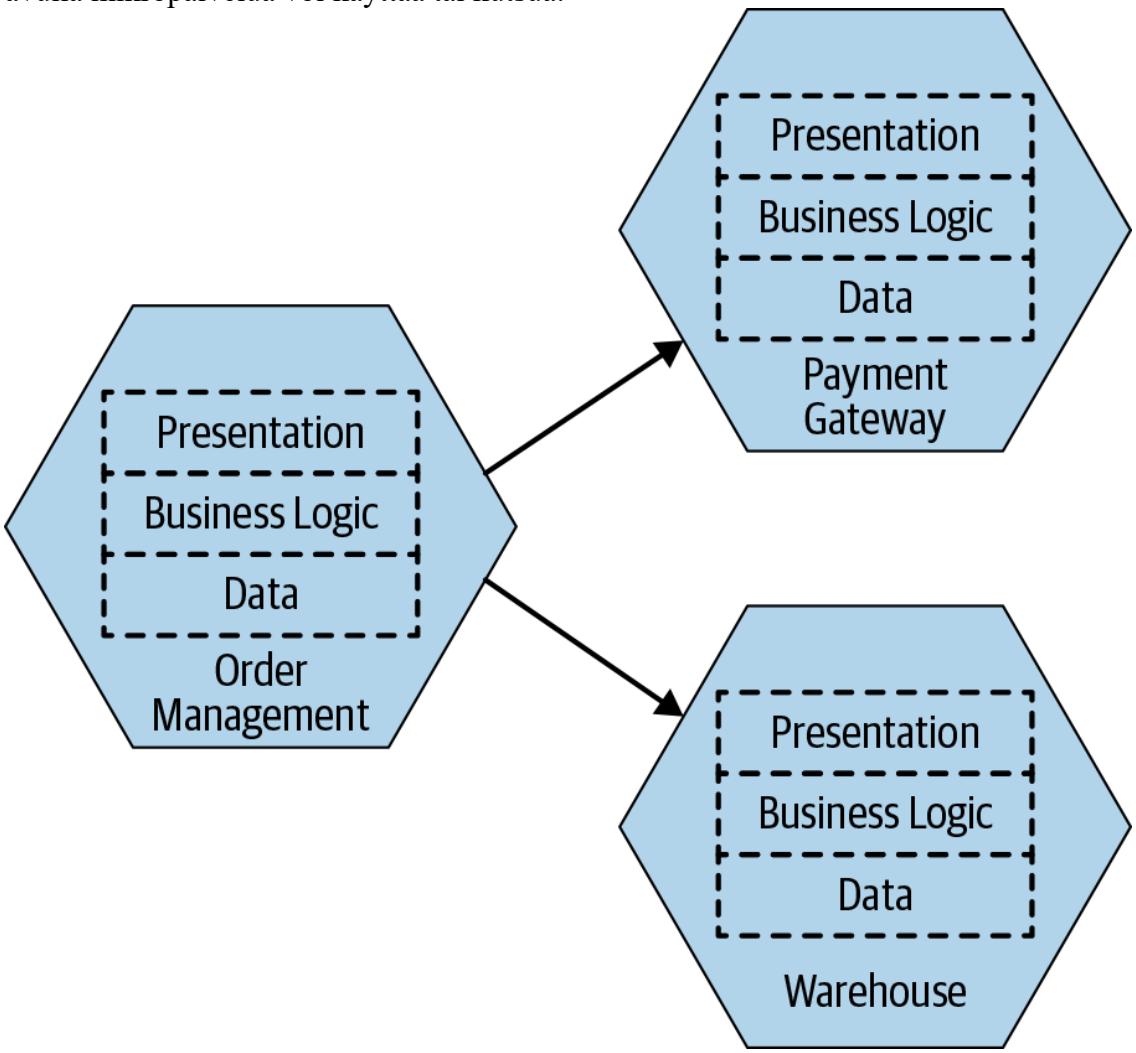
Ensimmäinen yhdistävä tekijä mikropalveluilla on se, että niitä voidaan julkaista itsenäisinä kokonaisuuksina. Tällä itsenäisellä julkaisulla tarkoitetaan sitä, että mikropalveluun voidaan tehdä muutos, julkaista se tuotantoon ja tarjota muutokset loppukäyttäjälle ilman, että mihinkään muuhun palveluun tarvitsee tehdä muutoksia. Tällöin jokainen mikropalvelu tulisi jo suunnitteluvaiheesta lähtien toteuttaa seuraten tätä teesiä. Vaikeuksia tähän tuo kuitenkin jo aiemmin tehdyt toteutuspäätökset, kuten esimerkiksi jaetut tietokannat tai muu riippuvuus lähdedatan suhteen. [Newman, 2021]

Itsenäisen julkaisun lisäksi mikropalveluita yhdistää myös se, että ne suunnitellaan usein palvelemaan tiettyä domainia. Mikropalveluarkkitehtuurissa määritellään palvelun reunaehdot käyttämällä liiketoiminta-alueitten tarpeita. Tällöin jokainen palvelu on kes-

kitetty palvelemaan ainoastaan tiettyä toiminta-aluetta, jolloin palveluiden itsenäinen päivittäminen on ketterämpää ja tehokkaampaa. [Newman, 2021]

Newman esittää kuvan 5 mukaisesti mikropalveluiden mahdollistavan sen, että jokainen palvelu itsessään kykenee tekemään koko palvelun tarjoaman end-to-end-putken: datan hakemisen, logiikan ja loppukäyttäjälle esittämisen. Perinteisessä mallissa jokaista kerrosta vastaisi oma tiiminsä, jolloin muutokset mihin tahansa kerrokseen vaatisivat muilta tiimeiltä panosta, kuten usein käy monoliittisissa palveluissa.

Datan hakemisella tarkoitetaan teknistä toimintoa, jossa joko ohjelmallisesti tai käyttäjän panoksella noudetaan dataa lähdejärjestelmästä. Usein tämä voi olla lähdejärjestelmän tekninen rajapinta tai esimerkiksi järjestelmän tuottama tiedosto, joka siirretään käsiteltäväksi. Kuvassa 5 esitetyn mikropalveluarkkitehtuurin logiikkakerros sisältää usein tarpeelliset päätökset datan käytön suhteen, kuten esimerkiksi dataan tehtäviä muutoksia tai rikastamista esimerkiksi toisen lähteen datalla. Kolmas osio arkkitehtuurissa, esittäminen, tarkoittaa käytännössä käyttöliittymä tai muuta rajapintaa, jonka avulla mikropalvelua voi käyttää tai kutsua.



Kuva 5. Jokaisella palvelulla on oma, kokonainen end-to-end-dataputki käytössään [Newman, 2021].

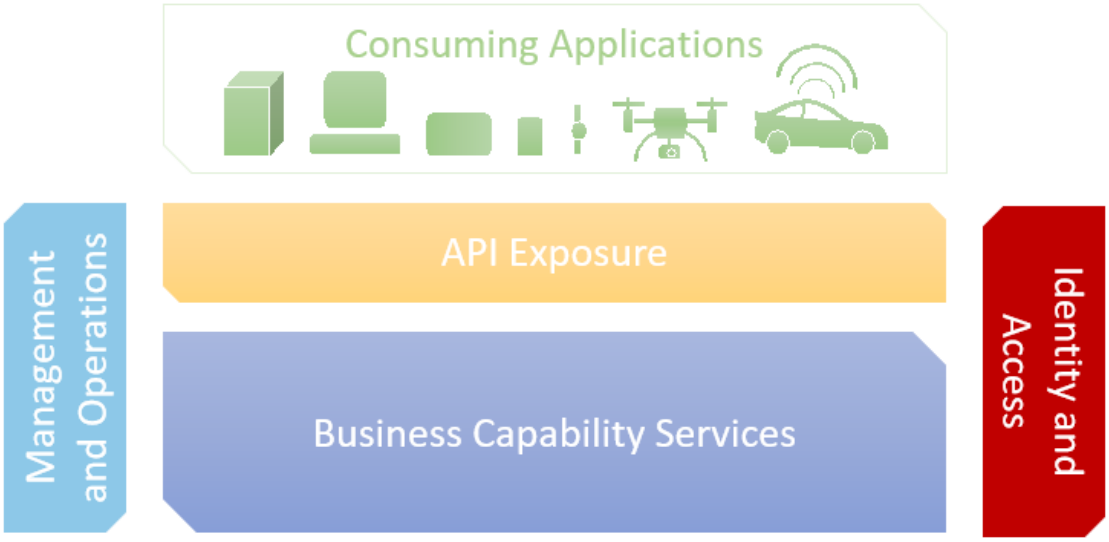
Viimeinen ja Newmanin mukaan hankalin yhdistävä konsepti mikropalveluarkkitehtuurissa on palvelun oman ”tilan” ymmärtäminen. Kuten jo aiemmin mainittiin, mikropalveluiden ongelmaksi muodostuu usein se, että jossain kohtaa prosessia on kuitenkin jaettu tietokanta, jolloin mikropalvelu ei pysty toteuttamaan aidosti toiminallisuuttaan alusta loppuun asti. Tämä pitäisi huomioida mikropalveluita toteutettaessa niin, että jokaisen muutoksen pitäisi olla kuitenkin takautuvasti toimiva; palvelua käyttävät osapuolet eivät joudu tekemään muutoksia olemassa oleviin rajapintoihinsa, vaan voivat ottaa uudet ominaisuudet käyttöön halutessaan. [Newman, 2021]

2.1.5 API-arkkitehtuuri

Vaikka API:t itsessään eivät ole tapoja integroida, ovat ne kuitenkin olennainen osa kuvattuja integraatiomenetelmiä. Käytännössä esimerkiksi mikropalvelut hyödyntävät usein nimenomaan toistensa API:ja, jolloin niiden hallinta ja käyttö nousee tärkeäksi osaksi yritysten integraatiostrategiaa.

API-toteutukset voivat olla joko ohjelmien sisäisiä tai niitä voidaan toteuttaa myös ohjelmien omien rajapintojen päälle. Tämänkaltaisia ulkoisia API-sovelluksia hallitaan joko ohjelmallisesti tai useammin käyttäen eri integraatioalusten API-kyvykkyyksiä.

API-arkkitehtuuri rakennetaan konseptuaalisesti Weirin ja Nemecin [2019] mukaan neljästä pääosasta. Kuvassa 6 horisontaaliset osat kuvaavat sovellustason ajonaikaisia ominaisuuksia, joita ilman API:t eivät olisi toteutettavissa. Pystysuorat osat muodostavat tukevat ominaisuudet, jotka avustavat API:n elinkaaren ylläpitoa, hallintaa, operaatioita ja analytiikkaa.



Kuva 6. API-lähtöinen arkkitehtuuri [Weir ja Nemec, 2019].

Wier ja Nemec [2019] määrittelevät kuvassa olevat osiot käyden läpi ensiksi horisontaalit ja sitten pystysuorat osat. Ylimpänä kuvassa ovat kaikki tietokoneohjelmistot,

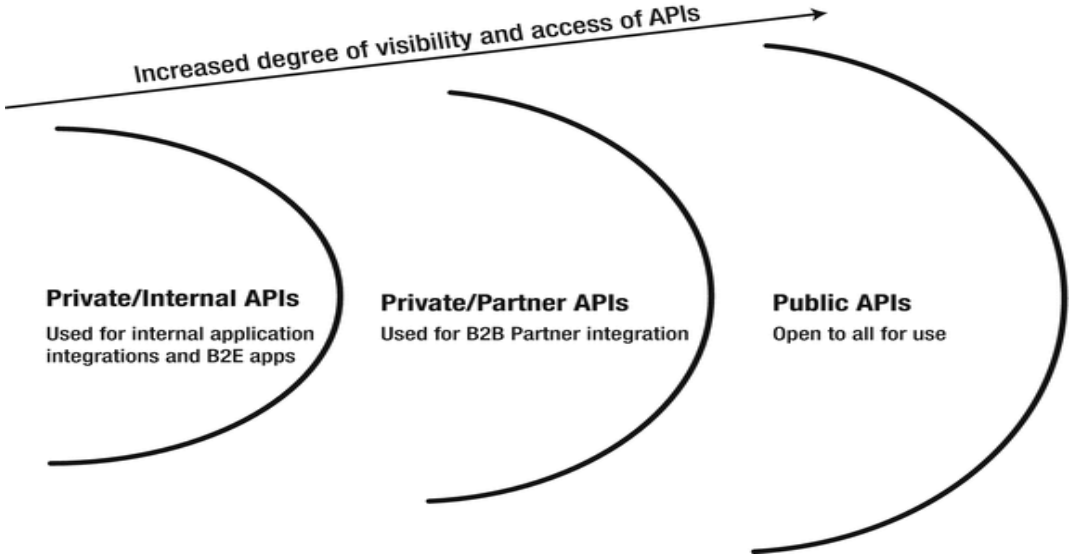
jotka kykenevät ottamaan yhteyttä ja käyttämään API:a. Käytännössä tämä sisältää niin perinteiset laskutusohjelmistot, mobiiliapplikaatiot, kuin älykellot ja jopa dronet.

