Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Tomi Lehto

Järjestelmäintegraatiot IBM:n työkaluilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

2.12.2019

Tekijä Otsikko Sivumäärä Aika	Tomi Lehto Järjestelmäintegraatiot IBM:n työkaluilla 37 sivua 2.12.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Simo Silander Senior Project Manager Pekka Saxberg

Tämän insinöörityön tavoitteena oli luoda järjestelmäintegraatiot tiettyjen yrityksen järjestelmien välille sekä luoda rajapinta, jota kautta yrityksen ulkopuoliset instituutiot voisivat kysellä järjestelmän tuottamaa tietoa REST-kutsuilla käyttämällä GET-verbiä.

Työ voidaan jakaa kahteen osaan. Ensimmäinen osa oli luoda integraatiot yrityksen sisäisten järjestelmien välille, joka sisälsi kaksi itsenäistä integraatioita. Yksi integraatio toimitti dataa tietovaraston FTP-palvelimelta projektia varten luodulle mikropalvelulle ja toinen otti vastaan REST-kutsuja asiakkuudenhallinnasta, rikasti kutsussa tulleen datan ja toimitti rikastetun datan mikropalvelulle. Toinen osa työtä oli tarjota rajapinta ulkopuolisille instituutioille, jota kautta voitaisiin kysellä tietoa mikropalvelulta, johon isona osana kuului molempien, palvelimen ja asiakkaan validointi käyttäen varmenteita. Molemmat osat toteutettiin käyttämällä IBM:n tuoteperheen työkaluja.

Työssä esitellään eri integraatiomalleja, jotka määrittelevät erilaisia tapoja luoda järjestelmäintegraatioita. Työssä tutustutaan myös varmenteilla tehtävään validointiin liittyvään teoriaan ja termeihin, joita ovat muun muassa SSL/TLS, TLS-kättely, SNI ja varmenteiden eri tasot ja niiden myöntäminen.

Työn tuloksena saatiin toimiva järjestelmäkokonaisuus, joka on tällä hetkellä testauksessa ja tullaan viemään tuotantoon, kun testaus on valmistunut.

Avainsanat	integraatio, IBM, varmenne, SSL, TLS, API, REST

Author	Tomi Lehto
Title	Enterprise Architecture Integrations Using IBM Tools
Number of Pages	37 pages
Date	2 December 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Simo Silander, Senior Lecturer Pekka Saxberg, Senior Project Manager

The goal of this engineering thesis was to create system integrations between enterprise's different systems and to create an interface through which third-party institutions could retrieve data from the system through REST requests using GET.

The study can be divided into two parts. The first part was to create integrations between the enterprise's internal systems. The integrations consisted of two separate connections: one delivered data from a data storage FTP server to a microservice created for this project, the other received data through a REST endpoint, enriched the data and delivered it to the microservice. The second part was to create an interface for third-party institutions to retrieve data from the microservice, of which mutual authentication through certificates was a big part. Both parts of the study were created using IBM tools.

The study presents a number of integration models that define different ways to create integrations and introduces theory and terms related to authentication using certificates such as SSL/TLS, TLS handshake, SNI and different levels of certificates and authorities that issue them.

As a result, a working system was created that is currently undergoing testing and will be deployed to production when the testing has finished.

Keywords	integration, IBM, certificate, SSL, TLS, API, REST

Sisällys

Lyhenteet

ACE	APP Connect Enterprise. IBM:n kehittämä ESB-alusta integraatiokehitykseen.
AIA	Agile Integration Architecture. IBM:n nimitys integraatiomallille, jossa integraatiot puretaan pienempiin osiin hallittavuuden skaalautuvuuden ja järjestelmän toimintavarmuuden parantamiseksi.
API	Application Programming Interface. Ohjelmointirajapinta.
BAR	Broker Archive. Pakattu tiedosto, joka sisältää resursseja kuten applikaatioita, kirjastoja yms.
CA	Certificate Authority. Luotettu instituutio, joka myöntää ja hallitsee varmenteita.
EAI	Enterprise Architecture Integration. Järjestelmäintegraatio. Eri teknologioista koostuva integraatiokehys, jonka avulla organisaation hajautunut data muokataan ja toimitetaan sitä tarvitseville järjestelmille.
ESB	Enterprise Service Bus. Integraatiomalli.
ESQL	Extended Structured Query Language. IBM Integration Busin määrittelemä ohjelmointikieli, jota voidaan käyttää datan manipulointiin viestivirroissa.
FTP	File Transfer Protocol. Tiedonsiirtomenetelmä palvelimen ja asiakkaan välillä. Käyttää TCP-protokollaa.
HTTP	Hypertext Transfer Protocol. Tiedonsiirtoon käytetty protokolla.
HTTPS	Hypertext Transfer Protocol Secure. Tiedonsiirtoa, jossa viestit salataan TLS-protokollalla ennen niiden lähettämistä HTTP-protokollalla.
REST	Representational State Transfer. Arkkitehtuurimalli web serviceille.

SNI	Server Name Indication. TLS-protokollan laajennus, joka mahdollistaa useampien palvelinvarmenteiden tarjoamisen.
SOA	Service-Orientend Architecture. Palvelukeskeinen arkkitehtuuri.
SSL	Secure Sockets Layer. Vanhentunut tietoliikenteen salausprotokolla. TLS:n edeltäjä.
TCP	Transmission Control Protocol. Tietoliikenneprotokolla, jolla järjestelmät pystyvät luomaan yhteyksiä ja lähettämään dataa internetin yli.
TLS	Transport Layer Security. Salausprotokolla, jota käytetään suojaamaan verkon yli tapahtuva viestiliikenne palvelimen ja asiakasohjelman (esimerkiksi selain) välillä.
UDP	User Defined Properties. Viestivirtatason muuttujia, joita voidaan määrittää ACE/IIB:llä.
XSLT	Extensible Stylesheet Language Transformations. XML-muunnoksiin käytettävä merkintäkieli.
YAML	YAML Ain't Markup Language. Yleensä konfiguraatioitiedostoissa käytetty kieli. API Connectin rajapintojen lähdekoodi on graafisessa käyttöliittymässä tässä muodossa.

1 Johdanto

Organisaatioiden laajentuessa ja vanhentuessa myös käytössä olevien järjestelmien kirjo kasvaa. Vanhojen järjestelmien päälle otetaan käyttöön yhä useampia uusia järjestelmiä, joilla halutaan tuoda lisäarvoa organisaatiolle. Usein nämä järjestelmät eivät kuitenkaan osaa suoraan keskustella keskenään, jolloin tarvitaan integraatiota. Integraatiolla automatisoidaan tiedon kuljettaminen organisaation eri järjestelmien välillä sekä sen muokkaaminen sellaiseen muotoon, jota kohdejärjestelmä pystyy käsittelemään.

Yleensä integraatiolla pyritään tehostamaan organisaation toimintaa ja tuomaan enemmän tietoa organisaation käytettäväksi. Tässä työssä tarve integraatiolle ei tullut kuitenkaan halusta saada lisää arvoa organisaation hallussa olevasta tiedosta, vaan lainsäädännöstä, joka vaatii, että tietynlainen tieto pitää olla saatavilla valituille instituutioille.

Työ on osa isompaa projektia, jota on tekemässä useita osapuolia. Työn tavoitteena on toteuttaa tarvittavat integraatiot organisaation sisäisten järjestelmien, kuten tietovaraston, asiakkuudenhallinnan ja tätä projektia varten luodun mikropalvelun välille, jotta niistä saadaan yksi toimiva kokonaisuus, joka pystyy tarjoamaan vaaditun tiedon kolmannen osapuolen instituutioille. Projektissa luodaan myös kyseisille instituutioille tapa noutaa tietoa suoraan edellä mainitusta mikropalvelusta, joka on toisen osapuolen tätä projektia varten tekemä palvelu.

Projekti koostuu kahdesta osasta:

- Integraatiot IBM:n App Connect Enterprise alustaa käyttäen. Näihin kuuluvat integraatio tietovaraston ja mikropalvelun välillä sekä integraatio asiakkuudenhallinnan ja mikropalvelun välillä.

- Mikropalvelun rajapinnan tarjoaminen turvallisesti kolmannen osapuolen instituutioiden käyttöön, mikä toteutetaan IBM:n API Connect -työkalulla.

2 Järjestelmäintegraatio

Yrityksien keräämän tiedon määrän kasvaessa on tärkeää, että tieto on myös saatavilla ja käytettävissä, kun sitä tarvitaan. Usein tieto on kuitenkin hajautuneena useisiin eri järjestelmiin. Yrityksen järjestelmät saattavat myös olla eri aikakausilta ja toteutettuina täysin eri arkkitehtuureilla, jolloin nämä järjestelmät eivät pysty suoraan keskustelemaan keskenään. Yritykset eivät myöskään yleensä pysty suoraan korvaamaan näitä vanhoja, niin kutsuttuja legacy-järjestelmiä, koska ne ovat usein kriittisiä yrityksen toiminnalle. Uudet järjestelmät ja applikaatiot on siis jotenkin saatava toimimaan vanhemmalla arkkitehtuurilla toteutettujen järjestelmien kanssa. Järjestelmäintegraatio, eli EAI (Enterprise Architecture Integration), on ratkaisu yllämainittuihin ongelmiin. [1, s. 5.]

Järjestelmäintegraatiolla pyritään yhdistämään kaksi tai useampi keskenään yhteensopimaton järjestelmä toisiinsa sekä automatisoimaan tiedonsiirto näiden järjestelmien välillä. Joskus järjestelmäintegraation tehtävä on yksinkertaisesti siirtää tietoa paikasta A paikkaan B. Näin ei kuitenkaan ole suurimmasta osassa tapauksia, sillä erilaisia taustajärjestelmiä voi olla useita, joten yhden järjestelmän tarjoama tieto ei yleensä ole yhteensopiva toisen järjestelmän käsittelyn kanssa. Integraation täytyy siis muuntaa data kohdejärjestelmän ymmärtämään muotoon ennen sen lähettämistä kohdejärjestelmälle. Nämä muunnokset voivat olla hyvinkin monimutkaisia. Yksittäinen työkalu ei pysty vastaamaan kaikkiin tarpeisiin, mitä integraatio saattaa vaatia, joten järjestelmäintegraatiota ei kannatakaan ajatella yksittäisenä työkaluna, vaan enemmänkin kokoelmana työkaluja, suunnitelmia ja menetelmiä [2, s. 57]. Järjestelmänintegraation ajatteleminen työkaluna onkin yksi ”ansoista”, johon integraatioprojekti voi kaatua [3].

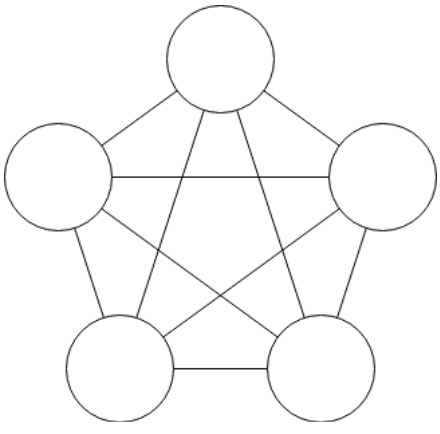
2.1 Integraatiomalleja

Koska järjestelmäkokonaisuuksia on monia erilaisia, se mitä integraatioilta vaaditaan, vaihtelee merkittävästi järjestelmästä toiseen. Eri tilanteisiin ja tarpeisiin on kehitetty erilaisia tapoja toteuttaa integraatio. Seuraavaksi käydään yleisesti läpi eri integraatioiden ratkaisumalleja, joihin ne soveltuvat, mitkä ovat niiden heikkoudet sekä mitä mallia tässä työssä käytettiin ja miksi.

2.1.1 Point-to-point-integraatio

Point-to-point-integraatiossa kaksi eri järjestelmää yhdistetään suoraan toisiinsa. Integraatio hoitaa kaikki tarvittavat toimenpiteet, kuten yhteyden lähtö- ja kohdejärjestelmiin sekä mahdolliset datamuunnokset. Käytännössä tällainen integraatio voitaisiin esimerkiksi tehdä, kun asiakas päivittää tietojansa asiakkuudenhallintaan, ja se tieto pitää saada päivitettyä myös toiminnanohjausjärjestelmään. Asiakkuudenhallinnan päässä integraatio luodaan ottamaan dataa vastaan juuri tästä järjestelmästä, ja toiminnanhallinnan päässä integraatio täytyy muokata lähettämään dataa toiminnanhallinnan vaatimusten mukaan. Välissä tietysti datan muokkaus toiminnanhallinnan vaatimaan muotoon. Molemmissa päissä on siis juuri sille järjestelmälle räätälöity integraatio. Point-to-point-integraatiot ovat yleensä siis järjestelmille yksilöityjä, eikä niitä voida uudelleen käyttää muissa integraatioissa.

Point-to-point-malli toimii hyvin, jos integroitavia järjestelmiä on pieni määrä. Kun järjestelmiä on kaksi tai kolme, pystytään integraatio pitämään vielä kevyenä sekä räätälöimään se tarkasti järjestelmien vaatimusten mukaiseksi. Kolmella järjestelmällä tarvitaan vain kolme point-to-point-integraatiota, jotta kokonaisuus saadaan täysin integroitua, mutta jo viisi järjestelmää lisää tarvittavien liitosten määrän kymmeneen (kuva 1). Kahdeksan tai yhdeksän järjestelmän kohdalla liitosten määrä nousee jo yli kolmenkymmenen. Tarvittavien liitosten määrä kasvaa siis hyvin nopeasti, kun liitettävien järjestelmien määrä lähtee nousuun, eikä point-to-point-malli ole enää käytännöllinen. [4.]



Kuva 1. Viiden järjestelmän välinen point-to-point integraatio.

