Tampereen yliopisto

Kim Milán

SKAALAUTUVAN PILVIPOHJAISEN JÄRJESTELMÄN KEHITYS

Skaalautuva, luotettava ja kustannustehokas mikropalvelujärjestelmä

Diplomityö
Tieto- ja sähkötekniikan tiedekunta
Tarkastaja: Kari Systä
Tarkastaja: Outi Sievi-Korte
Tammikuu 2022

TIIVISTELMÄ

Kim Milán: Skaalautuvan pilvipohjaisen järjestelmän kehitys
Diplomityö
Tampereen yliopisto
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Tammikuu 2022

Pilvipalvelut ja niihin pohjautuvien järjestelmien kehitys ovat viimeisen vuosikymmenen aikana nostaneet suosiotaan merkittävästi. Tämän myötä useat yritykset ovat alkaneet siirtää perinteisesti on-premise-ratkaisuina toteutettuja toimintojaan pilviympäristöihin.

Tässä työssä käsitellään eräälle energia-alan yritykselle tehtyä asiakasprojektia, jossa pilvipalveluympäristöön oli määrä toteuttaa järjestelmä yrityksen erilaisten käyttöpaikkojen, mittareiden, mittariasennusten, mittausdatan ja mittarikonfiguraatioiden hallintaan. Järjestelmä käsittelee sähkömarkkinadataa sisältäen muun muassa yksityistä, GDPR alaista ja asiakkaiden laskutukseen liittyvää henkilötieto- ja mittausdataa, ylläpitää ja välittää sähkömittareiden asennusprosesseihin liittyvää viestintää, ja hallinnoi sähkömittareille lähetettäviä ohjelmistokonfiguraatioita. Kehitettävän järjestelmän on siis asiakkaan toimialan ja liiketoiminnan luonteen vuoksi oltava skaalautuva, luotettava ja myös kustannustehokas muuttuvien datamäärien käsittelemiseksi ja esimerkiksi oikean laskutuksen ja mittausdatan eheyden varmistamiseksi.

Työssä tutkittiin pilvipalvelupohjaisia järjestelmiä ja niiden ominaisuuksia yleisellä tasolla, jotta käsiteltävän asiakasprojektin esimerkkijärjestelmän suunnitteluperiaatteita, arkkitehtuuria ja toteutusta sekä niistä tehtyjä päätelmiä voidaan arvioida. Pääasiassa työssä keskitytään kuitenkin itse esimerkkijärjestelmän toteutukseen, joka käydään skaalautuvuuden, luotettavuuden ja kustannustehokkuuden vaatimuksiin nähden oleellisilta osin läpi top-down-menetelmällä.

Työssä käsiteltävä esimerkkijärjestelmä on Amazonin AWS-ympäristöön tuotettu tapahtumapohjainen mikropalvelujärjestelmä, joka koostuu useista eri AWS tileille tuotetuista alipalveluista, joiden vastuualueet ja toiminnot on jaettu neljään päätyyppiin: käyttöpaikat ja mittarit, työtilaukset, mittarikonfiguraatiot ja järjestelmän ulkoinen viestintä. Järjestelmän eri osien välistä viestintää käsittelee keskitetty tapahtumaväylä-komponentti ja järjestelmät toiminnot on pääosin toteutettu palvelimettomien Lambda-funktioiden avulla.

Asiakasprojektin myötä syntyneen esimerkkijärjestelmän suunnittelussa ja toteutuksessa on pyritty noudattamaan AWS-ympäristön hyviä suunnitteluperiaatteita, joten tämänhetkisten tietojen ja kehityksen tilan valossa se täyttää sille asetetut vaatimukset skaalautuvuudesta, luotettavuudesta ja kustannustehokkuudesta, mutta lopullisten arvioiden tekeminen on tässä vaiheessa mahdotonta toistaiseksi puutteellisen testauksen ja vähäisen testidatan vuoksi. Järjestelmän kehitys jatkuu edelleen, erityisesti riittävän suorituskyvyn suurilla datamäärillä varmistamiseksi.

Avainsanat: Pilvipohjaiset-järjestelmät, pilvialustat, skaalautuvuus, Amazon Web Services, AWS Lambda

ABSTRACT

Kim Milán: Developing a scalable cloud-based system
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
Janurary 2022

Cloud platforms and the development of cloud-based systems have gained a significant amount of popularity in the last decade. Due to this, a number of companies have started moving their traditional on-premise-solutions to various cloud platforms.

This thesis focuses on a particular client project carried out for a company in the energy business, where the goal of the project was to develop an application for the management of various actions and data storage related to the client's metering points, meters, work orders, measurement data and meter configurations. The system manages GDPR-applicable data about the end clients and their billing, handles the data and communication related to meter installation processes, and manages the system configurations of metering devices. Due to the business area and the nature of client's business in general, the developed system must be scalable, reliable, and cost-effective to be able to adapt to changing data loads effectively and to ensure the integrity of the data for example for billing and energy measurement.

This thesis investigates cloud-based applications and systems, and their properties on a general level to make it possible to evaluate the example system's design principles, architecture, and overall execution, as well as the conclusions derived from them. The primary focus is on the actual execution and development process of the system, which is described from the top down based on the requirements of the system related to scalability, integrity, and cost-effectiveness.

The example system examined in this thesis is an event-based microservice application running on the Amazon AWS cloud platform. The system consists of several microservices, which have been divided into four primary responsibility areas: metering points and meters, work orders, meter configurations and communications to external systems. Communication between the microservices is primarily managed by a centralized event bridge component, and the majority of the actual functionality of the system has been developed using serverless Lambda-functions.

The design and development of the project's result used as an example system in this thesis has strived to adhere to best practices and design principles of the AWS platform and considering current test data and the state of development the system meets the requirements for scalability, integrity, and cost-effectiveness. However, due to a lack of adequate testing and test data, no final verdict can be made at this time. The development of the system is still on going, with a focus to ensure sufficient performance when handling large amounts of data.

Keywords: Cloud-based systems, cloud platforms, scalability, Amazon Web Services, AWS Lambda

ALKUSANAT

Diplomityö tehtiin töissä tekemääni asiakasprojektiin liittyen, mutta lähinnä kyseistä projektia esimerkkinä käyttäen eikä esimerkiksi toimeksiantona työnantajalleni tai asiakasyritykselle. Kiitokset Kari Syställe ohjaustyöstä ja tarkastuksesta, sekä Outi Sievi-Kortelle tarkastuksesta. Kiitokset myös läheisilleni ja ystävilleni tuesta kirjoitusprosessin aikana.

Tampereella, 17.1.2022

Kim Milán

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

NIST	National Institute of Standards and Technology
LAN	Local Area Network, lähiverkko
WAN	Wide Area Network, ulkoverkko, suuralueverkko
VPN	Virtual Private Network, virtuaalinen erillisverkko
SLA	Service Level Agreement, palvelutasosopimus
IaaS	Infrastructure as a Service, palvelumalli
PaaS	Platform as a Service, palvelumalli
SaaS	Software as a Service, palvelumalli
AWS	Amazon Web Service
GPGPU	General-Purpose computing on Graphics Processing Units
E2E	End-to-end, loppukäyttäjän kokemusta vastaava ohjelmistotestaus
CDK	(AWS) Cloud Development Kit
ETL	Extract, Transform & Load
CI	Continuous Integration, jatkuva integraatio
CD	Continuous Deployment, jatkuva jakelu

1. JOHDANTO

Tämän diplomityön tarkoituksena on käsitellä pilvipohjaisia, skaalautuvia mikropalvelujärjestelmiä, asiakasprojektina tuotetun järjestelmän kehityksen rinnalla. Projektityöhön liittyvän tutkimuksen tavoite on tarkastella ja arvioida kyseisen sähkömarkkinoille tarkoitetun pilvipohjaisen mikropalvelujärjestelmän suunnitteluperiaatteita ja arkkitehtuuria sekä tarkempaa teknistä toteutusta käyttötarkoitukseen, käsiteltävän datan vaatimuksiin ja skaalautuvien mikropalvelujärjestelmien yleisiin periaatteisiin nähden. Toteutetun järjestelmän prosessoima sähkömarkkinadata käsittää asiakasyrityksen käyttöpaikkojen eli käytännössä asiakaskohteiden ja niihin asennettujen mittalaitteiden tietoja, mittalaitteiden tuottamaa mittausdataa, sekä arkaluontoista ja esimerkiksi asiakkaiden laskutukseen liittyvää dataa, minkä vuoksi järjestelmän skaalautuvuus, luotettavuus ja toisaalta myös kustannustehokkuus ovat ensisijaisen tärkeitä tavoitteita järjestelmän kehityksessä.

Diplomityön aihe syntyi asiakasyrityksen tarvitseman järjestelmän kehitystyön aikana tarpeesta saada lisätietoa ja ymmärrystä skaalautuvista, pilvipohjaisista järjestelmistä myös yleisemmällä tasolla muita vastaavanlaisia kehitysprojekteja varten. Tämän vuoksi tutkimustyönä tässä diplomityössä keskitytään ensisijaisesti edellä kuvatun järjestelmän toteutukseen ja siitä syntyneiden tulosten arviointiin, yleiskäyttöisempiä huomioita ja päätelmiä silmällä pitäen.

Diplomityön ensimmäisessä osiossa, luvussa 2 käsitellään pilvipohjaisten järjestelmien määritelmää ja ominaisuuksia yleisellä tasolla. Ensimmäisen osion tarkoitus on kartoittaa teoriaa pilvipalveluihin ja -sovelluksiin liittyen, ja käydä lyhyesti läpi erilaisia vaihtoehtoja ja huomioon otettavia asioita pilvipalvelupohjaisen järjestelmän suunnittelussa. Ensimmäisessä osassa tarkennetaan myös skaalautuvuuden, luotettavuuden sekä kustannustehokkuuden määritelmiä, joihin toteutetun esimerkkijärjestelmän ominaisuuksia myöhemmin verrataan.

Toisessa osiossa, luvuissa 3 ja 4 keskitytään diplomityöhön tarkasteltavaksi valitun esimerkkijärjestelmän asiakasprojektin vaatimuksiin ja käytännön tekniseen toteutukseen. Teknisen toteutuksen top-down-tyylinen kuvaus ja käsittely aloitetaan järjestelmän arkkitehtuurista, josta edetään järjestelmän alempiin osakokonaisuuksiin ja komponenttei-

hin niiltä osin, kuin ne ovat diplomityön aiheen kannalta oleellisia. Toisessa osiossa kuvataan mitä osa-alueita toteutus pitää sisällään, miten eri osat ja ominaisuudet toteutettiin ja minkä vuoksi ne toteutettiin kyseisellä tavalla.

Viimeisessä osassa, luvussa 5, arvioidaan esimerkkijärjestelmän vaatimusten toteutumista, ja eri ominaisuuksien ja prosessien toteutuksen onnistumista järjestelmän skaalautuvuus, luotettavuus ja kustannustehokkuus huomioon ottaen. Luvussa käydään läpi myös yleisesti kehityksen aikana nousseita huomioita esimerkkijärjestelmän tilasta ja kehitystyöstä.

