Haaga-Helia
ammattikorkeakoulu Oy

Web-sovelluksen toteuttaminen palvelimettomalla arkkitehtuurilla

Iiro Niemi

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2021

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)

Iiro Niemi

Koulutusohjelma

Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi **Kirjoita tähän työsi pääotsikko.**	**Sivu- ja liitesivumäärä**
Web-sovelluksen toteuttaminen palvelimettomalla arkkitehtuurilla	23

Pilvipalvelujen kehityksen ja käytön kasvun myötä palvelumallit ja ohjelmistoarkkitehtuurit ovat muuttuneet johtaen uuden tyyppisiin ohjelmistokehityksen tapoihin ja haasteisiin. Palveluntarjoajat ovat alkaneet tarjota palveluja, joissa käyttäjän ei tarvitse ottaa vastuuta infrastruktuurin ja palvelinten hallinnoinnista. Käyttäjälle näkyvän korkeamman abstraktiotason myötä on alettu puhua palvelimettomasta laskennasta (engl. serverless computing), jossa sovellukset muodostuvat enemmän ulkoisista sovelluspalveluista (BaaS, engl. backend as a service) ja palvelinpuolen koodin suorittamisesta väliaikaisissa komponenteissa, kuten konteissa ja funktiopalveluissa (FaaS, engl. funcktion as a service). Samalla arkkitehtuureissa on siirrytty tapahtumapohjaisiin mikropalveluihin.

Tämän opinnäytetyön tavoitteena on kehittää palvelimettomalla arkkitehtuurilla toteutettava web-sovellus, joka auttaa toimeksiantajayritystä projektien suunnittelussa. Samalla tavoitteena on myös perehtyä teoreettisella ja konseptuaalisella tasolla palvelimettomiin pilvipalveluihin ja niitä hyödyntäviin arkkitehtuurimalleihin.

Opinnäytetyön ensimmäisessä osassa taustoitetaan palvelimettomaan paradigmaan johtaneita tekijöitä teknologioiden, palvelumallien ja arkkitehtuurien osalta. Samalla esitellään toiminnallisessa osiossa käytettävän Google Cloudin palvelimetonta tarjoomaa.

Toisessa osassa käsitellään toteutettavan sovelluksen kehityksen elinkaaren vaiheita määrittelystä suunnitteluun, kehitykseen ja testaukseen. Arkkitehtuurin ja toteutuksen osalta päädytään käyttämään laajasti Googlen Firebase-alustan palveluita.

Lopussa esitetään johtopäätöksiä ja kehitysehdotuksia. Pilvipalveluiden käyttö tulee kasvamaan voimakkaasti tulevaisuudessa sekä teknologiat kehittyvät vauhdilla. Palvelimettomat teknologiat soveltuvat vielä kapeaan segmenttiin sovellustyypeistä, joten konttipohjaiset ratkaisut säilyvät varteenotettavana vaihtoehtona rinnalla.

Toteutetun sovelluksen kaltaiselle pienelle ohjelmistoprojektille palvelimeton arkkitehtuuri ja Firebase-alusta sopi erinomaisesti. Laajemman sovelluksen kohdalla pitäisi kartoittaa uusimmat vaihtoehdot, koska uusia palveluita julkaistaan jatkuvasti ja ne useimmiten parantavat kehitystä tai ratkaisevat jonkun aiemman rajoitteen.

Asiasanat palvelimeton laskenta, palvelimeton arkkitehtuuri, pilvipalvelut, mikropalvelut

Sisällys

1 Käsitteet ja lyhenteet

Faas	Funktio palveluna (engl. Function as a Service)
BaaS	Palvelinpuolen komponentit palveluna (eng. Backend as a Service)
API	Sovellusrajapinta (engl. Application Programming Interface).
Palvelimeton laskenta	Engl. Serverless Computing. Pilvipalvelumalli, jossa palveluntarjoaja vastaa palvelimiin liittyvästä mm. infrastruktuurista, ylläpidosta ja skaalauksesta.
AWS	Amazon Web Services, Amazonin pilvipalvelualusta
Google Cloud	Googlen pilvipalvelualusta
Azure	Microsoftin pilvipalvelualusta
CNCF	Cloud Native Computing Foundation

2 Johdanto

Pilvialustoja hyödynnetään yhä useammin it-ratkaisujen tuottamisessa. Vanhoja ratkaisuja modernisoidaan pilvialustoille ja uutta liiketoimintaa rakennetaan niiden päälle. Vuonna 2016 oli näkyvissä jo, että noin kaksi kolmannesta yritysten it-budjetista pohjautuu pilven käyttöön vuoteen 2020 mennessä (Castro ym. 2019). Vuonna 2020 pilvi-infraan käytettävä rahamäärä ylitti omien datakeskuksien käytön, kasvaen 35% (Miller 2021) Gartnerin mukaan pilven käyttö kasvaa 18% vuoden 2021 aikana. Siitä summasta sovellusinfrastruktuuripalvelut (PaaS, engl. Platform as a Service) kasvaa 26,6% (Gartner 2020) IDC:n ennustuksen mukaan 2021 loppuun mennessä 80% yrityksistä keskittyy pilvipohjaiseen infraan ja sovelluksiin siirtymiseen (IDC 2020b).

Erityisesti ns. pilvinatiivien ratkaisujen suosio kasvaa koko ajan. Siinä pilvialustan tarjoamia palveluita käytetään mahdollisimman laajasti. Tämä ns. palvelimeton (engl. serverless) -malli on uusi paradigma, jolla ratkaisuja kehitetään ja jaellaan markkinoille ilman että ylläpidetään alla olevaa infrastruktuuria. Palvelimettomat ratkaisut tuovat kuitenkin mukanaan monia uusia haasteita.

Tässä opinnäytetyössä esitellään palvelimeton laskenta- ja arkkitehtuurimalli, kuvaillaan sen kehitystä ja tulevaisuuden haasteita sekä kokeillaan käytännössä ratkaista toimeksiantajayrityksen ongelmaa kehittämällä palvelimettomalla arkkitehtuurilla toteutettu työkalu.

3 Taustaa

3.1 Palvelimeton -määritelmä

Palvelimettoman laskennan (engl. serverless computing) määrittelemisessä on hyvä lähteä liikkeelle itse nimestä. Palvelimeton tarkoittaa pilvipalvelujen kontekstissa uutta abstraktiota, joka piilottaa palvelimet kehittäjän näkyviltä. Palvelimettomassa mallissa kehittäjien tarvitsee miettiä vähemmän palvelimia ja alemman tason yksityiskohtia joita palvelimien hallintaan liittyy.

Palvelimettomaan laskentaan löytyy monenlaista määritelmää. Tarkkoja rajoja on vaikea vetää, erilaiset määritelmät ovat enemmänkin termin harmaita sävyjä ja aiheeseen liittyvät muut termit limittyvät osittain palvelimettoman kanssa.

(Castro ym. 2019) määrittelee palvelimettoman tietojenkäsittelyn alustana, joka piilottaa palvelimien käytön kehittäjiltä ja ajaa koodia kysynnän mukaan automaattisesti skaalaten, laskuttaen ainoastaan ajasta joka kuluu koodin ajamiseen.

Paul Johnston (ServerlessDaysin perustaja) taas määrittelee palvelimettoman laskennan ratkaisuna, joka ei maksa mitään jos kukaan ei käytä sitä. (Castro ym. 2019)

CNCF (Cloud Native Computing Foundation) määrittelee palvelimettoman laskennan sovellusten kehittämisen ja ajamisen konseptina, joka ei vaadi palvelimien ylläpitoa. Palvelimeton kuvaa jakelumallin, jossa yhdestä tai useammasta funktiosta koostuvat sovellukset ajetaan, skaalataan ja laskutetaan täsmälleen tarvittavan kysynnän mukaan. (CNCF 2018a)

CNCF:n määritelmä mainitsee pilvifunktiot, joista palvelimeton-sanan voi sanoa saaneen alkunsa. AWS julkaisi ensimmäisenä vuonna 2015 Lambda-palvelunsa, jolla pystyi ajamaan pilvifunktioita. (Jonas ym. 2019)

AWS Serverless Hero ja iRobitin tutkija Ben Kehoe puhuu serverlessistä jatkumona, jolla palvelu/sovellus on enemmän palvelimeton, jos se on pitemmällä jollain seuraavista ulottuvuuksista (Kehoe 2017):

- Enemmän palveluita ja lyhytkestoista laskentaa
- Tiivis vastaavuus resurssien käytön ja laskutuksen välillä
- Pienempi ja enemmän abstraktoitu hallintakerros

Kun sovelluksesta tulee enemmän palvelimeton, sen rakennuspalikat ovat enemmän ulkoisia palveluita. Siksi termi palvelullinen (engl. serviceful) saattaa olla parempi kuin palvelimeton. Kuitenkin useimmiten puhutaan palvelimettomasta.