Exposure eli julkaisukerros on osa, joka tarjoaa luotettavasti ja turvallisesti pääsyn APIn ja mahdollistaa näin viestinnän alemman osan, eli palveluiden kanssa. Tämä osa sisältää ohjelmiston tai ohjelmistot, jotka ovat selkeitä toiminallisia kokonaisuuksia ja yhteydessä johonkin selkeään prosessiin. Tämä prosessin toteuttamiskyky on riippuvainen kyvystä suorittaa dataan liittyvää logiikkaa, transformaatioita ja muita orkestraatio- tai varmennustoiminnallisuuksia. Käytännössä julkaisukerros tarjoaa palveluiden API-päätepisteitä ja sitä kautta toiminallisuuksia ulkopuolisille järjestelmille.

Management and Operations -osa rakentuu muun muassa API:n elinkaaren hallinta, asennus, testaus, analytiikka ja käytöstä poistoon liittyvät toiminallisuudet. Lisäksi myös esimerkiksi API:a käyttävien kehittäjien tueksi tehdyt erilaiset API-portaalit ja avaimien hallinta kuuluvat tähän osioon. Toisaalta jos käytön mukaan tehdään laskutusta, tähän liittyvä metriikka tuotetaan tässä osiossa. Vastaavasti Identity and Access -osio sisältää kehittäjien tunnistamisen, pääsynhallinnan kehitysalustalle ja käyttäjäkohtaisen näkyvyyden eri API-rajapintoihin.

Näitä osioita toteuttavia ohjelmistoja on monia, käytännössä SaaS-tyyppisiä ratkaisuja, jotka tarjoavat koko arkkitehtuurin yhdestä paikasta. Toisaalta yleisimmät IPaaS-tuotteet tarjoavat myös omat ”API Management” -tuotteensa, joiden avulla API-arkkitehtuuria voidaan toteuttaa.

De [2017] esittää APIen jakamista kahteen tyyppiin: julkisiin ja yksityisiin. Julkiset APIt ovat saatavilla kaikille kiinnostuneille avoimessa internetissä, kun taas yksityisissä APIssa käyttäjäjoukko on rajattu. Nämä yksityiset APIt voidaan jakaa vielä kahteen tasoon, kumppanitasoon, jossa APIt on saatavilla yhteistyökumppaneille esimerkiksi B2B (Business to Business) yhteyksiä varten. Toisena yksityisenä tasona voidaan pitää sisäistä APIa, jonka avulla yritys voi rakentaa sisäisiä järjestelmiä.



Kuva 7. API tyypit [De, 2017].

Teknisesti API-rajapinnat määritellään usein ohjelmallisesti käyttäen yleistä API-kuvausta, kuten esimerkiksi Open API. Tämän avulla voidaan kertoa API:a käyttävälle taholle, minkä tyyppisiä operaatioita API käyttää. Lisäksi kuvauksessa esitellään API:n hyväksymä rakenne viesteille ja tarvittavat autentikaatiometodit, sekä muut API:n tarjoamat prosessit.

2.2 Integraatiotasot

Linthicum [2001] määrittelee integroinnille neljä eri tasoa, jotka eroavat toisistaan niin teknisesti kuin mahdollisuuksiensa puolesta. Vastaavasti jokaisella tasolla on omat rajoituksensa, jolloin niitä integroivien työkalujen toiminallisuudet voivat poiketa toisistaan paljonkin. Näitä tasoja on data-level, application interface-level, method-level ja user interface level. Nykyaikaisessa arkkitehtuurissa method-tason integrointi on väistymässä ja vastaavasti user interface -taso on kokemassa muutosta robottiavusteisten sovellusten myötä. Eniten resursseja keskitetään application-tasoon. Linthicumin määrittelyt ovat kuitenkin edelleen hyvin kuvaavia ja edelleen paikkansa pitäviä joiltain osin.

2.2.1 Integrointi data-tasolla

Datatason integroinnissa dataa siirretään kahden datavaraston välillä. Tällöin tätä prosessia voidaan kuvata yksinkertaisimmillaan siten, että dataa haetaan yhdestä tietokannasta ja se välitetään toiseen tietokantaan.

Datatason integrointi ei kuitenkaan ole varsinkaan nykyisessä pilviympäristössä enää vain kannan taulusta toiseen siirtelyä. Lähde- ja kohdejärjestelmien datarakenne voi olla kompleksinen, tietokannat voivat sisältää useita tauluja, data itsessään saattaa vaatia transformaatiota tai se tulee tallentaa tietyssä järjestyksessä eteenpäin ja niin edelleen.

Tämänkaltaisissa tilanteissa onnistuneen integraatioarkkitehtuurin lisäksi olennaisena osana on myös hyvin hallittu data, jolloin integraation kompleksisuus ei pääse kasvamaan liialliseksi.

2.2.2 Integrointi applikaatioiden rajapintatasolla

Rajapintataso (Application Programming Interface, API) on ohjelmistoon rakennettu ominaisuus, jonka avulla voidaan ohjelmallisesti keskustella käytössä olevan järjestelmän kanssa. Näiden rajapintojen avulla järjestelmät voivat jakaa dataa tai prosesseja keskenään, joka mahdollistaa järjestelmien yhdistelyn. Usein ainoa este kehittäjille onkin sovelluksen tarjoamien rajapintojen ominaisuudet ja toiminallisuudet, jotka on määritelty järjestelmän kehittäjien puolelta.

Useat toiminnanohjausjärjestelmät ovat esimerkkejä tämänkaltaisesta toiminallisuudesta, jossa järjestelmät tarjoavat rajapintoja, joista voidaan tietyin rajaehdoin hakea dataa; esimerkiksi verkkokauppajärjestelmä voi saada ohjelmallisesti saatavuustietoja varastotietojärjestelmästä rajapintojen avulla.

Varsinkin suurissa yrityksissä, joissa järjestelmiä on paljon ja ne ovat riippuvaisia toistensa tiedoista integroidutaan usein rajapintatasolla. Näissä tilanteissa käytetään useimmiten erilaisia viestinvälittäjäjärjestelmiä tai arkkitehtuureja, kuten aiemmissa luvuissa on todettu suorien point-to-point-viestien sijaan. [Linthicum, 2001]

Nykyaikaisessa kehityksessä painotetaan rajapintojen uudelleenkäytettävyyttä, hallintaa ja myös tuotteistamista. Näitä ominaisuuksia toteuttamaan on markkinoille tullut useita erilaisia hallintatyökaluja, joita kutsutaan yleisesti API management -tuotteiksi. Näiden avulla rajapintojen julkaisu voidaan tehdä keskitetysti ja käyttäjänhallinta tapahtuu yhdestä paikasta, eikä esimerkiksi sovellustasolla. Työkalut mahdollistavat myös eräänlaisten kehittäjäportaalien julkaisemisen, joissa kolmannet osapuolet voivat tutkia julkaistuja rajapintoja, pyytää niihin käyttöoikeuksia sekä ottaa niitä käyttöön omiin tarpeisiinsa ilman rajapintojen julkaisijan erillistä työpanosta. Tämä nopeuttaa uusien käyttäjien ja asiakkuuksien haltuunottoa, sekä edesauttaa viestinnän suoraviivaistamista tarjoamalla valmiit tavat luoda yhteyksiä.

2.2.3 Integrointi käyttöliittymätasolla

Käyttöliittymätason integrointi on Linthicumin mukaan samaan aikaan integraatiotasoista alkeellisin, mutta toisaalta niistä myös tarpeellisin. Muut tasot voivat olla tehokkaampia tai teknologisesti kehittyneempiä, mutta tosielämän realiteetti on kuitenkin se, että useissa tapauksissa käyttäjä on välttämätön toimija sovelluksen toiminnallisuuden kannalta.

Käytännössä teknisesti käyttöliittymätason integrointi ei ole kovin monimutkaista. Lyhimmillään se tarkoittaa tapoja ja tekniikoita, joilla voidaan tarkastella tai lisätä tietoa sovelluksessa. Usein käytetyt työkalut ovat kuitenkin kömpelöitä massiivisten datamäärien tarkasteluun. Datan visualisointiin tarkoitetut työkalut ja keskitetyt monitorointirat-

kaisut voivat tuoda tähän helpotusta. Nämä kykenevät parsimaan datan käyttäjäystävälliseen muotoon tai vastaavasti tarjoamaan kuratoidun tavan muokata dataa ilman taustajärjestelmän ylimääräistä kuormittumista.

Yleisesti ottaen itse applikaation käyttöliittymän käyttäminen ylläpito- tai muutostarpeisiin tulisi olla viimeinen vaihtoehto. Vastaavasti se vaatii myös applikaation syvällistä ymmärtämistä, jotta sen logiikka ja datavarastointi on käyttäjälle tuttua. Toisaalta data, joka ruudulla käyttäjälle näkyy, voi olla erilaista kuin tietovarastossa järjestelmän alla; joissain sovelluksissa logiikan toteuttaminen tai laskenta tapahtuu vain käytön aikana eikä sitä varastoida pysyvästi esimerkiksi tietokantaan.

Nykyisin erityisesti RPA-tyyppiset (Robotic Process Automation) toteutukset ovat korvaamassa tämän tason toimintatarpeita. Siinä missä perinteisesti käyttäjä joutuu esimerkiksi täyttämään tilauskaavakkeen saamansa sähköpostitilauksen perusteella, kehittynyt RPA-sovellus kykenee lukemaan sähköpostin ja ”kirjoittamaan” sisällön käyttöliittymän lomakkeeseen ilman ihmispanosta.

2.2.4 Metoditason integrointi

Metoditasolla integrointi on integraatiotasoista ehkä eniten sisäisten järjestelmien väliseen integrointiin sopivin. Sen pääperiaatteena on Linthicumin mukaan yksittäisten kirjastojen, objektien tai toisten applikaatioiden jaettu käyttö. Tällöin eri applikaatiot voivat uudelleen käyttää samaa toimintaperiaatetta ja näin jakavat tiettyjä toimintaperiaatteita. Erityisesti julkipilvessä tämä näkyy infrastruktuurikoodin jakamisena (Infrastructure as a Code), kun CI/CD (Continuous Integration / Continuous Development), versionhallinta ja muu sovelluksien ajoympäristötieto on jaettu yhdestä paikasta samaa logiikkaa käyttäen. Usein esimerkiksi Microsoftin Azuressa rakennetut applikaatiot ovat itsenäisiä kokonaisuuksia, mutta käyttävät kaikki samaa Terraform-työkalua provisiointiin.

Toisaalta aiemmin mainitut uudelleenkäytettävät API:t ovat rajapintoina metoditason integraatioita; usea eri verkkokauppa voi käyttää samaa rajapintaa valmistajan varastosaatavuuden tarkastamiseen ja ovat näin välillisesti integroituneita, vaikkakin eri toimijoilta. Toisaalta sisäisesti samaa rajapintaa voi käyttää tuotannonennustusjärjestelmä ja sisäinen myyntikanava.

Näin ollen metoditason integrointi on muuttanut muotoaan enemmän uudelleenkäytettävien palveluiden, rajapintojen ja yhtenäisten toimintatapojen suuntaan. Tämä on selkeä arkkitehtuurinen kehityssuunta pois eksplisiittisesti jaettujen ohjelmistokirjastojen yksittäisten metodien käytöstä, mitä se vuosituhannen alussa on Linthicumin mukaan ollut. Esimerkiksi samaa ohjelmointikieltä käyttävät myynti- ja CRM-järjestelmät ovat saattaneet käyttää jaettua metodia, joka luo uuden käyttäjän.

Enterprise-tason arkkitehtuurissa puhutaankin usein frameworkeista. Linthicum määrittelee teoksessaan niiden olevan kokoelma abstrakteja luokkia, objekteja tai aplikaatioita, joita voidaan käyttää uudelleen, mutta myös joiden avulla voidaan kehittää uusia toiminallisuuksia. Esimerkiksi sovelluksen käyttöliittymä voi perustua hyvin pitkälti suosittuun front-end frameworkkiin, vaikka kehittäjä on sitten räätälöitynyt sen sovelluksen tarpeita vastaavaksi. Tällöin yleiset toiminnallisuudet, kuten esimerkiksi täytettävät lomakkeet, ovat kehittäjän saatavilla heti, eikä niiden luomista tarvitse tehdä alusta asti kehittäjän toimesta.

2.3 Integrointitekniikat

Integrointitekniikat ovat tapoja yhdistää eri datalähteitä, järjestelmiä ja sovelluksia toisiinsa käyttäen tietynlaisia protokollia. Aiemmin kuvattuja integraatiopatterneja toteutetaan teknisesti usealla tavalla, mutta näistä yleisimpiä ovat RESTful API-kutsut, tiedostosiirrot sekä tietokantayhteydet. IPaaS-tuotteissa näitä toteuttavat valmiit komponentit, joihin tarvitsee määritellä ainoastaan yhteysparametrit.