2. PILVIPOHJAISET JÄRJESTELMÄT

2.1 Pilvipalvelupohjainen arkkitehtuuri ja palvelumallit

Pilvipalvelupohjaisen arkkitehtuurin määrittelemiseen kuuluu olennaisena osana selvitys siitä, mitä käsitteet ”pilvi” ja pilvipalvelut ylipäätään pitävät sisällään ja tarkoittavat. Pilven määritelmä onkin jatkuvasti muuttunut ja kehittynyt vuosien mittaan, alun perin tarkoittaen lähinnä joukkoa palveluita ja teknologioita, joiden sisäisistä suhteista ja toiminnasta ei käyttäjillä ole tietoa. Nykyään käytetyin määritelmä on peräisin yhdysvaltalaisen National Institute of Standards and Technology (NIST) -viraston tekemä määritelmä, joka koostuu kolmesta pääkomponentista: pilven ominaisuudet, deployment-mallit ja palvelumallit. [1]

NIST:n määritelmän [2] ensimmäisen osan mukaan pilveksi voidaan kutsua palvelua, joka täyttää seuraavat viisi ominaisuutta:

1. On-demand self-service eli käyttäjien on voitava hallinnoida sovellustensa resursseja ja palveluja ilman alustan ylläpitäjän tai muun tukihenkilön erillisiä toimia. Käytännössä pilvipalvelun käyttäjä voi siis esimerkiksi itse resursoida sovellukseensa lisää tallennustilaa, jolloin pilvipalvelun sisäiset palvelut hoitavat tarvittavat toimet automatisoidusti.
2. Broad internet access eli pilvipalveluun ja sen hallintaan on päästävä helposti käyttäjän internetyhteyden tilasta riippumatta. Käytännössä tämä tarkoittaa sitä, että palvelun käyttöön ei saa liittyä raskasta (tai lainkaan) erillistä client-sovellusta eikä käyttäjältä voida edellyttää nopeaa internetyhteyttä tai suurta kaistanleveyttä.
3. Resource pooling eli palvelun on kyettävä poolaamaan resursseja ja hyödyntämään niitä tehokkaasti ja joustavasti eri käyttäjien ja sovellusten muuttuvien tarpeiden mukaan. Toisin sanoen sen sijaan, että pilvipalvelun resurssit varattaisiin kokonaan yhden käyttäjän tai sovelluksen tarpeisiin, resurssit ja niiden käyttö pitää mukautua tilanteiden ja tarpeiden muuttuessa esimerkiksi niin, että ensimmäisen käyttäjän käytettävissä olevaa fyysistä muistia jaetaan tarvittaessa toisille käyttäjille, mikäli sitä on vapaana.
4. Rapid elasticity eli pilvipalvelun on kyettävä mukautumaan ja skaalautumaan käyttäjien tarpeiden mukaan. Käytännössä tämä tarkoittaa esimerkiksi erilaisten

resurssien kuten muistin ja tallennustilan sekä vaikka palvelinten määrän skaalautumista todellisen käyttötarpeen mukaan automaattisesti.

5. Measured service eli palvelun käyttöaste (käyttöaika, käytetty kaista, käytetty data…) on oltava mitattavissa. Mittaus mahdollistaa tarvittavien resurssien, suorituskyvyn sekä kenties oleellisimmin hinnan arvioimisen.

[2] [3]

Määritelmän mukaisen pilvipalvelun on toteutettava kaikki viisi ominaisuutta.

Määritelmän toinen osa käsittelee pilvipalveluiden neljää erilaista deployment- eli käyttöönottomallia:

1. Julkinen pilvi tarkoittaa pilvipalvelua, joka sijaitsee kokonaan ulkoisen palveluntarjoajan ympäristössä ja käyttäjät muodostavat yhteyden palveluun julkisen internetin kautta.
2. Yksityinen pilvi on julkisen mallin vastakohta, eli pilven kaikki järjestelmät ja resurssit ovat asiakkaiden omissa ympäristöissä ja yhteys pilveen mahdollinen vain LAN/WAN-verkossa tai VPN-yhteyden välityksellä.
3. Yhteisömallin mukainen pilvi on yksityisempi versio julkisesta pilvestä siten, että pilviympäristö on useamman osapuolen tai organisaation jakama mutta ei julkisesti saatavilla oleva palvelu.
4. Hybridimalli on kahden tai useamman muun mallin pilvistä koostuva kokonaisuus, jossa erilliset pilvet ovat toisiinsa liittyneitä.

[2]

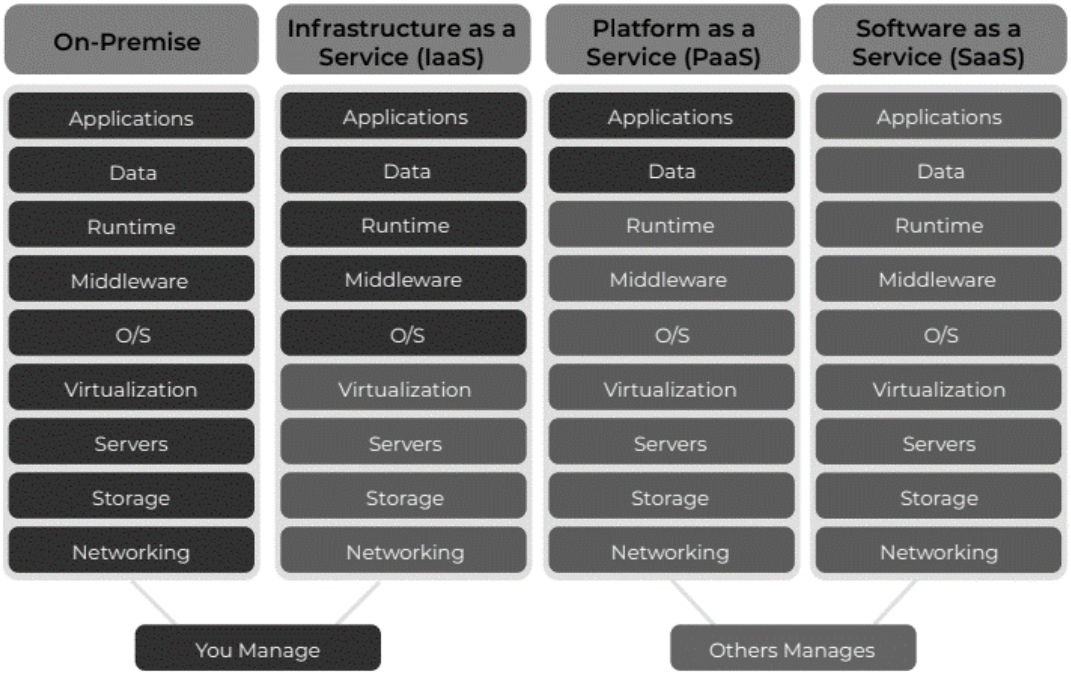
Arkikielessä pilvestä puhuttaessa viitataan useimmiten julkisiin pilvipalveluihin [1], joita tarjoavat toimijat kuten Amazon, Microsoft ja Google.

NIST:n määritelmän kolmas ja viimeinen osa käsittelee pilvipalveluiden palvelumalleja, joita ovat:

1. Infrastructure as a Service (IaaS) eli pilvipalvelun tarjoaja tarjoaa käyttäjille infrastruktuurin, jonka päälle käyttäjät voivat sovelluksensa rakentaa. Käytännössä tämä voi kattaa esimerkiksi sekä fyysisiä että virtuaalisia laitteita, kovalevytilaa ym. IaaS-palvelumalli tarjoaa käyttäjilleen eniten vapautta mutta sen myötä myös vastuu eri resursseista on käyttäjillä itsellään.

2. Platform as a Service (PaaS) eli pilvipalvelun tarjoaja tarjoaa alustan käyttäjien sovelluksille. Käytännössä siis käyttäjät varaavat sovellukselleen tietyt palvelut ja resurssit, joiden ylläpidosta vastaa palveluntarjoaja. Näin ollen vastuu infrastruktuurin eri osista siirtyy palveluntarjoajalle, ja käyttäjien vastuulle jää vastuu oman sovelluksen toiminnasta.
3. Software as a Service (SaaS) on kokonaisvaltaisin palvelumalli, jossa pilvipalvelun tarjoaja vastaa niin infrastruktuurista kuin käytettävästä sovelluksesta. SaaS on siis valmis ohjelmistopaketti, jonka ylläpito ei ole lainkaan käyttäjän vastuulla.

[2]



Kuva 1. *Vastuunjako eri palvelumalleissa*

Palvelumalli määrittää käytännössä siis vastuunjaon palveluntarjoajan ja käyttäjien välillä.

Pilviarkkitehtuuri käsittää kaikki osa-alueet, jotka muodostavat valmiin pilvisovelluksen, eli pilviarkkitehtuurin tarkoitus on määrittää pilvisovelluksen infrastruktuuri sekä siihen kuuluvat palvelut ja resurssit.

Pilvipalvelusovellus eroaa monin tavoin perinteisestä on-premise-ratkaisusta, jossa sovelluksen koko infrastruktuuri ja eri osa-alueet ovat asiakkaan omassa ympäristössä ja omalla vastuulla. Yksi suurimmista eduista pilvellä on-premiseen nähden on sovellusten

skaalautuvuus, jonka seurauksena pilvisovellusten ylläpito vaihtelevan käyttöasteen sovelluksissa on helpompaa ja edullisempaa kun esimerkiksi varatun fyysisen muistin tai kaistanleveyden kokoa voidaan kasvattaa tai pienentää todellisen tarpeen mukaan joustavasti. Toinen merkittävä etu on palvelumallin määrittämä vastuunjako palveluntarjoajan ja asiakkaan välillä, minkä seurauksena sovelluksen ylläpitoon asiakkaalta vaadittavat resurssit ja tietotaito vähenevät.

Datan käyttö ja omistajuus, ja toisaalta tietoturvakysymykset yleisesti ovat tekijöitä, jotka katsotaan on-premisen eduiksi verrattuna pilvisovelluksiin. Kun sovellukset palvelimet ja tietokannat sijaitsevat kolmannen osapuolen (pilvipalvelun tarjoaja) hallinnoimilla alustoilla, asiakkailla ei voi olla täyttä tietoa eikä varmuutta datan eheydestä tai luottamuksellisuudesta. Pilvipalveluihin kohdistetaan lisäksi paljon hyökkäyksiä ja onnistuneiden tietomurtojen seurauksena on usein suurten datamäärien ja yksityisten tietojen menetys ulkoisille tahoille.

2.2 Pilvipalvelualustat

Pilvipalveluita tarjoaa nykyisin suuri määrä eri kokoisia ja lähtöisiä toimijoita, joista oikean tarjoajan valintaan vaikuttaa monta tekijää. Pilvipalvelut koostuvat tarjoajasta riippuen erityyppisistä ja kokoisista ekosysteemeistä, ja saattavat soveltua tietynlaisen asiakaskunnan tarkoituksiin paremmin kuin toisten.

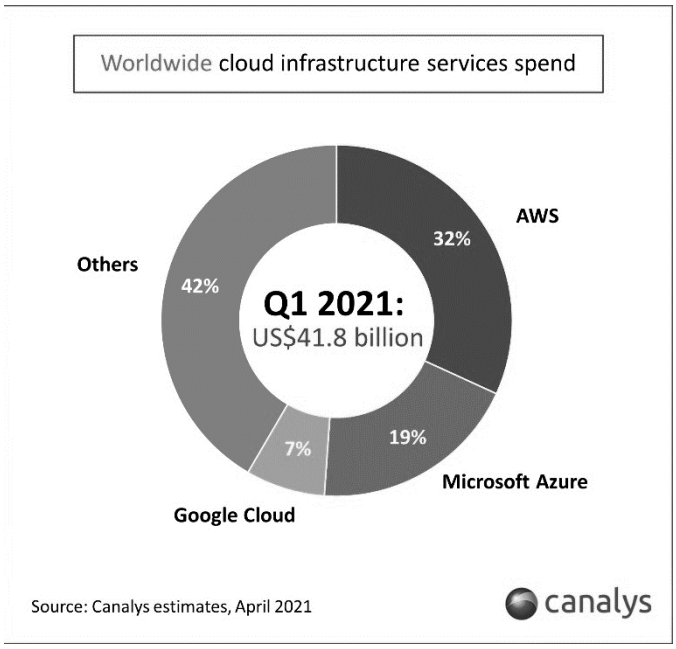
Pilvipalvelun ominaisuuksissa huomioon otettavia asioita ovat esimerkiksi palvelun eri osien resursoinnit ja monitoroinnin helppous, datamallien ja rajapintojen standardinmukaisuus sekä tarjoajan määrittämä palvelutasosopimus (Service Level Agreement, SLA), jonka mukaan palveluntarjoaja takaa nimen mukaisesti tietyn palvelun tason, pilven laadun, saatavuuden ja vastuunjaon suhteen.