Berkeleyn tutkimuspaperin määritelmä ottaa parhaiten huomion tämän jatkumon määrittelyssään. Heidän määritelmänsä mukaan vaikka palvelimettoman laskennan ydin on pilvifunktiot, pilvipalveluntarjoajat myös tarjoavat erikoistuneita palvelimettomia palveluita, jotka täyttävät tiettyjä sovellusten vaatimusten erityistarpeita (Jonas ym. 2019). Tiivistetysti Berkeleyn tutkimusryhmän määritelmä on Serverless = FaaS + BaaS (engl. Function as a service + Backend as a service), eli palvelimettomien funktioiden ja muiden palvelimettomien palveluiden yhdessä muodostama kokonaisuus. Myös CNCF:n palvelimettoman laskennan määritelmä ottaa huomioon FaaS:n ja BaaS:n kokonaisuuden (CNCF 2018b).

Erityisesti erotuksena palvelillinen (engl. serverful), palvelimettomaan kuuluu:

- Laskenta ja tiedon varastointi on erotettu toisistaan. Varastointi on erillinen palvelunsa ja laskenta on tilatonta (engl. stateless).
- Koodin ajo ilman resurssien allokoinnin ylläpitämistä. Pilvipalveluntarjoaja vastaa automaattisesti resurssien provisioinnista.
- Resurssien täsmällisestä käytöstä maksetaan, ei kuinka paljon resursseja on allokoitu.

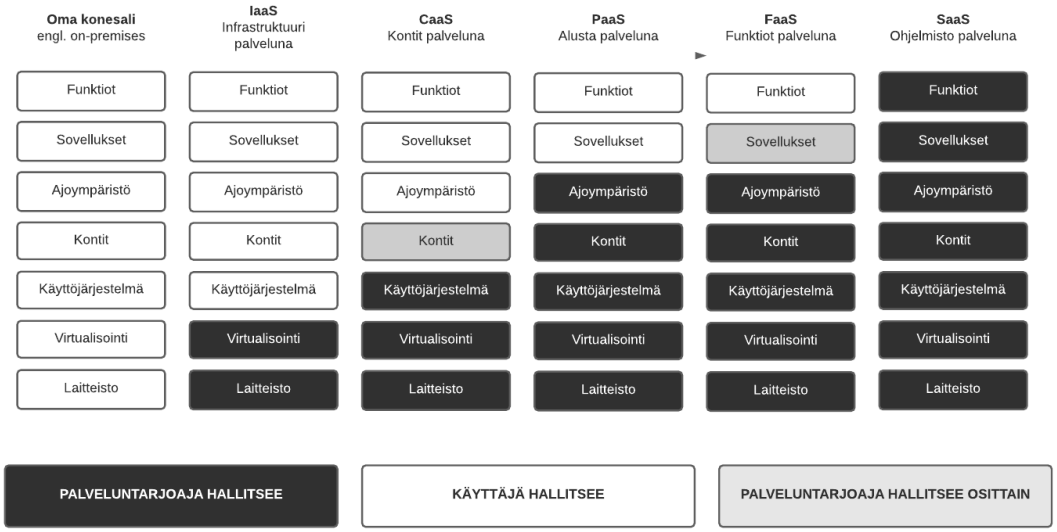
Termien määrittelyn sijaan on kuitenkin hyödyllisempää keskittyä siihen mitä palvelimettomilla ratkaisuilla pyritään saamaan aikaiseksi (Eismann ym. 2021):

- Operaatioiden taakkaa pienemmäksi.
- Nopeammin markkinoille.

- Fokus liiketoiminnan arvon luomisessa.

3.2 Pilvipalvelumallit

Pilvipalvelumallit voidaan karkeasti jakaa sen mukaan, kuinka paljon kontrollia ja vastuuta mm. ylläpidosta on palveluntarjoajalla ja kuinka paljon käyttäjällä tai kehittäjällä.



Kuva 1: Pilvipalvelumallit. Muokattu lähteestä (Onrego 2020)

Oman konesalin (engl. on-premises) -mallissa käyttäjä vastaa kaikesta lähtien fyysisestä laitteistosta, jonka päällä sovellukset pyörivät. Perinteisesti ennen pilvipalveluiden yleistymistä sovellusten työkuormat suoritettiin omissa konesaleissa tai ostetulla tai vuokratulla palvelinlaitteistolla.

IaaS (infrastruktuuri palveluna, engl. Infrastructure as a Service) -mallissa palveluntarjoaja vastaa laitteistosta ja virtualisoinnista. Käyttäjä voi konfiguroida esimerkiksi palvelin-, tallennus- ja verkkokapasiteettia ja ajaa sovelluksia niiden päällä. Tässä mallissa käyttäjällä on vastuu kaikesta käyttöjärjestelmästä ylöspäin. Käyttäjän tarvitsee huolehtia esimerkiksi korkeasta saavutettavuudesta, skaalauksesta, valvonnasta ja varmuuskopioinnista. Esimerkkejä IaaS-palveluista ovat mm. Amazon Elastic Compute Cloud (EC2), Google Compute Engine ja Azure Virtual Machines.

CaaS (kontit palveluna, engl. Container as a Service) -termi viittaa konttien orkesterointipalveluihin. Käytetyimmät palvelut pohjautuvat Kubernetekselle, kuten esim. Google Kubernetes Engine, Amazon Elastic Kubernetes Service tai Azure Kubernetes Service.

PaaS (alusta palveluna, engl. Platform as as Service) -mallissa palveluntarjoaja on vastuussa alustan infrasta, mutta kehittäjä on vastuussa koko koodin elinkaaresta joka jaellaan ja ajetaan alustalla. Alusta ei PaaS mallissa välttämättä skaalaudu nollaan, eli jouten olevista resursseista saattaa joutua maksamaan. Käyttäjä tyypillisesti pystyy muokkaamaan joitain alustan konfigurointiasetuksia.

SaaS (ohjelmisto palveluna, engl. Software as a Service) -mallissa palveluntarjoaja vastaa koko pinosta, myös sovelluksesta. Käyttäjän käyttää sovellusta usein selaimen tai muun käyttöliittymän kautta. SaaS-mallissa kehittäjällä ei ole kontrollia infraan, vaan pääsy vain olemassa oleviin komponentteihin. Kehittäjä saattaa pystyä ajamaan tiettyä koodia, mutta tämä koodi on tiukasti kytköksissä kyseiseen alustaan.

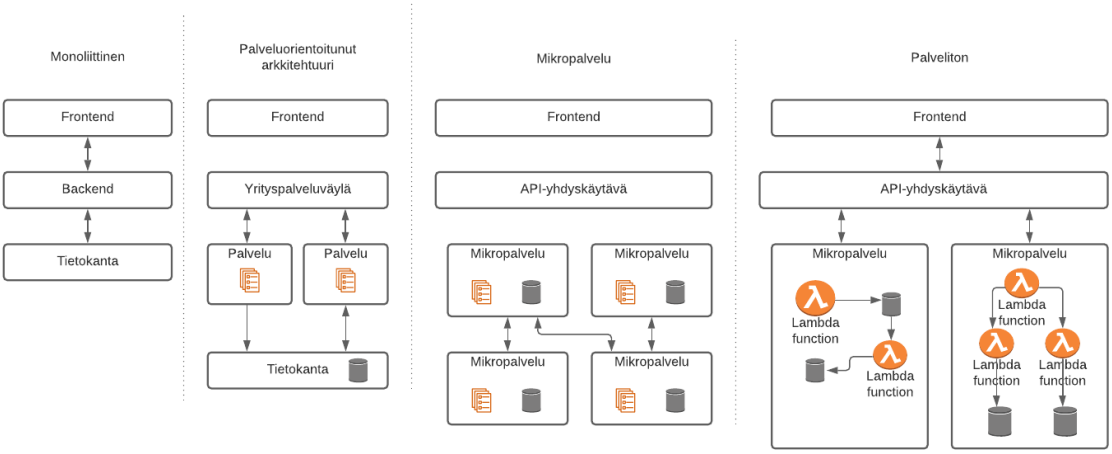
CNCF sekä Berkeleyn tutkimusryhmä jakavat palvelimettoman laskennan kahteen pilvipalvelumalliin: funktioihin palveluna (FaaS, engl. Function as a Service) ja backend palveluna (BaaS, engl. Backend as a Service).

BaaS viittaa kolmannen osapuolen API-pohjaisiin palveluihin, joilla pystytään korvaamaan jokin sovelluksen ydintoiminnallisuus. Nämä palvelut näyttäytyvät kehittäjälle palvelimettomina, koska ne skaalautuvat automaattisesti ja toimivat läpinäkyvästi. Usein BaaS-palveluja käytetään web- tai mobiilisovelluksissa, joissa hyödynnetään tietokantoja tai autentikaatiopalveluja. BaaS-mallin voisi sanoa sijoittuvan Kuva 1:ssa PaaS- ja FaaS-mallien väliin.