2.3.1 RESTful API

REST API (Representational Sate Transfer) on API, joka on suunniteltu käytettäväksi http-protokollalla erinäisten resurssien hallinnointiin tai käyttämiseen. Arkkitehtuurisesti API-client lähettää http-pyynnön serverille, käyttäen etukäteen määriteltyjä metodeja. Näitä metodeita ovat GET, POST, PUT ja DELETE, joita käytetään datan hakemiseen, luomiseen, päivittämiseen tai poistamiseen. Kun viesti on vastaanotettu serverin päässä, vastaa se etukäteen sovitussa muodossa pyydetyn resurssin mukaisesti. Yleisimpiä vastausviestien rakenteita ovat XML ja JSON.

2.3.2 Tiedostosiirrot

Hieman perinteisempi tapa integroida järjestelmiä on erilaiset tiedostonsiirtoprotokollat. Yksi näistä on SFTP (SSH File Transfer Protocol), joka käyttää SSH-protokollaa (Secure Shell) verkkoliikenteen salaamiseen. Käyttäjä voi esimerkiksi käyttää omaa SSH-avaintaan ottaessaan yhteyttä palvelimelle, jolla on tiedossa avaimen vastinpari. Tämän avulla palvelin voi tunnistaa yhteyttä ottaneen tahon ja luvittaa ainostaan oikeat tiedostot ja tiedostopolut käytettäväksi. [SSH, 2023]

Toinen tapa käsitellä järjestelmien tuottamia tiedostoja, on suorat levyjaot sisäverkossa. Näissä tapauksissa esimerkiksi iPaaS on asennettu paikallisesti ja sillä on pääsy suoraan tiedostoja luovan järjestelmän levylle, josta se voi tiedostot hakea. Tällöin liikenne tapahtuu sisäverkossa, eikä palvelua tarvitse avata julkiverkkoon, vaikka kohdejärjestelmä olisikin sisäverkon ulkopuolella.

2.3.3 Tietokantayhteydet

Usein järjestelmät säilövät datansa tietokannoissa. Tätä dataa voidaan lisätä, lukea tai muokata käyttäjätunnukselle määriteltyjen oikeuksien mukaisesti. Integraatiomielessä tämä tarkoittaa usein sitä, että järjestelmä tarjoaa tietyn tietokantataulun, josta integraatio voi dataa noutaa. Tällöin integraatiojärjestelmä tekee esimerkiksi tarpeellisen kyselyn tietokantaan käyttäen SQL-kyselykieltä (Structured Query Language). Tämä kieli mahdollistaa rakenteellisen komennon tietokannalle, jotta tietokantataulusta saadaan vain halutut, rajatut tiedot [Beaulieu, 2020]. Tämän tyyppinen integrointimalli tuo mukanaan riskejä, mikäli tietokantaa ei ole optimoitu jatkuvaan käyttöön integraatioalustan toimesta tai se ei kykene tuottamaan rajapintatauluja, vaan jakaa suoraan sisäisiä, operatiivisia tietokantatauluja integraatioiden käytettäväksi.

3 Integraatioalustat palveluina

Integraatioteknologioita ja integrointitapoja voidaan toteuttaa paikallisesti sovelluksien sisällä, mutta yleisesti erityisesti suuremmissa yrityksissä käytetään tätä varten tehtyjä sovelluksia ja palveluita. Tässä luvussa esitellään yleisellä tasolla mitä tarkoitetaan integraatioalustoilla, ja sen lisäksi esitellään muutamia Suomessa yleisesti käytössä olevia integraatioalustoja.

3.1 Integration Platform as a Service - IPaaS

IPaaS on lyhenne englannin kielen sanoista Integration Platform as a Service ja lyhykäisyydessään tarkoittaa integraatioalustatuotetta, jonka voi hankkia käyttöön palveluna. Tällöin alustatuotteen ylläpito on keskitetty palveluntarjoajalle, jolloin sen ylläpito- ja kehitysvastuu on muualla kuin käyttäjän omalla vastuulla. [Weir & Nemec, 2019]

Luis Augusto Weir määrittelee artikkelissaan [Weir, 2017] IPaaSien olevan todellisia IPaaSeja kun ne täyttävät mukaillen NIST:n määrittelemät [Weir, 2011] ominaisuudet. National Institute of Standards and Technology (NIST) on Yhdysvaltain kauppaministeriön alainen järjestö, jonka määrittelemiä ja tukemia standardeja tiede- ja teknologiainnovaatiot seuraavat Yhdysvalloissa. NIST:n mukaan pilvipalveluiden ominaisuuksia ovat:

1. **Saatavuus ja joustava hankinta:** Tuotteen lisenssi tai muu hankintahinta tulee olla maksettavissa ja tuotteen lisenssi saatavissa käyttöön välittömästi esimerkiksi palvelun kotisivuilta ilman tarvetta yhteydenottoon palveluntarjoajan myynti- tai muun henkilön suuntaan. Tuotteen käyttöönottoon ei tule myöskään kuulua minkäänlaista konfiguraatiota tai asennuspakettia, vaan sen on oltava käyttövalmis heti lisenssin rekisteröinnin jälkeen.
2. **Laaja verkkoyhteys**: Verkkoyhteyksien toimivuus iPaaS-tuotteeseen yhdistäessä on oltava turvallisia, nopeita ja luotettavia. Avoimet internetyhteydet voivat olla vaikeasti ennustettavia ja siksi julkipilvioperaattorien tulee mahdollistaa myös vaihtoehtoisia tapoja ottaa yhteyttä palveluun. Tälläisiä toiminallisuuksia on esimerksi Oraclen Fast Connect-palvelu ja Amazonin AWS Direct Connect.
3. **Resurssien käyttö:** Palvelun tulee tarjota laskentateho (esimerkiksi prosessoriteho, keskusmuisti, kovalevytila) tarpeen mukaan, jolloin resurssit tulevat automaattisesti tarjolle käytön tai konfiguraation perusteella. Tällöin resursseja tulee voida lisätä tai vähentää haluttaessa joko tarpeen tai erillisten sääntöjen mukaan.
4. **Ketterä skaalautumiskyky**: Korkean kysynnän aikoina resurssiennustaminen voi olla lähes mahdotonta, kuten esimerkiksi Black Friday -alennusmyyntien aikaan. Todellinen iPaaS-alusta kykenee skaalautumaan määriteltyjen sääntöjen ja resurssitarpeiden mukaisesesti. Tällöin transaktiohuiput kyetään hoitamaan skaalautumalla sekä horisontaalisesti, että vertikaalisesti ilman ihmisen tekemää työtä. Kun

transaktioiden määrä tasaantuu, alustan resurssit palautuvat normaalille tasolle. Vastaavasti automaattinen skaalaus alaspäin tulee olla mahdollista, jolloin tehokkaasti lisätyt resurssit eivät vähene huippujen jälkeen automaattisesti.

5. **Kulumittaroitu palvelu**: Palvelun hinnoittelu tulee olla joko käytön tai tilaussopimuksen mukaan tapahtuvaa ja täysin läpinäkyvää. Laskutuksen tulee näyttää käyttö ja niihin assosioidut hinnat ja maksut. Esimerkiksi jos iPaaS-tuotteen laskutus perustuu käytettyjen liittymien määrään, tulee käyttäjän tietää täsmälleen kuinka monta liittymää tilauksessa on käytössä mihin tahansa aikaan. Jos allokoitu liittymien määrä on täyttymässä, tulee tästä tulla notifikaatio, jotta käyttäjätaho voi puuttua tilanteeseen nopeasti. Vastaavasti alustan pitäisi lähettää huomatus käyttämättömistä resursseista, jotta ylimääräisiä maksuja voidaan karsia pois.
6. **Automaattinen alustan päivitys:** Suurin yksittäinen ominaisuus, joka erottaa iPaaS-tuotteen perinteisestä on-premise middlewaresta on sovelluksen päivitys. Pilvipohjaisten palveluiden tarjoajien tulee kantaa täysi vastuu alustan päivityksistä ja versionnostoista pilvipohjaiselle infrastruktuurille. Vastaavasti myös taaksepäin yhteensopivuus on palveluntarjoajan vastuulla, jolloin minkään päivityksen ei tule rikkoa olemassa olevia sovelluksia.

Weir [2017] jatkaa, että IPaaSien tarkoitus ei ole välttämättä korvata perinteisiä on-premise integraatiosovelluksia tai toimia pelkästään pilvessä, vaan mahdollistaa myös hybridiympäristöjen toiminnan. Näitä tilanteita voi olla esimerkiksi uusien tarpeiden täyttäminen, joita on-premise sovellukset eivät välttämättä tue; IPaaS-tuotteen käyttö voikin olla näissä tilanteissa tehokkaampaa. Kuvassa 8 on esitelty miten erilaiset integraatiopatternit voidaan toteuttaa käyttäen hybridimallia, jossa on-premise-sovellus hyödyntää pilvipohjaisen IPaaS:n ominaisuuksia.

Kuvassa 8 on esitelty patternien käyttömahdollisuuksia. Prosessiorkestrointi (Process orchestration) tarkoittaa sitä, että integraatio käynnistää sovelluksia määritetyn aikataulun mukaisesti. Julkaisu/tilausintegroinnissa (Publish/Subscribe) integraatiot julkaisevat tai seuraavat tapahtumia, joiden ilmaantuessa integraatio käynnistyy ja suorittaa prosessinsa. Vastaavasti muut sovellukset voivat kuunnella näitä tapahtumia jolloin viestinvälitys on lähes viiveetöntä. IPaaS-tuotteita tarvitaan usein myös hybridiympäristöissä siinä tapauksessa, että integraatio tapahtuu pyyntö/vastaus-periaatteella. Tällöin järjestelmä lähettää tietyn pyynnön, esimerkiksi REST-rajapintaan. Tällöin sovitun protokollan mukaisesti pyyntöviesti sisältää tietyt asiat, jolloin kysyjä saa vastauksena viestiinsä tarvitsemansa tiedon.

Integration Pattern	Type	Suitability	iPaaS Required?	On-premise capability required?
Process Orchestration	Cloud to Cloud	Orchestrate a process that spams across multiple cloud applications.	✓	✗
	Cloud to/from On-premises	Orchestrate a process that spams across multiple cloud applications and also requires integration with system located on-premises.	✓	✓
	On-premises to On-premises	Orchestrate a process that spams across multiple on-premises applications.	✗	✓
Publish / subscribe	Cloud to Cloud	Near real-time integration between systems in the cloud. Event triggered. More than one application (subscriber) interested in the same message.	✓	✗
	Cloud to/from On-premises	Near real-time integration between systems in the cloud and on-premises. Event triggered. More than one application (subscriber) interested in the same message.	✓	✓
	On-premises to On-premises	Near real-time integration between on-premises systems. Event triggered. More than one application (subscriber) interested in the same message.	✗	✓
Request / reply	Cloud to Cloud	Real time integrations (REST APIs) between systems in the cloud. Human triggered via UI (drives UX). One or many consumers reuse same API.	✓	✗
	Cloud to/from On-premises	Real time integrations (REST APIs) between systems in the cloud and on-premises. Human triggered via UI (drives UX). One or many consumers reuse same API.	✓	✓
	On-premises to On-premises	Real time integrations (REST APIs) between on-premises systems. Human triggered via UI (drives UX). One or many consumers reuse same API.	✗	✓

Kuva 8. IPaaS-tuotteen tarpeellisuus on-premise-sovellukseen peilaten [Weir, 2017].

3.2 Azure Logic Apps

Logic Apps on Microsoftin Azure-alustalla tarjoama pilvipohjainen iPaaS-tuote, joka tarjoaa mahdollisuuden ajaa automatisoituja prosesseja, jotka integroivat dataa, palveluita ja järjestelmiä. Tuote tarjoaa sadoittain valmiita yhteyspisteitä, joiden avulla liittymien teko API-rajapintoihin tai sovelluksiin on mutkatonta. Logic App toteuttaa workflownsa ennalta määritetyssä sekvenssissä, skaalautuen sääntöjensä ja toimintojensa mukaisesti, kun se laukaistaan määritellyn säännön mukaisesti. [Microsoft, 2021]

Logic Appsit voidaan liittää muihin Azuren palveluihin natiivisti, jolloin esimerkiksi Azuren jonojen tai API Managementin käyttö on helppoa. Palvelu voidaan myös kontittaa, jolloin applikaatioita voidaan asentaa ja ajaa pilvessä, paikallisesti tai on-prem-sovelluksessa. Palvelu tarjoaa myös valmiit DevOps-työkalut, jolloin asennusprosessit ovat turvallisia ja saumattomia. [Microsoft, 2021]

Logic Appsilla kehittämisen käynnistäminen ja ensimmäisen toimivan version tuottaminen on hyvin mutkatonta pilvessä tapahtuvan kehittämisen ja dynaamisen hinnoittelun takia. Alkuun pääsemiseksi ei tarvitse asentaa ohjelmistoja mihinkään, vaan kehityksen voi aloittaa sopivilla tunnuksilla Azure-portaalista.