Pilvipalvelutarjoajan valintaan vaikuttaa olennaisesti myös tarjoajan itsensä ominaisuudet. Tekijät, kuten palveluntarjoajan nykyinen ja odotettu taloustilanne, yleinen luotettavuus ja liiketoimintamalli vaikuttavat myös tarjotun palvelun eli pilviympäristön vakauteen ja luotettavuuteen.

Pilven tarjottujen ominaisuuksien ja ekosysteemin sekä palveluntarjoajan yleisen luotettavuuden ja vakauden lisäksi on otettava huomioon esimerkiksi mahdollisuus palveluntarjoajan vaihtamiseen ja siitä aiheutuviin datan- ja tulonmenetyksiin. Erityisesti jo palveluntarjoajaa valittaessa on pyrittävä välttämään niin kutsuttua ”vendor lock-in”-tilannetta, jossa käyttöehtojen tai esimerkiksi pilven arkkitehtuurin tai datamallien vuoksi pilvipalvelualustan vaihtamisesta toiseen aiheutuu kohtuuttomasti kuluja ja asiakassovelluksen seisokkiaikaa, tai on jopa kokonaan mahdotonta. ”Vendor lock-in”-tilanne on asiakkaan

näkökulmasta erityisen riskialtis palveluntarjoajan nostaessa palvelun hintoja merkittävästi, palvelun laadun heikentyessä tai esimerkiksi palveluntarjoajan joutuessa konkurssiin.

Markkinaosuuden mukaan kolme suurinta pilvipalveluiden tarjoajaa tällä hetkellä (viimeisin tieto Q1 2021) ovat Amazon Web Services, Microsoft Azure ja Google Cloud Platform, joiden yhteenlaskettu osuus markkinoista on noin 60 prosenttia.



Kuva 2. *Q1 2021 pilvipalveluiden markkinaosuudet, Canalys*

Amazonin, Microsoftin ja Googlen pilvialustoista lyhyesti:

Amazon Web Services (AWS)

Amazon Web Services on vuonna 2006 toimintansa aloittanut Amazonin julkinen pilvipalvelu, jonka pääpalvelumalli on IaaS [4]. AWS oli vuoden 2021 ensimmäisellä neljänneksellä markkinajohtaja 32 % markkinaosuudellaan [5]. AWS:n suurin etu markkinajohtajana on sen kattava kokoelma palveluita ja laaja maailmanlaajuinen saatavuus.

AWS koostuu erilaisista infrastruktuurikomponenteista kuten tallennustilasta (tiedostojen tallennukseen S3, tietokantapalvelut DynamoDB ja RDS), skaalautuvista laskenta- ja palvelinkomponenteista (palvelimettomat funktiot Lambda-palvelulla, virtuaalikoneet palvelinympäristöille Elastic Compute Cloud eli EC2-palvelulla) sekä monista muista websovelluksissa käytettävistä palveluista. AWS on määritellyt palvelukomponenttien ja palvelujen best practice -käyttöön AWS Well-Architected (AWS WA) -dokumentaation [6],

jolla pyritään ohjaamaan käyttäjiä AWS ympäristöä järkevästi käyttävän pilviarkkitehtuurin määrittämiseen ympäristön hyvien käytäntöjen mukaisesti.

Microsoft Azure

Microsoftin vuonna 2010 julkaisema, alun perin Windows Azure -nimellä tunnettu Microsoft Azure on Microsoftin vastine AWS:lle. Azure oli vuoden 2021 ensimmäisellä neljänneksellä toiseksi suurin pilvipalveluiden tarjoaja Canalyksen tekemän arvion mukaan [5]. Microsoft Azure tarjoaa pilvipalveluita IaaS-, PaaS- ja SaaS-mallien mukaisesti ja integroituu hyvin Microsoftin sovellusekosysteemin muiden palveluiden kanssa, mikä tekee siitä varteenotettavan vaihtoehdon erityisesti sellaisille enterprise-käyttäjille, joilla on jo käytössään muita Microsoftin ekosysteemin sovelluksia.

Kuten AWS, myös Azure koostuu laajasta kokoelmasta erilaisia komponentteja ja palveluita, joista useat vastaavat käyttötapauksiltaan lähes tai täysin AWS:n tarjoamia palveluita (esim. AWS S3 = Azure Blob Storage, AWS DynamoDB = Azure Cosmos DB, AWS Lambda = Azure Functions).

Google Cloud Platform (GCP)

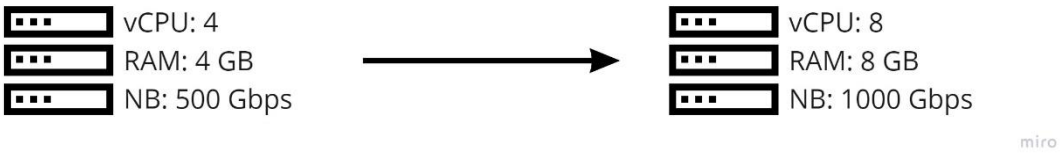
Vuonna 2011 Google julkaisi ensimmäisen pilvipalvelualustansa App Enginen, joka mahdollisti sovellusten suorittamisen pilvessä skaalautuvasti. Sittemmin Googlen pilvipalvelutarjonta on laajentunut ja tunnetaan nykyään yleisesti nimellä Google Cloud Platform tai lyhemmin Google Cloud. GCP oli vuoden 2021 ensimmäisellä neljänneksellä kolmanneksi suurin pilvipalveluiden tarjoaja Amazonin AWS:n ja Microsoftin Azuren jälkeen [5].

2.3 Pilvipohjaisen järjestelmän skaalautuvuus

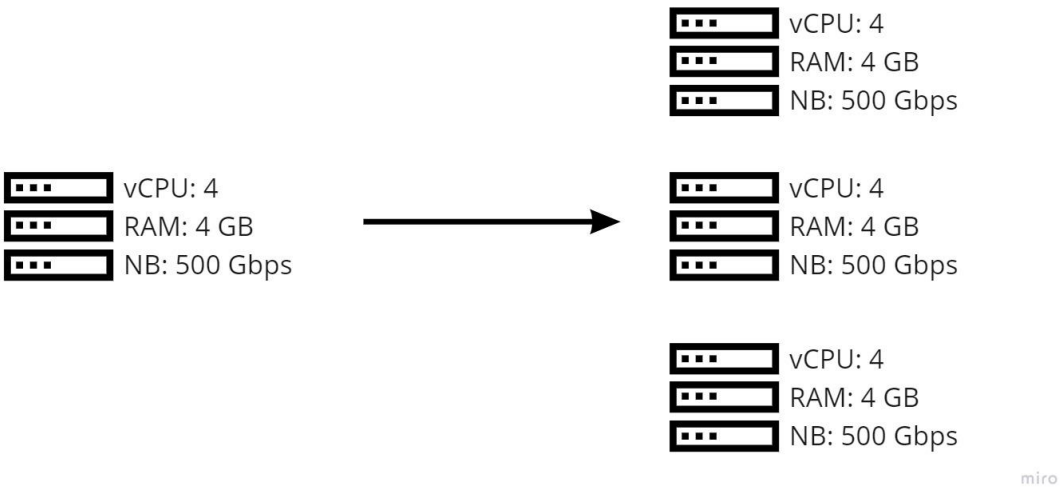
Skaalautuvuus on yksi pilvipalvelujärjestelmien tärkeimmistä, mutta samalla verrattain suppeasti ymmärretyistä ominaisuuksista [7]. Järjestelmän skaalautuminen voidaan toteuttaa monella tavalla eri komponenttien ja palveluiden osalta, mutta pääperiaate on, että järjestelmän varaamia resursseja ja suorituskykyä on mahdollista lisätä tai vähentää todellisen käytön tai tarpeen mukaan. Järkevällä skaalautuvuudella voidaan varautua ja reagoida järjestelmän suorituskyvyn pullonkauloihin, ja toisaalta minimoida käyttämättömistä resursseista aiheutuviin turhiin kustannuksiin.

Ohjelmistotekniikassa järjestelmän skaalautuminen voidaan yleisesti jakaa vertikaaliseen ja horisontaaliseen skaalautumiseen. Vertikaalisella skaalautumisella tarkoitetaan

olemassa olevien komponenttien käytettävissä olevien resurssien, kuten esimerkiksi välimuistin määrän säätelyä eli yksittäisen komponentin skaalausta ”ylöspäin”, kuten esitetty kuvassa 3. Horisontaalinen skaalautuminen vastaavasti tarkoittaa yksittäisten komponenttien, kuten palvelimien, lukumäärän kasvattamista tai pienentämistä toisiaan vastaavia instansseja luomalla, jolloin skaalautuminen tapahtuukin ”ulospäin” (kuva 4). Vertikaalinen skaalautuminen tuottaa lähtökohtaisesti suorituskyvyn kannalta parempia tuloksia, mutta on samalla käytettävien resurssien ja saatavilla olevan teknologian puolesta rajoitetumpaa kuin horisontaalinen skaalautuminen. Toisiaan vastaavien komponenttien lisääminen on yksinkertaisempaa ja lopulta edullisempaa, kuin teknisesti parempien/nopeampien/suurempien komponenttien tai resurssien hankinta tai kehitys ja ylläpito.



Kuva 3. *Vertikaalinen skaalautuminen*



Kuva 4. *Horisontaalinen skaalautuminen*

Pilvipohjaista mikropalveluarkkitehtuuria noudattavat, pienemmistä osa-alueista koostuvat järjestelmät soveltuvat tyypillisesti hyvin horisontaaliseen skaalautumiseen, sillä yksittäisten, itsenäisten palvelujen lukumäärää ja saatavuutta voidaan useimmiten muuttaa

tarpeen mukaan esimerkiksi käynnistämällä tarvittava määrä samanaikaisia suorituksia samasta funktionaalisesta komponentista ja poistamalla ne suorituksen jälkeen, tai replikoimalla tietokantoja tai rajapintoja esimerkiksi eri alueilla sijaitseville palvelimille. Kaikkia mikropalveluita ei kuitenkaan välttämättä ole perusteltua skaalata ulospäin, mikäli rinnakkaisten komponenttien yhtäaikaisesta suorituksesta seuraa ongelmia esimerkiksi viestijonon vaaditun käsittelyjärjestyksen tai mahdollisten tuplasuoritusten vuoksi.

Horisontaalisen skaalautumisen lisäksi pilvipohjaisen järjestelmän suunnittelussa on kuitenkin oleellista myös itse prosessien ja tarvittavien resurssien käytön optimointi. Pilviympäristössä toimivan järjestelmän ja sen käyttämien palveluiden skaalautuvuuteen vaikuttaa myös käyttäjistä täysin riippumattomia tekijöitä, kuten erilaisia laitteistopohjaisia ja verkonhallintaan liittyviä optimointitekijöitä. [7] Pilvialustojen skaalautuvuuden kehittämiseksi palvelujen tarjoajat ovat viime aikoina panostaneet nopeampiin ja kattavampiin verkkoihin, sekä alkaneet hyödyntää esimerkiksi näytönohjainten vahvaa rinnakkaista suoritusta General-Purpose computing on Graphics Processing Units (GPGPU) -tekniikan yleistyessä [8].