FaaS-palveluissa (funktio palveluna, engl. Function as a Service) ajetaan pieniä koodikokonaisuuksia jonkin tapahtuman sattuessa (ns. tapahtumapohjainen arkkitehtuuri). Tapahtuma voi olla esimerkiksi HTTP-kysely, tietokantaoperaatio, tai viesti jossain viestipalvelussa. Kehittäjät jakelevat koodin FaaS-palveluun, jossa koodi suoritetaan tarvittaessa tapahtumiin perustuen. Palveluntarjoaja vastaa automaattisesta skaalauksesta ja alla olevasta infrastruktuurista.

3.3 Arkkitehtuurisuunnittelumallit

Yritysarkkitehtuureissa on siirrytty pitkälti kontteihin ja mikropalveluihin. Palvelimettomassa tietojenkäsittelyssä on paljon yhteneväisyyksiä mikropalveluiden kanssa, mutta myös eroavaisuuksia. Ohjelmistoarkkitehtuurien evoluutiossa mikropalveluarkkitehtuuri syntyi vähentämään monoliittisten systeemien ja palveluorientoituneiden arkkitehtuurien kompleksisuutta. (Taibi, Spillner & Wawruch 2021)



Kuva 2: Ohjelmistoarkkitehtuurien evoluutio (muokattu lähteestä (Taibi, Spillner & Wawruch 2021)

Kuva kuvaa ohjelmistoarkkitehtuurien evoluutiota. Mikropalvelut ovat pieniä ja itsenäisiä palveluita, joilla on yksi selkeästi määritelty tarkoitus tai tehtävä. Ne keskustelevat toistensa kanssa hyvin määriteltyjen sovellusrajapintojen (engl. API, application programming interface) kautta. Mikropalveluiden ideana on, että jokaista palvelua pystytään kehittämään, jakelemaan ja testaamaan itsenäisesti ja erillään muista. Eri mikropalvelut saattavat myös muodostua erilaisista teknologiapinoista ja olla eri tiimien vastuulla. (Taibi, Spillner & Wawruch 2021)

Palvelimettoman arkkitehtuurin ero mikropalveluihin on siinä, että palvelimettomat mallit pohjautuvat tapahtumille, kun mikropalvelut vastaavat usein API-kutsuihin. Palvelimettomassa mallissa funktioita pystytään laukaisemaan myös ilman erillistä API-kerrosta, esimerkiksi tietomuutoksen tai sanoman välityksen seurauksena. (Taibi, Spillner & Wawruch 2021)

Mikropalvelut soveltuvat paremmin pitkään ajettaviin palveluihin, jotka vaativat paljon resursseja. Palvelimettomat funktiot taas ovat ajoajaltaan lyhytkestoisia. (Taibi, Spillner & Wawruch 2021)

Palvelimeton malli on toisaalta niin uusi, että on vähän empiiristä dataa kuinka palvelimettomat arkkitehtuurit pitäisi muodostaa (Taibi, Spillner & Wawruch 2021). Eri pilvipalveluntarjoajilla on työkaluja arkkitehtuurin parhaiten käytänteiden soveltamiseen (Narang 2021). Pisimmällä lienee arkkitehtuurityökaluissa tällä hetkellä AWS, joka tarjoaa myös työkalun nimenomaan palvelimettoman arkkitehtuurin tarkasteluun, nimeltään AWS Serverless Lens. (AWS 2019)

3.4 Palvelimettoman laskennan erityispiirteitä

Pilvipalvelujen ensimmäinen vaihe yksinkertaisti järjestelmäylläpidon helpottamalla infrastruktuurin konfigurointia ja hallintaa virtuaalikoneiden ja virtuaaliverkkojen avulla, jotka pyörivät datakeskuksissa. Käynnissä olevassa toisessa vaiheessa palvelimet piilotetaan tarjoamalla ohjelmointiabstraktioita sovellusten kehittäjille, mikä yksinkertaistaa pilvikehitystä ja tekee pilviohjelmistoista helpompia kirjoittaa. (Johann Schleier-Smith ym. 2021)

Ensimmäisen vaiheen kohde oli järjestelmäylläpitäjät ja toisen vaiheen kohde on ohjelmoijat. Tämä muutos vaatii sen, että pilvipalvelujentarjoajat ottavat vastuulleen operaatiot joiden päällä sovellukset pyörivät. Korostaakseen fokuksen muutosta palvelimista sovelluksiin, uutta vaihetta on alettu kutsua palvelimettomaksi tietojenkäsittelyksi, vaikka etäpalvelimet ovat edelleen pohjalla pyörivä perusta. Perinteistä ensimmäistä vaihetta kutsutaan palvelilliseksi tietojenkäsittelyksi (engl. serverful computing). (Johann Schleier-Smith ym. 2021)

Taulukko 1 kuvaa palvelimettoman ja palvelillisen mallin eroja, vertailukohtina AWS:n Lambda -pilvifunktiot ja EC2-virtuaalikoneet. (Jonas ym. 2019)

Ominaisuus	**Palvelimeton**	**Palvelillinen**
Milloin ohjelma ajetaan	Käyttäjän määrittelemän tapahtuman sattuessa	Jatkuvasti kunnes pysäytetään
Ohjelman tilan käsittely	Jossain ulkopuolisessa varastossa (storage), ohjelma tilaton (stateless)	Missä tahansa (stateful or stateless)
Muistin maksimimäärä	3 GB (käyttäjä valitsee)	> 10 TB (käyttäjä valitsee)
Maksimi ajoaika	Mitataan minuuteissa	Rajaton
Käyttöjärjestelmä, kirjastot ja palvelimen tyyppi	Palveluntarjoaja valitsee	Käyttäjä valitsee
Skaalaus, Deployment, Virhetoleranssi, Monitorointi ja Loggaus	Palveluntarjoaja vastuussa	Käyttäjä vastuussa

Taulukko 1: Palvelimettomien pilvifunktioiden ja virtuaalikoneiden vertailua (muokattu lähteestä (Jonas ym. 2019)

(Eismann ym. 2021) tutkimuksessa analysoitiin 89 erilaista palvelimettomalla laskennalla toteutettua sovellusta. Tutkimuksessa selvisi muun muassa syitä miksi palvelimeton malli otettiin käyttöön. Motivaatioista suurin oli raha – 47% valitsi palvelimettoman mallin vähentääkseen kustannuksia. 34% valitsi palvelimettoman laskennan, koska se vähentää operationaalisia huolia. Palvelimettomassa mallissa kehittäjät voivat keskittyä uusiin ominaisuuksiin jakelun, skaalaamisen tai valvonnan sijaan. Kolmas tärkein syy oli (34%) palvelimettomien sovellusten skaalaamisen helppous. (Eismann ym. 2021)

Toisessa 160 vastaajan kyselytutkimuksessa selvisi myös, että palvelimettoman mallin omaksumisen hyötyjä ovat tapahtumapohjainen arkkitehtuuri (51%), kustannukset (41%), kehittämisen nopeus (36%), skaalautumisen joustavuus (31%) sekä sovellusten suorituskyky (19%). (Eismann ym. 2021)

Palvelimettomilla järjestelmillä on mm. seuraavia hyötyjä verrattuna perinteisiin palvelillisiin malleihin (Taibi, Spillner & Wawruch 2021):

- Ei operaatioita: Pilvifunktioita käytettäessä tarvitsee vain jaella koodia. Ei tarvitse miettiä skaalaamista tai orkestrointia. Operaatioiden näkökulmasta suurin osa resurssien ylläpidosta on ulkoistettu, mm. tietokannat.

- Nopea ja itsenäinen jakelu: Kehittäjät voivat jaella ja uudelleenjaella yksittäisiä funktioita ilman koko systeemin jakelua. Sama pätee myös mikropalveluihin, mutta pilvifunktioiden kohdalla on kyse vielä pienemmästä kokonaisuudesta.

- Ei kiinteitä infrakustannuksia: Palvelimettomassa mallissa kustannukset ovat muuttuvia, ainoastaan resurssien käytön mukaan. Varatusta kapasiteetista ei laskuteta. Lyhytaikaisten tehtävien kohdalla kustannukset saattavat vähentyä dramaattisesti. Jos kuitenkin on kyseessä pitkäkestoinen laskenta, pitää ottaa huomioon suurien kustannusten mahdollisuus.

- Automaattinen skaalaus: Palvelimettomat sovellukset pystyvät käsittelemään ison määrän samanaikaisia kutsuja.