Hinnoittelu määräytyy käytännössä siten, että jokainen komponentti prosessissa maksaa sentin osia ajon aikana, jokaisen ajon maksaessa erikseen. Tällöin Azuren käyttöönotossa ei tarvitse käyttää suurta määrää resursseja ympäristöjen pystytykseen ja maksamaan konesalikuluja ennen kuin ensimmäinenkään integraatio on valmis.

3.3 MuleSoft

MuleSoft on perustettu 2006, kun perustajat Ross Mason ja Dave Rosenberg kyllästyivät tekemään ”aasin työtä” dataintegroinnissa. Ajatuksena oli tehdä uudelleenkäytettä-

vää koodia, jotta samankaltaisia tehtäviä voitaisiin tehdä samoista osista ilman, että pyörää tarvitsisi keksiä uudestaan. [Mulesoft, 2021]

Teknisesti MuleSoft tarjoaa erilaisia tuotteita eri tilanteita varten. Pohjalla on Anypoint Studio, jonka avulla voidaan rakentaa Mule-integraatioita, jotka toimivat Mule-ajoympäristöissä. Näitä ajoympäristöjä voidaan ajaa joko paikallisesti omissa konesaleissa tai vastaavasti pilvessä. Muita yleisesti integroinnissa käytettäviä tuotteita on Anypoint Design Center jonka avulla kehittäjät voivat toteuttaa APIen koko elinkaarta, sekä siihen liittyvä Anypoint Exchanges, jossa API-tarjoajat voivat jakaa tuotteitaan. Lisäksi Anypoint Studioon kuuluu myös Anypoint Management Center, jonka avulla voidaan monitoroida integraatioiden ja APIen käyttöä. [Mulesoft, 2021]

MuleSoftin tuotteita voidaan käyttää kaikkien yleisimpien integraatioarkkitehtuurien toteuttamiseen: mikropalveluihin, SOA-tyyppiseen toiminnallisuuteen tai puhtaana iPaaS-palveluna.

Gartner on valinnut vuonna 2020 MuleSoftin johtavaksi teknologiaksi Enterprise-tason iPaaS vertailussaan. Vahvuudeksi mainitaan aloittamisen helppous, joka mahdollistaa nopean käyttöönoton ja välittömät hyödyt. [Gartner, 2020]

Integraatioita rakennetaan käyttämällä Anypoint Studiota, joka on graafinen IDE. Valmiita komponentteja voi ”vetää” kanvakselle, jolloin IDE kirjoittaa taustalla tarvittavan koodin erilaisiin XML-tiedostoihin. Näitä tiedostoja voi myös kirjoittaa käsin, mutta pääosin kehitystyö tapahtuu käyttämällä näitä valmiita komponentteja ja niiden ominaisuuksia muokkaamalla.

3.4 Dell Boomi

Dell Boomi Atmosphere (Boomi) on pilvipohjainen integraatiopalvelu (IPaaS, integration platform as a service), joka mahdollistaa erilaisten järjestelmien välisen integroinnin yksinkertaisella kanvaksella. Integraatioiden tuottaminen työkalulla on yksinkertaista ja tapahtuu vetämällä erilaisia komponentteja kanvakselle ja näiden konfiguroinnilla. [Boomi, 2021]. Gartner määrittelee Boomin yhdeksi IPaaS-markkinan johtajaksi ja visionääriksi, johtuen alustan valmiista tarjoamasta ja kyvystä toteuttaa tarpeet. [Gartner, 2020]

Alkuperäinen Boomi perustettiin jo 2000, kunnes Dell osti yrityksen ja sen tarjoaman AtomSpheren omaan portfolioonsa 2010. Sittemmin vuoteen 2021 mennessä yrityksellä on maailmanlaajuisesti jo yli 12 000 asiakasta ja se on kehittänyt tuotevalikoimaansa ostamalla yrityksiä mm. tekoälyn, Master Data ja low code -tuoteperheiden kehittämiseksi. [Boomi, 2021]

Boomi AtomSpheren ajoympäristö, Atomi (Atom) tai Molekyyli (Molecule), voi olla pilvipohjainen, jolloin integraatioinstanssit ja sovellus toimii Dellin omassa konesalissa. Usein kuitenkin alustaa käyttävät tahot asentavat sovelluksen omille palvelimil-

leen tai käyttävät hybridimallia, jolloin esimerkiksi testiympäristöt pysyvät pilvessä, mutta tuotantoympäristö asennetaan paikallisesti. [Boomi, 2021]

Atomi on kevyt Java-pohjainen sovellus, jota voidaan pitää kevyenä integraatiomoottorina, joka hoitaa asennettujen integraatioiden toiminnot, sisältäen kaikki tarpeelliset paketit ja komponentit prosessin ajoon alusta loppuun. Vastaavasti Molekyyli on single-tenant multi-node -ajoympäristö, joka mahdollistaa HA (High Availability) -ympäristön tarjoamalla samanaikaisen ajon kahdella eri Atomilla tai kolmannen backup, ns. kylmän ympäristön olemassaolon. [Boomi, 2021]

Boomin kehitystyö tapahtuu AtomSpheressä, jota käytetään selaimella. AtomSphere tarjotaan ainoastaan Dellin pilvipalveluna, eli sen käyttämiseen tarvitaan internetyhteys eikä AtomSphereä voi ajaa lokaalisti. AtomSpheressä on myös mahdollisuus käyttää muita Boomin tuotteita, kuin Integrations-tuotetta, kuten esimerkiksi Master Data Hub tai Boomi Flow. Näiden tuotteiden monitorointiin ei kuitenkaan tässä työssä oteta kantaa, vaan työ keskittyy nimenomana Integrations-tuotteeseen ja sen toiminnallisuuksiin.

4 Monitorointityökalut

Monitorointi on terminä melkein mahdoton määritellä yksiselitteisesti, koska jokainen monitorointiratkaisu on käytännössä erilainen. Ratkaisuun vaikuttavia tekijöitä voi olla esimerkiksi näkökulma monitorointiin ylipäätänsä ja ratkaisussa käytettävät komponentit, jotka muodostavat kokonaisuuden. Toisaalta jo pelkästään tapa miten ja mitä dataa kerätään monitoroitavaksi määrittelee ratkaisun laajuutta. [Bastos ja Araujo, 2019]

Digitalisaation ja sen aiheuttaman aiemmin mainitun yritysten muuttumisen ohjelmistotaloiksi on myös erinäisten järjestelmien määrä lisääntynyt. Tähän kun lisätään esimerkiksi mikropalveluarkkitehtuuri, voi seurattavien järjestelmien määrä kasvaa satoihin isoissa yrityksissä, jolloin niiden manuaalinen seuraaminen käy mahdottomaksi.

Bastos ja Araujo [2019] mainitsevatkin nykyisten monitorointiratkaisuiden olevan jatkuvan toiminallisuuden ja sen ylläpitämisen kannalta kriittisiä monessa yrityksessä. Integraatioalustojen ylläpidossa tärkeitä toiminallisuuksia on tehokas kehitysympäristö, integraatioiden toiminnan seuranta viestitasolla, ongelmatilanteiden automaattinen ja manuaalinen monitorointi, sekä virheistä tointuminen.

4.1 Monitorointityökalujen käyttö

Erilaisia monitorointityökaluja yhdistää pohjimmiltaan samat tekniset periaatteet ja käyttöliittymäsuunnitteluperusteet. Jotta työkalut voivat monitoroida eri palveluita tai toimintoja, tulee niiden vastaanottaa dataa seuratusta kohteesta. Tämä voidaan toteuttaa joko passiivisesti tai aktiivisesti.

Passiivisella vastaanottamisella tarkoitetaan sitä, että monitoroinnin kohde lähettää tarvittavan datan tai tilatietonsa itsenäisesti monitorointialustalle, joka tarjoaa rajapinnan tätä varten. Tämä rajapinta voi olla teknisesti esimerkiksi REST API tai tiedostopohjainen sisäänluku. Aktiivinen monitorointialusta taas kykenee käyttämään esimerkiksi tietokantakyselyitä, http-pyyntöjä tai sftp-palvelinta lukevia protokollia, joiden avulla data saadaan luettua sisään monitorointiratkaisuun. Joissain tilanteissa, esimerkiksi useita sovelluksia valvoessa, työkalu voi käyttää molempia keinoja datan hankintaan.

Datan sisäänluvun lisäksi alustoja yhdistää myös datan visualisointi ja erilaiset parametrisoinnit. Käytännössä tämä tarkoittaa erilaisten näkymien ja kuvaajien tuottamista käyttöliittymään. Näitä voidaan muokata tai laajentaa riippuen käytetystä datasta, mutta ne ovat usein tärkein osa tämänkaltaista monitorointia.

Parametrisoinnilla ja erityisillä ”hakukonekielillä” voidaan työstää sisäänluettua dataa etukäteen määritellyin säännöin. Näin saadaan tuotua esille joko tekstimuodossa tai kuvaajana aina samantyyppistä, tarvittua dataa. Nämä hakuehdot voidaan tallentaa ja näin uudelleen käyttää aina tarvittaessa.

4.2 Työkaluvaihtoehtojen esittely

Tässä kohdassa esitellään markkinoilla yleisesti käytössä olevia valmiita keskitettyjä monitorointiratkaisuja. Alustavan kartoituksen perusteella valittiin kolme työkalua, jotka voivat potentiaalisesti täyttää tämän opinnäytetyön tarpeet. Nämä työkalut ovat: Elastic Stack, Datadog ja Splunk.

4.2.1 Elastic Stack

ELK-stackin pohjalta kehittynyt Elastic Stack on avoimeen lähdekoodiin perustuva keskitetty monitorointiratkaisu. ELK-nimi on peruja alkuperäisestä kolmesta projektista: Elasticsearch, Logstash ja Kibana. Mukaan liittyi myös Beats-projekti, jonka myötä nimenmuutos tuli ajankohtaiseksi. [Elastic, 2021]

Elasticsearch on haku- ja analytiikkamoottori, joka kykenee käsittelemään sisääntulevan datan logeista, palvelinmetriikoista ja web-sovelluksista sisäisiin Elasticsearch indekseihin. Tämä raakadata käsitellään, normalisoidaan ja rikastetaan tarpeen mukaan ennen indekseihin kirjoittamista ja näitä indeksejä vasten voidaan toteuttaa monimuotoisia hakuehtoja.

Logstash on työkalu, joka aggregoi ja prosessoi datan Elasticsearchin indekseihin käyttäjän tarpeiden mukaisesti. Se kykenee tekemään datalle transformaatioita, keräämään sitä useista lähteistä ja tarvittaessa rikastamaan sitä ennen indekseihin lähettämistä.

Kibana visualisoi ja hallinnoi kerättyä dataa Elasticsearchin indekseissä ja mahdollistaa myös käyttäjiä tekemään datan pohjalta monimuotoisia dynaamisia infograafeja tai maantieteellisiä raportteja datasta.

4.2.2 Datadog

Datadog on pilvipohjainen Service as a Service (SaaS) -tuote, joka tarjoaa servereiden, tietokantojen, työkalujen ja palveluiden monitorointia data-analytiikka-alustansa kautta. Se on perustettu vuonna 2010 ja on sittemmin kasvanut muun muassa yritysostojen myötä yhdeksi johtavista monitorointityökaluja tarjoavaksi toimijaksi. [Datadog, 2021]

Datadog tarjoaa integraatioita erinäisiin sovelluksiin ja lokiagregaatteihin, kuten esimerkiksi aiemmin mainittuun Logstashiin. Etuna tähän on se, että käytettäessä valmista Datadog-integraatiota, lokin muotoilu ja toimintaperiaate on valmiina tiedossa, eikä sen esikäsittelyä tarvitse toteuttaa itsenäisesti.