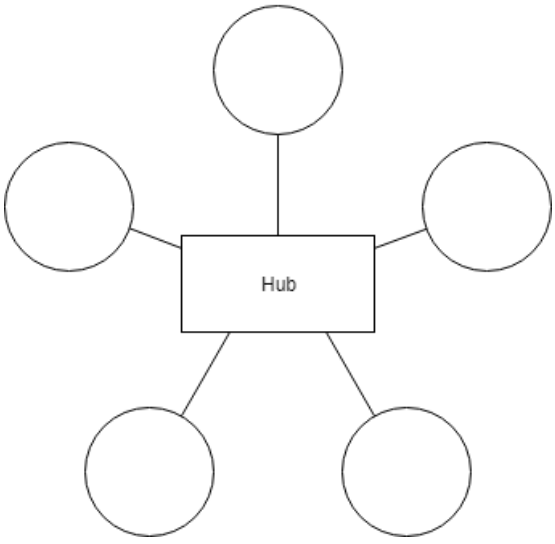
Tässä työssä tarvittiin vain kaksi liitäntää:

- tietovarasto -> mikropalvelu
- asiakkuudenhallinta -> mikropalvelu.

Koska integroitavia järjestelmiä on pieni määrä, ja molemmat yhteydet vaativat omanlaisensa liitännät ja datamuunnokset, olisi point-to-point-malli varsin toimiva vaihtoehto työn toteuttamiseen.

2.1.2 Hub and spoke -integraatio

Point-to-point-malli toimii, kun integroitavien järjestelmien määrä pysyy pienenä, mutta entä jos kyse on suuremmasta järjestelmäkokonaisuudesta tai jos se halutaan pitää skaalautuvana tulevaisuuden laajennuksia varten? Hub and spoke -malli pyrkii löytämään ratkaisuja näihin ongelmiin. Päinvastoin kuin point-to-point-mallissa, hub and spoke -malli, tai niin kutsuttu keskitetty malli, ei vaadi suoria yhteyksiä järjestelmien välille, vaan kaikki integroitavat järjestelmät ottavat yhteyden samaan keskeiseen komponenttiin (kuva 2). Tämä komponentti hoitaa liikenteen reitityksen dataa jakavien ja vastaanottavien järjestelmien välillä.



Kuva 2. Viiden järjestelmän hub and spokes -mallin mukainen integraatio.

Etuina point-to-point-malliin verrattuna keskitetty malli skaalautuu huomattavasti paremmin ja uuden komponentin lisääminen integraatioon ei vaadi useita uusia liittymiä. Keskitetty malli myös yksinkertaistaa asioita niin arkkitehtuurin kuin tietoturvan kannalta sekä vähentää mahdollista toistoa, jos samankaltaisia integraatioita on useita [5].

Haittapuolina ovat resurssit, jotka keskitetty integraatio vaatii alkupanostuksena. Malli vaatii myös, että integraatiot, jotka liittyvät keskukseen, noudattavat tiettyä mallia, joten voi olla erittäin vaikeaa luoda integraatiokeskusta, joka pystyy käsittelemään dataa hyvin erilaisista lähteistä. Kaikki sanomat kulkevat keskitetyn hubin kautta, joten sanomien määrän kasvaessa siitä voi muodostua järjestelmän pullonkaula. Järjestelmäkokonaisuus on myös täysin keskuksen varassa, joten jos se kaatuu, koko järjestelmä kaatuu. [4.]

2.1.3 Enterprice Service Bus

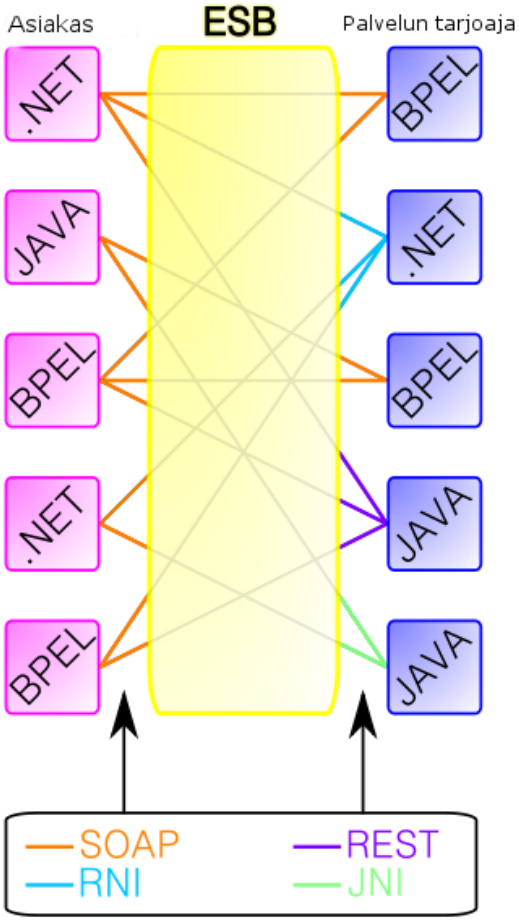
Vaikka hub and spoke -malli ratkaisi joitakin point-to-point-mallin ongelmakohtia, se toi mukanaan omat ongelmansa. Keskitetyn komponentin rakentaminen ja käyttöönotto vaativat ison määrän resursseja, ja vaikka uusien järjestelmien lisääminen on helpompaa, täytyy lisättävien järjestelmien olla keskitetyn komponentin kanssa yhteensopivia. Enterprise Service Bus tai ESB pyrkii tarjoamaan paljon samoja hyötyjä kuin aiemmin mainittu hub and spokes -malli, mutta korjaamaan sen joustamattomuuden, mitä tulee lisättävien järjestelmien yhteensopivuuteen.

ESB:n yhteydessä täytyy myös mainita palvelukeskeinen arkkitehtuuri (SOA), jota ESB on keskeinen osa. Palvelukeskeinen arkkitehtuuri määrittelee järjestelmäkokonaisuuden, jossa ohjelmistokomponentit tarjoavat palveluja toisilleen uudelleenkäytettävien rajapintojen kautta. Kun nämä rajapinnat käyttävät yleisiä kommunikointistandardeja, saadaan uudet komponentit integroitua helposti osaksi järjestelmäkokonaisuutta. Komponenttien uudelleenkäytettävyys on yksi periaate, johon SOA pyrkii. Palvelut ovat lisäksi löyhästi kytkettyjä, joten niitä voidaan kutsua tietämättä paljoakaan siitä, miten integraatio on pinnan alla toteutettu. [6.]

ESB-mallin mukaisessa integraatiossa sanomat kulkevat edelleen keskitetyn reitittimen kautta (kuva 3). Mutta toisin kuin yhteen komponenttiin luottavassa integraatiossa, ESB:llä ei ole vain yhtä keskitettyä komponenttia (hubia), joka hoitaisi kaikki integraation

reititykset, datamuunnokset, tietoturvan yms. EBS:ssä nämä kaikki eri toiminnallisuudet voidaan kapseloida omiksi kokonaisuuksikseen, jolloin yhdelle komponentille ei aiheudu niin suurta taakkaa, vaan se jakautuu tasaisemmin useille yhteen toimiville palveluille, jotka keskustelevat keskenään [4]. Näin pullonkaulaa ei synny niin helposti kuin hub and spokes -mallin mukaisesti toteutetulla integraatiolla. ESB pystyy myös hoitamaan kuluvalvonnan, sanomien lokituksen ja virhetilanteiden hallinnan, mikä lisää järjestelmän tietoturvaa.

ESB on pohjimmiltaan siis arkkitehtuurimalli, jonka mukaan integroidaan useita eri järjestelmiä keskustelemaan keskenään väylämäisen infrastruktuurin ylitse [7]. ESB on erityisen hyödyllinen, kun halutaan tuoda legacy-järjestelmän tieto modernien järjestelmien käytettäväksi. ESB:llä pystytään reaaliajassa hakemaan tietoa vanhasta järjestelmästä, tekemään sille tarvittavat muunnokset ja julkaisemaan sen vaikkapa REST-rajapinnan kautta. Tätä kautta kaikki järjestelmät, jotka pystyvät lähettämään pyyntöjä REST-arkkitehtuuriin pohjautuvaan web serviceen, saavat käyttöön vanhan taustajärjestelmän datan, aivan kuin kysely olisi mennyt suoraan taustajärjestelmään. Parhaimmillaan integraatio on siis täysin näkymätön.



Kuva 3. Esimerkki ESB-arkkitehtuurista [8].

Täydellinen ratkaisu ESB ei kuitenkaan ole. Se on edelleen yksittäinen entiteetti, joka kaatuessaan kaataa kaikki yhteydet. Myös ESB:n käyttöönotto ja muutosten teko integraatioihin voi olla monimutkaista ja kallista. Muutosten tekeminen yhteen integraatioon saattaa myös vaikuttaa muihin integraatioihin.

ESB ”valittiin” työn malliksi, sillä käytössä oli ennalta määrättynä työkaluna IBM:n ESB-alusta. ESB kuitenkin toimii hyvin työssä toteutetuissa integraatioissa, sillä integroitavat järjestelmät käyttävät tiedon lähettämiseen ja noutamiseen yleisiä teknologioita, joten niille on helppo tarjota toimivat palvelut ESB-mallin mukaisesti.

2.1.4 Mikropalveluarkkitehtuuri

Samoin kuin hub and spokes -mallissa, myös ESB:n ongelmaksi muodostuu sen koko, joka aiheuttaa ongelmia palveluiden keskinäisen riippuvuuden, skaalautuvuuden ja muutosten hallinnan kanssa. Vastaamaan tarvetta saada järjestelmistä ketterämpiä ja skaalautuvampia, täytyy ESB pilkkoa pienempiin itsenäisiin integraatiokomponentteihin. Tähän tarpeeseen vastaa mikropalveluista mallia ottava mikropalveluarkkitehtuuri.

Mikropalveluarkkitehtuurin ja palvelukeskeisen arkkitehtuurin suurin ero on skaala. SOA pyrkii luomaan koko yrityksen kattavan järjestelmän ja hallitsemaan integraatioita kaikkien sovellusten välillä, kun taas mikropalvelumalli keskittyy yksittäisten applikaatioiden toimintaan. Mikropalveluilla halutaan varmistaa eri integraatioiden toiminnan riippumattomuus muista integraatioista, sekä hyvä skaalautuvuus että helppo muokattavuus. ESB:llä kaikki toiminnallisuus on saman prosessin alla ja kun yhtä palvelua halutaan skaalata, täytyy koko monoliittinen prosessi monistaa. Mikropalvelumallissa jokainen toiminnallisuus on oma palvelunsa, joita voidaan skaalata itsenäisesti [9]. Mikropalvelumallilla voidaan olla varmoja, että muutos yhteen integraatioon ei vaikuta muihin integraatioihin. Integraatioiden uudelleenkäytettävyys on yksi SOA:n tavoitteista, mutta mikropalveluihin perustuvassa mallissa se toimii mallin periaatteita vastaan, sillä se lisää sitä, kuinka vahvasti eri applikaatiot ovat liittyneet toisiinsa, joten koodin ja datan monistaminen on parempi vaihtoehto riippumattomuuden ja skaalautuvuuden kannalta. IBM kutsuu tällaista mallia, jossa ESB puretaan pienempiin hallittavampiin komponentteihin nimellä: ”agile integration architecture” (AIA). [10.]

Mikropalveluarkkitehtuuri sopii myös erinomaisesti nykyaikaiseen pilvimaailmaan. Kun integraatiot on eroteltu pienemmiksi, yhdestä muutamaan integraatiota sisältäviksi kokonaisuuksiksi, on ne helppo paketoida kontteihin ja ottaa käyttöön pilvipalvelussa käyttämällä esimerkiksi Docker- ja Kubernetes-teknologioita.

2.2 Integraatioprojekti (erot tyypilliseen ohjelmistoprojektiin)

Integraatioprojekteissa on usein mukana monta eri toimijaa. Vaikka ESB-tyyliseen malliin perustuvan integraatiopalveluväylän toteuttaisi yksi toimija, ovat integraatiot aina riippuvaisia integroitavista järjestelmistä, ja tätä kautta muista toimijoista. Tämä aiheuttaa väistämättä joitakin ongelmia, kuten projektin etenemisen hidastumista ja mahdollisia

väärinkäsityksiä tai virheitä johtuen huonosta kommunikoinnista toimijoiden kesken. Selkeään viestintään panostaminen onkin erittäin tärkeätä integraatioprojektissa, sillä usein kehittäjät näkevät vain oman osa-alueensa eivätkä ymmärrä, mitä muut toimijat tarvitsevat heiltä, jotta projekti saadaan maaliin.

Integraatioprojekteissa usein itse toteutus onkin pienemmässä osassa verrattuna perinteisiin ohjelmistoprojekteihin. Pääpaino on kommunikoinnissa ja suunnittelussa ja itsenäistä tekemistä on pieniä palasia kerrallaan. Integraatioprojektissa vaaditaan siis muutakin kehittäjiltä kuin syvä tekninen osaaminen. Integraatioissa asiat, joita ei aina tule mietittyä, kuten merkistö, välimerkit ja rivinvaihdot, aiheuttavat usein päänvaivaa integraatiokehittäjille. [11.]

Integraatioprojektit eroavat myös siinä muista ohjelmistoprojekteista, että tiukkoja aikarajoja on vaikea asettaa. Koska integraatiot riippuvat niin monesta eri toimijasta, aina ei voida tietää, milloin integraatio saadaan valmiiksi ja testattua. Usein tulee tilanteita, joissa integraatio saattaa olla muuten valmiina pitkänkin aikaa, mutta kohdejärjestelmä puuttuu. Toinen yleinen tilanne voisi olla, että tiedetään, mistä integroidaan ja mihin, mutta ei vielä ole tietoa siitä, minkälaisia muunnoksia dataan tarvitaan. Tämä taas palautuu kommunikoinnin ja suunnitellun tärkeyteen. Koska tarve integraatioille ja datalle tulee usein asiakkaalta, myös määrittely sille, millaista dataa integraatiosta halutaan ulos, täytyy kommunikoida integraatiokehittäjille. Koska integraatioiden aikarajat liikkuvat, ei projekteissa myöskään yleensä voida käyttää suljettuja sprinttejä. Tässäkin työssä oli käytössä Scrum-viitekehys, mutta toteutusta ei voitu jakaa sprintteihin samalla tavalla kuin tyypillisessä ohjelmistoprojektissa tehtäisiin. [11.]