Palvelimettomat järjestelmät ovat hajautettuja järjestelmiä ja siten niiden mukana tulee omat haasteensa, erityisesti erilainen kehitysparadigma, mm. tapahtumapohjainen arkkitehtuuri, eri palvelujen välinen kommunikointi, valvonnan, testauksen ja tietoturvaan liittyvät haasteet: (Taibi, Spillner & Wawruch 2021)

- Testaus ja debuggaus: Ympäristön lokaali replikointi ei ole aina mahdollista, vaan funktioiden testausta pitää tehdä tuotannossa. Debuggaus voi olla haastavaa hajautetun luonteen vuoksi sekä koska taustaprosessit saattavat olla kehittäjiltä piilossa.

- Pitkäkestoiset prosessit: Perinteisessä palvelillisessa arkkitehtuurissa pitkäkestoisten prosessien ajaminen saattaa tulla halvemmaksi.

- Suorituskyky: Palvelimettomassa mallissa resurssien käynnistämiseen saattaa liittyä "kylmäkäynnistyksen" ongelma, mikä saattaa aiheuttaa ylimääräistä latenssia.

- Lukittautuminen palveluntarjoajaan: Toiseen palveluntarjoajaan vaihtaminen voi olla vaikeaa. Toisella palveluntarjoajalla vastaavat pilvinatiivit teknologiat saattavat erota mm. tietokantojen, viestipalveluiden, api-rajapintojen, tai muiden natiivien teknologioiden osalta.

- Tapahtumapohjainen paradigma: Voi olla kehittäjälle uusi, jos ei ole aikaisemmin ollut tekemisissä palvelimettoman laskennan kanssa.

3.5 Google Cloud palvelimettomana alustana

Kaikilla isoimmilla pilvialustoilla (AWS, Azure, Google Cloud) on lähes vastaavat palvelut, joista palvelimettomien sovelluksien perusarkkitehtuurin voi rakentaa (viesti-, tallennus-, tietokanta-, autentikaatio ja api-palvelut jne.) (Google 2021c)

Uusia palveluita julkaistaan jatkuvasti lisää. Tämän opinnäytetyön toiminnallisessa osuudessa käsiteltävän sovelluksen suunnittelun jälkeen on julkaistu Googlen palvelimettomien palveluiden tarjoomaan mm. EventArc (Google 2021e) ja Workflows-palvelut (Google 2020), jotka parantavat palvelimettomien sovellusten tapahtumien orkestrointia.

Googlen palvelimettoman laskennan (engl. serverless computing) tarjooma koostuu kolmesta palvelusta:

- Cloud Run: konttien ajoon tarkoitettu palvelu.

- Cloud Functions: pilvifunktiopalvelu.

- App Engine: Kokonaisten web-sovelluksien ajamiseen tarkoitettu palvelu.

Yksi Google Cloudin ensimmäisiä palveluja on App Engine, joka julkaistiin jo vuonna 2008. Silloin termiä palvelimeton ei oltu vielä edes keksitty. App Engine on PaaS-alusta sovellusten ajamiseen. Käyttäjän tarvitsee sovelluskoodin ohella määritellä vain muutamia resurssivaatimuksia, ja Googlen vastuulla on alla oleva infra virtuaalikoneista, konteista, käyttöjärjestelmästä ajoympäristötasolle asti. (Sullivan 2019)

Viimeisin Googlen palvelimeton palvelu on Cloud Run, joka suorittaa ja skaalaa käyttäjän kontteihin perustuvia sovelluksia. Cloud Run luo kontin perusteella palvelun ja liittää siihen kutsuttavan HTTPS-endpointin. Kehittäjän tarvitsee paketoida HTTP-serverin sisältämä koodi konttikuvaksi ja jaella konttikuva Cloud Runille. (Venema 2021)

Cloud Functions on Googlen pilvifunktiopalvelu. Palvelu on vastaavanlainen kuin muillakin pilvipalveluntarjoajilla. Käyttäjän tarvitsee tarjoilla vain koodi ja määritellä ajoympäristö, palveluntarjoaja vastaa skaalauksesta ja ylläpidosta.

Näiden lisäksi Googlella on oma BaaS-kokonaisuus nimeltään Firebase. Firebase on tullut suosituksi erityisesti web-sovellusten ja mobiilisovellusten alustana. Firebase oli alun perin oma yrityksensä, mutta Google osti sen vuonna 2014 ja on sittemmin integroinut alustan kiinteäksi osaksi Google Cloudia. Firebase tarjoaa helposti sovellukseen integroitavina komponentteina mm. tietokannan, autentikaatiopalvelun, tallennuksen, pilvifunktiot, isännöinnin (engl. hosting) ja analytiikan. (Google 2021f)

Tämän opinnäytetyön toiminnallisessa osuudessa arvioitiin kaikkia edellä mainittuja vaihtoehtoja. Lopulta päädyttiin käyttämään puhtaasti pilvifunktioita, koska ne ovat tyypillisin palvelimettomien sovellusten rakennuspalikka, sekä pisimmällä palvelimettomien palveluiden jatkumolla.

4 Palvelimeton sovellus Googlen Firebase-alustalla

4.1 Johdanto

Tämän opinnäytetyön toimeksiantajana toimi Alfame Systems Oy. Opinnäytetyössä lähdettiin kokeilemaan, voisiko Gantt-kaavion pohjalle rakentuva suunnittelutyökalu ratkaista tai helpottaa sisäiseen projektien ja henkilöiden koordinointiin liittyvää ongelmaa.

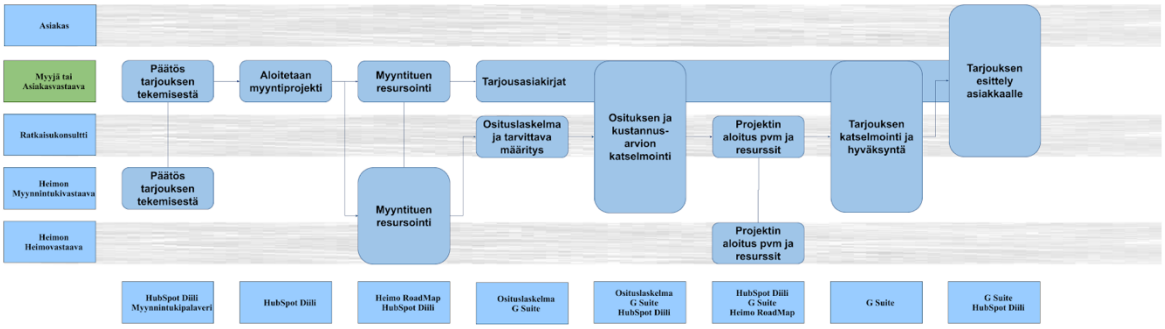
Alfame Systems Oy on it-alan konsulttiyritys, joka suunnittelee ja toteuttaa liiketoiminnalle kriittisiä API- ja integraatioratkaisuja, prosessiautomaatiota sekä sovellusten ohjelmistokehitystä. Henkilöstöä on noin vajaa 100 henkilöä. (Alfame 2021)

4.2 Vaatimusmäärittely

Taustaa ongelmista, tarpeista ja tavoitteista

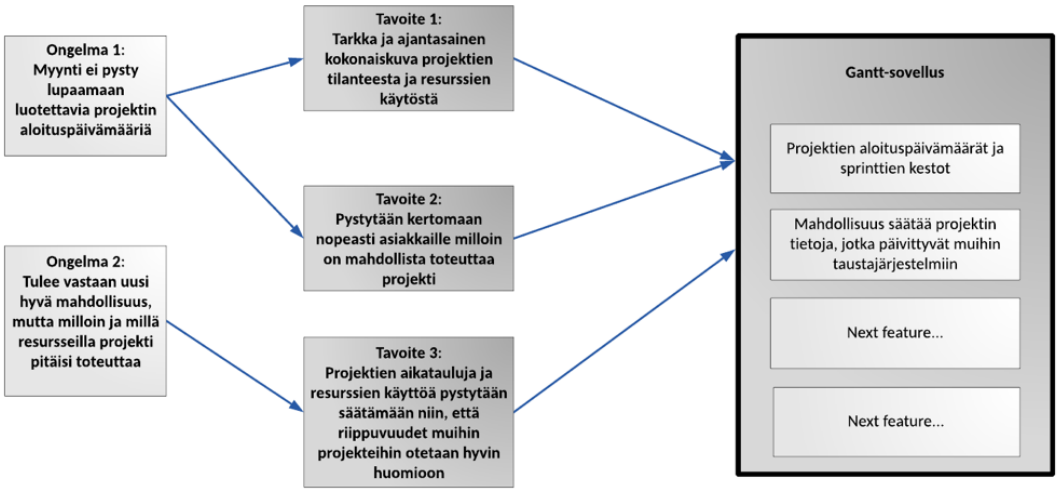
Kohdeyrityksessä henkilöstö on organisoitunut hallintoon ja heimoihin. Hallinto koostuu johtoryhmästä, myynnin, markkinoinnin ja viestinnän henkilöistä sekä henkilöstöhallinnosta. Heimot taas koostuvat projektipäälliköistä, asiakasvastaavista, ohjelmistoarkkitehdeistä ja -kehittäjistä. Heimot on muodostettu siten, että ne työskentelevät samantyyppisten asiakastarpeiden tai teknologioiden ympärillä.