Datadog kykenee myös toteuttamaan laajaa analytiikkaa ja sen voi asentaa useille alustoille. Sen API-kyvykkyydet mahdollistavat tehokkaan räätälöinnin ja sovellus onkin yksi johtavista lokitustyökaluista markkinoilla. Se tarjoaa myös automaattiset ilmoituspalvelut esimerkiksi sähköpostiin tai yleisimpiin viestintäpalveluihin, kuten Slack tai Atlassianin HipChat. [Malepati et al., 2019]

4.2.1 Splunk

Splunk on markkinoiden ensimmäinen ”data-to-everything” alusta, jonka tarkoitus on poistaa datan ja toimimisen väliset esteet. Alustan avulla niin IT-osastot, data-analyytikot, kuin myös devops- ja IT-turvatiimit pystyvät hyödyntämään organisaationsa dataa mistä tahansa lähteestä millä tahansa aikaikkunalla. [Splunk, 2021]

Yritys on perustettu vuonna 2003 ja tuotteesta on saatavilla erilaisia lisenssejä ja hinnoitteluvaihtoehtoja. Lisenssin mukaan tuotteen ominaisuudet voivat vaihdella, mutta karkeasti Splunk Coren lisenssimallit ovat: Free, Cloud ja Enterprise. Näissä hinnoittelu määräytyy Cloudin tilanteessa käytön mukaan ja enterprisessä haluttujen ominaisuuksien. Ilmaisversiossa ominaisuudet ovat hyvin riisutut ja sopii käytön testailuun. Lisäksi on hankittavissa dataprosessori, joka prosessoi lokit ja muut tietolähteet valmiiksi Splunkia varten. Tätä tarvitaan, jos kyseessä on esimerkiksi suuri konserni, jolla datamäärät ovat terabittiluokkaa. [Splunk, 2021]

5 Monitorointityökalun ominaisuuksien kartoittaminen

Kohdeyritys on suomalainen, perinteinen valmistavan tuotantotalouden yritys, joka myy tuotteitaan lähinnä tukkureille ja jälleenmyyjille. Sen liikevaihto on satoja miljoonia euroja ja se työllistää noin tuhat henkilöä ja sillä on toimintaa usealla suomalaisella paikkakunnalla. Yritys käynnisti toiminnanohjausjärjestelmänsä päivitysprojektin, jonka yhteydessä todettiin myös vanhan integraatioalustan vaativan päivitystä. Uudeksi integraatioalustaksi valikoitui Dell Boomi, jonka käyttöönottoprojektiryhmässä opinnäytetyöntekijä oli myös mukana. Käyttöönottoprojektin aikana huomattiin ylläpidollisia puutteita teknologioissa, jonka perusteella aloitettiin keskustelut monipuolisemmasta tavasta hallita ja monitoroida integraatioratkaisuja. Tämä opinnäytetyö pyrkiikin auttamaan löytämään ratkaisuja näihin keskusteluihin.

Työtä varten on haastateltu yrityksessä työskentelevää projektiryhmää, jonka työnä on ylläpitää Dell Boomi alustalle rakennettuja integraatioratkaisuja. Ryhmään kuuluu opinnäytetyön tekijän lisäksi Dell Boomi -kehittäjiä ja integraatioarkkitehti, joiden kanssa on käyty määrittelykeskusteluita mahdollisimman kattavan monitorointiratkaisun tuottamiseksi, jonka pääasiallisena tarkoituksena on auttaa yrityksen liiketoiminnalle kriittisen integraatioalustan ylläpidollisissa tehtävissä. Näistä tarpeista on muodostettu lista kohdista, joita monitorointiratkaisun tulee täyttää.

Haastattelua toteutettiin vapaamuotoisesti erinäisissä workshopeissa, suunnittelupalavereissa ja projektiryhmän työn ohella ilman varsinaista rakenteellista haastattelupohjaa tai agendaa. Tarkoituksena oli selvittää eri tahojen ajatuksia asiaan liittyen ja näin löytää käytännönläheinen listaus ongelmista ja vaatimuksista, joita lähteä ratkaisemaan suunnitelmallisesti.

5.1 Monitorointiratkaisun vaatimukset

Boomin AtomSpheressä on integraatioiden rakentamisen lisäksi tarjolla myös monitorointiratkaisu, jonka avulla voi seurata jokaisen prosessin suorituksia tietyllä tasolla. Tällöin esimerkiksi onnistuneet prosessiajot näkyvät monitorilokissa onnistuneina ja epäonnistuneet punaisina. Tämä ominaisuus perustuu graafiseen käyttöliittymään, joka tarjoaa Atomien kirjoittamista lokeista tietoja toivotulla tasolla. Käytön myötä on kuitenkin ilmentynyt joitain kohtia, joiden selvittäminen on ollut käyttäjille vaikeaa, aikaa vievää tai jopa mahdotonta. Keskitetyn monitorointiratkaisun tulisikin kyetä tarjoamaan ne ominaisuudet, jotka Boomin omassa työkalussa jo on, mutta myös korjaamaan puutteet tai ainakin toteuttamaan ne paremmin.

5.1.1 Virheiden hallinta

Sisäinen monitorointiratkaisu nostaa virheet näkymään punaisilla palloilla merkattuina. Näitä virheeseen menneitä suorituksia voi ajaa uudelleen suoraan käyttöliittymästä,

mutta ongelmaksi muodostuu se, että tästä uudelleenajosta ei jää merkintää. Tällöin esimerkiksi ylläpitävän tahon on mahdoton löytää tuhansien viestien joukosta niitä virheitä, mitkä on onnistuneesti jo uudelleenlähetetty.

Keskitetyn monitorointityökalun pitää pystyä vastaamaan tähän ongelmaan esimerkiksi lokittamalla uudelleenlähetys ja yhdistämään se alkuperäiseen virheeseen ja sen jälkeen tuomaan tämä tieto käyttäjälle graafisesti tai ohjelmallisesti selvästi esille, jotta käyttäjä pystyy tarkastamaan yhdestä paikasta virheeseen menneiden viestien tilanteet.

Boomin työkalusta myös virheiden etsiminen on hyvin vaikeaa, virheviestit ovat usein lyhennettyjä tai eivät sisällä koko virhettä esimerkiksi Javan heapia kokonaisuudessaan. Tällöin teknisen virheen etsiminen voi olla haastavaa.

5.1.2 Viestien sisällön lokitus

Boomin monitorointityökalusta viestien sisällön etsiminen on hankalaa ja vaatii useamman klikkauksen. Tällöinkin vaaditaan erillinen lokitusparametri käyttöön prosessissa, jolloin kehittäjillä on oltava myös tieto siitä, mitä viestintää loppukäyttäjä haluaisi seurata.

Keskitetyn monitorointiratkaisun tulisi sisältää myös sopivissa kohdin viestien sisältö alkuperäisten sisään tulevien viestien osalta, mahdollisten viestimuunnoksien jälkeen sekä ulospäin integraatioprosessista lähtevien viestien suhteen. Näin ylläpitävät tahot pystyvät tarkastamaan epäonnistuneiden viestien datan oikeellisuuden ja tekemään siitä tarvittavia johtopäätöksiä.

5.1.3 Irrallisuus pilviratkaisusta

Mahdollinen kriittinen ongelma Boomin alustassa liittyy AtomSphereen, joka on saatavilla ainoastaan Dellin pilvipalveluna. Tämä tarkoittaa sitä, että mikäli AtomSphere ei ole saatavilla esimerkiksi huoltoikkunan tai vikatilanteen takia, ei monitorointisovellukseen pääse myöskään käsiksi.

Sovelluksen ollessa asennettu paikallisesti, integraatiot kuitenkin jatkavat toimintaansa, mikä tarkoittaa myös sitä, että mahdollisia virhetilanteita voisi selvittää palvelintasolla, vaikka AtomSphere ei toimisi. Näin ollen keskitetyn monitorointiratkaisun tulisi olla irrallinen pilvipalvelusta, jotta se on saatavilla myös silloin, kuin AtomSphere ei ole.

5.1.4 Nykyiset ominaisuudet

Uuden monitorointiratkaisun tulee kyetä viestimään ongelmista sähköpostitse, mutta myös esimerkiksi SMS- ja pikaviestinhälytykset voivat olla hyödyllisiä ominaisuuksia. Erilaiset seurantagraafit tulee myös olla saatavilla, kuten esimerkiksi viestien määrä, käytetyt prosessit ja tiedostojen koko. Lisäarvoa toisi myös mahdollisuus tuoda palvelimen kuormitus- ja tilaseuranta samaan paikkaan, jotta sitä ei tarvitse eriyttää toisiin palveluihin.

Monitorointiratkaisuun on hyvä tuoda myös tilastotietoa virheiden lukumäärästä ja vasteajoista. Lisäksi prosessien tai palveluiden yhteysongelmien ja vikatilanteiden aiheuttamien palveluhäiriöiden kesto on tieto, mikä on hyvä tuoda keskitettyyn monitorointiratkaisuun.

5.1.5 Monitorointityökalun käyttöönotto

Jotta ulkoinen monitorointityökalu olisi hyödyllinen lisä integraatioiden ylläpitoon, tulisi sen olla helposti käyttöönotettava. Käytännössä tämä tarkoittaa sitä, että työkalun asentamisen tulisi olla mahdollista ilman suurta ymmärrystä itse sovelluksen käytöstä. Tämä voisi tarkoittaa esimerkiksi yksittäisen asennusohjelman tai paketin ajamista, sekä muutaman muuttujan asettamista palvelimella.

5.1.6 Kuormantasaajien ja palvelimien seuranta

Integraatiosovellusten palvelimilla käytetään usein myös kuormantasaajia. Nginx tarjoaa avoimen lisenssin työkalun kuormantasaajaksi ja sitä käytetäänkin usean Enterprise-tason IPaaS-palvelun yhteydessä tuottamaan kuormantasausta ja pääsynhallintaa. Tällöin sen lävitse kulkee liikennettä ja se muodostaa lokitietoa, jolloin myös sen tuonti keskitettyyn monitorointiratkaisuun vähentää ylläpidollista kuormaa. Vastaavasti palvelinten ollessa yrityksen sisäisessä seurannassa, vaativat ne myös monitorointia. Jotta tätä seurantaa varten ei tarvita erillistä palvelua, tulee keskitetyn ratkaisun kyetä myös tuottamaan monitorointidataa palvelimista.

5.1.7 Hinnoittelu

Kohdeyritys on uusimassa koko monitorointitoteutustaan, jolloin tuotteen tulisi vastata myös yrityksen muihin integraatioiden ulkopuolisiin tarpeisiin. Koska kyseessä on strateginen hankinta, myös tuotteen hinnoittelu-, ylläpito- ja lisenssiperiaatteet ovat osana hankintaprosessia.

5.2 Valitun monitorointityökalun esittely

Esitellyistä monitorointityökaluista Elastic Stack on ainoa avoimen lähdekoodin tuote, joka mahdollistaa käyttöönoton ilman lisenssimaksuja. Tarjolla on myös lisensoituja vaihtoehtoja, mutta pohjimmiltaan tuote on kuitenkin räätälöitävissä omaan käyttöön ja näin ollen potentiaalinen vaihtoehto tuotteeksi monitorointia varten.

Tässä piilee myös tuotteen heikkous, eli tuotteessa ei ole valmiina tukea Boomin lokitiedoille, jolloin koko sisäänlukuarkkitehtuuri tulisi hoitaa räätälikoodilla. Tällöin esimerkiksi lokien parsinta on määrämuotoista ja mikäli niihin tulisi muutoksia, pitäisi koko monitorointiputki rakentaa uusiksi.

Elasticin käyttöehdoissa on myös määritelty, että mikäli tehdyllä tuotteella on tarkoitus tehdä liikevaihtoa, tulee koko toteutus julkaista avoimesti saataville. Tämä voi

aiheuttaa tulojen menetyksiä tai joissain tapauksissa ongelmia, mikäli toiminallisuuksien julkaisemisesta on haittaa käyttävälle taholle.

Datadog on ominaisuuksiensa puolesta valmiimpi paketti ja eikä vaadi erillistä yksittäisten osien asennuksia käyttöympäristöön palvelun ollessa pilvipohjainen SaaS-ratkaisu. Vaikka suoraa Boomi-agenttia ei Datadogilta toistaiseksi löydy, on sen sisäänrakennetut ominaisuudet kuitenkin helpommin käyttöönotettavia, kuin avoimen lähdekoodin ratkaisussa.

Käyttötarkoitukseen soveltuvimmaksi osoittautui kuitenkin Splunk, joka yhdistää kahden muun tuotteen parhaat ominaisuudet hinnoittelun ja valmiiden ominaisuuksien suhteen.

Splunkilla on myös oma yhteisösivustonsa, jossa valmiiden Splunk-applikaatioiden jakaminen on vaivatonta. Täällä applikaatioille voidaan myös määritellä käyttömaksu, eli mikäli toteuttava taho haluaa tuotteistaa monitorointiratkaisun, on se myös mahdollista. Käyttämällä tätä palvelua, voi Splunk-applikaation tekijä tuoda applikaationsa saataville kaikille Splunkia käyttäville tahoille.

Tutkinnan aikana Splunkilta löytyikin jo erään konsulttiyrityksen tekemä toteutus Dell Boomia varten, joka vahvisti entuudestaan käsitystä siitä, että tämä tuote kykenee palvelemaan Boomin lokistruktuuria ja näin tuomaan lisäarvoa Boomia integraatioalustana käyttäville tahoille.

Lisähuomiona myös kehittävän tiimin kokemukset näistä tuotteista tukevat Splunkia, jonka ominaisuudet ovat parhaiten käytettävissä Boomin kanssa. Muut tuotteet vaativat enemmän räätälöintiä tai vastaavasti lisenssikustannuksia verrattuna Splunkiin, jonka käyntiin saaminen on verrattain kevyttä, eikä myöskään liian kallista. Alla olevassa taulukossa verrataan vielä, mitä toivottuja ominaisuuksia kustakin tuotteesta löytyi. Vaatimukset on numeroitu samassa järjestyksessä kuin kohdassa 5.1 ne on lueteltu.