2.4 Datan eheys pilvipohjaisessa järjestelmässä

Datan eheydellä tarkoitetaan arviota siitä, kuinka laadukasta, kokonaista ja tarkkaa kerätty ja käsiteltävä data on. Datan eheys vaikuttaa sekä datan perusteella tehtyihin ulkoisesti loogisiin päätöksiin, esimerkiksi mittaamalla saatujen arvojen tai laskettujen ennusteiden mukaan, että järjestelmien sisäisiin prosesseihin edellä mainittujen tekijöiden lisäksi esimerkiksi dataformaattien vaikutuksesta. Datan eheyteen vaikuttavia tekijöitä ovat esimerkiksi inhimilliset virheet, kuten tietokantaan käsin tehtyjen muutosten aiheuttamat virheet, dataformaattien yhteensopimattomuudet esimerkiksi eri tiedostomuotoja ja niiden sisältämiä asetteluita ja muiden määrityksiä tulkitessa, sekä tietomurrot ja muut dataan kohdistuvat ulkoiset uhat. [9]

Pilviympäristöissä datan säilytys ja käsittely tapahtuu toteutettavasta palvelumallista riippuen enemmän tai vähemmän pilvipalveluiden tarjoajan toimesta, jolloin asiakasjärjestelmissä käytettävän datan eheyden arviointi on mahdollisista ennalta-arvaamattomista tuntemattomista tekijöistä johtuen entistä kriittisempää. Fyysinen laitteisto, kuten tietokantojen tallennukseen käytettävät palvelimet ja kovalevyt, on tyypillisesti jaettu samanaikaisesti useiden pilvipalvelun käyttäjien tarpeisiin ja käyttöön. Näin ollen esimerkiksi erilaiset pääsynhallintaan liittyvät virhetilanteet tai muut ohjelmistovirheet voivat sallia asiattomille käyttäjille tai järjestelmille pääsyn toisten pilvialustaa käyttävien järjestelmien dataan tai prosesseihin. [10] Erityisesti tunnetuimpiin pilvipalvelualustoihin kohdistuu lisäksi jatkuvasti ja lisääntyvissä määrin uusia ulkoisia hyökkäysyrityksiä [11], mikä lisää

tarvetta datan eheyden ja luotettavuuden varmistamiselle esimerkiksi jatkuvalla seurannalla.

Mikropalveluarkkitehtuurin mukaisella toteutuksella olennaista on myös datan säilytyksen vastuuttaminen eri palveluiden välillä ja dataformaattien ja rakenteiden yhtenäistetty ja tarkoituksenmukainen käyttö. Tiedonsiirto eri mikropalveluiden välillä on toteutettava niin, ettei esimerkiksi lukuarvojen tarkkuus tai datan sisältö muuten muutu palvelusta toiseen tai oleellista tietoa muuten menetetä tiedonsiirtoon käytettävien palveluiden, tiedonvälitysprotokollien tai esimerkiksi salauksen myötä.

2.5 Pilvipohjaisen järjestelmän kustannustehokkuus

Pilvipalvelupohjaisen järjestelmän kustannustehokkuuden arvioinnissa käytetään yleensä vertailukohtana vastaavan järjestelmän on-premise-toteutuksen eri tekijöistä johtuvia kustannuksia.

On-premise-järjestelmän kustannusarvio koostuu tyypillisesti:

- infrastruktuurin laitteiston alustavista hankintahinnoista
- infrastruktuurin asennuksiin ja ylläpitoon käytettävistä henkilötyötunneista
- infrastruktuurin ylläpidon muista kustannuksista, kuten energiankulutuksesta

Pilvipalvelualustalla toimivan järjestelmän kustannukset koostuvat fyysisen ylläpidettävän laitteiston puuttuessa:

- palveluntarjoajalta varattavien ja käytettävien resurssien käytöstä
- palvelumallista riippuen järjestelmän ylläpidon henkilötyötunneista

Pilvipalvelupohjaisen järjestelmän kustannustehokkuuden arvioidaan yleisesti olevan parempi kuin on-premise-järjestelmän käytettävien infrastruktuuriresurssien joustavuuden ansiosta, mutta infrastruktuurin ja resurssien puutteellisesti suunniteltu käyttö johtaa silti usein tarpeettomasti kasvaviin kuluihin [12]. Pilvipalvelualustat kuten AWS tarjoavat erityyppisiin käyttötarkoituksiin useita vaihtoehtoisia ratkaisuja, joiden soveltuvuus kustannustehokkuuden näkökulmasta on arvioitava aina käyttötarkoitus- ja palvelukohtaisesti. Esimerkkinä taulukossa 1 vertailu AWS Relational Database Service (RDS) ja DynamoDB -tietokantapalveluiden välillä. RDS ja DynamoDB ovat molemmat datan säilytykseen ja jäsentelyyn käytettäviä palveluita, mutta niiden toteutus, laskutus ja sopivat käyttötarkoitukset eroavat merkittävästi.

Taulukko 1. RDS – DynamoDB -vertailu

Palvelu	RDS	DynamoDB
Kuvaus	instanssioitu relaatiotietokanta-alusta erilaisten SQL-kantojen ylläpitoon	palvelimeton NoSQL-tietokanta
Laskutus	kuukausittainen maksu käytössä olevista tietokantainstansseista	erilliset maksut tietokantaan tehtävistä luku- ja kirjoitusoperaatioista sekä tallennetusta datasta
Tyypilliset käyttötapaukset	tyypillisiin relaatiotietokantaa vaativiin toteutuksiin, kuten ERP- ja CRM-järjestelmiin	reaaliaikaisiin, nopeaa IO-suoritusta vaativiin web-sovellutuksiin kuten verkkokauppojen ostoskorien, käyttäjädatan ym. hallintaan

[13]

Käyttötarkoitukseen sopivan palvelun ja riittävän, muttei kohtuuttoman suuren resurssikapasiteetin valinnalla on suora vaikutus pilvialustan käytöstä aiheutuviin kustannuksiin, joten kustannustehokkaan järjestelmän suunnittelu vaatii käyttäjältä kokemusta ja asiantuntemusta valitun pilvialustan käytöstä, sillä väärän palvelun tai komponentin käyttö voi hyvinkin nopeasti johtaa suorituskyky- tai kustannusongelmiin, joiden korjaamiseen vaaditaan laajempia arkkitehtuurimuutoksia.

3. ESIMERKKIJÄRJESTELMÄN VAATIMUKSET

3.1 Järjestelmän korkean tason vaatimukset

Asiakasyrityksen toimialueen laajuuden ja yli 400-tuhannen sähkömittarin jatkuvan mittauksen vuoksi järjestelmän on kyettävä käsittelemään suuria määriä dataa kerralla. Osa käsiteltävästä datasta koostuu yksittäisistä, suuremmista kokonaisuuksista kuten uusien mittareiden massa-asennuksista tai koko tietokannan kaikkien mittareiden ja käyttöpaikkojen tietojen päivityksestä, joten järjestelmän on kyettävä reagoimaan muuttuviin taakkoihin nopeasti ja käsittelemään suuretkin datapurskeet kohtuullisessa ajassa. Sähkömittausdatan kannalta reaaliaikaisuus on eduksi, mutta oleellisinta on kuitenkin datan jatkuvuus, joten tehokkaalla skaalautumisella pyritään ehkäisemään erilaisista tukoksista johtuvia aukkoja tai epäsäännöllisyyksiä tallennetussa datassa. Muuttuvien datamäärien myötä skaalautumisessa on kuitenkin otettava huomioon eri prosessien perusteltu käsittelynopeus, koska suoritusprosessien määrän, kaistanleveyden ja muistin kasvattamisesta seuraa luonnollisesti lisäkuluja. Näin ollen jokaisen prosessin kohdalla on erikseen arvioitava, onko kyseinen prosessi kriittinen suoritusnopeuden suhteen, vai voidaanko käsiteltäviä tehtäviä esimerkiksi odotuttaa SQS-jonossa prosessin ruuhkautumisen uhalla.

Järjestelmän käsittelemä data sisältää sekä asiakasyrityksen yritys- ja yksityisasiakkaiden luottamuksellisia tietoja, että esimerkiksi laskutuksen ja toiminnan kannattavuuden arvioinnin kannalta oleellista dataa, joten datan ja toisaalta koko järjestelmän luotettavuus ja luottamuksellisuus on ensisijaisen tärkeää. Datan siirron, varastoinnin ja pääsynhallinnan on oltava hyvin suojattu ulkoisilta uhilta, kuten haittaohjelmilta ja tietomurroilta. Tiedonsiirron ja tietokantojen on kuitenkin myös oltava varmistettuja ja hyvin monitoroitu sisäisten virheiden varalta, jotta järjestelmän omista vioista johtuvalta datan menetykseltä ja sirpaloitumiselta voidaan välttyä.

3.2 Järjestelmän toiminnot

Järjestelmän suorittamat prosessit koostuvat mittariasennuksiin liittyvien työtilausten käsittelystä, asiakastietojärjestelmän päivittyvän datan koostamisesta, mittalaitteiden konfiguraatioiden määritysten ja lähetysten ylläpidosta, ja useista eri ulkoisten järjestelmien ja palveluiden kanssa kommunikoinnista edellä mainittujen prosessien tai esimerkiksi käyttöliittymän datan tarkastelun yhteydessä. Yksittäiset prosessit on kuvattu tarkemmin luvussa 4 teknisen toteutuksen yhteydessä.

4. ESIMERKKIJÄRJESTELMÄN TEKNINEN TOTEUTUS

Työ toteutettiin asiakasprojektina energia-alalla toimivalle yritykselle yhteistyössä asiakkaan oman projektiryhmän sekä muiden alihankkijoiden kanssa. Työn tavoitteena oli suunnitella ja toteuttaa järjestelmä, joka käsittelee sähkönmittausdataa, mittariasennuksia, työtilauksia, sekä monia muita asiakasyrityksen toimenkuvaan kuuluvia toimintoja. Järjestelmä toteutettiin AWS-pohjaisena mikropalvelujärjestelmänä, jossa datan kulkua eri palveluiden välillä ohjaa keskeinen EventBridge-palvelu eli järjestelmän ”keskuskeittiö”-tapahtumaväylä.

4.1 Stackit ja mikropalvelut

Koko järjestelmän päätoiminnallisuus koostuu neljästä stack-kokonaisuudesta, joiden vastuualueet on jaettu seuraavasti:

MeteringPoint-stack:

- Käyttöpaikkojen ja mittarien, sekä niihin liittyvien tietojen ja tapahtumien hallinta. Tämän kokonaisuuden tehtävä on lukea EnerimCIS hallintajärjestelmästä tallennettavia synkronointitietoja käyttöpaikoille ja mittareille, ja tallentaa ne yhdessä mittariasennuksiin liittyviltä työtilauksilta sekä järjestelmän käyttöliittymältä tulevan datan kanssa koostetuiksi kokonaisuuksiksi DynamoDB-tietokantaan. MeteringPoint-stack osallistuu myös työtilausprosessien hallintaan tiettyjen ulkoisten järjestelmien kanssa viestimällä, mikä johtuu vain MeteringPointilta löytyvästä prosessille oleellisesta datasta. Skaalautuvuuden näkökulmasta MeteringPoint-stackin suurin mahdollinen pullonkaula on edellä mainittu EnerimCIS-järjestelmän tietojen synkronointi, joka saattaa kerralla käsittää joko yksittäisten käyttöpaikkojen ja mittarien tietojen päivityksiä tai sitten koko käyttöpaikkakannan päivityksen. CIS-synkronointiin liittyviä prosesseja on käsitelty tarkemmin luvussa 3.3.1.

WorkOrder-stack:

- Työtilausten hallinta, joka käsittelee eri lähteistä tulevia työtilauksiin liittyviä tietoja, kuten työtilausten vastaanottamisviestejä, mittariasennusten aloitus- ja valmistumisviestejä, ym. WorkOrder-stack vastaanottaa siis dataa ulkoapäin, suorittaa tarvittavia toimenpiteitä, tallentaa työtilausten dataa kantaan ja välittää sitä

tarvittaessa muille stackeille, kuten MeteringPoint-stackille. Työtilausten datamäärä on yksittäisten työtilausten osalta pieni, mutta järjestelmän on toisinaan käsiteltävä laajoja ”massatyötilauksia”, jotka käynnistävät samanaikaisesti työtilausprosesseja satojen työtilausrivien osalta. Niin yksittäisten kuin massatyötilaustenkin tilausrivien eri käsittelyvaiheiden työn aloitus- ja valmistumiskuittaukset voivat saapua järjestelmän käsiteltäväksi vaihtelevissa järjestyksissä ja aikoina, joten yksittäisten prosessien ja niiden tilojen hallinta vaatii useita toisistaan riippumattomia aliprosesseja ja kattavaa virheidenhallintaa, jotta myös paljon manuaalista työtä (asentajat) sisältävien työtilausten käsittely pysyy jatkuvasti ja luotettavasti käynnissä kaikissa tilanteissa.