Tätä opinnäytetyötä varten yrityksessä kaivattiin ratkaisua ongelmaan, joka oli huomattu erityisesti myynnin ja projektiensuunnittelun yhteydessä, kun pitäisi pystyä esittelemään asiakkaalle milloin uusi projekti voidaan aloittaa ja millaisella kokoonpanolla. Kuva 3 näyttää visualisointuna prosessin kulun.



Kuva 3: Tarjouksentekoprosessi

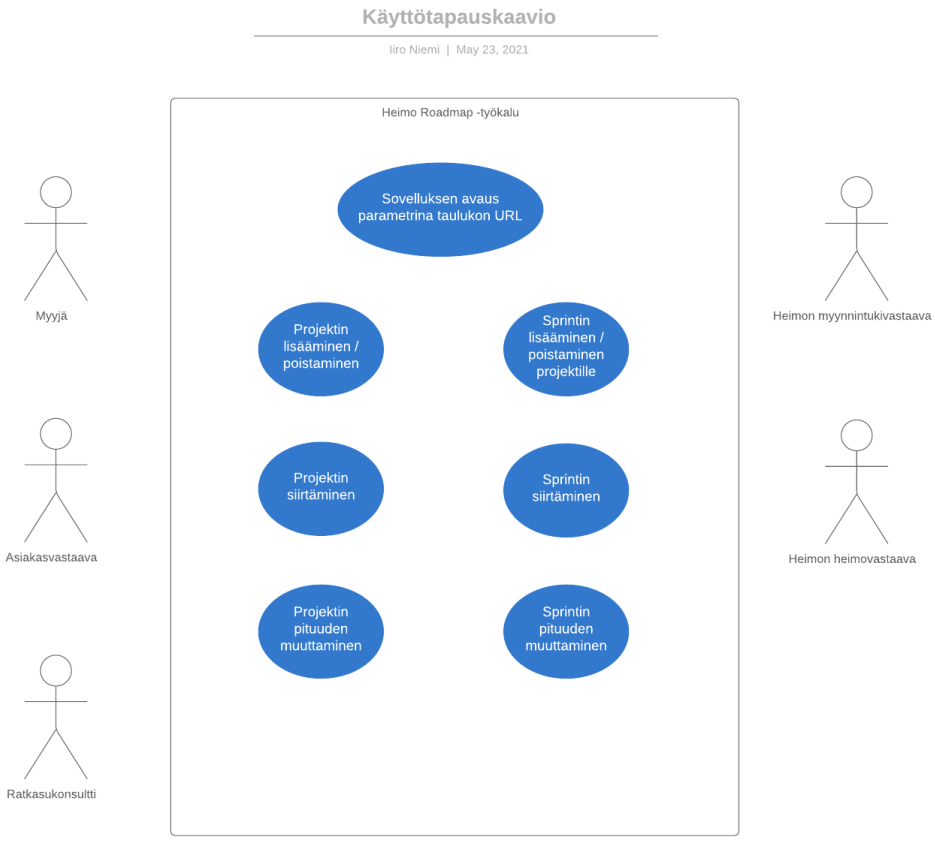
Projektien ja tiimien kokoonpanojen suunnitteluun tarvittiin työkalu, jolla voisi nopeasti hahmottaa nykyisten projektien ja resursoinnin kokonaisuus. Yrityksessä tätä ongelmaa ja ratkaisua oli jo siinä määrin tutkittu, että oli havaittu potentiaalinen komponentti jonka pohjalta voitaisiin rakentaa konseptitodistus (engl. Proof of Concept). Tämä komponentti oli Javascript-pohjainen interaktiivinen Gantt-kaaviokirjasto (DHTMLX 2021). Muitakin oli tutkittu yrityksen henkilöstön toimesta, mutta tämä oli havaittu mahdollisista vaihtoehdoista parhaaksi tähän tarkoitukseen. Opinnäytetyöprojektin osalta kartoitin myös muita vaihtoehtoja, mutta parempia ei tullut vastaan. Ongelmaa ja ratkaisua on hahmotettu alla (Kuva 4).



Kuva 4: Ongelman ja ratkaisun hahmottelua

Sovelluksen toiminnoista ja vaatimuksista

Sovelluksessa tarvittavia käyttötapauksia tunnistettiin erityisesti myynninvastaavien, heimovastaavien, asiakasvastaavien ja tekijöiden osalta. Niitä on hahmotettu alla.



Kuva 5: Käyttötapauskaavio

Lisäksi sovelluksen tulisi kytkeytyä muihin yrityksen taustajärjestelmiin, joista saataisiin dataa ja joihin data päivittyisi sovellusta käytettäessä. Näitä on esimerkiksi Google Sheets, BigQuery, HubSpot.

4.3 Suunnittelu

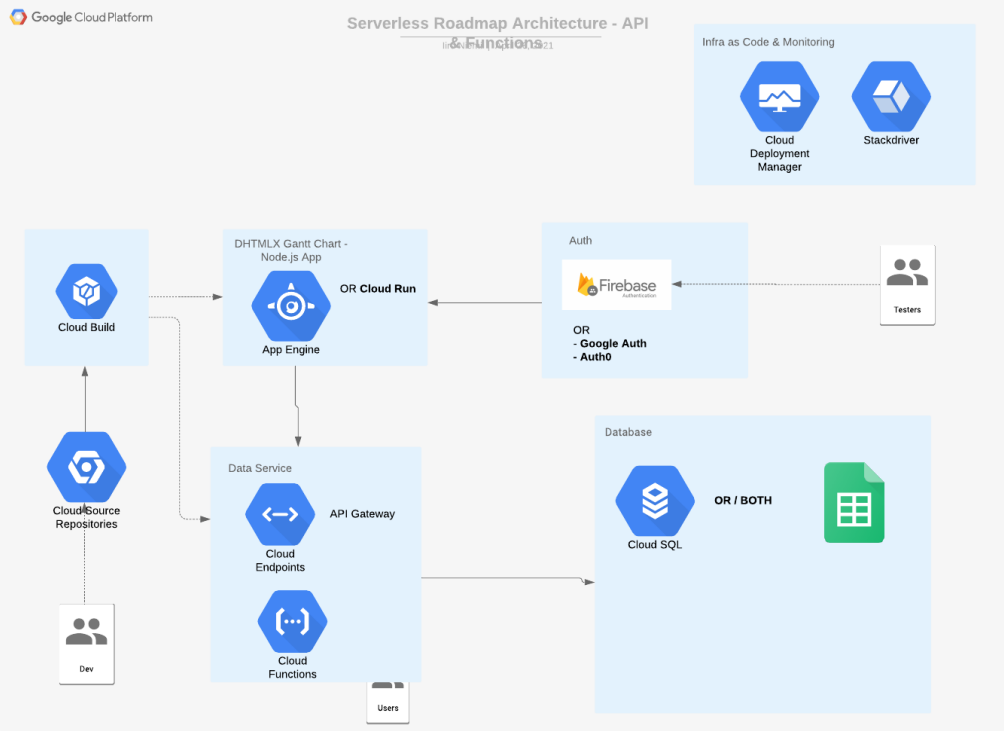
Teknologioiden osalta opinnäytetyössä haluttiin kokeilla jotain uutta, jota ei aiemmin oltu käytetty. Sellaiseksi valikoituivat Googlen pilvipalvelut ja erityisesti palvelimettoman laskennan teknologiat.

Samalla kun rinnalla opiskeltiin Google Cloudiin liittyvää materiaalia, alkoi myös hahmotelmat sovelluksen arkkitehtuurista selkiytyä. Arkkitehtuurisuunnitelmissa edettiin useamman iteraation kautta, kunnes lopulta löytyi opinnäytetyön rajaukseen sopiva.