Splunkin käyttöönotto on suoraviivaista. Lisenssin valinnan jälkeen siirrytään alustalle, jota voi käyttää soveltuvalla selaimella. Käyttöönotossa määritellään ensiksi seurattava kohde. Se voi kohdistua esimerkiksi kovalevyn valittuun kansioon, yksittäiseen tiedostoon, verkkoliikenteeseen tai http-liikenteeseen.

Valittu lähde määrittelee sen, miten käyttöliittymä käsittelee dataa ja formatoi sen nähtäville käyttäjälle. Mikäli seurattava kohde ei ole vakiomuotoinen, voi käyttäjä määritellä seurattavat parametrit itse ja miten ne parsitaan sisään Splunkiin.

Seuraavaksi määritellään datan indeksointi eli miten data tallennetaan. Kun asetukset ovat valmiit, voi käyttäjä tarkastella määrittelyt ja varmistaa, että monitoroitava tapahtuma tulostuu halutunlaisesti käyttöliittymään ja indeksoituu oikein.

Vaihtoehtoisia tapoja tuottaa dataa käyttöliittymään on käyttää joko Splunkin sisäisiä välittäjiä, jotka vastaanottavat ulkoisesta lähteestä tulevan datan. Lisäksi Splunkilla on valmiiksi rakennettuja niin sanottuja Splunk-applikaatioita, jotka virtaviivaistavat

datan sisäänlukua. Näissä Splunkin sisäisissä applikaatioissa parametrisointi ja muu konfiguraatio on tehty jo valmiiksi lähdejärjestelmän mukaiseksi.

Kun data on saatu luettua sisään Splunkiin, voi sitä tarkastella hakutoiminnallisuuden avulla. Nämä haut ovat eräänlaisia funktioita, joiden avulla datasta voidaan luoda visualisaatioita, raportteja ja dashboardeja. Käytännössä hakuihin syötetään erilaisia kenttä=arvo -pareja, Boolean-operaattoreita kuten AND, OR tai NOT ja merkkijonoja. Lisäksi hakuehdoiksi voi antaa aikamääreitä, kuten esimerkiksi viimeiset 24h tai päivämääräväli.

Työkalu tarjoaa myös oman logiikkansa mielestä tiettyjä esiintyviä avainsanoja ja näitä voi myös itse määritellä, jolloin lokirivit voidaan niputtaa kategorioihin hakujen optimointia varten. Näiden kategorioiden alta voidaan myös esimerkiksi tarkastella yleisimpiä arvoja, esimerkiksi esiintyykö operaationa useimmiten POST vai GET http-liikennettä tarkastellessa.

Erilaisia visualisointiopohjia on tarjolla useita ja niiden sisältöä voi määritellä käyttämällä hakuehtoja. Kaikki haut voidaan tallentaa käyttöliittymään ja sitä kautta myös dashboardeille, jolloin hakujen toistaminen on helpompaa, eikä niitä tarvitse kirjoittaa uudelleen aina.

Dashboardit ovat hyvin kustomoitavia ja niitä voi jakaa myös kehittäjien ulkopuolisille tahoille. Vastaavasti automaattiset päivitykset ovat myös mahdollisia, jolloin esimerkiksi graafien heijastaminen ajantasaisesti monitorille on helppoa. Dashboardeista voi ottaa myös PDF-kopion, josta dataa voi tarkastella irrallisena käyttöliittymästä. Tämän tiedoston säännöllinen lähettäminen esimerkiksi sähköpostiin onnistuu myös.

5.3 Toteutusarkkitehtuuri

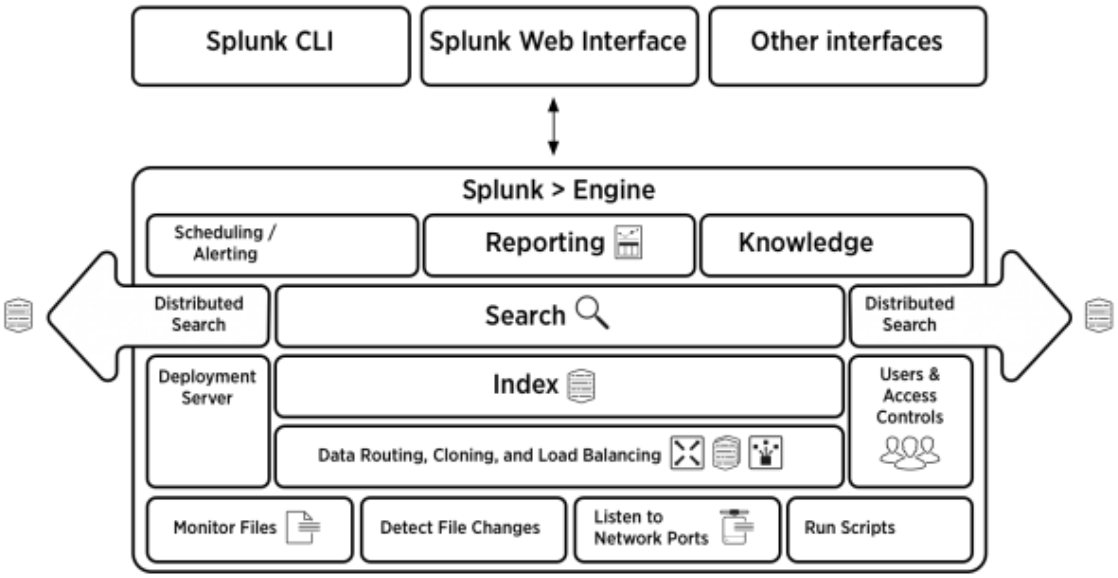
Boomi tuottaa päiväkohtaista, määrämittaista lokia ajoympäristöstänsä levylle. Tähän lokiin se kirjoittaa jokaisesta prosessista Notify-shapella tuotetut merkkijonot, mutta myös muuta metatietoa mm. Atomien ja Molekyylien tilasta, yhteysvirheistä ja muista ympäristön tapahtumista. Tämän lisäksi Boomi kirjoittaa mahdolliset Javan virhestackit lokiin, joiden avulla epäonnistuneiden prosessien vianselvitys on mahdollista.

Vastaavasti lokiin voidaan kirjoittaa myös haluttuja virheilmoituksia hallituista tilanteista, kuten esimerkiksi yhteysvirheestä johonkin kohdejärjestelmään. Prosessin sisältä voidaan kirjata myös viestien sisältöä tai prosessissa syntyvää metatietoa, joka kirjautuu lokille myös määrämittaisena.

Lokin määrämittaisuus on hyvin tärkeä käyttäessä lokeja sisään lukevia monitorointiratkaisuja. Nämä yleensä käsittelevät määritetyin ehdoin lokeja, jolloin on toivottavaa, että sisäänluettu loki täyttää ehdot.

Kuvassa 9 esitellään Splunkin yleinen enterprise-tason arkkitehtuuri [Splunk, 2021]. Tapausesimerkissä käyttö on alkuun hyvin yksinkertaista, kuten aiemmin esitetyssä tarvekartoituksessa on todettu. Näin ollen arkkitehtuuri ja vaatimukset sille ovat kevyem-

mät kuin suuremmissa käyttötapauksissa, joissa käyttökohteita ja käyttäjiä voi olla useita. Vastaavasti säilöttävien lokien määrä ei ole suuri, koska liikennettä integraatioalustalla ei ole tässä vaiheessa suuresti.



Kuva 9. Splunk Enterprise -arkkitehtuuri [Splunk, 2022].

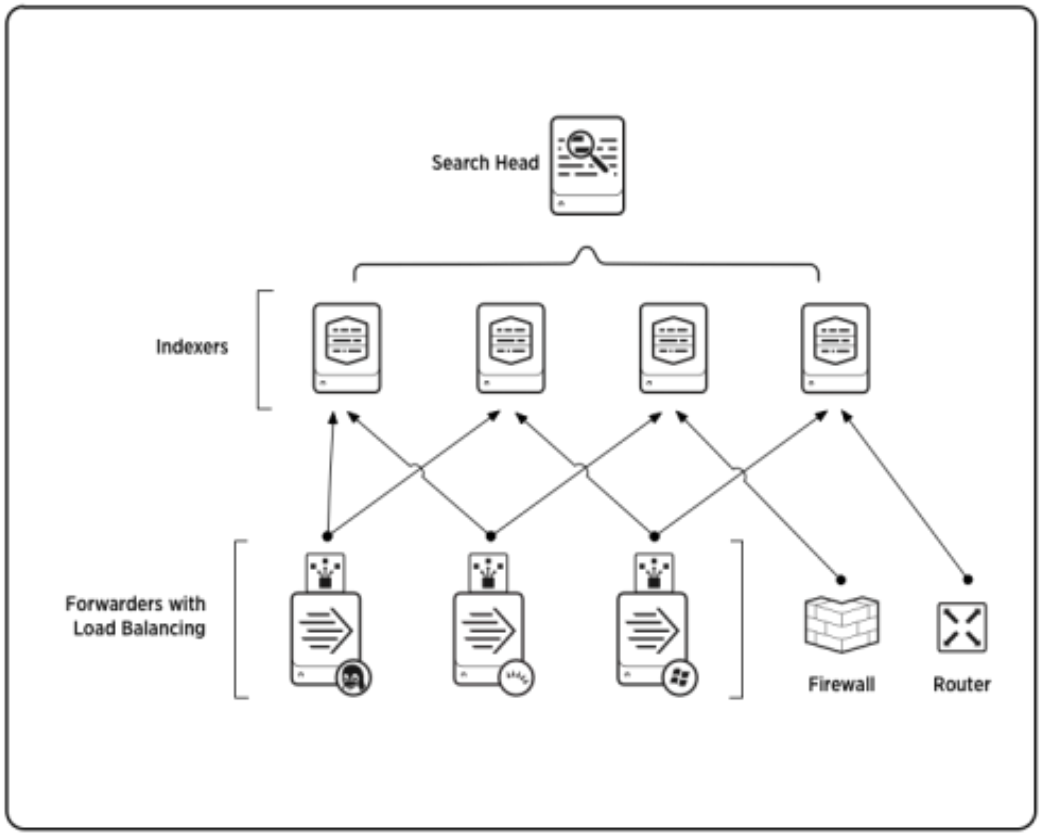
Lähtökohtaisesti Splunk-ratkaisu rakentuu kolmesta päätoiminallisuudesta, joita ovat keräys (Collection), indeksointi (Indexing) ja haku (Search). Keräystoiminnallisuudella tarkoitetaan Splunkin tarjoamia datan sisäänlukuominaisuuksia. Näitä on esimerkiksi verkkotason syötteet, tiedostomuutokset tai erilaiset http-tapahtumaherätteet. Verkkotasolta Splunk voi saada prosessorikuorman tasoa kuvaavaa dataa tai verkkoliikenteen määrää. Tiedostomuutoksia seuraava prosessi voi aloittaa datan sisään luvun, jos seurattuun kansioon muodostuu uusi tiedosto tai olemassa olevan tiedoston aikaleima muuttuu. Http-heräte voi olla sovelluksen lähettämä suojattu http-kutsu, joka esimerkiksi sisältää dataa Splunkin käsiteltäväksi. [Splunk, 2021]

Keräys-toiminnallisuus sisältää myös sisään tulevan datan käsittelyn toisen vaiheen eli käsittelyn. Tämä vaihe valmistelee sisääntulevan datan valmiiksi indeksointia varten esimerkiksi merkkien enkoodauksen, rivittämisen, erittelyn, aikaleimojen harmonisoinnin ja muiden vastaavien yleisluontoisten toimien avulla. Näin saatu data on samassa mallissa muun jo indeksoidun datan kanssa. [Splunk, 2021]

Indeksit ovat Splunkin sisäisiä tietovarastoja. Ne sisältävät kahdentyyppisiä tiedostoja: raakadatan pakattuna sekä indeksejä, jotka osoittavat raakadataan. Näiden lisäksi indekseissä on myös joitain metadatatiedostoja. Nämä tiedostot ovat ”bucketeissa”, jotka ovat aikaleiman perusteella järjestettyjä hakemistoja. Tämä mahdollistaa nopean ja joustavan hakutoiminnallisuuden, sekä ajastetun arkistoinnin. [Splunk, 2021]

Haku-toiminnallisudella tarkoitetaan Search & Reporting -applikaatiota, joka on primäärinen käyttöliittymä Splunkin käyttämiseen. Sen avulla voidaan tehdä hakuja, tallentaa raportteja ja tehdä dashboard-näkymiä. Applikaatiossa käytetään hakujen muodostamiseen Search Processing Languagea (SPL), joka mahdollistaa mm. tapahtumien hakuja indeksistä, laskea metriikkoja tai määritellä ajanjaksoja jolloin ajoja on tehty. [Splunk, 2021]

Asennustyö aloitetaan kohdeympäristössä asentamalla Splunkin windowspaketti halutulle palvelimelle. Valitun palvelimen ei tarvitse olla sama, mihin Boomi-instanssi on asennettu. Varsinkin tilanteissa, joissa Splunkin käyttö on laajamittaista, voi oman palvelimen tai palvelinklusterin rakentamista harkita. Kun ympäristö on valittu, ajetaan siinä sopiva asennuspaketti, joka asentaa kuvan 10 mukaiset komponentit.