MeterConfiguration-stack:

- Mittarikonfiguraatioiden hallinta, jonka tehtävä käytännössä on ylläpitää tietoja käyttöpaikoilla olevien mittarien ohjelmistokonfiguraatioista ja lähettää varsinaiset konfiguraatiopäivitykset mittareille. Konfiguraatioiden lähetys voidaan suorittaa samanaikaisesti useille mittareille (lähetyserä saattaa kattaa esimerkiksi kolmen mittarikonfiguraation lähetyksestä sadalle mittarille), joten yksittäisten päivitysprosessien suoritukseen ja ylläpitoon tarvitaan joustavaa infrastruktuuria.

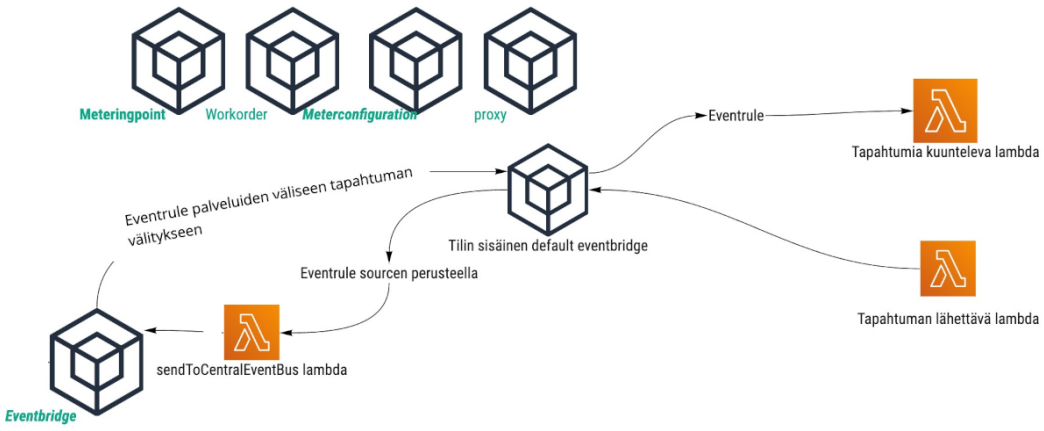
Proxy-stack

- Järjestelmän ulkoisten palveluiden tapahtumien välityksestä vastaavan Proxy-stackin tehtävä on nimensä mukaisesti toimia välikätenä esimerkiksi asiakasyrityksen mittarikumppanin järjestelmistä tulevan datan tuomisesta muiden alitilien käsiteltäväksi, ja vastaavasti esimerkiksi työtilausprosesseihin liittyvien mittari-käyttöpaikka-linkitysten lähetyksestä ulos päin mittarikumppanille. Mittareilta tulevan aikasarjadatan tallennus ja välitys eteenpäin on näin ollen myös Proxy-stackin vastuulla, joten tiedonsiirron jatkuvuus vaikuttaa oleellisesti myös asiakasyrityksen laskutukseen ja sitä myötä tavallisiin kuluttajiin.

Näiden neljän oleellisimman stackin lisäksi järjestelmästä löytyy omia stack-kokonaisuuksia erilaisille apu- ja ohjauspalveluille esimerkiksi pääsynhallintaan ja datan monitorointiin liittyen, mutta järjestelmän käyttötarkoituksen keskeisimmät tehtävät jakautuvat kuitenkin edellä listatuille kokonaisuuksille.

4.2 Tapahtuma-arkkitehtuuri

Suurin osa järjestelmän sisäisestä kommunikaatiosta tapahtuu keskitetyn, järjestelmän ”keskuskeittiöksi” nimetyn EventBridge-tapahtumaväyläkomponentin kautta eri palveluista lähetettävien tapahtumien avulla. Tapahtumapohjaisen arkkitehtuurin kantavana periaatteena on toteuttaa itsenäisiä palveluita, joiden toiminta voidaan mahdollistaa myös muiden liittyvien palveluiden ollessa poissa käytöstä tai saavuttamattomissa. Kuvassa 5 on esitelty järjestelmän tapahtumaviestintää yleisellä tasolla. Alitilien lambda-palveluista lähetettävät tapahtumat kulkevat ensin niiden oman alitilin oletus-EventBridgeen (kuvassa keskellä, ”Tilin sisäinen default eventbridge”), joka välittää ne edelleen keskuskeittiö-EventBridgelle erillisen välitys-lambdan avulla. Keskuskeittiöltä (kuvassa vasemmalla alhaalla) viestit välitetään alitilien oletus-EventBridgeille, joista ne vastaavasti ohjautuvat lopulta niitä kuunteleville Lambda-palveluille tai viestijonoihin. Kokonaisuus noudattaa tapahtumapohjaista arkkitehtuuria, jossa viestejä lähettävät osapuolet eivät ota mitään kantaa siihen, mitkä tilit ja niiden palvelut viestejä vastaanottavat, vaan viestien päätymien oikeille vastaanottajille riippuu eri EventBridge-komponenteille asetetuille säännöille (Event Rule). Alitilien EventBridge-komponentit ovat edellytys monen eri AWS tilin väliselle (cross-account) tapahtumien viestinnälle, sillä keskuskeittiöltä ei ole mahdollista ohjata tapahtumia suoraan alitilien kuuntelijoille EventBridge-palvelun rajoitusten vuoksi.



Kuva 5. *”Keskuskeittiö”-tapahtumaväylä ja tapahtuma-arkkitehtuuri*

Asynkronisesti toimiva tapahtuma-arkkitehtuuri tekee järjestelmän toiminnasta joustavampaa ja eri komponenttien kehityksestä toisistaan paremmin riippumatonta, mutta toisaalta järjestelmän end-to-end (E2E) -testaus ja prosessien seuranta kokonaisten suoritusketjujen osalta vaikeutuu. Tapahtumapohjaisen arkkitehtuurin etuihin lukeutuu myös järjestelmän vaivaton skaalautuminen mikropalveluinstanssien määrää kasvattamalla. Jotkin järjestelmän yhteyksistä, esimerkiksi kaikki käyttöliittymän ja muun järjestelmän välinen liikenne, eivät käytettävyyden kannalta voi toimia asynkronisesti. AWS EventBridge-palvelu rajoittaa myös lähetettyjen viestien koon 256 kilotavuun, joten suurten datalohkojen lähettäminen suoraan ei välttämättä ole mahdollista. EventBridgen alueellinen rajoitus lähettävien viestien määrälle on asiakasjärjestelmän alueella 10000 viestiä sekunnissa, mikä ei järjestelmän kannalta aiheuta ongelmia tällä hetkellä, mutta saattaa kuitenkin vaikuttaa järjestelmän skaalautumiseen rajoittavasti.

4.3 Lambda-pohjaiset datan käsittelijät

Suurin osa datan käsittelystä järjestelmän sisällä tapahtuu AWS Lambda-palvelun funktioiden avulla. Lambdat ovat palvelimettomia (serverless) ohjelmaprosesseja, joita voidaan suorittaa aluekohtaisesti rinnakkain jopa 1000 kappaletta. Yhden Lambda-prosessin ajoa varten käynnistetään aina uusi ajoympäristö, esimerkiksi Node.js, joka sitten poistetaan prosessin suorituksen päätyttyä. Lambdoille on CDK:n (Cloud Development Kit, käsitellään luvussa 4.5 - Työkalut) avulla määritetty erilliset herätteet, syötteet ja oikeudet sen mukaan, mikä kyseisen käsittelijän käyttötarkoitus on. Esimerkkiprosessina käyttöpaikka- ja mittaritietojen tuonti EnerimCIS-palvelusta, ja siitä käynnistyvien kahden Lambdan määritykset:

```
// Upload-lambda
const uploadProcessor = new lambda.Function(this, "uploadProcessor",
{
   runtime: lambda.Runtime.NODEJS_12_X,         // ajoympäristö
   code: lambda.Code.fromAsset('bin/upload'),   // lähdekoodin sijainti
   handler: 'uploadProcessor.handler',
   timeout: cdk.Duration.minutes(15),           // max ajoaika
   tracing: lambda.Tracing.ACTIVE,              // tracing monitoroinnille
   environment: {                               // ympäristömuuttujat
      meterSqs: setUploadCompleteMeterSqs.queueUrl,
      ...rawDataTables
   },
   memorySize: 2048                             // lambdalle varattu muisti
});

// Upload-lambdalle lukuoikeudet käyttöpaikkadatan S3-buckettiin
incomingS3.grantRead(uploadProcessor);

// Asetetaan Upload-lambda kuuntelemaan S3-bucketin tapahtumia
S3EventTopic.addSubscription(new subs.LambdaSubscription(uploadProcessor));
```

```
// Upload-lambdalle lähetysoikeus ”käsittely valmis” -SQS-jonoon
setUploadCompleteSqs.grantSendMessages(uploadProcessor);

// Merge-lambda
const mergeProcessor = new lambda.Function(this, "mergeProcessor", {
   runtime: lambda.Runtime.NODEJS_12_X,
   code: lambda.Code.fromAsset('bin/merge'),
   handler: 'meterMergeProcessor.handler',
   timeout: cdk.Duration.minutes(15),
   tracing: lambda.Tracing.ACTIVE,
   environment: {
     ...rawDataTables,
     processedBatches: processedBatchesTable.tableName,
     meterData: meterDataTable.tableName,
   },
   layers: [meteringPointLambdaLayer],
   reservedConcurrentExecutions: 1,
   memorySize: 2048
});

// Asetetaan Merge-lambda kuuntelemaan ”käsittely valmis” -SQS-jonoa
mergeProcessor.addEventSource(
   new lambdaEvents.SqsEventSource(setUploadCompleteSqs, {
      batchSize: 1
   })
);
```

Ensimmäinen Lambda-käsittelijä (*Upload*) saa herätteenään tiedon määritetyn S3-bucketin (*incomingS3*) kirjoitustapahtumasta, jonka seurauksena käsittelijä lukee S3:een kirjoitetut csv-muotoiset tiedostot ja syöttää niitä vastaavat rivit raakadata-tauluihin (*rawDataTables*) DynamoDB-tietokantaan. Lopuksi käsittelijä lähettää viestin raakadatan latauksen valmistumisesta SQS-palveluun(setUploadCompleteSqs). Tätä prosessia varten Lambda-käsittelijälle on CDK:lla määritetty S3-heräte, oikeudet lukea tiedostoja määritetystä S3-bucketista, oikeudet kirjoittaa dataa määriteltyihin DynamoDB-tauluihin ja oikeus lisätä viestejä määritettyyn SQS-jonoon.

Tuontidatan seuraava käsittelijä (Merge) käyttää herätteenään ensimmäisen käsittelijän lähettämiä valmistumisviestejä, joiden perusteella käsittelijä lukee tietyn raakadatapaketin sisällön raakadata-tauluista, tekee tarvittavat muutokset ja koostaa datasta päivityksiä DynamoDB:ssä sijaitseviin käyttöpaikkojen ja mittarien master-tietokantoihin. Näitä toimintoja varten Lambdalle on määritetty oikeudet lukea viestejä määritetystä SQS-jonosta, oikeus lukea dataa DynamoDB-raakadatatauluista, ja täydet käyttöoikeudet (luku + kirjoitus) käyttöpaikkojen sekä mittarien master-tietotauluihin.