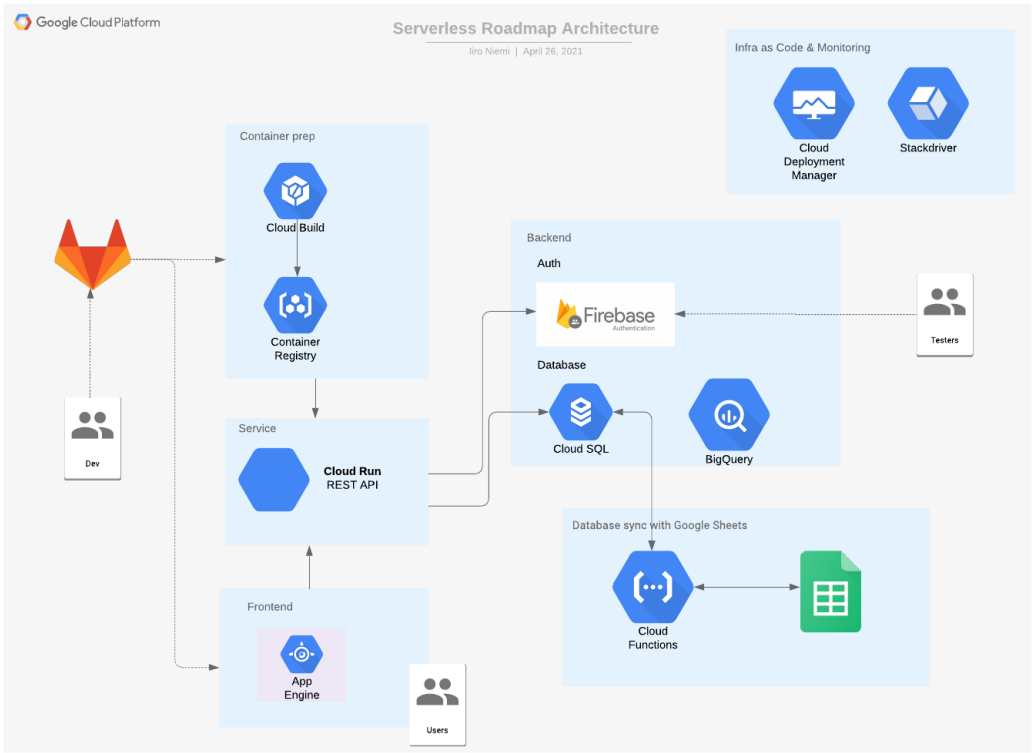
Arkkitehtuurin kehityspolkua

Arkkitehtuurimalleissa on pyritty kuvaamaan käytettävät palvelut ja kuinka data liikkuu palvelusta tai työkalusta toiseen.

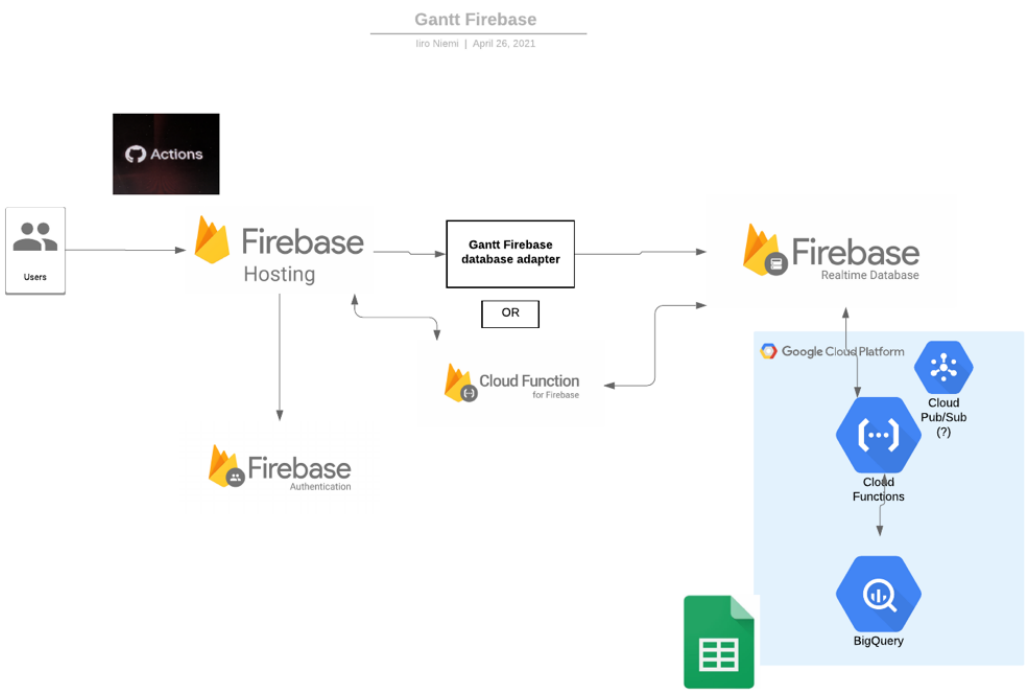
Ensimmäisessä (Kuva 6: Arkkitehtuuri versio 1) ja toisessa (Kuva 7: Arkkitehtuuri versio 2) versiossa keskityttiin pitkälti suunnittelussa Googlen palveluihin ja Firebase ei vielä tässä vaiheessa ollut vahvasti pöydällä. Kontitus houkutteli erityisesti sen siirrettävyyden takia - kaikista suurimmista pilvipalveluista (Google, AWS, Azure) löytyy omat vaihtoehtonsa konttien orkestrointiin.



Kuva 6: Arkkitehtuuri versio 1

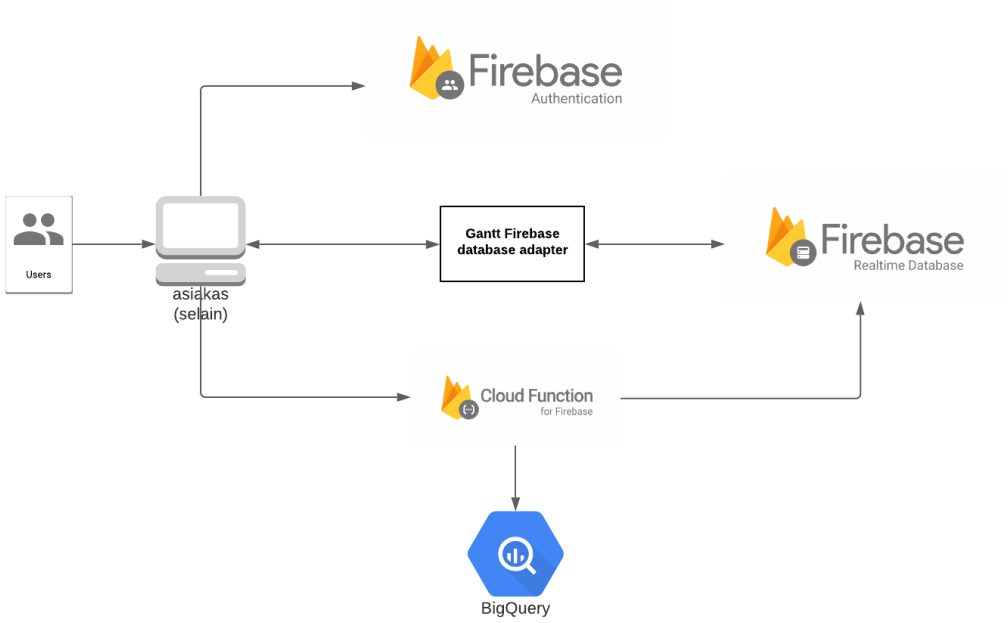


Kuva 7: Arkkitehtuuri versio 2



Kuva 8: Arkkitehtuuri versio 3

Firebaseen päädyttiin lopulta arkkitehtuurisuunnittelun viimeisessä vaiheessa (Kuva 8: Arkkitehtuuri versio 3), koska siinä yhdistyi hyvin integroidussa kokonaisuudessa kaikki tämän lopputyön sovelluksen kannalta tarvittavat palvelut. Lisäksi Firebase on kiinteä osa muuta Google Cloudin tarjontaa, joten sen käyttö ei rajannut mitään Googlen vaihtoehtoja varsinaisesti pois.



Kuva 9: Viimeisin arkkitehtuuri yksinkertaisimmillaan

Viimeisintä arkkitehtuuria yksinkertaisimmillaan voisi kuvata Kuva 9 tavalla. Tarvittavat palvelut integroituivat hyvin selaimessa pyörivään Gantt-sovellukseen erillisinä valmiina Firebase-kirjastoina (firebase-auth.js, firebase-database.js, firebase-functions.js). Käyttäjän kirjautuessa sovellukseen päivitetään viimeisimmät tiedot BigQuery-tietovarannosta pilvifunktion avulla Firebasen tietokantaan. Käyttäjät pystyvät katselemaan ja muokkaamaan tietokannan tietoja Gantt-sovelluksen käyttöliittymän kautta, mutta tämän toteutuksen rajaukseen ei sisältynyt tietojen päivitys takaisin muihin yrityksen tietojärjestelmiin, joissa päivitetyistä tiedoista voisi olla hyötyä.

4.4 Kehitys

Isännöinti-palvelut Firebasessa

Sovelluksen isännöinti (engl. hosting) -palveluna käytetään Firebasea. Isännöintipalveluiden kautta käyttäjät saavat nopeasti sovelluksen selaimella ajamiseen tarvittavat tiedostot globaalin sisällönjakeluverkon (engl. CDN, Content Delivery Network) kautta. Yhteyden salaus on integroituna Firebasen isännöintipalveluun.