3 Työkalut ja teknologiat

3.1 IBM Integration Bus v9 / IBM App Connect Enterprise v11

IBM App Connect Enterprise, lyhyesti ACE, on IBM:n kehittämä alusta integraatioiden välitykseen, niin kutsuttu ”integration broker”. Tuotteen aikaisempia nimiä olivat WebSphere Message Broker ja IBM Integration Bus versiosta 9 (IIB) eteenpäin. Tuotteen nimi vaihtui IBM App Connect Enterpriseen version 11 myötä. Kirjoittamishetkellä

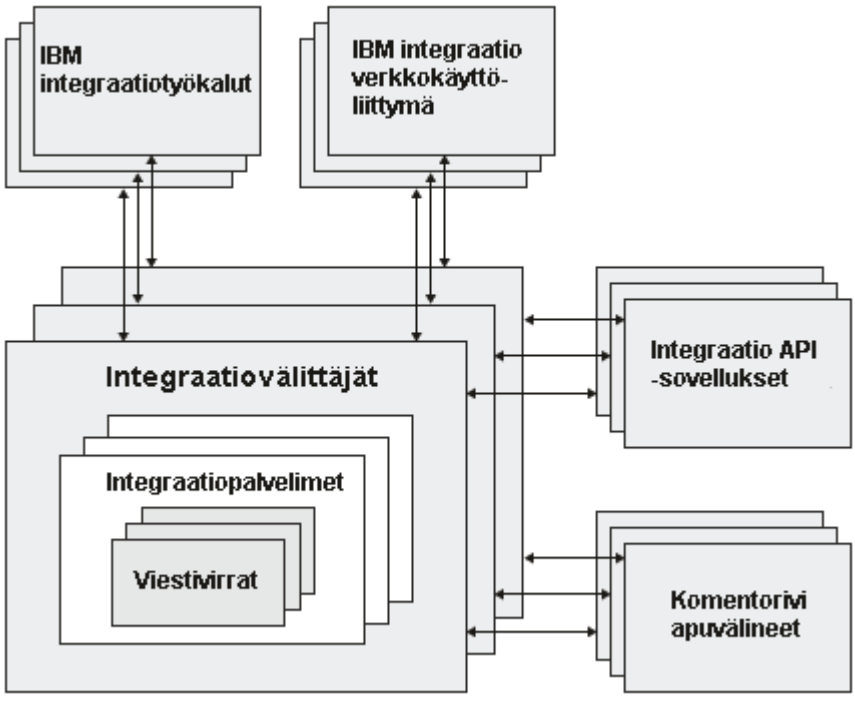
10.10.2019 uusin saatavilla oleva versio tuotteesta on Fix Pack 11.0.0.6. Tämä työ aloitettiin IBM Integration Bus v9:llä, mutta työn loppuvaiheissa vaihdettiin käyttämään IBM App Connect Enterprise v11:sta, josta tullaan jatkossa käyttämään nimeä ACE. Onneksi toteutuksen siirtäminen IIB:ltä ACE:lle kävi vaivattomasti, ja se saatiin testiin ACE:lla. Työ myös tullaan viemään tuotantoon ACE:llä.

ACE on pohjimmiltaan Enterprise Service Bus -tuote, jolla pystytään yhdistämään sovelluksia ja palveluita toisiinsa aiemmin mainitun SOA-arkkitehtuurin mukaisesti. Uutena ominaisuutena ACE:hen on tullut parempi liitäntä nykyaikaisiin pilvi- ja hybridiympäristöihin [12]. ACE itse voitaisiin myös pakata esimerkiksi Dockerilla kontteihin ja siirtää pilviympäristöön, mutta tässä työssä käytetty ACE oli ”on-premises”, eli se on asennettuna fyysiselle palvelimelle, josta sitä ajetaan. ACE tukee monia eri protokollia, kuten IBM MQ:ta, JMS 1.1 ja 2.0:aa, HTTP ja HTTPS:ää, web servicesiä (SOAP and REST), Filea, Enterprise Information Systemsiä (sisältäen SAP:n ja Siebelin), ja TCP/IP:tä [12]. ACE pystyy siis liittämään melkein mitä tahansa mihin tahansa. Vaikka tässäkin työssä tehdyt integraatiot saattavat vaikuttaa point-to-point-mallin-integraatioilta, joissa yksi järjestelmä liitetään toiseen, on erona juuri ACE:n ja yleisesti ESB-mallin kyky käyttää niin monia eri protokollia. Tämä mahdollistaa rajapintojen ja palveluiden tarjoamisen ja käytön sen sijaan, että tehtäisiin räätälöity monimutkainen integraatio molempien osapuolten päihin.

ACE:llä toteutettujen integraatioiden perustana ovat viestivirrat (message flow). Viestivirrat ovat sarja peräkkäisiä prosesseja, jotka ajetaan sisään tulevan viestin seurauksena [13]. Viestivirroilla pystytään hallitsemaan, miten viestiä ja dataa muokataan ja mihin se lähetetään seuraavaksi. Viesti voi myös kulkea useamman viestivirran läpi, kuten tässä työssä tehtiin yhden integraation kohdalla. Integraation selkeyttämiseksi on usein järkevää luoda useampi viestivirta siten, että jokaisella niistä on yksi tehtävä, jota ne toteuttavat. Esimerkiksi yksi viestivirta ottaa vastaan tai noutaa viestin, toinen muokkaa viestiä ja kolmas lähettää viestin eteenpäin. Viestin liikuttelu virtojen välillä onnistuu jonojen kautta. Viesti pystytään julkaisemaan suoraan toisella viestivirralla sijaitsevaan jonoon tai käyttämään publish-subscribe-mallin mukaista ratkaisua, jossa viesti julkaistaan aiheeseen (topic) ja yhdistetään jonoon (queue) tilauksella (subscription). Tämä mahdollistaa, että viestivirtojen välisiä liitäntöjä pystytään helposti muokkaamaan ilman viestivirran toteutuksen muokkaamista, sekä tarvittaessa viestin jakamisen useisiin eri kohteisiin. Tästä on hyötyä, jos toteutukseen halutaan esimerkiksi lisätä jonkinlainen tiedon

esikäsittely. Tarvitaan vain käsittelyn tekevä uusi viestivirta, jolle viestit ohjataan muokkaamalla jonojen tilauksia, eikä olemassa oleviin viestivirtoihin tarvitse koskea. Tällaista toteutusta käydään tarkemmin läpi myöhemmässä vaiheessa raporttia.

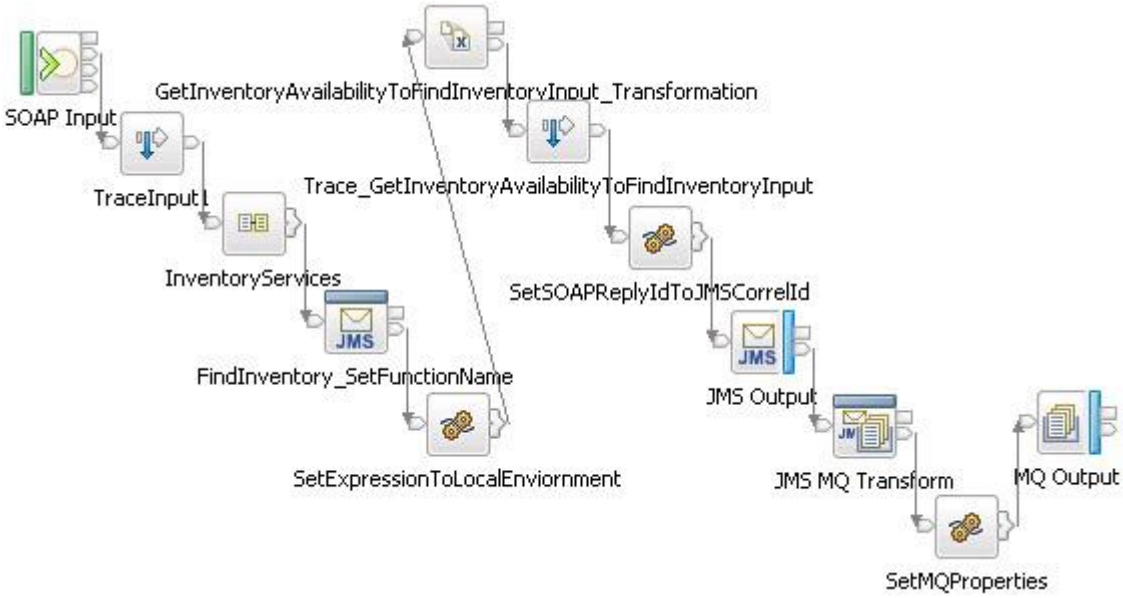
Integration node eli broker, jota kutsun tulevaisuudessa integraatiovälittäjäksi, on ACE:ssä se komponentti, joka pyörittää integraatioiden toiminnallisuutta. Niitä voidaan luoda useita jakamaan taakkaa (kuva 4) tai jokaiselle ympäristölle (esimerkiksi test, staging, production) voidaan luoda oma integraatiovälittäjä. Integraatiovälittäjälle viedään BAR (broker archive) -tiedosto, joka sisältää tarvittavat tiedot viestivirtojen suorittamiseen. Integraatiovälittäjällä olevien viestivirtojen suoritus eri osoiteavaruuksissa tai ainutkertaisina prosesseina voidaan varmistaa lajittelemalla ne ryhmiksi käyttämällä integraatiopalvelimia (integration servers), joita ennen kutsuttiin nimellä execution group [14]. Viestivirrat saadaan näin eristettyä toisistaan.



Kuva 4. IBM Integration Bus -ympäristö [15].

Viestivirtojen luomiseen käytetään IBM ACE toolkitiä, joka oli tärkein työssä käytetty työkalu. Se on Eclipseen pohjautuva graafinen työkalu, jolla pystytään rakentamaan viestivirtoja pudottamalla tiettyjä tehtäviä suorittavia komponentteja, joita tullaan tästä lähtien kutsumaan solmuiksi, kanvaasille ja yhdistämään niitä. Näin saadaan luotua putki (kuva

5), jota pitkin sisään tullut viesti matkustaa ja muokataan. Aivan kaikkeen ACE:ssä valmiiksi olevat perussolmut eivät pysty, joten toolkit:ssä on myös mahdollisuus luoda omia solmuja ja kirjoittaa omaa koodia käyttämällä Java- tai ESQL-ohjelmointikieliä.



Kuva 5. Esimerkki viestivirroista, joita ACE toolkitillä pystytään tekemään [16].

Kuva 5 on esimerkki viestivirrasta, jossa sisään tuleva viesti otetaan vastaan SOAP-kutsuna, viestille tehdään erinäisiä operaatioita ja lopulta viesti kirjoitetaan jonoon odottamaan jatkokäsittelyä. Rataspyöräkomponentit ovat mukautettua ESQL-koodia, mutta samat asiat voitaisiin tehdä myös Javalla. ACE:llä on omat Java-luokat ja kirjastot viestin ja ympäristön manipulointiin. Tämä esimerkki näyttääkin hyvin, millä tavoin mukautettua Java-koodia voidaan käyttää viestivirroissa. Seuraavat tavat ovat ainakin tämän työn kohdalla ne yleisimmät tapaukset, joissa mukautetusta Java-koodista on hyötyä.

- Tiedon tallentaminen ympäristömuuttujiin. Viestin tietosisältöä tai metadataa voidaan tallentaa lokaaliin tai globaaliin ympäristöön ja käyttää myöhemmin.
- Tietosisällön muokkaaminen. Java-koodissa voidaan hakea sisään tulevan viestin tietosisältö ja muokata esimerkiksi JSON-elementtien arvoja tai lisätä ja poistaa kenttiä.
- Java-koodia seuraavien solmujen konfiguraation asettaminen. Esimerkiksi HTTP kutsun osoitteen rakentaminen ja muiden tietojen asettaminen on helppoa Java-

koodissa, sillä Javalla pystytään hakemaan tarvittavaa tietoa monista eri paikoista. Esimerkiksi Java-koodissa voidaan esimerkiksi hakea tietokannasta kohdejärjestelmään liittymiseen tarvittavat pääsytunnukset, jolloin niitä ei tarvitse kovakoodata viestivirtaan. Jos kohdejärjestelmään tulee muutoksia, viestivirtaan kovakoodattujen arvojen muuttaminen olisi huomattavasti suurempi työ kuin vai yhden rivin muokkaaminen tietokannassa. Tämä on yksi esimerkki siitä, miksi muiden solmujen konfiguraatioiden asettaminen Javalla saattaa olla hyödyllistä.

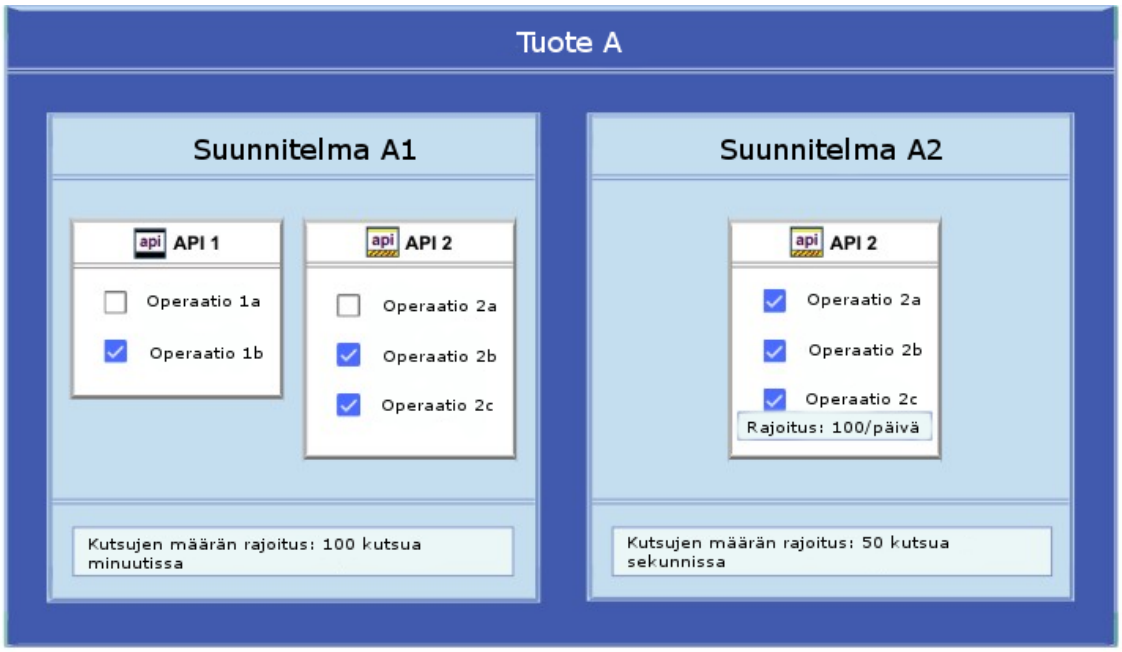
3.2 IBM API Connect

Yksi osa työtä oli toteuttaa rajapinta, jota kautta täysin ulkopuoliset instituutiot voisivat noutaa tarvittavaa tietoa mikropalvelusta. Mikropalvelu haluttiin pitää erillään ulkoverkosta, joten tarvittiin integraatio, joka pystyisi tarjoamaan ulospäin rajapinnan, joka välittäisi liikenteen mikropalvelulle ja samalla valvoisi mikropalveluun sisään tulevaa liikennettä. Periaatteessa tämä olisi voitu tehdä myös ACE:llä, mutta IBM:n tuoteperheessä on toinen, juuri tähän erikoistunut tuote, jota jo ennestään käytettiin useissa rajapinnoissa. Tästä syystä yhteen osaan työtä päädyttiin käyttämään IBM:n API Connect -tuotetta.