Esimerkkinä käytettyjen funktioiden ja muiden prosessiin liittyvien resurssien määrittelyn kannalta oleellista on se, että käsiteltävää EnerimCIS-muutoksista johtuvaa dataa saattaa kerralla tulla paljon, joten funktioille on pyritty varaamaan riittävä määrä muistia (2048

MB). Muutosten käsittelyn on lisäksi tapahduttava FIFO-periaatteen mukaisessa, oikeassa järjestyksessä, joten kunkin käsittelyyn käsittelijä-Lambdoja saa kerrallaan olla ajossa vain yksi kappale mahdollisten virheiden varalta. Rinnakkaisten suorituksien avulla käsittely olisi tehokkaampaa, mutta tämän prosessin osalta suorituskykyä oleellisempaa on datan eheys. Lambdalle varatun muistin määrä on arvioitu suurimman mahdollisen datamäärän käsittelyyn tarvittavan muistin mukaisesti, mikä vaikuttaa prosessin suorituskykyyn myönteisesti. AWS Lambda-prosessien hinnoittelu riippuu sekä prosessille varatun muistin että suoritukseen kuluvan ajan mukaisesti, joten kustannustehokkuuden kannalta ylimitoitetusta muistikapasiteetista ei aiheudu ongelmia – tiettyyn pisteeseen asti. Muistin kasvaessa prosessien suoritusajat lyhenevät, jolloin prosessien kustannukset pysyvät keskimäärin samoina, kuin pienemmällä muistikapasiteetilla. Prosessien suoritusnopeus ei voi kuitenkaan ylimääräisen muistin myötä kasvaa rajattomasti, jolloin edellä mainitun ”tietyn pisteen” ylittyessä lisämuisti ei enää tuota nopeampia suoritusaikoja, mutta kasvattaa silti suorituksen hintaa varatun, ylimääräisen muistikapasiteetin myötä.

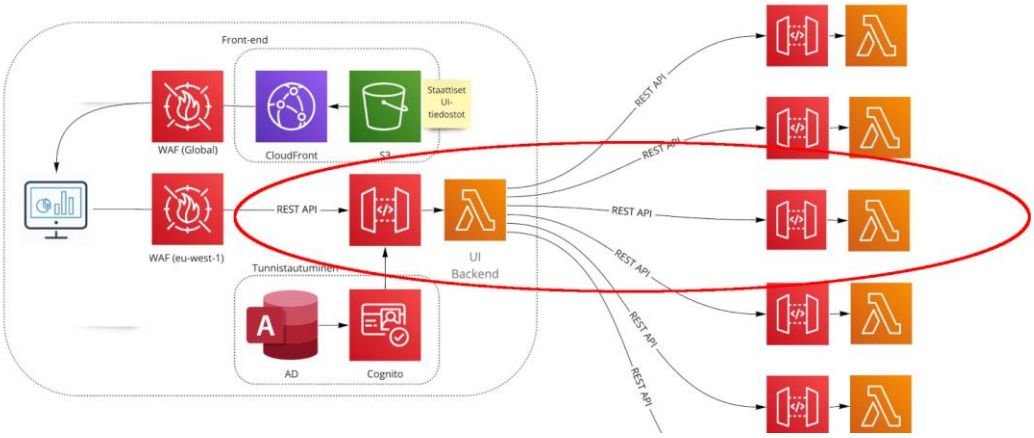
Esimerkkeinä käytetyn prosessin Lambdojen lähdekoodeissa käsiteltävän datan määrä on otettu huomioon muodostamalla käsittelystä usean samanaikaisen säikeen suoritusvirran, jotta kerralla tarvittavan välimuistin määrä sekä toisaalta suoritusaika pysyvät riittävän matalalla tasolla. Seuraavassa käyttöpaikkadatan päivitysprosessin otteessa käytetään Highland.js-streamkirjastoa päivitettävien käyttöpaikkojen käsittelyn jatkuvana datavirtana kerralla tarvittavan muistikapasiteetin vähentämiseksi. Prosessissa on lisäksi otettu huomioon DynamoDB-palvelun rajoitukset samanaikaisille luku- ja kirjoitusoperaatioille (vaiheet 3-5).

```
   // 1. Muunnetaan käyttöpaikkaId-lista stream-muotoon
      hl(<rawMeteringPointIds>)
   // 2. Käsitellään yksittäisen käyttöpaikan päivityskooste
      .map((rawMeteringPointId: any )=> hl(handleMergePremise(rawMeteringPoin-
tId, batchId, rowNo++)))
   // 3. Suoritetaan 2. vaihetta samanaikasesti 24 kp:lle rinnakkain
      .parallel(24)
   // 4. Koostetaan 2. ja 3. tulokset 24 rivin ryhmiin
      .batch(24)
   // 5. Suoritetaan päivitysten DynamoDB-kirjoitus
      .map((meteringPointArray: any) => hl(handleWriteBatch(meteringPointAr-
ray)))
   // 6. Suoritetaan 5. vaihe viidellä ryhmällä rinnakkain
      .parallel(5)
   // Tulosten lokitus
      .each((data: any) => {
        console.log("Updated", data.amount, "metering points")
      }
```

Muiden prosessien, kuten työtilausten käsittelyn, eri vaiheiden suoritusjärjestykselle ei ole asetettu erityisiä vaatimuksia eikä niihin liittyvien lambda-prosessorien tarvitse kerralla lukea suurta määrää dataa, joten muuttuviin datamääriin skaalautuminen onnistuu useiden lambda-prosessien samanaikaisen ajon ja käsittelijöiden osittamisen avulla.

4.4 REST-rajapinnat

Erilaisia REST-rajapintoja käytetään osana järjestelmän sisäistä viestintää, esimerkiksi jonkin mikropalvelutilin tarvitessa suuremman määrän dataa toisen tilin resursseista, kuin olisi EventBridgen välityksellä mahdollista lähettää tai kun kysely tehdään synkronisesti. Esimerkkinä järjestelmän webkäyttöliittymän backend (kuva 6), joka koostuu ApiGateway-rajapinnasta ja siihen tehtäviä kutsuja käsittelevästä Lambda-prosessorista. Muut REST-rajapinnat koostuvat samanlaisista ApiGateway + Lambda -yhdistelmistä. ApiGatewaylle tehtävä kutsu käynnistää aina uuden Lambda-prosessin, joten rajapinnat skaalautuvat tarvittaessa tehokkaasti rinnakkaisten suoritusten avulla.



Kuva 6. *UI Backend REST toteutus*

Tyypillisessä käyttötapauksessa selaimelta lähetetään pyyntö UI-backendille, joka puolestaan kutsuu muiden tilien (MeteringPoint, WorkOrder, …) vastaavia REST-rajapintoja, yhdistelee ja käsittelee saadut vastaukset ja palauttaa koostetun vastauksen selaimelle. Dataa siirtyy potentiaalisesti kerralla suuri määrä ja sen on palauduttava käyttäjälle synkronisesti, joten REST-rajapinnan käyttö on perusteltua. Vaihtoehtoisesti selaimelta voitaisiin kutsua suoraan muiden tilien REST-rajapintoja, mutta nykyinen malli mahdollistaa varmemman pääsynhallinnan rajapintojen välillä, koska muiden tilien rajapinnat voidaan

sulkea kokonaan järjestelmän ulkopuolisilta kutsuilta, jättäen ainoaksi ulospäin näkyväksi rajapinnaksi UI-backendin, jolloin mahdolliset palomuurimääritykset ja muut suojaustoimenpiteet ulkoisia uhkia vastaan on tarpeen toteuttaa vain yhdellä rajapinnalla.

4.5 Työkalut

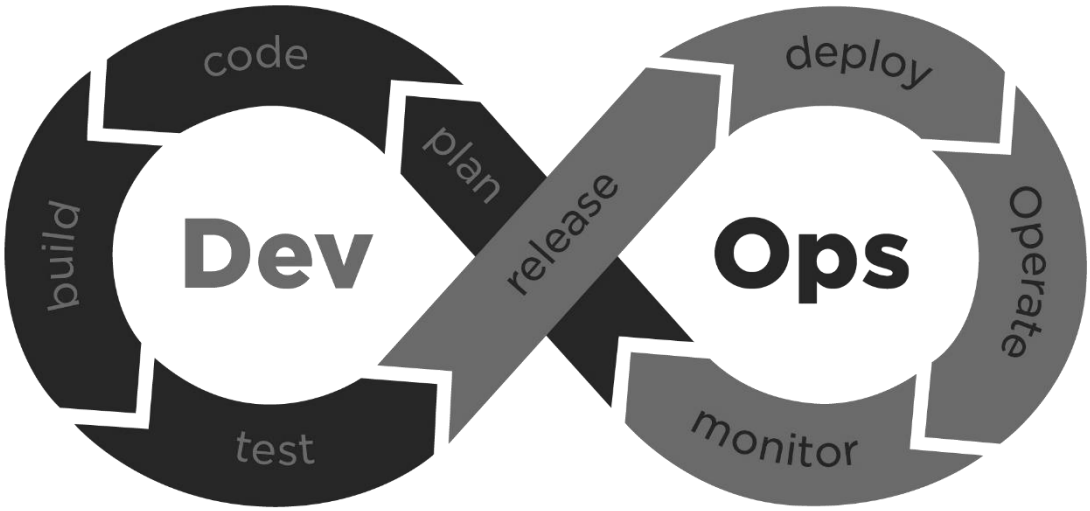
Järjestelmän kehitys on jaettu toisistaan pääasiassa riippumattomiin AWS Cloud Development Kit (CDK) -stackeihin, joiden julkaisu eri ympäristöihin tehdään Gitlab CI-putkien kautta. CDK tuottaa lopulta CloudFormation-mallin, jonka mukaan eri AWS resurssit varataan ja määritellään, mutta CDK:n avulla mallien kehitys on joustavampaa ja tehokkaampaa kuin suoraan CloudFormation-malleja käsittelemällä, koska varattavien resurssien eri ominaisuuksien ja erityisesti niiden välisten riippuvuuksien ja luparakenteiden määrittäminen hoituu suurelta osin automatisoidusti ”pellin alla”. Automatisoinnin haittapuolena rakenteiden määrittämisen vapaus rajoittuu CDK:n sisäisen kehityksen eli projektista riippumattomien toimijoiden tekemiin rakennevalintoihin, jotka ovat kuitenkin toistaiseksi osoittautuneet tämän järjestelmän kehitykseen sopiviksi. Resurssien määrittämiseen CDK:lla käytetään valitulle kehitysohjelmointikielelle ominaisia rakenteita ja syntaksia, mikä yhtenäistää kehitystyötä ja kehittäjiltä vaadittua tietotaitoa. Jakamalla järjestelmä erillisiin stackeihin jakautuvat samalla myös järjestelmän vastuut ja palveluiden käyttöoikeudet paremmin hallittaviksi kokonaisuuksiksi.

Ensisijaiseksi kehityskieleksi järjestelmälle valikoitui Typescript ja ajoympäristöksi Node.js pääasiassa projektin kehittäjien aiemman kokemuksen perusteella. Typescript mahdollistaa tyypitetyn Javascript-koodin kirjoittamisen ja Node.js on ympäristönä alustariippumaton. Molempien käyttö lisäksi on AWS SDK:n ja CDK:n puolesta vahvasti tuettua ja hyvin dokumentoitua, joten ne sopivat lähes kaikkiin järjestelmän mikropalvelutarpeisiin. Poikkeuksellisesti AWS Glue-palvelun eli AWS:n tarjoaman ETL-alustan oletuskirjastot ovat toteutettu vain Python ja Scala -ohjelmointikielille, joten yksittäisiä, suuria tietokantojen käsittelyoperaatioita varten tehtyjä lähdekoodeja on toteutettu myös Pythonilla/Scalalla.

Kolmansien osapuolien framework-puitekirjastojen valinnassa oleellisimpia huomioon otettavia asioita ovat tietoturva ja lisensointi. Ulkoisten riippuvuuksien valinta noudattaa samoja tietoturvaperiaatteita kuin järjestelmä sisäisesti, ja moduulien sekä niiden omien riippuvuuksien haavoittuvuudet on pyritty kartoittamaan mahdollisimman laajasti. Lisenssien osalta järjestelmässä käytetään vain avoimen lähdekoodin täysin vapaasti käytettävissä olevia riippuvuuksia (MIT, Apache, BSD), koska järjestelmän lähdekoodista ei haluta tehdä avointa, kuten esimerkiksi GPL-lisensoidut riippuvuudet edellyttäisivät. [14]

4.6 DevOps

Järjestelmän kehityksessä on pyritty noudattamaan hyviä DevOps-periaatteita jatkuvan integroinnin ja jakelun (CI/CD) sekä järjestelmän jatkuvan monitoroinnin avulla. Näiden toimintaperiaatteiden tavoitteena on potentiaalisiin ongelmiin nopeampi puuttuminen ja kehityksen yleinen joustavuus ja kustannustehokkuus.