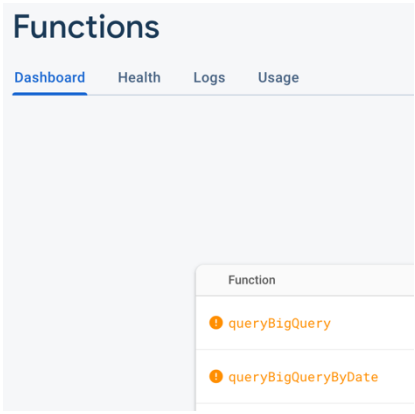
Isännöinti-käyttäytymistä voi melko pitkälle erikseen määritellä, mutta tässä projektissa pysyttiin lähes täysin oletusasetuksissa.

```
1.  "hosting": {
2.      "public": "public",
3.      "ignore": [
4.        "firebase.json",
5.        "**/.*",
6.        "**/node_modules/**"
7.      ],
8.      "rewrites": [
9.        {
10.         "source": "**",
11.         "destination": "/index.html"
12.       }
13.     ]
14.   }
```

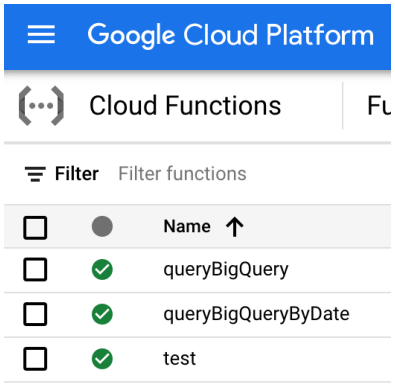
Asiakasselain lataa public -kansion sisällön (Firebase ei tarjoile isännöinti-palvelun kautta mitään muuta), sekä Firebase ohjaa kaikki kutsut aina index.html :ään.

Pilvifunktioiden käyttö sovelluksessa

Sovelluksessa käytin Firebasen alustan omia pilvifunktioita, jotka ovat käytännössä samat kuin Googlen Cloudin konsolissa näkyvät Cloud Functions. Samat funktiot näkyvät sekä Firebasen että Google Cloudin käyttöliittymissä (Kuva 10 ja Kuva 11).



Kuva 10: Funktiot Firebasen konsolissa



Kuva 11: Funktiot Google Cloudin konsolissa

Funktiot on toteutettu Node.js:llä käyttäen Firebasen ja Googlen kirjastoja:

```
1.  const functions = require('firebase-functions');
2.  const admin = require('firebase-admin');
3.  const { BigQuery } = require('@google-cloud/bigquery');
4.  admin.initializeApp();
5.
6.  exports.queryBigQueryByDate = functions.https.onCall(async (req, res) => {
7.    const bigquery = new BigQuery();
8.    console.log(req.date);
9.
10.   const query = `SELECT
11.                                       Name, CloseDate
12.                                 FROM
13.                                       \`alfame-bi.hubspot.deals\`
14.                                 WHERE
15.                                       Stage ="closedwon"
16.                                       AND CloseDate > cast(TIMESTAMP("${req.date}")
    as datetime)
17.                                 ORDER BY
18.                                       CloseDate ASC`;
19.
20.   const options = {
21.             query: query,
22.             location: 'US',
23.   };
24.
25.     const [job] = await bigquery.createQueryJob(options);
26.   const [rows] = await job.getQueryResults();
27.
28.   return rows;
29. });
```

Funktioiden tarkoitus on hakea tarvittavat tiedot Google Cloudin BiqQuery-datavarastosta. Käyttäjän kirjautuessa sovellukseen, sovellus käynnistyy ja laukaisee funktion, joka palauttaa BigQuerystä löytyvät tiedot. Sovellus tallentaa haetut tiedot tietokantaan. Jos BigQuerystä löytyy uusia tietoja, joita ei vielä ole sovelluksen tietokannassa, päivitetään tietokannan tiedot.

Sovelluksessa toteutetut funktiot ovat ns. ”HTTPS callable functions”, joita kutsutaan suoraan firebase-sovelluksesta. Näin autentikaatiotokenit sisältyvät automaattisesti funktiokyselyihin, eikä funktiokyselyä voi tehdä kirjautumatta sovellukseen.

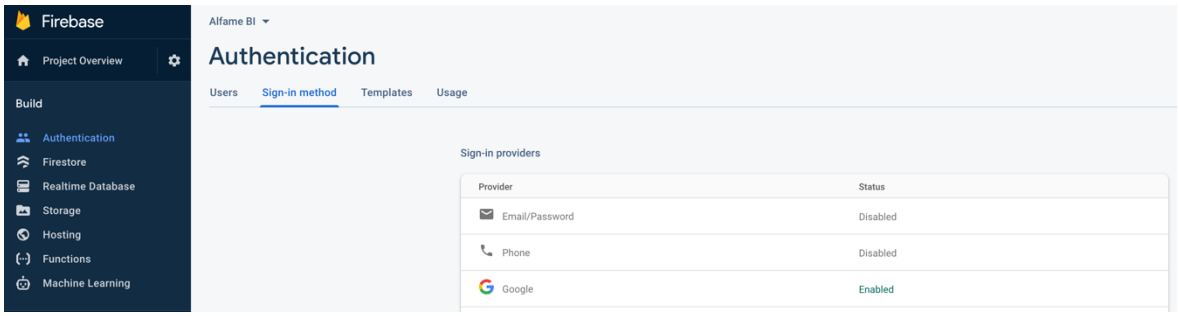
Asiakassovelluksen ja tietokannan välinen yhteys

Firebasen "Realtime Database" -tietokanta sopi tähän projektiin erinomaisen hyvin, koska Gantt-kaavion kehittäjä tarjosi hieman valmista javascript-koodia gantt-sovelluksen ja tietokannan synkronisointiin. Kuitenkin valmis koodi oli sen verran vanhaa, että osa tietokannan kanssa keskusteluun tarkoitetuista metodeista oli hieman vanhentunut. Joten tarkistin ja kirjoitin osan metodikutsuista uudelleen. (DHTMLX 2015)

Realtime Database sopi myös siitä syystä erinomaisesti tähän Gantt-sovellukseen, että se ilmoittaa sovelluksen asiakkaille reaaliajassa tietokannassa tapahtuvista muutoksista, ja näin kaikilla käyttäjillä on reaaliajassa oikea data sovelluksessa. (Google 2021g)

Autentikaatio

Autentikaatioon ja autorisaatioon käytettiin Firebasen ja Googlen valmiita työkaluja. Sovellus kuuntelee autentikaation tilaa ja näyttää sisältöä sivulla sen mukaan onko käyttäjä kirjautunut vai ei. Kirjautuminen on toteutettu käyttäen Firebasen Javascript SDK:ta (Software Development Kit) ja Googlen omaa kirjautumismetodia. (Google 2021a)



Tietokannan tietoturvasäännöt

Tietoturvan kannalta on tärkeää, että pääsy sovelluksen tietokantaan on vain kirjautuneilla käyttäjillä. Tähän sovellukseen tietokantasäännöt määriteltiin niin, että vain toimeksiantajayrityksen domain-päätteellä kirjautuneet käyttäjät pystyvät lukemaan ja kirjoittamaan tietokantaan (Kuva).

Firebase-projektin määrittelytiedostossa firebase.json sekä database.rules.json määritellään tietokantasäännöt:

firebase.json

```
1.  "database": {
2.      "rules": "database.rules.json"
3.  }
```

database.rules.json

```
1.  {
2.    "rules": {
3.      ".read": "auth.token.email.endsWith('@alfame.com')",
4.              ".write": "auth.token.email.endsWith('@alfame.com')"
5.    }
6.  }
```

Jatkuva jakelu

Projektin versionhallintatyökaluna käytettiin Githubia, koska se oli käytössä myös toimeksiantajayrityksessä. Google Cloudissa on myös oma palvelunsa ohjelmistokoodin versionhallintaan (Google 2021b), mutta ainostaan tätä projektia varten ei haluttu käyttää erillistä palvelua.

Automaattista jakeluputkea varten otettiin käyttöön Github Actions. Se on Githubin ominaisuus, joka mahdollistaa jatkuvan jakelun (engl. continuous delivery). Tämän sovelluksen kirjoitushetkellä Github Actions oli sen verran uusi ominaisuus, että Firebaseen liittyen siihen oli olemassa vain kolmannen osapuolen avointa lähdekoodia. Sittemmin Firebase on integroinut Github Actionit omaan palveluunsa helposti käyttöönotettavaksi (Google 2021d)

Kuvassa oleva koodi määrittelee, että kun uutta koodia päivitetään tietovaraston (engl. repository) master-haaraan, niin Github Actions jakelee sovelluksen automaattisesti Firebaseen. Koodissa oleva itse jakelu (rivi 16-17) Firebaseen käyttää Github Marketplacessa olevaa valmista Github Actionia.