API Connect on IBM:n tuottama tuote rajapintojen hallintaan koko niiden elinkaaren ajaksi. API Connectilla pystytään luomaan rajapintoja, hallitsemaan niiden versioita ja julkaisua, hallitsemaan ja rajoittamaan käyttäjien pääsyä rajapintoihin ja järjestelmiin sekä analysoimaan rajapintojen käyttöä.

API Connect toimii tässä työssä eräänlaisena kulunvalvojana, joka validoi sitä kutsuvat applikaatiot ja välittää kutsut taustajärjestelmän rajapintaan. Kulunvalvonta ei kuitenkaan ole ainoa asia, jota API Connectilla voi tehdä. Rajapintoihin sisään tuleviin kutsuihin on mahdollista tehdä muunnoksia ja reitityksiä hyvin samaan tapaan kuin ACE:ssä luotiin viestivirtoja. API Connectin editori sisältää jonkin verran valmiita solmuja, mutta jos ne eivät riitä, voidaan sisään tullutta kutsua tai taustajärjestelmästä palautunutta viestiä muokata XSLT-muunnoksilla tai kustomoidulla JavaScript-koodilla.

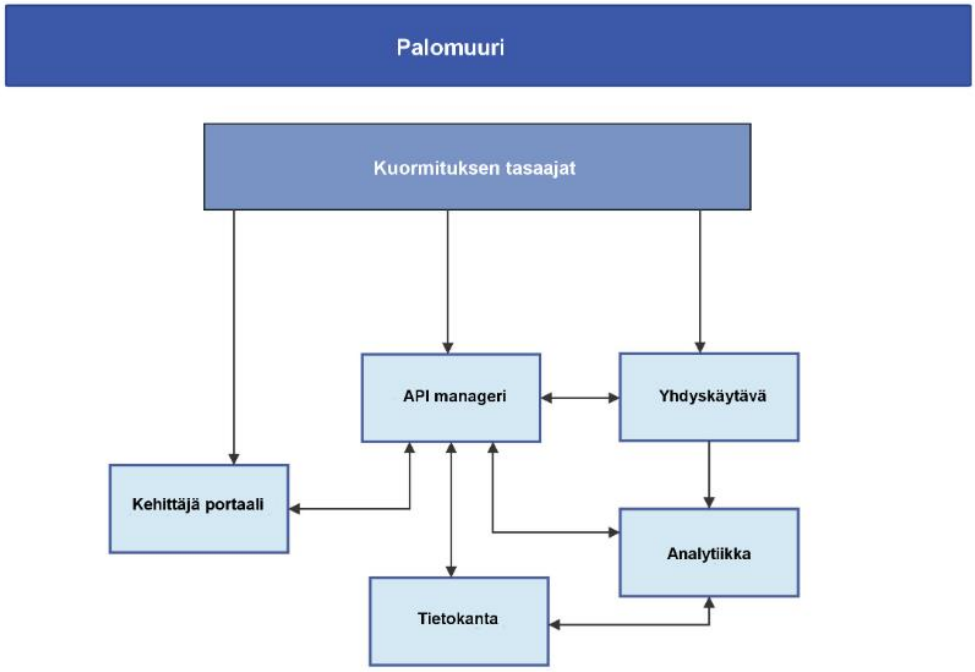
API Connectilla luodut rajapinnat paketoidaan tuotteeksi (product). Tuotteeseen voi sisältyä yksi tai useampi API. Tuotteeseen paketoidaan myös suunnitelmat (plans), joilla rajapinnan tarjoaja (provider) pystyy kontrolloimaan vaikkapa, kuinka monta kertaa tämän tuotteen rajapintoja yksi käyttäjä voi kutsua tietyn aikamääreen sisällä. Kuva 6 esittelee, miten tuotteet on mahdollista rakentaa. Tässä työssä tuotteiden rakenne on kuitenkin hyvin yksinkertainen, jokaisella tuotteella on yksi rajapinta ja yksi suunnitelma.



Kuva 6. API Connect tuotteiden, suunnitelmien ja rajapintojen välinen yhteys [17].

Tuotteet julkaistaan katalogeihin (catalog), joihin kuluttajat (API consumer), eli sovellusten kehittäjät, saavat oikeudet. Katalogeilla pystytään erottamaan rajapintoja toisistaan ja hallitsemaan, mitkä rajapinnat näkyvät millekin kuluttajalle. Usein kehityksen alla olevat rajapinnat pidetään omassa katalogissaan testausta varten ja julkaisuvalmiit rajapinnat viedään omaan katalogiinsa. Jotta applikaatio pystyy käyttämään tiettyä rajapintaa ja applikaation lähettämä pyyntö ohjautuu oikein, täytyy kehittäjän lisätä kehittäjäportaalin kautta tilaus (subscription) halutulle tuotteelle. Tämä mahdollistaa yhden rajapinnan usean eri version samanaikaisen julkaisemisen, sillä tilaus ohjaa pyynnön halutulle versiolle. [17.]

API Connectin toiminta perustuu neljään pääkomponenttiin. Kuva 7 esittelee API Connectin topologian ja komponenttien väliset yhteydet.



Kuva 7. API Connectin topologia [18].

Seuraavaksi käydään läpi API Connectin eri komponenttien toimintaa, mitä niiden vastuualueeseen kuuluu ja miten ne tekevät yhteistyötä toistensa kanssa.

- Tarvitaan ensinnäkin tapa, jolla luoda rajapinnoille kuvaukset ja mahdolliset API Connectin vastuulla olevat toimenpiteet, kuten tietomuunnokset tai tunnistautumistietojen lisääminen kohdejärjestelmää varten. Tähän tarvittavat työvälineet tarjoaa API manager -käyttöliittymä. API Managerin (management portal) käyttöliittymän kautta pystytään hallitsemaan rajapintojen julkaisua, käyttäjien oikeuksia, applikaatioiden tilauksia ja myös luomaan rajapintoja. Rajapintoja luodaan käyttäen managerin graafista käyttöliittymää tai kirjoittamalla suoraan rajapinnan lähdekoodia YAML-muodossa. Tämä tarkoittaa myös sitä, että rajapintoja voidaan suoraan tuoda API Connectiin antamalla API Connectille valmis swagger/OpenAPI-rajapintakuvaus YAML-muodossa, jolloin API Connect luo kuvatut polut ja parametrit automaattisesti.

- Seuraavaksi tarvitaan tapa julkaista luodut rajapinnat kehittäjien saataville. Tarvitaan siis komponentti, joka ottaa vastaan ulkoa tulevat pyynnöt ja suorittaa API managerissa rajapinnalle määritetyt toiminnot. DataPower Gateway on komponentti, joka mahdollistaa API Connectin toiminnan. Joka kerta kun API managerissa luotu rajapinta julkaistaan, se lähetetään IBM:n DataPower teknologialla toimivalle palveluväylälle. Rajapinnan kutsuja lähettää HTTP-pyynnön DataPower Gatewayn paljastamalle päätepisteelle (endpoint). DataPower validoi käyttäjän oikeuden kutsua rajapintaa ja valvoo, että rajapinnalle asetetut käytännöt (policies), kuten aikaisemmin mainittu pyyntöjen määrän rajoitus, toteutuvat, ennen kuin se ohjaa kutsun taustajärjestelmiin. DataPower Gateway myös kerää dataa sille tulleista pyynnöistä ja välittää sen analytiikalle.

- Applikaatioiden kehittäjät tarvitsevat tavan löytää rajapinnat ja niiden kuvaukset, sekä hallita omaa applikaatioita API Connectin sisällä. Tähän tarkoitukseen API Connect tarjoaa kehittäjäportaalin (developer portal). API Connectin kehittäjäportaali on komponentti, jonka kautta applikaatioiden kehittäjät pystyvät löytämään rajapintoja, sekä ottamaan ne käyttöönsä lisäämällä omalle applikaatiolle tilauksen tiettyyn rajapintaan. Kehittäjäportaali hakee rajapinnan kuvauksen API-managerilta aina kun rajapinta viedään DataPower Gatewaylle. Tietyn katalogin kehittäjäksi kutsun saanut organisaatio pystyy luomaan API Connectiin applikaatioita, joille API Connect luo sekä ID:n että Secretin, jotka applikaation pitää asettaa otsikkotietoihin (headers) joka kerta kutsuessaan API Connectin rajapintaa.

- Koska API Connectissa on julkaistuna monia eri rajapintoja, olisi myös hyvä saada jonkinlaista dataa siitä, kuinka paljon kutsuja liikkuu tässä työssä toteutetun rajapinnan läpi. Ongelmatilanteitakaan ei yleensä pystytä täysin välttämään, joten niiden selvittämiseen tarvitaan myös loki, josta nähdään virheeseen menneet pyynnöt. API Connectissa kaiken tämän hoitaa analytiikka (analytics) -komponentti. Palveluväylän palauttama analytiikka on nähtävissä API managerin käyttöliittymän kautta. Analytiikasta löytyvää dataa pystytään käsittelemään ja järjestelemään yleisellä tasolla, jotta nähdään esimerkiksi, kuinka monta pyyntöä palveluväylän läpi kulki viime kuussa, minkä rajapinnan läpi kulkee eniten pyyntöjä tai minkälainen on jakauma eri vastauskoodien välillä. Kehittäjille mielenkiintoisempaa tietoa on analytiikasta myös löytyvä tarkempi kutsukohtainen data,

jota voidaan tiettyyn pisteeseen asti käyttää virheenselvitykseen. Usein analytiikasta löytyvä loki ei kuitenkaan ole riittävä, vaan on parempi kääntyä ympäristön lokeihin, kuten Google Cloudin Stackdriver-lokeihin tämän työn kohdalla.

Tässä työssä API Connectia käytettiin mikropalvelun rajapinnan julkaisemiseen turvallisesti ulkopuolisille instituutioille. Koska kyseessä oli ulkopuolinen käyttäjä, eikä API Connectia käyttävä, yrityksen kanssa yhteistyöstä tekevä applikaatiokehittäjä, ei työssä pystytty käyttämään API Connectille normaalia tunnistautumismenetelmää. Tunnistautuminen täytyi tehdä molemminpuolisella autentikoinnilla (mutual authentication tai two-way authentication), API Connectille yleisemmän ID/Secret-otsikkotietoihin ja rajapintojen tilaukseen perustuvat systeemin sijaan. Toisin kuin ACE, työssä käytetty API Connect sijaitsee pilvessä. Se on kontitettu ja sitä ajetaan Kubernetesen avulla Googlen pilviympäristössä. Uudempana teknologiana API Connect haluttiin ottaa käyttöön nykyisen trendin mukaisesti pilvipalvelussa ja luultavasti myös ACE seuraa API Connectia pilvimaailmaan jossakin vaiheessa.

3.3 Cloud Pub/Sub

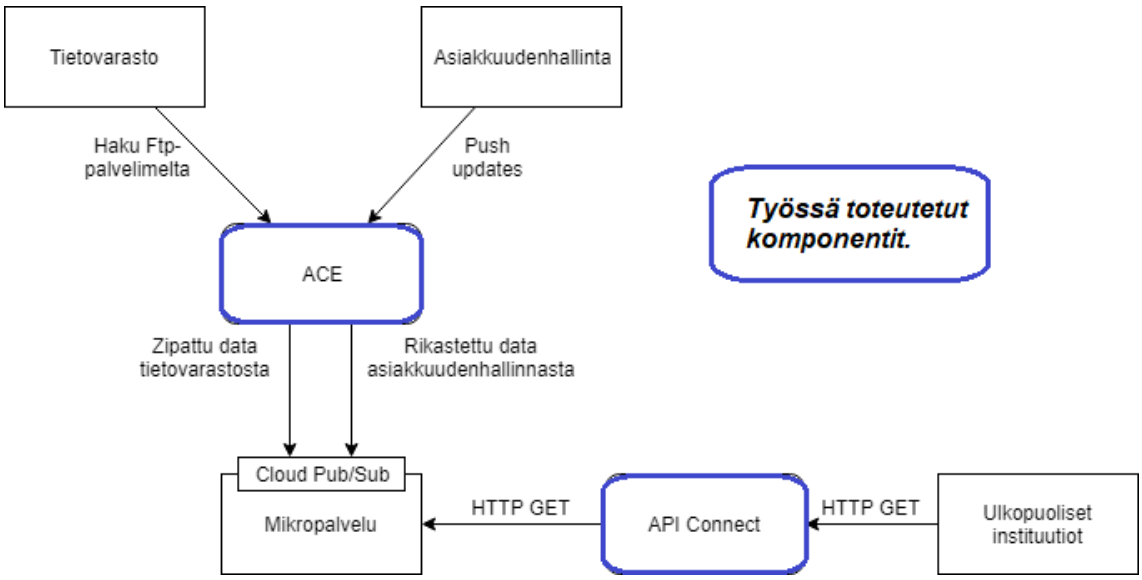
Pilvialustalla sijaitsevat komponentit tai palvelut tarvitsevat tehokkaan ja varman tavan lähettää viestejä toisilleen. Ei ole myöskään toivottavaa, että palvelut joutuvat odottamaan vastauksia lähettämiinsä viesteihin estäen niiden toiminnan odotuksen ajaksi, jos viestin vastaanottava komponentti on alhaalla. Ratkaisuna tähän on publish-subscribe-malli, jolla hoidetaan komponenttien välinen viestintä asynkronisesti. Viestin lähettäjä eli julkaisija julkaisee viestin tiettyyn aiheeseen tai merkkaa, minkälaista sisältöä viesti sisältää. Tilaajat pystyvät lisäämään tilauksen tiettyyn aiheeseen tai aiheisiin, jolloin ne saavat kaikki viestit näistä aiheista. Toinen mahdollisuus on suodattaa halutut viestit niiden sisällön perusteella. Tämänlaisella viestintämallilla erotetaan viestin lähettäjä ja vastaanottaja toisistaan, jolloin viestin lähettäjä pystyy lähettämään viestin tuntematta vastaanottajia ja jatkamaan välittömästi omaa toimintaansa välittämättä vastaanottajien tilasta. Vastaanottajien lisääminen onnistuu myös kevyesti, sillä tilaajien täytyy vain lisätä tilaus vastaanottaakseen haluamaansa dataa, eikä julkaisijan tarvitse lähettää samaa viestiä useita kertoja. [19.]