Kuva 7. *DevOps vaiheet, lähde: Pease 2017*

Continuous Integration (CD) ja Continuous Deployment (CD) eli jatkuva integrointi ja julkaisu on järjestelmän kehityksessä toteutettu Gitlab CI:n ja CDK:n kirjastojen avulla ja kattaa DevOps-kehityskaaren (kuva 7) ”code”, ”build”, ”test”, ”release” ja ”deploy” vaiheet.

Jokaiselle kehitystilille on luotu oma Gitlab-repositorio, joka sisältää automatisoidut rakenteet koodikannan rakentamiseen, E2E- ja yksikkötestaukseen ja lopulta kohdeympäristöön julkaisuun:

```
stages:
  - build
  - deploy

cache:
  paths:
    - .npm/

before_script:
  - ci/init

build:
```

```
    stage: build
    script:
      - ci/build
    rules:
      - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
        when: always
      - if: '$CI_COMMIT_BRANCH =~ /^(master|develop|sandbox)$/'
        when: always
      - when: never

  deploy:
    variables:
      STAGE: $CI_COMMIT_BRANCH
    stage: deploy
    script:
      - ci/build
      - ci/deploy
    rules:
      - if: '$CI_COMMIT_BRANCH =~ /^(master|develop)$/'
        when: on_success
      - if: '$CI_COMMIT_BRANCH == "sandbox"'
        when: manual
      - when: never
```

Julkaisua varten jokaiselle kehitystilille (stack-toteutukset) on määritetty kolme AWS ympäristöä: sandbox, QA ja production. Sandbox-ympäristö toimii kehittäjien ensisijaisena testiympäristönä, joka on irrallaan järjestelmän ulkoisista palveluista ja jossa muutoksia ja komponentteja voi testata täysin vapaasti. Sandbox-testauksen jälkeen ohjelmistoversio julkaistaan QA-ympäristöön (develop), jonka pääasiallinen käyttötarkoitus on toimia asiakasyrityksen testiympäristönä ennen lopullista hyväksymistä tuotantoon (production), joka on nimensä mukaisesti järjestelmän lopullinen käyttöympäristö. Gitlab-repositorion haarat on määritetty siten, että sandbox-nimiseen haaraan yhdistäessä käynnistyy automaattisesti julkaisu tiliä vastaavaan AWS sandbox-ympäristöön. Repositorion develop- ja production-haarat toimivat vastaavasti QA- ja tuotantoympäristöjen suhteen.

Järjestelmän monitoroinnin (kuva 5, vaihe ”monitor”) keskeisimpänä tekijänä on DataDog-toteutus, johon kerätään lokitietoja ja metriikkaa kaikista järjestelmän osa-alueista. DataDog valittiin järjestelmän monitorointipalveluksi ensisijaisesti sen kattavien integrointimahdollisuuksien vuoksi. Monitoroinnin kannalta oleellista dataa syntyy esimerkiksi Lambda-prosessorien suorituslokeista, SQS-jonojen liikenteestä ja DLQ-jonoihin päätyneistä viesteistä sekä järjestelmän sisäisten ja ulkoisten yhteyksien virheistä. Eri metriikoiden avulla voidaan lisäksi seurata esimerkiksi eri prosessien suoritusaikoja, suoritusajankohtia ja käsiteltyjen pyyntöjen määrää, eli käytännössä kaikkia suorituskykyyn ja kustannuksiin vaikuttavia tekijöitä. DataDog-toteutuksen monitoreille on myös määritetty

automatisoituja hälytyksiä asianomaisille (ts. kehitystiimi tai asiakkaan oma henkilökunta) esimerkiksi tiettyjen parametrien kuten lambdan suoritusvirheiden ylittäessä sallitun määrän.

DataDog-metriikkaa varten eri resurssien CDK-määrityksiin on lisätty DataDogin luennan mahdollistavia komponentteja, joista esimerkkinä Lambdojen monitoroinnin mahdollistavat rakenteet liitteessä 1.

Jatkuvan monitoroinnin avulla järjestelmän virheisiin ja mahdollisiin pullonkauloihin tai muihin suorituskykyyn ja kustannuksiin vaikuttaviin tekijöihin puuttuminen helpottuu ja aikaistuu, mistä on suurta hyötyä pitkällä aikavälillä.

4.7 Tietoturva

Datan eheyden ja arkaluontoisen datan luottamuksellisuuden vuoksi onnistuneesti suunniteltu ja toteutettu tietoturvakokonaisuus on luonnollisesti ensisijaisen tärkeä osa projektikehitystä. Tietoturvaominaisuuksien suunnittelussa ja toteutuksessa luotetaan AWS-alustan tarjoamiin komponentteihin mahdollisten omien toteutuksien sijaan. Amazonin tietoturvarakenteet ovat hyvin kehittyneellä tasolla, ja mahdollistavat eri komponenttien ja niiden määritysten laajan kustomoinnin järjestelmän tarpeiden mukaan.

Järjestelmän keskeisimpänä pääsynhallintakomponenttina toimii AWS Identity and Access Management (IAM), joka autentikoi järjestelmän sisäiset yhteydet, kehittäjien ja muun vastuuhenkilökunnan pääsyn AWS ympäristöihin, ja automaattiset CI/CD-prosessit. IAM-palvelukomponentti löytyy jokaiselta järjestelmään kuuluvalta tililtä, ja jokaiselle tilille on määritelty erikseen pääsynhallinta käyttäjille ja rooleille ja näiden tilienvälisille yhteyksille. Rajaamalla järjestelmän sisäisten REST-kutsujen pääsynhallinnan tileille määritettyjen roolien IAM-tunnusten mukaan voidaan tehokkaasti ehkäistä ulkoisten uhkien pääsy rajapinnoille sekä yhtenäistää järjestelmäkokonaisuuden tietoturvakäytäntöjä.

Käyttöliittymän ja sen backend-rajapinnan käyttäjien autentikointi on toteutettu AWS:n käyttäjä- ja pääsynhallintapalvelun Cogniton sekä Microsoft Azuren käyttäjähallintapalvelun Active Directoryn (AD) avulla. AWS Cognito-komponentille on määritelty käyttäjäpool, jonka sisällöstä ja varsinaisesta käyttäjien autentikoinnista vastaa kuitenkin loppupäässä asiakkaan tarjoama Azure AD toteutus. Järjestelmän käyttöliittymä ottaa Cogniton kautta yhteyttä Azure-palveluun, jossa käyttäjä kirjautuu AD tunnuksillaan, ja josta lopulta palautuu käyttöliittymälle autentikointivastaus, joka sisältää backend-kutsujen tekemiseen tarvittavat session tokenit ja käyttäjälle määritetyt roolit ja ryhmät eri toimintojen ja oikeuksien rajaamista varten.

Eri käyttäjäryhmien oikeudet käyttöliittymässä ja sen rajapinnoilla on rajattu asiakkaan määrittämien käyttötarpeiden ja henkilökunnan roolien mukaisesti. Tämä helpottaa käyttäjävirheistä johtuvien ongelmien selvitystä ja ehkäisee samalla mahdollisia väärinkäytöksiä ja GDPR-rikkeitä jakamalla tietojen ja toimintojen käyttöoikeudet sen mukaan, mitkä roolit niitä tosiasiassa tarvitsevat.

Cognito-toteutuksen lisäksi käyttöliittymälle ja REST-rajapinnoille on asetettu AWS Web Application Firewall (WAF) -palvelu, jolle AWS Core Rule Setin (CRS) lisäksi on määritetty IP-osoitteiden suodatukseen liittyvä säännöstö, joka estää kaikki sallittujen IP-osoitteiden listaan kuulumattomien lähteiden tekemät kutsut REST-rajapintoihin sekä pääsyn AWS:n jakelupalvelu Cloudfrontin kautta tarjottavaan käyttöliittymään.

Pääsynhallinta WAF-palvelulla määritetään luomalla Web ACL (Access Control List), jolle listataan eri tilanteiden säännöstöt ja oletustoimenpiteet. Esimerkkinä käyttöliittymän Cloudfront-jakeluun liitetty ACL JSON-muodossa liitteessä 2. Käytetyn säännöstön periaate on oletusarvoisesti estää kaikki erikseen määriteltyjen sallittujen tapausten ulkopuolinen liikenne ja toiminnot.

Amazonin hallinnoima Core Rule Set on Amazonin mukaan [15] määritelty kattamaan mahdollisimman laajasti erilaiset tunnetut tietoturvauhat, kuten OWASP:n top 10 -listauksen [16] mukaiset kriittisimmät web-applikaatioita koskevat riskit. Vaikka kolmannen osapuolen (AWS) tekemän määrityksen osalta ei luonnollisesti pystytä täysin varmistamaan tietoturvan riittävyyttä kaikkien muuttuvien tietoturvauhkien osalta, järjestelmän kehityksessä on päädytty luottamaan Amazonin toimiin pilvipalvelualustan tarjoajana.

Esimerkkijärjestelmän tietoturvatestaus suoritettiin toisen alihankkijan toimesta, eikä siihen perehdytä tämän työn osalta tarkemmin.

5. ESIMERKKIJÄRJESTELMÄN ARVIOINTI

Järjestelmän ensimmäinen korkean tason tavoite oli kyky käsitellä muuttuvia, toisinaan suuriakin datamääriä joustavasti ja järkevästi skaalautuen. Järjestelmän skaalautuvuuden ja suorituskyvyn voidaan prosessikohtaisiin vaatimuksiin nähden todeta olevan toistaiseksi riittävällä tasolla, mutta lopullisia, koko tuotantodatamäärän huomioon ottavia arvioita skaalautuvuudesta ei voida nykyisellään perustellusti esittää. AWS alusta ja sen tarjoamat palvelut kuten Lambda-funktiot mahdollistavat prosessien skaalautumisen testiympäristöjen ja alustavan, osittaisen tuotannon tarpeen mukaan riittävän tehokkaasti ja useissa tapauksissa jopa oletusmäärityksillä. Järjestelmän suorituskykytestaus on kehityksen aikana jäänyt kuitenkin valitettavan kevyeksi, eikä kaikkien prosessien vaatimia todellisia resurssitarpeita ole saatu selvitettyä ennen ongelmien ilmaantumista. Suorituskyvyn kehittäminen on näin ollen ollut pääasiassa reaktiivista ongelmien ratkomista puutteellisten vaatimusmääritysten ja virheellisesti tehtyjen arvioiden vuoksi. Suurten datamäärien ja järjestelmän skaalautumisen kannalta perusteellinen suorituskykytestaus ja sen myötä tarvittavien resurssien arviointi on ensisijaisen tärkeää. Puutteellisen suorituskykytestauksen vuoksi järjestelmän kustannustehokkuutta on siis myös nykyisellään vaikea arvioida.

Datan eheyden kannalta suuren datamäärän ylläpitäminen ja käsittely usean alitilin ja mikropalvelun avulla samanaikaisesti on osoittautunut paikoin haastavaksi. Monen prosessin muokatessa samaa dataa on otettava huomioon, mitkä osat datasta on sallittua kirjoittaa yli ja mitkä datalohkot ovat jonkin muun prosessin vastuulla. Järjestelmän kehityksen aikana on jouduttu turvautumaan kokonaisten tietokantataulujen korjaamisprosesseihin datan eheyden palauttamiseksi virheellisesti toimineiden käsittelyprosessien aiheuttaman tietojen menetyksen seurauksena.