Kuva 10. Splunk Enterprise asennuksen komponentit [Splunk, 2022].

Näitä komponentteja on aiemmin mainitut indeksoijat, jotka mahdollistavat datan prosessoinnin ja tallentamisen. Search head on Splunk Enterprise -instanssi, joka toimii hakujen jakajana indeksoijille. Välittäjät (Forwarder) välittävät dataa toisaalla oleville indeksoijille datan prosessointia ja tallentamista varten, ilman että toteuttavat indeksointia itsenäisesti.

Kun asennus on tehty, tulee määritellä mille muille palvelimille välittäjiä tarvitsee asentaa. Tämän määrittelee se, että onko tarkoituksena monitoroida myös muita palve-

limia, vai keskittyä pelkästään integraatioalustan palvelimeen ja integraatioalustan toiminnallisuuden seurantaan.

Tässä tapauksessa palvelin- ja verkkoliikennetason monitorointia on toteutettu toisella ohjelmistolla, joten niiden monitorointia ei tarvitse toteuttaa Splunkilla. Tämä sisältää myös integraatioalustan tiedostojen luku- ja kirjoituskohteen käyttämän SFTP-palvelimen toiminnallisuuden.

6 Toteutuksen arviointi

Monitorointiratkaisua varten tunnistettiin etukäteen yhdessä kehitystiimin arkkitehdin kanssa seitsemän vaatimusta, joiden toteutumisen perusteella toteutuksen onnistumista voidaan arvioida. Käytännössä nämä ovat asiakkaan prosessia varten suunniteltuja toiminnallisuuksia, mutta myös hyvin yleisiä integraatioratkaisuihin liittyviä monitorointitarpeita.

6.1 Arviointi

Tärkeimpänä kohtana arkkitehti nosti virheiden hallinnan. Boomin sisäänrakennetussa monitoroinnissa virheet voidaan tuoda esille selkeästi, mutta suurimpana ongelmana pidettiin yksittäisen viestin löytämistä nopeasti. Hankalaksi koettiin myös viestin sisällön tarkastelu, sillä varsinaisen sanoman etsiminen prosessilokilta vaatii paljon navigointia usean eri valikon välillä.

Keskitetyssä monitorointiratkaisussa tähän pystytään vastamaan hyvin. Kustomoituja lokeja parsimalla voidaan nostaa haluttu tekstikenttä suorana käyttöliittymään, jolloin käyttäjälle näkyy ainoastaan tarvittava tieto. Pelkästään tämä riittää useassa tilanteessa asiakasyritykselle, mikäli virhe on esimerkiksi yhteyksiin liittyvä tai virheelliseen tunnukseen liittyvä.

Toinen asiakkaan esittämä tarve monitorointityökalulle oli mahdollisuus nähdä helposti myös tiettyjen viestien sisältö virhetilanteiden sattuessa selvitystyötä varten. Käyttöliittymä mahdollistaa myös tämän toiminnallisuuden ja viestin sisältöä pystytään tarvittaessa katselmoimaan. Virhetilanteesta syntynyttä riviä klikkaamalla voidaan avata viestin sisältö, jos se on määritelty kyseisessä prosessissa tallennettavaksi. Esimerkki tilanteesta, jossa tietosisältö on hyvä tallentaa, on tilausliikenne. Näissä viesteissä monitorointityökalun käyttäjä pystyy etsimään suoraan epäonnistuneen tilausviestin lähettävän osapuolen ja ottamaan yhteyttä tähän tahoon virheiden selvittämiseksi.

Splunkin toiminnallisuus ei ole riippuvainen deployment-tyypistä eli instanssi voidaan asentaa täysin paikallisesti ilman julkipilviriippuvuutta. Käytännössä asennus tehdään tietylle virtuaalikoneelle, jonka kautta Splunk saa yhteyden tarvittaviin toisiin virtuaalikoneisiin ja näin ollen pystyy vastaanottamaan tarvitsemansa tiedot ympäristöistä.

Monitorointityökalu pystyy nostamaan joko suoria applikaation kohtaamia virheitä virheviesteiksi tai sitten tuottamaan kustomoitua, käyttötarpeen mukaista sisältöä virheviesteihin. Tarpeeksi todettiin mm. kustomoidut virheviestit sähköpostitse, joten myös tämä asiakkaan esittämä toiminnallisuusvaatimus voidaan toteuttaa monitorointiratkaisulla. Lisäksi työkalu tarjoaa myös integrointimahdollisuuden esimerkiksi Microsoft Teamsiin tai Slack-pikaviestinpalvelimiin, jolloin asiakkaan mahdollisiin uusiinkin tarpeisiin voidaan vastata ilman suurempaa kustomointia tai lisäkehitystä.

Myös viestien tarkempi tutkinta ja helppokäyttöisyys ei-teknisille henkilöille todettiin hyväksi nykyisessä monitorointiratkaisussa ja näitä ominaisuuksia asiakasyritys

toivoi myös uudelta monitorointiratkaisulta. Tuottamalla soveltuvat yksinkertaiset ohjauspaneelit, voidaan myös nämä datat tuoda esille Splunk-näkymiin. Näiden suunnittelussa on hyvä osallistaa ketterien kehitysmenetelmien mukaisesti loppukäyttäjiä, jotta oikeanlaiset käyttöliittymät ja raportit voidaan rakentaa tarvetta vastaaviksi.

Ketterissä kehitysmenetelmissä tunnustetaan 12 periaatetta, joista ensimmäinen on ”korkein prioriteettimme on täyttää asiakkaan tarve aikaisella ja jatkuvalla arvokkaan ohjelmiston tuottamisella” [Stellman & Greene, 2014]. Tällä tarkoitetaan yleistäen sitä, että pyritään luomaan mahdollisimman nopeasti arvoa asiakkaalle tekemällä pieniä osakokonaisuuksia ja tuomalla näitä loppukäyttäjän saataville mahdollisimman nopeasti sen sijaan, että kasataan suuri kokonaisuus ja julkaistaan se yhdellä kertaa.

Splunkin pilviversion käyttöönotto on verrattain suoraviivaista, eikä paikallisasennuskaan vaadi suuria investointeja tai prosesseja varsinkin, kun monitoroitavien ympäristöjen määrä on maltillinen. Kuten jo arkkitehtuurikuvauksessa todettiin, ei varsinainen asennus vaadi kuin tarvittavien asennuspakettien ajamista. Kun oikeat palvelimet on tunnistettu, pystytään asennukset tekemään näille palvelimille. Indeksoijien asentamiset tapahtuvat yksinkertaisesti dokumentaatiota seuraamalla ja ne on helppo saada tarvittaville palvelimille toimintaan: esimerkiksi asiakkaan ympäristössä Boomi-palvelimille, kuormantasaajapalvelimelle ja FTP-palvelimelle.

Asennuksen myötä saadaan lisäksi kuormantasaajapalvelin ja FTP-palvelin seurannan piiriin, mikä oli myös etukäteen vaadittu toiminnallisuus. Tässä vaiheessa muita seurattavia palvelimia ei ole tunnistettu, joten niitä ei tuoda monitorointialustan valvontaan. Näille palvelimille, joille lisäasennukset tehdään, voidaan toteuttaa muun muassa verkkoliikenne-, kovalevy- ja keskusyksikköseuranta. Näille seurannoille voidaan antaa kriittiset raja-arvot, joiden ylityttyä Splunk lähettää esimerkiksi sähköpostiviestin asiakkaan infrastruktuuritiimille tai muulle ylläpitävälle taholle.

Hinnoittelu Splunkilla on asiakaskohtaista, mutta yleisesti jaetusta materiaalista löytyy vaihtoehtoisia laskutusmalleja muutama. Näissä malleissa hinta määräytyy joko laskentatehon, määriteltyjen entiteettien (ympäristöt, sovellukset) mukaan tai sitten perinteisesti sisään indeksoidun datan määrän mukaan. Näistä asiakasyritykselle parhaiten soveltuu indeksoidun datan määrän mukaan tapahtuva hinnoittelu, koska sisään luettavaa dataa on suhteellisen vähän. Mikäli monitoroitavien palvelimien tai sovelluksien määrä kasvaa, voi Splunkin hinnoittelua muuttaa tarvittaessa, joten hinnoittelun puolesta asiakasyrityksen tarpeeseen pystytään alustalla myös vastaamaan.

6.2 Kehityskohdat

Vaikka Splunkissa on mahdollisuus kolmansien osapuolien tuottamiin liittymiin, ei varsinaista natiivia tukea Boomia varten ollut olemassa tätä työtä tehdessä. Näin ollen kaikki mahdolliset toiminallisuudet tulee toteuttaa käyttöönottovaiheessa, mikä tarkoittaa lisätyötä. Joissain kilpailevissa työkaluissa komponentit on viety pitemmälle ja räätälöity vastaamaan esimerkiksi Boomin tuottamaa lokia suoraan, jolloin muotoilua ei tarvitse toteuttaa itse.

Lisäksi esimerkiksi kilpailevassa tuotteessa Nodinitessä on rakennettu integraatiota vielä pitemmälle Boomin ja monitorointityökalun välille. Boomin toiminnallisuuksia voidaan käyttää myös AtomSpheren APIen kautta, joten esimerkiksi epäonnistuneiden ajojen tai lokitietojen kyseleminen onnistuu käyttämällä näitä rajapintoja. Nodinitessä nämä on yhdistetty monitorointityökaluun siten, että suoraan nappia painamalla työkalu voi laukaista epäonnistuneen ajon uudelleen, ilman Boomin käyttöliittymän käyttämistä. [Nodinite, 2023] Tämä Boomin integrointi valmiiseen pakettiin olisi myös Splunkissa hyödyllinen ominaisuus, vaikka sanottakoon, että Splunk ei ole profiloitunut pelkästään integraatiomonitorointityökaluksi, toisin kuin Nodinite.

Verrattuna Datadogiin Splunkin käyttöliittymässä on parannettavaa. Oikeiden lokitietojen löytäminen ja niihin liittyvien dashboardien tekeminen tuntuu välillä hieman vaikealta, mutta toisaalta tehtyjen tuotoksien käyttämien pohjana seuraaville toteutuksille oli kohtalaisen käytännöllistä.

Suuret datamäärät eivät ole Splunkille ongelma ja kun lokien parsimiseen käytetyn hakukielen on opetellut, sitä pystyy myös käsittelemään melko tehokkaasti. Toisaalta tässä on myös oma heikkoutensa, koska työkalun tehokas käyttö vaatii tuon kielen syntaksin ja toiminallisuuksien opettelua ja sisäistämistä kehittäjältä, joka työkalua käyttää. Tällöin uusien ominaisuuksien tuottaminen ja käyttöönotto vaatii aina kehittäjän, jolla on käsitys työkalusta ja sen kehtitämisestä. Gartner [2022] kertoo kuitenkin, että keskitetyn IT:n merkitys vähenee tulevaisuudessa ja järjestelmien kehitys- ja ylläpitovastuu siirtyy enenemissä määrin lähemmäksi liiketoimintaa. Tässä yhteydessä se tarkoittaisi sitä, että liiketoiminnalla tulisi olla kyky ja ymmärrys tulkita ja kehittää heille tärkeitä monitorointikokonaisuuksia ilman erillistä kehittäjäresurssia.

7 Yhteenveto

Opinnäytetyön tarkoitus on tuoda esille yleisesti integraatioalustojen toimintaperiaatteita ja niiden takana olevia arkkitehtuurisia ratkaisuja, esitellä muutama moderni integraatioalusta, sekä keskitetty monitorointiratkaisu. Työn pohjana toimineessa asiakasprojektissa on käytössä Dell Boomi -alusta, joten se valikoitu myös kohteeksi monitorointityökalun sovitukseen. Monitorointityökaluista käyttöön valikoitui Splunk helpon käyttöönoton, laajojen valmiiden kirjastojen ja erilaisten hinnoitteluvaihtoehtojen vuoksi. Lisäksi tiimistä löytyi jo aiempaa kokemusta Splunkin käytöstä, jolloin kehitystoimien aloittaminen todettiin kevyemmäksi.

Integraatioprosessien tuottamien virhelokien muotoilu on monitorointiratkaisun käyttöönoton kannalta tärkein askel. Boomin muodostama ”container-loki” on jo lähtökohtaisesti luettavassa muodossa monitorointialustalle, joten se ei vaadi lisätyötä. Kuvassa 11 on esimerkki lokin rakenteesta.