Integraatioiden kohdejärjestelmänä oleva mikropalvelu ottaa dataa vastaan Googlen Cloud Pub/Sub-teknologian kautta, joten integraation piti pystyä julkaista viestejä sitä kautta. Cloud Pub/Sub on Googlen pilvessä toimiva publish-subscribe-mallin viestintäteknologia, jonka avulla pystytään lähettämään viestejä eri applikaatioiden välillä. Tätä teknologiaa käytettiin viestien välittämiseen ACE:ltä mikropalveluun, joka sijaitsee Googlen pilvessä. Myös IBM WebSphere MQ publish/subscribe, jota käytetään paljon ACE:ssä viestien välittämiseen viestivirtojen välillä, käyttää samaa konseptia.

Myöhemmin työssä käydään yksityiskohtaisemmin läpi, mitä kaikkea viestien välittämiseen Cloud Pub/Subilla tarvitaan, mutta yleisellä tasolla se toimii seuraavalla tavalla: Tietoa julkaiseva applikaatio lähettää viestejä aiheeseen (topic). Tietoa käyttävät applikaatiot tekevät tilauksen aiheeseen, josta haluavat saada viestejä. Julkaisijat pystyvät julkaisemaan viestejä samaan aiheeseen ja samaan aiheeseen voi olla kiinnittynyt useampi applikaatio samaan aikaan. Viestin kyytiin voidaan myös laittaa avain-arvo-pareja, joilla pystytään kuvaamaan viestin sisältöä. [20.]

4 Toteutetut integraatiot

Tässä työssä piti toteuttaa kolmea liittymää, joista kaksi tehtiin ACE:llä ja kolmas API Connectilla. Kuvassa 8 näkyy suurempi kokonaisuus tähän työhön liittyvistä järjestelmistä. ACE:n kautta tehtiin kaksi integraatiota, johon kuului datan hakemista eri lähteistä, sen muokkausta ja toimittamista mikropalvelulle Cloud Pub/Sub:n kautta. Toinen osa työtä oli luoda REST-rajapinta, jonka kautta ulkopuoliset instituutiot pääsisivät noutamaan dataa mikropalvelusta GET-kutsuilla. Alkuun API Connectilla toteutettu rajapinnan julkaisu vaikutti yksinkertaisimmalta osuudelta, mutta lopulta sen toteuttaminen vei huomattavasti enemmän aikaa kuin ACE:llä toteutetut integraatiot. ACE-integraatiot saatiin tehtyä melko vaivattomasti. Cloud Pub/Subin käyttö vaati hieman opettelua, mutta siinä suurin osa työstä tehtiin mikropalvelun puolella, joten integraation hoidettavaksi jäi viestin julkaisu annettuun aiheeseen.



Kuva 8. Kokonaiskuva työssä toteutetuista integraatioista.

4.1 Tiedoston haku tietovaraston FTP-palvelimelta

Tietovarasto luo tiedoston FTP-palvelimelleen, josta se pitäisi saada toimitettua mikropalvelulle pakattuna zip-tiedostona. Ensimmäinen integraatio oli siis melko yksinkertainen. Työkalut tiedoston noutamiseen tietovarastosta löytyvät valmiina ACE:stä ja loput toimenpiteet pystytään tekemään yksilöidyllä Java-koodilla.

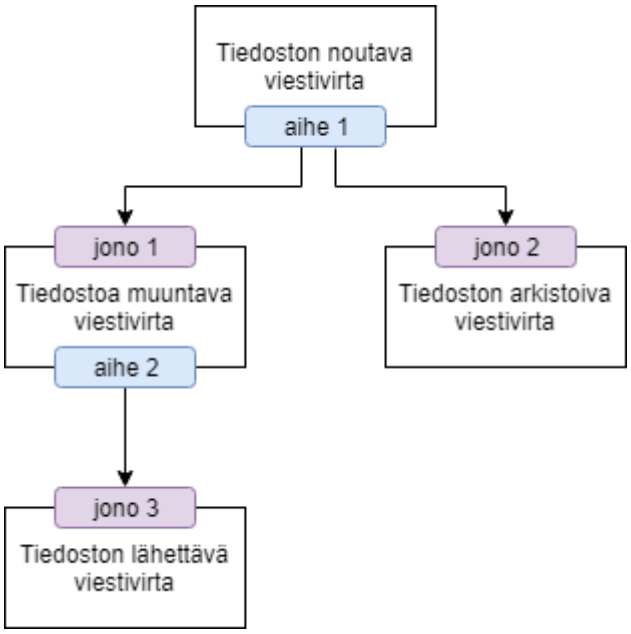
ACE:n File Input -solmu voidaan asettaa kyselemään tiedostoja tietovaraston FTP-palvelimelta säännöllisin väliajoin ja noutaa sieltä tiedostot, joiden nimet osuvat määriteltyyn tiedostomaskiin. Tiedostomaskin lisäksi voidaan asettaa toinen tiedostomaski, jolla solmulle kerrotaan, minkä nimisiä tiedostoja ei noudeta palvelimelta. Myös kyselyiden tiheys voidaan asettaa suoraan solmulle. Usein ympäristöissä saattaa kuitenkin olla eroja. Esimerkiksi tuotannon FTP-palvelimelle ei välttämättä haluta kovin tiuhaa kyselyväliä, sillä monet eri integraatiot saattavat kysellä sieltä tiedostoja. Nämä kaikki asetukset voidaan myös ylikirjoittaa luomalla tiedosto, jossa määritellään sekä viestivirtojen yleisiä että niissä olevien solmujen asetuksia uusiksi. Tämän jälkeen tiedostossa määritellyt asetukset voidaan ajaa manuaalisesti rakennetulle BAR-tiedostolle IBM:n mqsiapplybaroverride-komennolla. Näin jokaiselle ympäristölle voidaan luoda oma BAR-tiedosto. Tämä prosessi kannattaa tietenkin automatisoida CI-putkessa, ettei BAR-tiedostolle tarvitse joka kerta ajaa manuaalisesti ympäristökohtaisia asetuksia.

Tiedosto piti pakata mikropalvelua varten zip-tiedostoksi. Tiedoston pakkaamiseen ei kuitenkaan löydy valmista ACE-solmua, joten pakkaaminen tehtiin Javalla. Koodissa avataan ZipOutputStream, jonka avulla viestin mukana tavuina (byte) tullut tiedosto kirjoitetaan uuteen Zip-tiedostoon. Ei siis ole kyse ollenkaan monimutkaisesta koodista, mutta mainitsemisen arvoista on, että ACE:n viestivirroille pystytään määrittelemään parametrejä nimeltä UDP (User Defined Properties). UDP:t ovat viestivirtatason muuttujia, ja ne ovat erityisen hyödyllisiä Java-koodin kanssa. UDP-parametreille pystytään määrittelemään arvoja, joita voidaan lukea Javassa, mikä mahdollistaa sen, ettei kaikkia muuttujia tarvitse määritellä Java-solmun sisällä, vaan sellaisia arvoja, joita usein tarvitsee muuttaa, pystytään helposti muuttamaan. Käytännön esimerkkinä tässä työssä zip-tiedoston luovalla viestivirralla on määritetty UDP:t uuden zip-tiedoston nimelle ja pakkauksessa käytetyn kompressoinnin tasolle. Tällä tavalla näitä parametreja pystytään muuttamaan helposti koskematta Java-koodiin.

Pakkauksen jälkeen viesti, jonka mukana tiedosto liikkuu viestivirtojen läpi, piti vielä julkaista Cloud Pub/Sub -aiheeseen. Googlelta löytyivät hyvät ohjeet ja esimerkit viestien julkaisemiseen [21]. Toteutus tehtiin suurimmaksi osaksi suoraan edellä mainitun Googlen esimerkin pohjalta, mutta ongelmaksi muodostui tapa, jolla pääsytiedot asetetaan oletuksena. Oletuksena pääsytiedot asetetaan ympäristömuuttujaan nimeltä GOOGLE_APPLICATION_CREDENTIALS, mikä kyllä toimii, mutta ei jätä vaihtoehtoja auki tulevaisuuden laajennuksia varten. Jos tulevaisuudessa halutaan ottaa käyttöön toinen integraation, joka käyttää eri Cloud Pub/Sub -pääsytietoja, niin tarvitaan tapa, jolla voidaan osoittaa mitä pääsytunnustiedostoa käyttää. Ratkaisuksi löydettiin GoogleCredentials-luokka, jolle oli mahdollista antaa FileInputStreamin, joka sisälsi polun levyllä sijaitsevaan pääsytunnustiedostoon. GoogleCredentials-luokan ilmentymä, joka siis sisälsi polun pääsytunnustiedostoon, taas liitettiin Publisher-luokan ilmentymään. Publisheriin lisättiin myös Googlen esimerkin mukaisesti ProjectTopicName-luokan ilmentymä, joka sisälsi viestin vastaanottajan määrittelemän aiheen nimen, johon viesti julkaistaan, sekä projektin Id:n.

Tässä integraatiossa viestivirrat oli koteloitu omiksi kokonaisuuksikseen, jotka kuljettavat viestiä aiheiden, jonojen ja tilausten avulla (kuva 9). Jokaisella viestivirralla on oma tehtävänsä. Yksi viestivirta noutaa tiedoston tietovarastolta, seuraava viestivirta pakkaa tie-

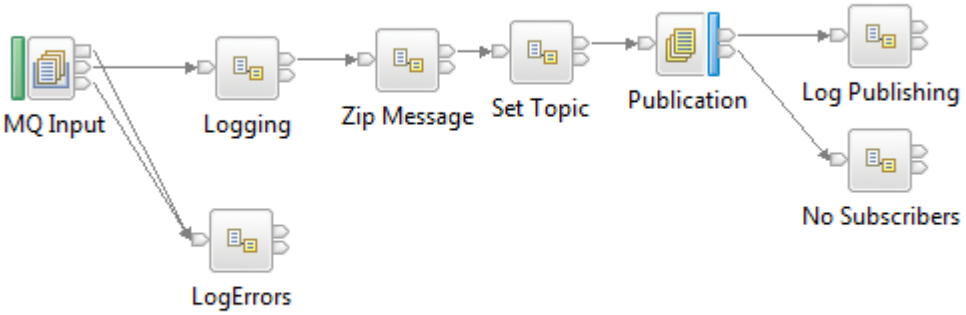
doston ja viimeinen viestivirta julkaisee tiedoston mikropalvelun saataville. Kaikissa vaiheissa tehdään myös lokitusta ja asetetaan metadataa viestin kyytiin, mikä näkyy paremmin seuraavan integraation viestivirtaesimerkissä (kuva 12).



Kuva 9. Viestin kulku eri viestivirtojen välillä.

Kuva 9 esittää, kuinka aiheita ja jonoja on käytetty viestin välittämiseen viestivirtojen välillä. Viivat aiheiden ja jonojen välillä kuvaavat tilauksia, joilla jonot ilmaisevat halunsa ottaa vastaan aiheeseen julkaistut viestit. Oikeassa toteutuksessa tiedostoa ei arkistoida, mutta kuva 9 on esimerkki siitä, mitä jonoja ja aiheita käyttämällä pystyttäisiin tekemään. Mallin yksi vahvuus on, että saman aiheen voi tilata useampi jono, jolloin julkaisija voi lähettää viestin usealle jonolle yhdellä lähetyksellä. Kuva 10 on esimerkki työssä toteutetusta viestivirrasta, joka muuttaa viestin mukana tulleen tiedoston zip-muotoon. MQ Input -solmuun on määritelty jono, jolle on lisätty tilaus edellisen, tiedostoja FTP:ltä noutavan viestivirran määrittelemään aiheeseen. MQ Input -solmussa on kolme porttia, joista viesti voi tulla ulos. Ylimmäinen suorakaiteen muotoinen portti ottaa kiinni virheet, jotka tapahtuvat solmun prosessissa. Suorakaiteen muotoisia portteja voisi ajatella eräänlaisena solmun try-catch:n catch-haarana. Keskimmäinen on normaali onnistunut tapaus, ja alin portti on virheportti. Molemmat virheportit on yhdistetty lokitukseen. Kun edellinen viestivirta julkaisee viestin aiheeseen, toimitetaan se välittömästi tälle viestivirralle ja viesti lähtee etenemään virrassa. Viestin julkaisu aiheeseen tapahtuu Publication-

solmulla. Publish-subscribe -mallin mukaisesti julkaisija ei välitä, kuka tilaa sen viestejä, mutta täysi tilaajien puuttuminen aiheuttaa virheen, joka saadaan ulos kuvassa 10 näkyvästä Publication-solmun toisesta portista erillisiä toimenpiteitä tai lokitusta varten.