Mikropalveluarkkitehtuurin periaatteiden mukaisesti alitileillä ja niiden sisältämillä palveluilla on omat, toisistaan riippumattomat vastuut ja kehityskaaret. Käytännössä tämän periaatteen toteuttaminen ei kuitenkaan jokaisen prosessin osalta ole onnistunut täydellisesti. Esimerkiksi luvussa 4.1 WorkOrder-stackin yhteydessä esitelty työtilausprosessi, johon kuuluu työtilausdatan vastaanottamista, käsittelyä ja lähettämistä eri palveluille niin järjestelmän sisäisesti, kuin myös ulkoisiin järjestelmiin, kuten asiakasyrityksen mittarikumppanin asennusjärjestelmiin, on aiheuttanut vastuiden ristikkäisyyttä stackien välillä. Työtilausprosessin eri vaiheissa tarvitaan dataa sekä työtilaus-stackin että käyttöpaikka-stackin sisältämistä tietokannoista, mikä johtaa järjestelmästä ulospäin tapahtuvan viestinnän vastuiden jakautumiseen näiden stackien välillä, mikä taas johtaa siihen,

että joidenkin osaprosessien osalta käyttöpaikka-stack sisältää työtilauksiin liittyvää dataa, josta työtilaus-stack ei ole lainkaan tietoinen. Nimensä mukaisesti työtilaus-stackin on tarkoitus olla työtilauksiin liittyvissä asioissa ”master”-tietolähde, mutta todellisuudessa esimerkiksi web-käyttöliittymälle tarvittavan työtilausdatan tuomiseen soveltuu lopulta paremmin käyttöpaikka-stackin rajapinnat työtilaus-stackin sijaan.

Järjestelmän kehitystä jatketaan uusien ja alati muuttuvien vaatimusten lisäksi erityisesti eri prosessien seurantaa ja lokitusta parantamalla, jotta kehityksen aikana huomattuja puutteita ja suorituskykyä voidaan korjata saadun datan perusteella. Projektin aikana hankitun tiedon ja ammattitaidon myötä järjestelmän prosesseja voidaan myös kehittää edelleen paremmin skaalautuviksi ja luotettavammiksi, mutta tällä hetkellä järjestelmän koetaan täyttävän sille asetetut vaatimukset riittävällä tasolla ja kehityksessä keskitytään yksittäisten prosessien testaamiseen ja toiminnan varmistamiseen lopullisen tuotantovalmiuden saavuttamiseksi.

6. YHTEENVETO

Skaalautuvan, luotettavan ja kustannustehokkaan pilvipalvelupohjaisen järjestelmän kehitykseen vaikuttaa useita tekijöitä valitusta pilvipalvelualustasta ja palvelumallista aina yksittäisten järjestelmäkomponenttien määritykseen ja tekniseen toteutukseen asti. Julkisten pilvipalveluiden markkinoilla on useita eri kokoisia tarjoajia, joista suurimmat pyrkivät tarjoamaan mahdollisimman kattavan kokoelman palveluita erilaisille käyttäjille. Sopivan pilvipalveluiden tarjoajan valinta onkin usein pilvipohjaisen järjestelmän kehityksen ensimmäisiä vaiheita, sillä käytettävissä olevien palveluiden ja niiden hinnoittelulla on suoraan vaikutusta suunniteltavan järjestelmän arkkitehtuuriin ja sitä myötä skaalautuvuuteen ja kustannustehokkuuteen.

Tässä diplomityössä skaalautuvan pilvipohjaisen järjestelmän kehitystä ja teknistä toteutusta käsiteltiin suurelta osin asiakasprojektiin liittyvän AWS-alustalle toteutetun järjestelmän avulla. Esimerkkijärjestelmän käyttötarkoitus muuttuvan sähkönmittausdatan ja suuryrityksen asiakas-, mittari- ja käyttöpaikkatietojen käsittelijänä teki siitä hyvin diplomityön aiheeseen soveltuvan tutkimuskohteen. Tapahtumapohjaisuus ja mikropalveluarkkitehtuuri ovat molemmat järjestelmän skaalautuvuuden kannalta merkittäviä tekijöitä, sillä ne mahdollistavat järjestelmän eri osien skaalautumisen sekä horisontaalisesti että vertikaalisesti. Kustannustehokkuuden kannalta näiden vaikutus riippuu ennemmin yksittäisten tekijöiden kuten esimerkiksi Lambda-prosessorien hinnoittelusta ja tarkoituksenmukaisesta määrityksestä ja käytöstä. Esimerkkijärjestelmän toteutuksen mukaisella tapahtumien käsittelyllä ja väliaikaisella varastoinnilla datan eheys ja sitä myötä luotettavuus on saatu energia-alan yrityksen vaatimalle tasolle.

Todellisen suorituskyvyn osalta esimerkkijärjestelmän arviointi ja siitä tehtyjen päätelmien määrä jäivät valitettavan vähäiselle tasolle järjestelmän puutteellisen suorituskykytestauksen ja kehitysvaiheessa käytettävissä olevan datan vähäisen määrän vuoksi. Kattavammalla testauksella ja suuremmilla testatuilla datamäärillä olisi päästy parempiin johtopäätöksiin myös esimerkkijärjestelmän skaalautuvuuden ja kustannustehokkuuden näkökulmasta.

LÄHTEET

[1] D. Rountree, I. Castrillo, The Basics of Cloud Computing: Understanding the Fundamentals of Cloud Computing in Theory and Practice, julkaistu 2013, saatavissa: https://www-sciencedirect-com.libproxy.tuni.fi/book/9780124059320/the-basics-of-cloud-computing

[2] P. Mell, T. Grance, NIST, The NIST Definition of Cloud Computing, NIST SP 800-145, 2011, saatavissa: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[3] M. Anandarajan, B. Arinze, Cloud Computing, Volume 13. Technology and Innovation Management, julkaistu 22.1.2015, saatavissa: https://onlinelibrary-wiley-com.libproxy.tuni.fi/doi/full/10.1002/9781118785317.weom130068

[4] Amazon AWS, About Us, https://aws.amazon.com/about-aws/

[5] Canalys, Global cloud services market reaches US$42 billion in Q1 2021, päivitetty 29.4.2021, saatavissa: https://canalys-com-public-prod.s3.eu-west-2.amazonaws.com/static/press_release/2021/1143234453Canalys_cloud_pr_Q1_2021.pdf

[6] Amazon AWS, AWS Well-Architected framework, https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/index.en.html

[7] L. M. Vaquero, L. Rodero-Merino, R. Buyya, Cloud scalability: building the Millennium Falcon, päivitetty 19.2.2013, saatavissa: https://onlinelibrary-wiley-com.libproxy.tuni.fi/doi/full/10.1002/cpe.3008

[8] Expósito E, Taboada G, Ramos S, Touriño J, Doallo R. General-purpose computation on GPUs for high performance cloud computing, julkaistu 15.5.2012, saatavissa: https://onlinelibrary-wiley-com.libproxy.tuni.fi/doi/full/10.1002/cpe.2845?sid=vendor%3Adatabase

[9] C. Cote, Harvard Business School, What is data integrity and why does it matter, päivitetty 4.2.2021, saatavissa: https://online.hbs.edu/blog/post/what-is-data-integrity

[10] N. Al-Saiyd, N. Sael, Data integrity in cloud computing security, 2013, saatavissa: https://www.researchgate.net/publication/329512861_DATA_INTEGRITY_IN_CLOUD_COMPUTING_SECURITY

[11] IBM Security X-Force, 2021 Cloud Threat Landscape Report, julkaistu 2021, saatavissa: https://www.ibm.com/downloads/cas/WMDZOWK6

[12] A. Loten, The Wall Street Journal, Rush to the Cloud Creates Risk of Overspending, julkaistu 25.7.2018, saatavissa: https://blogs.wsj.com/cio/2018/07/25/rush-to-the-cloud-creates-risk-of-overspending/

[13] Digital Cloud, Amazon RDS vs DynamoDB, saatavissa: https://digitalcloud.training/amazon-rds-vs-dynamodb/

[14] Open Source Initiative, Licenses & Standards, saatavissa: https://opensource.org/licenses

[15] Amazon AWS, Baseline rule groups, saatavissa: https://docs.aws.amazon.com/waf/latest/developerguide/aws-managed-rule-groups-baseline.html#aws-managed-rule-groups-baseline-crs

[16] OWASP – Top 10 Web Application Security Risks, päivitetty 2021. Saatavissa: https://owasp.org/www-project-top-ten/

LIITE 1: CDK-MÄÄRITYKSET LAMBDA-METRIIKALLE

```
// Kinesis rooli AWS logituksen välittämiseen Firehose-palvelulla
const kinesisRole = new iam.Role(this, 'kinesisRole', {
      assumedBy: new iam.ServicePrincipal('firehose.amazonaws.com'),
});
// S3-bucket monitoroitavan lokidatan tallennusta varten
const datadogDataBucket = new s3.Bucket(this, 'datadogDataBucket', {
      versioned: true,
      bucketName: `${resPrefix}-datadog-data`,
      encryption: s3.BucketEncryption.KMS,
      encryptionKey: kmsKey,
      publicReadAccess: false,
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
});
datadogDataBucket.grantReadWrite(kinesisRole);
kmsKey.grantEncryptDecrypt(kinesisRole);
// Firehose deliveryStream, joka ohjaa S3-bucketiin tallennettua dataa Da-
taDog-endpointille
const deliveryStream = new kinesisfirehose.CfnDeliveryStream(this, 'data-
dogDeliveryStream', {
   deliveryStreamType: 'DirectPut',
   deliveryStreamName: `${resPrefix}-datadog-stream`,
   httpEndpointDestinationConfiguration: {
        endpointConfiguration: {
          url: datadogConfiguration.url,
          accessKey: datadogConfiguration.apiKey,
        },
        requestConfiguration: {
          commonAttributes: deliveryStreamPrameters,
        },
        s3Configuration: {
          bucketArn: datadogDataBucket.bucketArn,
          roleArn: kinesisRole.roleArn,
        },
      },
});
// Cloudwatch rooli Lambdojen logituksen ohjaamiseksi deliveryStreamiin
const cloudwatchRole = new iam.Role(this, 'kinesisCloudwatchRole', {
      assumedBy: new iam.ServicePrincipal('logs.amazonaws.com'),
});
cloudwatchRole.addToPolicy(
      new iam.PolicyStatement({
        resources: [deliveryStream.attrArn],
        actions: ['firehose:*']
}));
new cdk.CfnOutput(this, 'datadogDeliveryStreamARN', {
      value: deliveryStream.attrArn,
      exportName: `${resPrefix}-DatadogDeliveryStreamARN`,
});
return (lambda: lambda.Function) => new logs.CfnSubscriptionFilter(this,
lambda.node.id + 'LambdaKinesisSubscription', {
         destinationArn: deliveryStream.attrArn,
         logGroupName: lambda.logGroup.logGroupName,
         filterPattern: logs.FilterPattern.allEvents().logPatternString,
         roleArn: cloudwatchRole.roleArn});
```

LIITE 2: AWS WAF MÄÄRITYKSET KÄYTTÖLIITTYMÄLLE


```
{"Name": "UI-WAF",
  "Id": "<id>",
  "ARN": "arn:aws:wafv2:us-east-1:<tunnus>:global/webacl/UI-WAF/<>",
  "DefaultAction": {
    "Block": {}
  },
  "Description": "",
  "Rules": [
    {
      "Name": "AWS-AWSManagedRulesCommonRuleSet",
      "Priority": 0,
      "Statement": {
        "ManagedRuleGroupStatement": {
          "VendorName": "AWS",
          "Name": "AWSManagedRulesCommonRuleSet"
        }
      },
      "OverrideAction": {
        "None": {}
      },
      "VisibilityConfig": {
        "SampledRequestsEnabled": true,
        "CloudWatchMetricsEnabled": true,
        "MetricName": "AWS-AWSManagedRulesCommonRuleSet"
      }
    },
    {
      "Name": "Allowed-IP-Set-rule",
      "Priority": 1,
      "Statement": {
        "IPSetReferenceStatement": {
          "ARN": "<ARN>"
        }
      },
      "Action": {
        "Allow": {}
      },
      "VisibilityConfig": {
        "SampledRequestsEnabled": true,
        "CloudWatchMetricsEnabled": true,
        "MetricName": "Allowed-IP-Set-rule"
      }
    }
  ],
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "UI-WAF"
  },
  "Capacity": 701,
  "ManagedByFirewallManager": false,
  "LabelNamespace": "awswaf:<tunnus>:webacl:UI-WAF:"
}
```