Tämä ”container-loki” muodostuu ajoympäristön tapahtumista, joita ovat esimerkiksi:

- Atomien tila (esim. käynnistymässä, sammunut, uudelleen käynnistys)
- Odottamattomat ajoympäristön virhetilanteet
- Boomi-prosessien kuuntelijoiden käynnistymis/sammumisilmoitukset
- Väliaikaisten tietojen automaattisten tyhjennysprosessien viestit
- AtomSphere-yhteyden kanssa kommunikointi

Boomi esittääkin, että ”container-lokien” monitorointi on tärkeä osa kokonaisvaltaista operationaalista monitorointistrategiaa [Boomi, 2022]. Kuten mainittu, loki on määrämittaista, jolloin sen sisäänlukeminen monitorointityökalun käytettäväksi on helppoa.

Column	Example Value
Timestamp (Local Timezone where Atom is installed)	Feb 28, 2017 12:08:51 PM EST
Log Level	INFO
Java Library and Class	[com.boomi.container.core.StatusReporter start]
Message	Starting Status Reporter

Kuva 11. Boomi-alustan ”Container-lokin” muotoilu.

Lokitasoja on neljä. Yleinen taso, INFO, indikoi tehtäviä, joita Atomi on toteuttanut. FINE-taso ilmaisee Atomin toimivan kuten on odotettu. WARNING indikoi virheestä, joka voi olla hetkellinen tai johtaa SEVERE-tason viestiin. Tämä taso indikoi, että on tapahtunut virhe, johon on puututtava. Näiden viestien mukana on yleensä myös virheviesti ja siihen liittyvä stack trace. Näiden avulla voidaan rakentaa tarvittavat herätteet jo suoraan monitorointialustalle, kuten esimerkiksi sähköpostiviesti ylläpitävälle taholle jokaisesta SEVERE-tason lokimerkinnästä. Yksityiskohtaisempaa tietoa lokille saa käyttämällä debug-tilaa hetkellisesti, jolloin yksittäisten virhetilanteiden selvittäminen saattaa olla helpompaa.

”Container-lokien” lisäksi voidaan nostaa määrämittaisia tekstitiedostoja, joiden avulla voidaan lokittaa myös muuta prosesseissa syntyvää meta- ja logiikkatietoa, joka ei varsinaisesti estä Atomin tai prosessien toimintaa. Tämänkaltaista tietoa voi olla esimerkiksi luettujen tiedostojen nimet, loogiset virheet datassa tai vaikka prosessin käynnistänyt partneri. Lokitiedostojen muotoilun kannalta oleellista on saada yhtenevät virheilmoitukset, jotta ylläpitävä taho saa tarvittavan tiedon riippumatta siitä, missä kohtaa prosessia tai missä integraatiossa virhe tapahtuu. Tarvittaviksi tiedoiksi tunnistettiin:

- Ajankohta
- Virheen nostanut prosessi
- Prosessin ajokohtainen-id
- Virheviesti
- Mahdollinen viestisisältö

Toinen käyttökohde löytyi myyntiprosessin valvontaan, jossa tarkoituksena on seurata liiketoimintakriittistä myyntitilaus- ja laskuliikennettä. Tilausputkessa tilaukset tulevat EDIFACT-muotoisina tiedostoina, jotka muunnetaan integraatioprosessissa XML-muotoiseksi, connectSpec-standardia (aiemmin OAGIS) käyttäväksi viestiksi. ConnectSpec-standardi on vastaava tuotanto- ja logistiikkataloudessa käytetty viestimuoto, kuin EDIFACT, mutta pohjaa XML-datatyyppeihin eikä EDI-tiedostoihin [OAGi, 2023]. Taustajärjestelmän rajapinnat odottavat saavansa tietueet tässä connectSpec-muodossa, joten tämä muunnettu viesti välitetään taustajärjestelmän tarjoamalle rajapinnalle.

Tästä kokonaisuudesta tunnistettiin tietoja, jotka tulisi nostaa monitorointialustalle ja myös tilanteita, joista tulee lähettää heräte vastuuhenkilöille.

Toiminnan seurannan kannalta keskitetyn monitorointialustan käyttöliittymässä tulee näkyä myös onnistuneet viestit. Määrittelyvaiheessa todettiin, että viestien sisältöä ja aihetta tulee voida seurata helposti ja niihin tehdä kohdennettuja hakuja. Seurattavia arvoja on esimerkiksi tietyn partnerin lähettämien tilauksien lukumäärä, kokonaislukumäärät ja myös hiljaisuuden valvonta; mikäli tilauksia ei tiettyjen reunaehtojen sisällä saavu, tulisi tästä nostaa hälytys sovitulle taholle. Yksittäisten tilauksien metatietoja voi

myös nostaa tilauksien sisältä nähtäville, kuten esimerkiksi tilausrivien lukumäärä, euromäärät tai toimituspaikat. Näitä toivottavia ominaisuuksia voidaan kehittää tuotantoon viennin jälkeen, kun liiketoiminnalta tarpeita nousee esiin.

Metriikan on hyvä näyttää myös kaikkien viestien lukumäärää, jotta mahdolliseen kuorman äkkinäiseen kasvuun voi reagoida reaaliaikaisesti tai vastaavasti seurata graafeja pitemmältä aikajanalta. Tämä auttaa ylläpidollisissa toiminnoissa, jos esimerkiksi kuormantasaajassa on vikaa tai jokin tietty integraatio kuluttaa liikaa resursseja.

Virhetilanteiden ollessa yleensä joko huomautuksia tai fataaleja, on hyvä virittää automaattihälytysviestintä siten, että ainoastaan reagointia vaativia hälytyksiä nostetaan käyttäjille asti esimerkiksi sähköpostin avulla. Varoitukset ja muut alemman tason lokientryt voidaan nostaa kyllä monitorointiratkaisun seurantasivulle, mutta niistä ei kannata lähettää erillistä viestiä, jotta oikeat hälytystilanteet eivät jää huomaamatta.

Tämänkaltaisia kriittisiä, heti toimintaa vaativia virhetilanteita tunnistettiin muutama erityisesti liike- tai varastotoimintakriittisistä integraatioista. Kohdeyrityksen tuotantoprosessia ohjaa edellisten öiden aikana tulleet tilaukset, joten jos tilauksia ei ole kuulunut, myös tuotannon ohjaus vaikeutuu. Näin ollen hiljaisuudenvalvonta ja epäonnistuneet tilauskirjaukset kohti toiminnanohjausjärjestelmää tulee nostaa päivystävälle taholle välittömästi.

Integraatioalusta kirjoittaa lokeille hyvin paljon kaikesta toiminnastaan, mikä tarkoittaa runsasta määrää erilaisia warning-tason ilmoituksia, jotka eivät vaadi minkäänlaisia toimenpiteitä. Toisaalta joissain tilanteissa alusta nostaa myös error-tason hälytyksiä, kuten esimerkiksi mikäli jokin pakolliseksi määritelty elementti puuttuu tilaussanomasta. Nämä eivät kuitenkaan estä teknisesti eivätkä liiketoiminnallisesti tilauksien läpikulkua, joten ne voidaan tulkita turhiksi. Tämänkaltaisten virhetilanteiden hallintaan ei löytynyt sopivaa keinoa, joten ainakin toistaiseksi myös näistä nousee virhe ylläpitotaholle.

Paikallisasennuksen edut suhteessa pilvipalveluversioon nousivat myös esiin projektin aikana. Käyttäjäyrityksen toimintojen ollessa lähinnä paikallisasennuksia iPaaSia myöten, on myös luonnollista liittää monitorointisovellus samaan sisäverkkoon. Tällöin esimerkiksi lokeja lukevien prosessien käyttäminen ei vaadi erillistä palomuurien avausta tai erillistä verkkotason konfiguraatiota, mitä pilviversio puolestaan vaati. Toisaalta tämän kaltaisella arkkitehtuurilla asiakkaan infrastruktuuri- ja liiketoimintatiedot pysyvät täysin omassa hallinnassa, kun tietoa ei viedä julkipilven puolelle.

Tulevaisuudessa monitorointiratkaisua voi kuitenkin vielä uudelleen evaluoida, jos esimerkiksi yrityksen tarpeet tulevat muuttumaan radikaalisti tai esimerkiksi integraatioalusta vaihtuu. Jatkotutkimuksia tulee tehdä kuitenkin Splunkin suhteen mm. realisoituneista kustannuksista ja käytännön tason kehittämisestä: Onko järjestelmää tarpeeksi helppo kehittää, että uusia lähteitä ja raportteja on riittävän helppo tehdä? Mikäli järjes-

telmä osoittautuu vaikeakäyttöiseksi tai vaikeaselkoiseksi käytännössä, on myös näiden taustoja hyvä selvittää esimerkiksi käyttökokemuskatselmoinnin kautta. Vastaavasti toteutuksen oikea suorituskyky ja lisenssin määrittämien ehtojen seuraaminen voidaan toteuttaa vasta pidemmän tuotantokäytön avulla.

Sovelluskehittäjien mielestä jatkossa hyväksi todettua arkkitehtuuria ja valvontalogiikkaa voitaisiin käyttää myös muissa toimituksissa, joissa teknologiavalikoima on sama. Tällöin jokaisessa toimituksessa monitorointiratkaisua ei tarvitsisi kehittää uudelleen tai selvittää yhteensopivuutta Boomin kanssa, vaan voitaisiin lähteä rakentamaan monitorointia valmiilla sapluunalla. Tätä skaalautuvuutta ja yleiskäyttöisyyttä, sekä siihen liittyviä sudenkuoppia voidaan selvittää lisää tulevaisuudessa.

8 Lähdeluettelo

Anandamurugan, S. & Priyaa, T. 2014. Service Oriented Architecture. Nova Science Publishers.

Araujo, P. & Bastos, J. 2019. Hands-On Infrastructure Monitoring with Prometheus. Packt Publishing.

Beaulieu, A. 2020. Learning SQL, 3rd edition. O'Reilly Media, Inc.

Boomi. 2023. Verkkosivu. *https://boomi.com/platform/#/d48d01/home*

Bye, P. &, Britton, C. 2004. Middleware: Strategies for Building Large, Integrated Systems. Addison-Wesley Professional.

Byrne C. 2018. Integration and the Path to Becoming a Digital Business. In: O'Reilly Strata Conference

Chappell, D. 2004. Enterprise Service Bus. O'Reilly Media, Inc.

De, B. 2017. API Management: An Architect's Guide to Developing and Managing APIs for Your Organization. Apress

Elastic. 2023. Verkkosivu. *https://www.elastic.co/what-is/elasticsearch*

Fowler, M. 2014. Verkkosivu. *https://martinfowler.com/articles/microservices.html*

Gartner. 2023. Verkkosivu. *https://www.gartner.com/en/information-technology/glossary/information-platform-as-a-service-ipaas*

Greene, J. & Stillman, A. 2014. Learning Agile. O'Reilly Media, Inc.

Giordano, A. 2010. Data Integration Blueprint and Modeling: Techniques for a Scalable and Sustainable Architecture. IBM Press.

Gulledge, T. (2006), What is integration? Industrial Management & Data Systems, 106(1), 5–20

Guttridge, K. & Zakheim, B. 2018. Where Does Point-to-Point Integration Belong in Your Integration Strategy? Gartner Research.

Linthicum, D. 1999. Enterprise Application Integration. Addison-Wesley Professional

Malepati, T., Shah B. & Vanier V. 2019. Advanced MySQL 8. Packt Publishing.

National Institute of Standards and Technology (NIST). 2011. Erikoisjulkaisu. *https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf*

Newcomer, E. 2002. Understanding Web Services. InformIT.

Newman, S. 2021. Building Microservices, 2nd Edition. O'Reilly Media, Inc.

Nodinite. 2023. Verkkosivu. *https://docs.nodinite.com/*

OAGi. 2023. Verkkosivu. *https://oagi.org/pages/connectspec*

Reeve, A. 2013. Managing Data in Motion. Morgan Kaufmann.

Richards, M. 2016. Microservices vs. Service-Oriented Architecture. O'Reilly Media, Inc.

Schneider, J. 2020. SRE with Java Microservices. O'Reilly Media, Inc.

Splunk. 2023. Verkkosivu. *https://docs.splunk.com/Documentation/Splunk/8.2.3/Installation/Splunksarchitectureandwhatgetsinstalled*

SSH Communications Security. 2023. Verkkosivu. *https://www.ssh.com/academy/ssh/sftp-ssh-file-transfer-protocol*

UNECE. 2023. Verkkosivu. *https://unece.org/trade/uncefact/introducing-unedifact*

Welkenbach, P., Liebhart, D. & Schmutz, G. 2010. Service-Oriented Architecture: An Integration Blueprint. Packt Publishing.

Weir, L. & Nemec, Z. 2019. Enterprise API Management. Packt Publishing

Weir, L. 2017. Verkkosivu. *https://www.soa4u.co.uk/2017/03/ipaas-what-is-it-exactly-is-it-on.html*