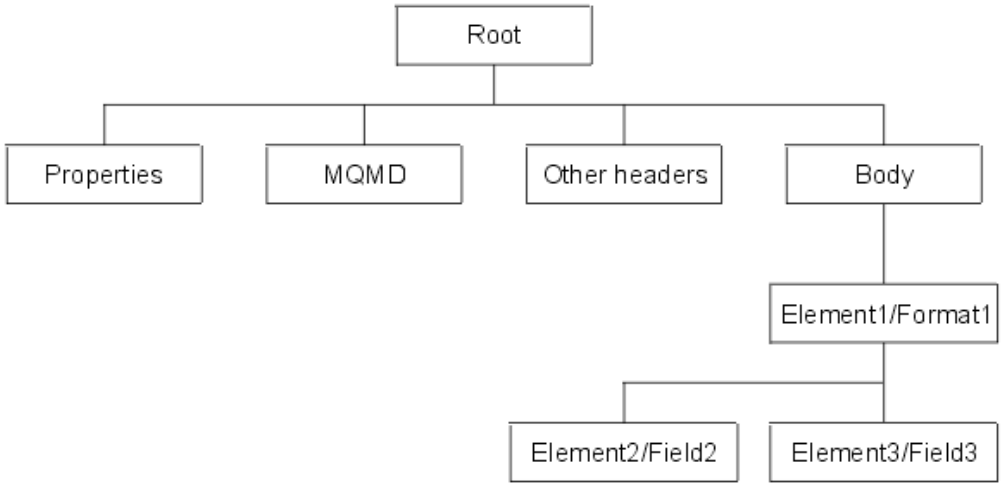
Kuva 10. Esimerkki tiedoston zip-muunnoksesta vastaavasta viestivirrasta.

4.2 HTTP tiedonsiirto asiakkuudenhallinnasta

Toisessa integraatiossa päästiin jo hieman monimutkaisemman toteutuksen kimppuun. Asiakkuudenhallinta puskee dataa web servicen kautta ja kuten edelläkin, se piti välittää mikropalvelun käytettäväksi Cloud Pub/Sub -aiheen kautta. Tässä integraatiossa jouduttiin myös rikastamaan asiakkuudenhallinnan puskemaa dataa toiminnanohjausjärjestelmästä haetulla datalla.

Asiakkuudenhallinnalle piti siis saada jokin tapa, jolla se pystyy lähettämään dataa ACE:lle. Tätä varten viestivirralle luotiin REST-päätepiste, johon voidaan lähettää dataa POST-metodilla. REST oli tähän hyvä valinta, koska tiedettiin, että sisään tulevassa viestissä ei tule olemaan kovin montaa kenttää ja JSON-muotoista tietosisältöä olisi helppo muokata ja rikastaa Javalla. Asiakkuudenhallinnan järjestelmä myös varmasti pystyisi lähettämään pyyntöjä REST-rajapintaan. Tämä toteutettiin konfiguroimalla ACE:n HTTP Input -solmulle haluttu polku, joka ottaa viestejä vastaan ja välittää ne eteenpäin viestivirrassa käsiteltäviksi.

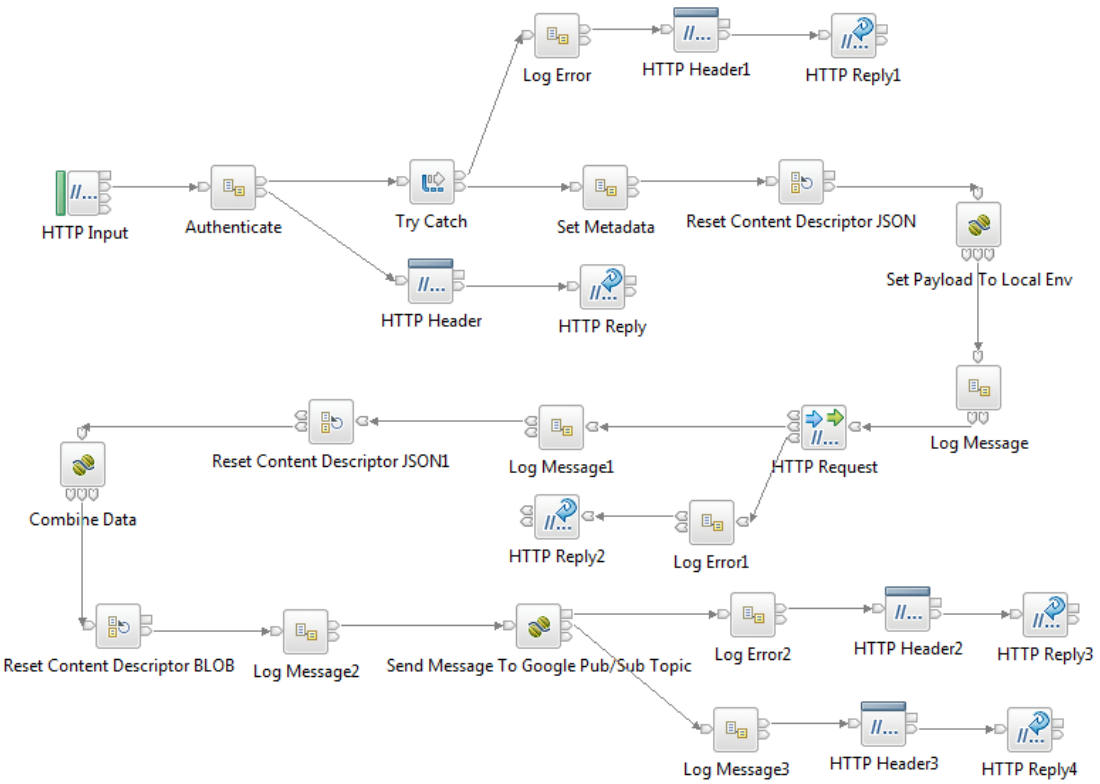
Pelkkä asiakkuudenhallinnan lähettämä data ei siis riittänyt, vaan se piti rikastaa toiminnanohjausjärjestelmästä saatavalla datalla. Alkuperäisen sisään tulleen viestin JSON-muotoinen tietosisältö tallennettiin lokaaliin ympäristöön talteen, jonka jälkeen tarvittavat tiedot pyydettiin toisen järjestelmän REST-rajapinnasta GET-metodilla. Vastauksena ACE saa tiedot taas JSON-muodossa, mutta ennen kuin tietosisältöä päästään muokkaamaan, täytyy ACE:lle kertoa, että viesti on JSON-muotoista. Kuva 11 on esimerkki ACE:n parsimasta viestipuurakenteesta, jossa viestisisältö tallennetaan Body-elementtiin. Käyttämällä ResetContentDescriptor-solmua pystytään kertomaan ACE:n parserille, että viesti on JSON-muotoista, jolloin rajapinnasta vastauksena saatu tieto löytyy viestijuuren alta ”/JSON/Data”-polun päästä. Nyt saatua dataa pystytään käsittelemään ja muokkaamaan JSON-muotoisena esimerkiksi Javalla.



Kuva 11. ACE-viestipuun kuvaus [22].

Nyt kun asiakkuudenhallinnasta saatu tieto on tallennettuna paikalliseen ympäristöön, ja toiminnanohjausjärjestelmästä haettu tieto on viestin mukana, voidaan Javalla hakea molemmat JSON tietosisällöt ja luoda uusi JSON-viesti, jossa ne on yhdistetty.

Toisin kuin ensimmäisessä integraatioissa, tämä integraatio toteutettiin käyttäen vain yhtä viestivirtaa (kuva 12). Yleisesti tämänkaltaisissa web serviceissä koko integraatio hoidetaan yhdessä viestivirrassa, sillä kohdejärjestelmän vastauksen välittäminen alkuperäiselle kutsujalle on tällä tavalla huomattavasti helpompaa, sillä HTTP-vastaussanoma ei automaattisesti välity viestivirralta toiselle.

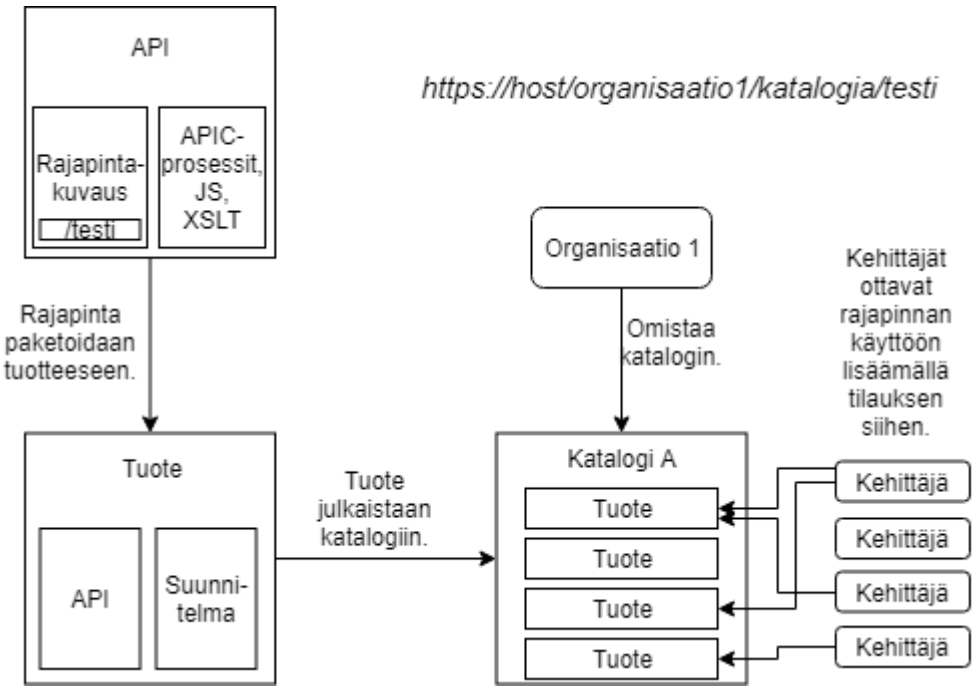


Kuva 12. Asiakkuudenhallinnan ja mikropalvelun välinen viestivirta yleisellä tasolla.

Kuvassa 12 on esitetty asiakkuudenhallinnan ja mikropalvelun välisen integraation viestivirta. Viestivirta alkaa HTTP Input -solmulla, joka ottaa vastaan REST-pyyntöjä. Tämän jälkeen kutsujan identiteetti varmistetaan, ennen kuin viesti pääsee pidemmälle viestivirrassa. Jos kutsujalla ei ole oikeutta kutsua tätä rajapintaa ja validointi epäonnistuu, niin viesti tulee authenticate-solmun alemmasta portista ja kutsujalle palautetaan 401-virhekoodi. Try-Catch ottaa kiinni mahdolliset virheet viestin käsittelyssä ja ohjaa viestin toiseen haaraan, jossa virheet lokitetaan ja virhekoodi palautetaan kutsujalle. HTTP Request -solmu on se, joka lähettää GET-pyynnön toiminnanhallinnalle, ja kun tähän pyyntöön saadaan vastaus, jatkaa viesti matkaansa viestivirrassa. Jos vastauksena saadaan virhekoodi, palautuu viesti HTTP Request -solmun alemmasta portista, josta virhe palautetaan alkuperäiselle kutsujalle ja viestin matka keskeytyy. Lopulta parseria pyydetään parsimaan viesti BLOB-muotoon, jotta se saadaan lähetettyä Cloud Pub/Sub-aiheeseen, jonka jälkeen lähetyksen onnistuminen lokitetaan ja vastauskoodi palautetaan alkuperäiselle kutsujalle.

4.3 Rajapinnan julkaisu kolmannen osapuolen instituutioille

Viimeiseksi luotiin rajapinta, jonka kautta ulkopuoliset instituutiot pystyisivät turvallisesti kyselemään mikropalvelulta tietoa. Koska tarkoitus oli, että API Connect välittää sille tulevat pyynnöt mikropalvelulle, suurin osa työstä oli API Connectin konfigurointia. Aikaisemmin sivuttiin jo hieman sitä, miten API Connectissa rajapinnat julkaistaan, mutta kerrataan se vielä tässä (kuva 13). API Connectin graafisessa käyttöliittymässä luodaan API-luonnos (draft), joka sisältää kuvauksen rajapinnasta (polut, parametrit, turva-asetukset yms.) ja toiminnallisen osion, joka minimissään sisältää API Connectista ulospäin lähtevää liikennettä ohjaavan proxy-solmun, mutta voi sisältää erilaisia API Connectin tukemia prosesseja, datan muuntamista kohdejärjestelmän ymmärtämään muotoon, reititystä, lokitusta, XSLT-muunnoksia tai mukautettua JavaScript-koodia. Tämä luonnos paketoidaan tuotteeseen ja siihen voidaan liittää erilaisia rajapinnan käyttöä kontrolloivia käytäntöjä (suunnitelmia). Tämä jälkeen tuote julkaistaan rajapintoja tarjoavan organisaation omistamaan katalogiin, jolloin API tulee saataville niille, joilla on oikeudet tähän katalogiin. Koska katalogeja voi olla useita, on URL, jolla rajapintoja kutsutaan, katalogikohtainen. URL on muotoa: *https://host/<organisaation nimi>/<katalogin nimi>/<API polku>*.



Kuva 13. Prosessi rajapinnan julkaisemiseksi API Connectissa.

API Connectille luotiin mikropalvelua vastaava rajapintakuvaus, joka määrittelee, minkälaisia pyyntöjä API Connectin rajapintaan lähetetään. Tämä rajapinta luotiin tarkoituksella samanlaiseksi kuin API Connectin käyttämä mikropalvelu-API. Näin pystytään välittämään API Connectille tuleva pyyntö mahdollisimman helposti eteenpäin. Parhaassa tapauksessa sisään tulevasta pyynnöstä saadaan suoraan polku ja parametrit, jotka voidaan kiinnittää kohdejärjestelmän DNS-nimen tai IP:n perään, jolloin suoraan saadaan kohdejärjestelmän rajapintaan toimiva HTTP-pyyntö. Koska tätä rajapintaa käyttävällä instituutiolla oli vaatimuksia polun suhteen, ei API Connectin rajapinnan polku voinut kuitenkaan olla sama kuin mikropalvelu-API:n. Tämä ratkaistiin rakentamalla kohdejärjestelmään (mikropalvelu-API) sopiva peruspolku JavaScript-koodia käyttäen. Haluttu operaatio ja mahdolliset GET -parametrit (path & query) haettiin sisään tulevasta pyynnöstä.

Itse rajapinnan konfiguraatio ei siis ollut kovinkaan monimutkainen. Rajapinta täytyi kuitenkin suojata, jotta kuka tahansa ei pystyisi sitä kutsumaan. Normaalisti rajapintojen suojaukseksi projektissa mukana olevan kehittäjän täytyy saada kutsu rekisteröityä API Connectin katalogikohtaiseen kehittäjäportaaliin, minkä jälkeen hän pystyy luomaan applikaation, jolle API Connect määrää Id:n ja salaisuuden. API Connectia kutsuvan applikaation täytyy liittää jokaiseen tekemäänsä kutsuun edellä mainitut Id ja salaisuus, jotta API Connect ei hylkää kutsua. Kehittäjä pystyy lisäämään tilauksia applikaatiolleen vain siitä katalogista, johon hänet kutsuttiin. Tämän rajapinnan suojaukseen ei kuitenkaan voitu käyttää tätä normaalia tapaa, sillä rajapinnan käyttäjä oli kolmannen osapuolen organisaatio, jota ei voitu kutsua API Connectiin käyttäjäksi. Tämä API piti suojata varmenteilla siten, että id:n ja salaisuuden sijaan rajapinnan kutsujaa pyydetään lähettämään asiakasvarmenne, joka validoidaan API Connectissa (mutual authentication). Myös rajapintaa kutsuvalla organisaatiolla oli vaatimuksia rajapinnan suojauksesta ja käytettävistä varmenteista. Seuraavaksi kuvataan TLS-protokollaa ja varmenteilla autentikoitaessa tapahtuvaa kättelyä, sekä miten tällainen molemminpuolinen varmenteilla autentikointi voidaan toteuttaa API Connectilla.

4.3.1 SSL/TLS-kättely

TLS (Transport Layer Security) on salausprotokolla, jota käytetään suojaamaan verkon yli tapahtuva viestiliikenne palvelimen ja asiakasohjelman (esimerkiksi selain) välillä. SSL (Secure Sockets Layer) on TLS-protokollan edeltäjä, joka on vanhentunut. Jos ny-

kyaikana kuulee puhuttavan SSL:stä, tarkoitetaan yleensä TLS:ää. TLS on kerroksittainen protokolla, jota yleensä ajetaan tietoliikenneprotokollan, kuten TCP:n (Transmission Control Protocol) päällä. TLS:n yli taas voidaan käyttää sovelluskerroksen protokollia kuten HTTP (Hypertext Transfer Protocol) [23, s. 1]. HTTPS (Hypertext Transfer Protocol Secure) onkin laajennus HTTP-protokollaan, jossa tietoliikenne salataan käyttäen TLS:ää.

Asiakkaan ja palvelimen välinen tietoliikenne täytyy kryptata käyttämällä salausavaimia. Salausavain on merkkijono, joka kryptografisen algoritmin kanssa käytettynä muuttaa selkokielisen tekstin ei-luettavaan muotoon [24]. Tehokkaaseen tiedonvaihtoon olisi paras, jos molempien osapuolien viestien salaamiseen ja purkamiseen pystyttäisiin käyttämään samaa yksityistä salausavainta (symmetrinen salaus), mutta ongelma on, että tämä salausavain pitää ensin välittää molemmille osapuolille turvallisesti. Yksityisiä ja julkisia avaimia käyttävässä asymmetrisessä salauksessa tätä ongelmaa ei ole, sillä julkista avainta voi jakaa vapaasti, mutta asymmetrinen salaus yleensä vaatii enemmän aikaa, eikä ole yhtä tehokas [25]. Tähän on ratkaisuna TLS kättely -prosessi, jossa asymmetristä salausta käyttäen asiakas ja palvelin luovat uuden yhteisen salausavaimen, jota voidaan käyttää tietoliikenteen salaukseen.

TLS-kättely on menettelytapa, jolla pystytään validoimaan sekä palvelimen että asiakasohjelman oikeellisuus ja varmistetaan, ettei mahdollinen hyökkääjä pääse muokkaamaan tietoa kesken tiedonvaihdon. Kättelyssä asiakas ja palvelin varmistavat toistensa luotettavuuden varmenteiden avulla. Varmenteen tarkoituksena on kertoa verkossa muille, kuka varmenteen omistaja todella on. Tätä varten luodaan julkinen avain ja yksityinen avain. Tämä avainpari toimii siten, että julkisella avaimella salattu viesti pystytään purkamaan vain sitä vastaavalla yksityisellä avaimella, mikä varmistaa viestityksen luottamuksellisuuden. Julkisella avaimella taas pystytään varmistamaan yksityisellä avaimella salatun viestin lähettäjän luotettavuus, luomalla viestistä tiiviste (hash) ja salaamalla se yksityisellä avaimella. Viestin vastaanottaja pystyy laskemaan viestistä saman tiivisteen ja purkamaan salatun tiivisteen lähettäjän julkista avainta käyttäen, ja jos tiivisteet ovat samat, voidaan olla varmoja lähettäjän henkilöllisyydestä. Organisaatiota edustava varmenne luodaan sisällyttämällä julkinen avain varmenteeseen ja mukaan laitetaan tietoja varmenteen omistajasta. CA (Certificate Authority) varmistaa, että hakija edustaa sitä organisaatiota, jota se sanoo edustavansa, jonka jälkeen CA allekirjoittaa varmenteen [26].

Kättelyn eri vaiheet (kuva 14):

1) Tervehdysviesteillä asiakas ja palvelin aloittavat turvatun yhteyden muodostamisen ja sopivat minkälaista konfiguraatiota tullaan käyttämään kättelyssä. Asiakas lähettää tervehdysviestin palvelimelle ja sisällyttää viestiin käytettävän TLS-version sekä listan asiakkaan tukemista cipher suiteista. Cipher suitet ovat lista hyväksyttyjä kryptografisia algoritmeja, joilla palvelimen ja asiakkaan välinen yhteys validoidaan ja salataan [23, s. 14].

2) Palvelin valitsee asiakkaan listasta cipher suiten, jota molemmat voivat käyttää. Se vastaa asiakkaan tervehdykseen omalla tervehdyksellä ja lähettää asiakkaalle viestin, joka sisältää asiakkaan listasta valitun cipher suiten, jotta molemmat osapuolet osaavat käyttää samoja kryptografisia algoritmeja kättelyn edetessä, sekä palvelinvarmisteensa (julkinen avain). Palvelin voi myös lähettää asiakkaalle pyynnön saada asiakkaan varmenne validoitavaksi, jos kyse on molemminpuolisesta varmenteiden validoinnista.

3) Asiakasohjelma varmentaa palvelimen lähettämästä varmenteesta, että lähde on luotettu. Varmenteen vahvistuksessa tarkastetaan varmenteen sähköinen allekirjoitus, sertifikaattiketju, varmenteen myöntämis- ja erääntymispäivä sekä varmistetaan, ettei varmennetta ole peruttu [27].

4) Asiakas lähettää satunnaisen tavumerkkijonon salattuna palvelimen julkisella avaimella (palvelimen asiakkaalle lähettämä varmenne). Jos palvelin pystyy tästä merkkijonosta laskemaan saman salausavaimen kuin asiakas, voi asiakas olla varma, että palvelimella on hallussaan julkista avainta vastaava yksityinen avain (asymmetrinen salaus), mikä taas tarkoittaa sitä, että palvelin todella on, kuka se sanoo olevansa [23, s. 7].

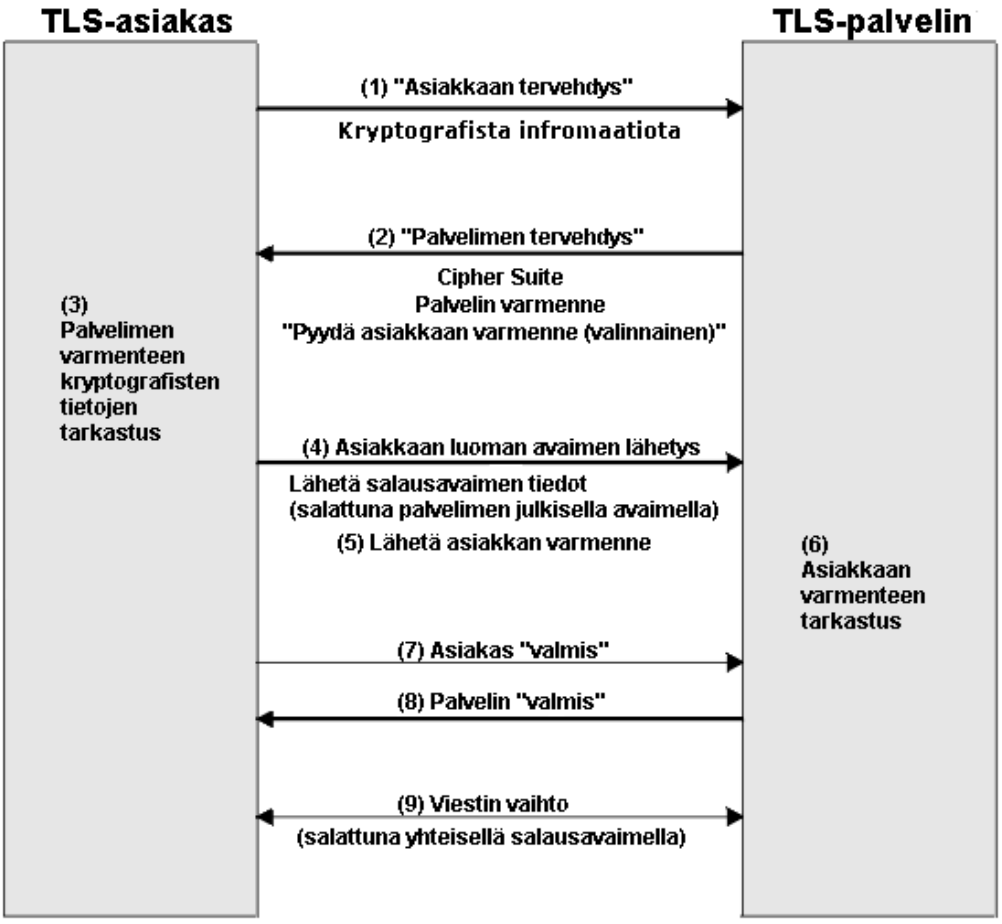
5) Jos palvelin pyysi tai vaati asiakkaalta varmennetta, lähettää asiakas sen palvelimelle tässä vaiheessa. Asiakas myös lähettää palvelimelle taas tavumerkkijonon, mutta tällä kertaa se on salattu asiakkaan yksityisellä avaimella.

6) Samoin kuin asiakas varmisti, että palvelimen lähettämä varmenne oli validi, palvelin varmistaa tässä vaiheessa asiakkaan lähettämän varmenteen validiuden. Palvelin varmistaa myös asiakkaan sähköisen allekirjoituksen purkamalla edellisessä kohdassa saamansa viestin käyttämällä asiakkaan julkista avainta, ja jos tulos vastaa aikaisemmassa vaiheessa saatua merkkijonoa, voidaan olla varmoja siitä, että viestin lähettäjällä on hallussa palvelimen saamaa julkista avainta vastaava yksityinen avain.

7) Asiakas lähettää palvelimelle viestin salattuna yhteisellä salausavaimella, mikä varmistaa autentikoimisen onnistumisen.

8) Palvelin lähettää asiakkaalle viestin salattuna yhteisellä salausavaimella, mikä varmistaa autentikoimisen onnistumisen.

9) Palvelin ja asiakas voivat nyt vaihtaa yhteisellä salausavaimella (symmetrinen salaus) salattuja viestejä, kunnes TLS-sessio vanhenee. [28.]



Kuva 14. TLS-kättelyn vaiheet [28].

4.3.2 TLS-profiilit API Connectissa

API Connectiin täytyy jotenkin saada rakennettua molemminpuolinen varmenteen validointi, jotta rajapinnan kutsuja pystytään tunnistamaan. Edellä kuvatun mukainen kättelyprosessi pitäisi siis saada aikaiseksi palveluväylään, kun työssä luodulle rajapinnalle lähetetään pyyntö. Onneksi API Connectista löytyy tähän valmis ratkaisu. TLS-palvelinprofiili (TLS server profile) on palveluväylään asetettava profiili, joka kertoo palveluväylälle, minkä varmenteen palvelin lähettää asiakkaalle. TLS-profiilin perusteella validoidaan myös asiakkaan varmenne, jos profiili on konfiguroitu kysymään sitä. Kättelyyn tarvittavat varmenteet asetetaan avainsäilöön (keystore) ja luotettujen varmenteiden säilöön (truststore), jotka sitten otetaan käyttöön liittämällä ne TLS-palvelinprofiiliin.

Avainsäilöön asetetaan palvelinpuolen varmenteet, eli varmenne, jonka palvelin lähettää asiakkaalle kättelyssä ja josta asiakas varmistaa palvelimen aitouden. Palvelinvarmenteita on kolmea eri laatua: DV (Domain Validated), OV (Organization Validated) ja EV (Extended Validation), jotka ovat toinen toistaan turvallisempia. DV on yleisimmin käytetty varmenne, mutta se on validoitu vain verkkotunnuksella [29]. Tämä tarkoittaa sitä, että varmenne varmistaa vain sen, että varmenteen haltija omistaa domainin, joka on varmenteessa mainittu. OV-varmenne vaatii jo enemmän tietoja hakijalta. CA varmistaa, että OV-varmenteen hakijan yritys on todella, kuka se sanoo olevansa ja varmenteeseen tulee yrityksen nimi. OV-varmennetta käyttävä lähinnä yritykset, jotka haluavat lisäsuojaa asiakkaillensa. EV on kaikista turvallisin varmenne. Yrityksen muiden tietojen lisäksi CA varmistaa, että yritys on fyysisesti olemassa esimerkiksi puhelinsoiton kautta. EV on käytössä suurissa yrityksissä. [29.]

Tässä työssä asiakkaan puolelta oli vaatimuksena, että palvelinvarmenne on joko EV-laatua, tai jos se on otettu asiakkaan määrittelemältä varmenteiden myöntäjältä, se voi olla OV-laatua. Projektia varten päädyttiin ottamaan EV-varmenne samalta myöntäjältä, jolta muissakin projekteissa käytetyt varmenteet ovat. Avainsäilöön asetetaan sekä yksityinen avain että palvelimen varmenne, joka siis sisältää julkisen avaimen. TLS-kättelyn mukaisesti varmenne lähetetään asiakkaalle ja yksityistä avainta käytetään asiakkaan lähettämän viestin purkamiseen. Nyt kun palvelimen varmenne on saatu kuntoon, täytyy vielä asiakkaan varmenne validoida.

Luotettujen varmenteiden säilö on lista varmenteita, johon palvelin luottaa. Kun asiakas TLS-kättelyssä lähettää oman varmenteensa palvelimelle, verrataan sitä tätä listaa vasten. Säilöön voitaisiin asettaa vaikkapa asiakkaan varmenteen myöntäjän juurivarmenne, jolloin asiakkaan varmenne hyväksyttäisiin. Tämä ei kuitenkaan ole suositeltavaa, sillä kuka tahansa, jolla on hallussa saman myöntäjän varmenne, pääsisi autentikoinnista läpi. On siis turvallisempaa käyttää pidemmällä varmenneketjussa olevia, enemmän tietoa sisältäviä varmenteita. Tässä työssä luotettujen varmenteiden säilöön asetettavat varmenteet saatiin rajapintaa käyttäviltä instituutioilta.

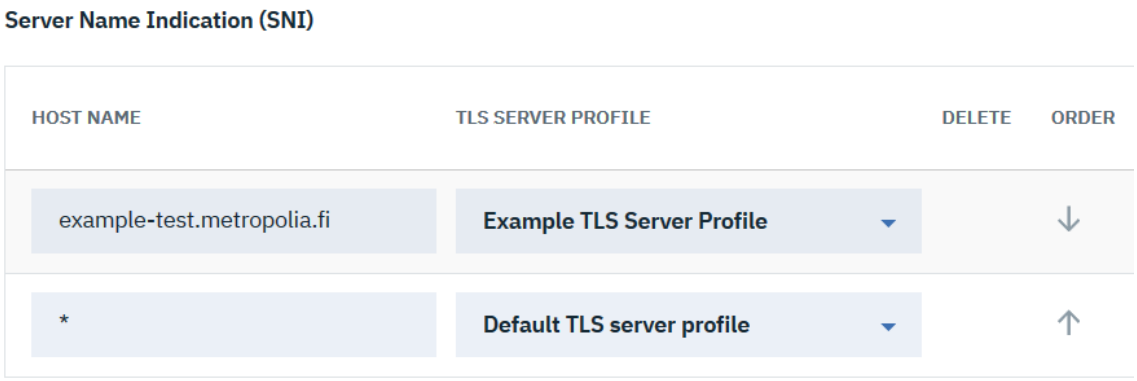
Kun molemmat varmennelistat oli luotu, valittiin profiiliin tuetut TLS-versiot. Tässä työssä valittiin, että ainut tuettu versio on TLS1.2, sillä se on turvallisempi kuin aikaisemmat

versiot ja uusin TLS1.3 ei ole vielä vaihtoehtona API Connectilla. Myös tuetut cipher suitet on listattu profiilissa ja listaa voi muokata profiilikohtaisesti. Profiiliin asetetaan vielä se, pyydetäänkö asiakkaalta varmennetta kättelyn aikana. Tähän on kolme vaihtoehtoa:

- None: Asiakkaan varmennetta ei kysytä.
- Request: Asiakasta pyydetään lähettämään varmenne, mutta kättely ei epäonnistu suoraan, vaikka asiakas ei lähettäisi varmennetta, vaan varmenne lähetetään palveluväylälle, jossa varmenne voidaan hyväksyä tai hylätä. [30.]
- Require: Asiakasta vaaditaan lähettämään varmenne. Jos asiakas ei lähetä omaa varmennettaan palvelimelle, kättely epäonnistuu. Tässä työssä tehdyssä rajapinnassa asiakasta vaaditaan lähettämään varmenne.

Kun uusi profiili on luotu, asetetaan se palveluväylälle. Samalla palveluväylällä voi olla useampia TLS-palvelinprofiileja, minkä tekee mahdolliseksi SNI (Server Name Indication). SNI on laajennus TLS-protokollaan ja mahdollistaa sen, että sama palvelin ja IP pystyy tarjoamaan useita eri palvelinvarmenteita [31]. SNI kertoo TLS-kättelyn alussa, mihin isäntänimeen (hostname) asiakas haluaa yhdistää, josta palvelin tietää, minkä varmenteen se tarjoaa asiakkaalle [31]. Mahdolliset SNI:t asetetaan palveluväylään ja jokaiselle voidaan antaa oma TLS-palvelinprofiili. Kun palvelin saa yhteydenoton, käy se läpi SNI-listan ja ottaa käyttöön sen profiilin, johon asiakkaan käyttämä isäntänimi osuu. Isäntänimi voi sisältää jokerimerkin (wildcard), ja onkin järkevää asettaa SNI-listan viimeiseksi nimeksi ”*”, joka ottaa kiinni kaikki yhteydet, jotka eivät osu tiettyyn SNI:hin. Vaihtoehtoinen tapa olisi ollut luoda toinen palveluväylä, jolle pätisivät aina tämän uuden profiilin asetukset, mutta se olisi ollut turhankin iso työ yhden rajapinnan takia.

Kuvassa 15 on luotu uusi isäntänimi ”example-test.metropolia.fi”. Kun palveluväylä vastaanottaa kutsun, se käy kuvan 15 listan läpi ja jos asiakkaan käyttämä isäntänimi osuu luotuun nimeen, niin palveluväylä ottaa käyttöön uuden ”Example TLS Server Profile” -nimisen profiilin. Jos taas asiakkaan käyttämä isäntänimi ei osu listan ensimmäiseen vaihtoehtoon, palveluväylä tarkistaa listan seuraavan nimen, kunnes lista loppuu. Tässä esimerkissä listan viimeisenä on jokerimerkki, joka ottaa vastaan kaikki kutsut, jotka eivät ole osuneet listan aiempiin nimiin.



Kuva 15. SNI-asetus TLS-palvelinprofiilissa.

Pilviympäristön takia, ennen kuin palveluväylä tunnisti uuden SNI:tä varten luodun isäntänimen, piti se lisätä Kubernetes-klusterin palveluväylän Ingress-asetuksiin. Ingress on Kubernetes-komponentti, joka kontrolloi ulkopuolisia yhteyksiä klusterin palveluihin. Tämän jälkeen oli mahdollista testata, että instituutiot pystyvät noutamaan dataa mikropalvelulta GET-pyynnöillä onnistuneesti, mutta väärällä varmenteella yritettäessä kutsu pysähtyy TLS-kättelyyn.

5 Yhteenveto

Tässä työssä toteutettiin integraatiot yritysjärjestelmien välille käyttämällä IBM:n tuotteita. Työ voidaan jakaa kahteen osaan. Ensimmäinen osa on kaksi IBM:n App Connect Enterprise -alustalla toteutettua integraatiota tietovaraston ja mikropalvelun, sekä asiakkuudenhallinnan ja mikropalvelun välille. Toinen osa oli mikropalvelun rajapinnan julkaisu ulkopuolisille instituutioille käyttäen IBM:n API Connect työkalua. Projektihallinnassa käytettiin Scrum-viitekehystä, joka sisälsi jokapäiväisen kokouksen integraatiotiimin kesken. Kaikkien projektin osapuolten kanssa pidettiin kokouksia kahdesti viikossa.

Projektin toteutus aloitettiin kesäkuussa 2019, ja se on testauksessa tämän kirjoittamisen aikaan lokakuussa 2019. Näillä näkymin toteutus viedään tuotantoon marraskuussa 2019.

Työkalut olivat ennalta määrättyjä, joten toisia vaihtoehtoja ei tutkittu. Integraatiot sujuivat kuitenkin suurimmaksi osin ongelmitta IBM:n tarjonnalla. ACE:n edeltäjä IBM Integration Bus oli jo ennestään minulle tuttu ja siirtyminen uuteen versioon sujui melko vaivattomasti, sillä ne ovat hyvin samanlaisia, mitä tulee integraatioiden rakentamiseen. API Connectista minulla oli hieman vähemmän kokemusta. Suurin osa sillä tehtävästä työstä oli myös asiakkaan varmenteen validointia, mitä en ollut toteuttanut ennen API Connectilla.

Työssä tuli siis vastaan paljon uutta, vaikka itse työkalut ja työtavat olivatkin ennestään ainakin jollain tasolla tuttuja. ACE:llä tehtyjen integraatioiden kanssa uutena teknologiana tuli Google Cloud Pub/Sub -prosessi, kun taas API Connectin rajapinnan julkaisussa pääsin perehtymään syvemmin varmenteisiin ja palvelimen ja asiakkaan väliseen validointiprosessiin. API Connectin kanssa pääsin myös tutustumaan pilvimaailmaan Google Cloud Platformin muodossa, sekä Docker- ja Kubernetes-teknologioihin, joita käyttäen API Connect oli asennettu Googlen pilveen.

Työtä tehdessäni tutustuin siis uusiin teknologioihin ja sitä kirjottaessa pääsin myös perehtymään hieman syvemmin teknologioiden taustalla olevaan teoriaan, joten insinöörityö oli hyvin opettavainen kokemus.

Lähteet

1 Juric M.B., Jennings F., Sarang P., Loganathan R. 2007. SOA Approach to Integration. Packt Publishing.

2 Lee, Jinyoul & Siau, Keng & Hong, Soongoo. 2003. Enterprise integration with ERP and EAI. Commun. ACM. 46.

3 Dancing Around EAI 'Bear Traps'. Verkkodokumentti. <http://www.ebizq.net/topics/int_sbp/features/3463.html?page=2>. Luettu: 07.10.2019.

4 Understanding enterprise application integration - The benefits of ESB for EAI. Verkkodokumentti. <https://www.mulesoft.com/resources/esb/enterprise-application-integration-eai-and-esb>. Luettu: 07.10.2019.

5 Hubs, spokes, and point-to-point: a quick guide to common integration terminology. Verkkodokumentti. <https://www.ellucian.com/insights/hubs-spokes-and-point-point-quick-guide-common-integration-terminology>. Luettu: 08.10.2019

6 What is an ESB (enterprise service bus)? Verkkodokumentti. <https://www.ibm.com/cloud/learn/esb>. Luettu: 08.10.2019.

7 What is an ESB? Verkkodokumentti. < https://www.mulesoft.com/resources/esb/what-esb>. Luettu: 08.10.2019.

8 Enterprise Service Bus. Verkkokuva. <https://en.wikipedia.org/wiki/Enterprise_service_bus>. Kuva haettu: 08.10.2019.

9 Microservices versus ESB. Verkkodokumentti. <https://blogs.mulesoft.com/dev/microservices-dev/microservices-versus-esb/>. Luettu: 20.10.2019.

10 The fate of the ESB. Verkkodokumentti. <https://developer.ibm.com/articles/cl-lightweight-integration-1/>. Luettu: 09.10.2019.

11 Haastattelu. Senior Project Manager Pekka Saxberg. 10.10.2019.

12 IBM App Connect Enterprise introduction. Verkkodokumentti. <https://www.ibm.com/support/knowledgecenter/en/SSTTDS_11.0.0/com.ibm.etools.mft.doc/bb43020_.htm>. Luettu: 10.10.2019.

13 Message flows overview. Verkkodokumentti. <https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ac00310_.htm>. Luettu: 12.10.2019.

14 Integration servers. Verkkodokumentti. <https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ae00270_.htm>. Luettu: 12.10.2019.

15 The IBM Integration Bus environment. Verkkodokumentti. <https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft.doc/be43400_.htm>. Luettu: 12.10.2019.

16 Inventory check availability flow. Verkkokuva. < https://www.ibm.com/developerworks/websphere/library/techarticles/1112_ren/1112_ren.html >. Kuva haettu: 13.10.2019.

17 Packaging strategy and terminology in API Connect. Verkkolähde. <https://www.ibm.com/support/knowledgecenter/en/SSFS6T/com.ibm.apic.overview.doc/capim_overview_orgsprodsplansapis.html>. Luettu: 15.10.2019.

18 API Connect Reserved Instance. Verkkokuva. < https://www.ibm.com/support/knowledgecenter/en/SSFS6T/com.ibm.apic.overview.doc/capic_reservedinstance.html>. Kuva haettu: 14.10.2019.

19 Publisher-Subscriber pattern. Verkkodokumentti. <https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber>. Luettu: 30.10.2019.

20 What Is Cloud Pub/Sub? Verkkolähde. < https://cloud.google.com/pubsub/docs/overview>. Luettu: 16.10.2019.

21 Quickstart: Using Client Libraries. Verkkodokumentti. <https://cloud.google.com/pubsub/docs/quickstart-client-libraries>. Luettu: 19.10.2019.

22 Message tree stucture. Verkkokuva. < https://www.ibm.com/support/knowledgecenter/en/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ac12610_.htm>. Kuva haettu: 19.10.2019.

23 Polk, Tim; McKay, Terry; Chokhani, Santosh. 2014. "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations". National Institute of Standards and Technology.

24 What Is a Cryptographic Key? | Keys and SSL Encryption. Verkkodokumentti. < https://www.cloudflare.com/learning/ssl/what-is-a-cryptographic-key/>. Luettu: 30.10.2019.

25 Symmetric vs. Asymmetric Encryption – What are differences? Verkkodokumentti. <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>. Luettu: 30.10.2019.

26 Certificate Authorities & Trust Hierarchies. Verkkodokumentti. <https://www.globalsign.com/en/ssl-information-center/what-are-certification-authorities-trust-hierarchies/>. Luettu: 17.10.2019.

27 How TLS provides identification, authentication, confidentiality, and integrity. Verkkolähde. <https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.sec.doc/q009940_.htm>. Luettu: 17.10.2019.

28 An overview of the SSL or TLS handshake. Verkkokuva ja verkkolähde. <https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm>. Kuva haettu ja lähde luettu: 17.10.2019.

29 DV OV and EV Certificates. Verkkodokumentti. <https://www.ssl.com/article/dv-ov-and-ev-certificates/>. Luettu: 21.10.2019.

30 Creating a TLS Server Profile. Verkkolähde. < https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.cmc.doc/ssl.html>. Luettu: 19.10.2019.

31 What is Server Name Indication (SNI)? Verkkodokumentti. < https://www.globalsign.com/en/blog/what-is-server-name-indication/>. Luettu: 19.10.2019.