Tämän projektin laajuuteen ei kuulunut automaattisten testien tai päähaaran sääntöjen konfigurointi osaksi jakeluputkea.

./.github/workflows/main.yml

```yaml
name: Deploy to Firebase

on:
  push:
    branches:
      - master

jobs:
  deploy:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repo
      uses: actions/checkout@master
    - name: Deploy to Firebase
      uses: w9jds/firebase-action@master
      with:
        args: deploy
      env:
        FIREBASE_TOKEN: ${{ secrets.FIREBASE_TOKEN }}
```

4.5 Testaus

Palvelimettomien sovellusten paikallinen kehittäjän omalla koneella tapahtuva testaus on edelleen haaste. Parhaiten testaus onnistuu mahdollisimman lähellä tuotantoa olevassa ympäristössä. Firebasen tapauksessa kuitenkin tiettyjä toimintoja pystyy lokaalisti emuloimaan. (Google 2021h) Tämän opinnäytetyön rajaukseen ei kuitenkaan kuulunut emulaatioympäristöön perehtyminen, vaan testaus tapahtui lähellä tuotantoympäristöä.

Opinnäytetyössä toteutetusta sovelluksesta pidin esittelytilaisuuden toimeksiantajayrityksen henkilöstölle. Esittelyn demon aikana sovelluksella oli 14 samanaikaista käyttäjää, eikä merkittäviä bugeja, virheitä tai haittoja ilmennyt sovelluksen käytössä.

5 Pohdinta ja johtopäätökset

Berkeleyn tutkimusryhmä ennustaa palvelimettomalle laskennalle räjähdysmäistä kasvua. Omiin konesaleihin pohjautuvat ratkaisut vähenevät ajan mittaan, ainoastaan lainsäädäntö, regulaatio tai datan hallintasäännöt saattavat pitää omissa konesaleissa kiinni. Berkeley tutkimusryhmä ennustaa, että sovellusten kirjo laajenee, mikä tahansa sovellus missä tahansa skaalassa tulee halvemmaksi palvelimettoman laskennan mallisena, sekä he ennustavat jopa niin radikaalisti että perinteinen palvelillinen malli tulee kuolemaan ja sen myötä asiakas-palvelin malli. (Jonas ym. 2019)

On todennäköistä, että kasvuluvut jatkuvat korkeina myös jatkossa ja toiminta on kannattavaa sekä palvelutarjoajille että käyttäjille. COVID-19 pandemia on vain kiihdyttänyt pilven omaksumista ja laajenemista. IDC-tutkimusyhtiön mukaan pilveen liittyvä kulutus kokonaisuudessaan maailmassa ylittää vuonna 2024 yhden biljoonan euron. (IDC 2020a)

Konttien ja virtuaalikoneiden orkestrointi on vahva kilpailija myös jatkossa. Palvelimettomat sovellukset kuitenkin palvelevat tällä hetkellä vielä kapeaa segmenttiä sovellustyypeistä. Vaikka tarjolla on viitekehyksiä, joilla pystyy mm. kuvaamaan funktiot koodina niin että ne eivät ole sidoksissa mihinkään tiettyyn palveluntarjoajaan, voi resurssien siirtäminen toiselle palvelutarjoajalle olla haaste.

Toiminnallisen osuuden osalta sovellus saavutti tavoitteet pääosin. Toimiva sovellus saatiin tuotantoon, konseptitodistuksen osalta riittävin ominaisuuksin. Lopullista konseptitodistusta pystyttiin esittelemään henkilöstölle ja käytettävissä olevat henkilöt testasivat sovellusta onnistuneesti.

Projektin aikana oppi todella paljon. Aihe osoittautui erittäin hedelmälliseksi, ajankohtaiseksi ja hyödylliseksi tulevaisuuden kannalta. Palvelimettomat teknologiat kehittyvät vauhdilla, ja jo vuoden päästä optimaalinen toteutus saattaisi näyttää arkkitehtuuriltaan hyvin erilaiselta.

Toteutettu sovellus osoitti käytännössä, että suhteellisen yksinkertaisesta palvelimettomasta web-sovelluksesta on melko helppo ja nopeaa kehittää prototyyppi ilman että kehittäjällä on valtavaa asiantuntemusta alla olevasta infrastruktuurista.

Jatkokehityksenä työkalussa olisi paljonkin tehtävää. Käyttäjäkokemus ja käyttöliittymä vaatisi paljon parantamista - käyttöön pitäisi ottaa jokin frontend-kehys (esim. React, Vue, Svelte tai Angular).

Toiminnan laajetessa arkkitehtuuria pitäisi miettiä uudelleen ja pohtia mahdollisia uusia palvelimettomia palveluja. Myös optimaaliset arkkitehtuurimallit tulevat todennäköiset muuttumaan ja kehittymään jatkossa.

Välttääkseen lukkiutumista palveluntarjoajaan tai helpottaakseen toiseen palveluntarjoajaan siirtymistä infrastruktuuri kannattaisi määritellä mahdollisimman paljon koodina jotain palveluntarjoaja-agnostista työkalua käyttäen (esim. Serverless Framework, Terraform, Pulumi) ja harkita konttipohjaisia ratkaisuja. Palvelimettomat kontitus-ratkaisut tarjoavat myös pidempää ajoaikaa pitkäkestoisille prosesseille.

Varsinaiseen yrityksen liiketoimintaongelmaan kehitetty työkalu ei tuonut vielä lopullista ratkaisua. Mutta tältä pohjalta uusien ratkaisujen kehittäminen ja jatkojalostaminen on varmasti halvempaa, helpompaa ja tuottavampaa.

6 Lähdeluettelo

Alfame 2021, *Alfame Systems Oy*. Saatavilla: https://www.alfame.com/ (Haettu 19.5.2021).

AWS 2019, *Serverless Applications Lens - AWS Well-Architected Framework*. Saatavilla: https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/welcome.html (Haettu 16.5.2021).

Castro, P., Ishakian, V., Muthusamy, V. & Slominski, A. 2019, The rise of serverless computing, *Communications of the ACM,* **62**(12), s. 44-54. DOI 10.1145/3368454.

CNCF 2018a, *The CNCF takes steps toward serverless computing* (Haettu 16.5.2021).

CNCF 2018b, *Serverless White Paper*. Saatavilla: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview (Haettu 23.5.2021).

DHTMLX 2021, *Interactive JavaScript/HTML5 Gantt Chart*. Saatavilla: https://dhtmlx.com/docs/products/dhtmlxGanttchart/ (Haettu 23.5.2021).

DHTMLX 2015, *How to Use DHTMLX Gantt Chart with FireBase Platform*. Saatavilla: https://dhtmlx.com/blog/use-dhtmlx-gantt-chart-with-firebase/ (Haettu 23.5.2021).

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C.L. & Iosup, A. 2021, Serverless Applications: Why, When, and How? *IEEE Software,* **38**(1), s. 32-39. DOI 10.1109/MS.2020.3023302.

Gartner 2020, *Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 18% in 2021*. Saatavilla: https://www.gartner.com/en/newsroom/press-releases/2020-11-17-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-18-percent-in-2021 (Haettu 23.5.2021).

Google 2021a, *Authenticate Using Google Sign-In with JavaScript*. Saatavilla: https://firebase.google.com/docs/auth/web/google-signin (Haettu 23.5.2021).

Google 2021b, *Cloud Source Repositories*. Saatavilla: https://cloud.google.com/source-repositories (Haettu 23.5.2021).

Google 2021c, *Compare AWS and Azure services to Google Cloud*. Saatavilla: https://cloud.google.com/free/docs/aws-azure-gcp-service-comparison (Haettu 16.5.2021).

Google 2021d, *Deploy to live & preview channels via GitHub pull requests*. Saatavilla: https://firebase.google.com/docs/hosting/github-integration (Haettu 23.5.2021).

Google 2021e, *Eventarc: A unified eventing experience in Google Cloud*. Saatavilla: https://cloud.google.com/blog/topics/developers-practitioners/eventarc-unified-eventing-experience-google-cloud (Haettu 16.5.2021).

Google 2021f, *Firebase -dokumentaatio*. Saatavilla: https://firebase.google.com/docs (Haettu 23.5.2021).

Google 2021g, *Firebase Realtime Database -dokumentaatio*. Saatavilla: https://firebase.google.com/docs/database (Haettu 23.5.2021).

Google 2021h, *Introduction to Firebase Local Emulator Suite*. Saatavilla: https://firebase.google.com/docs/emulator-suite (Haettu 23.5.2021).

Google 2020, *Accelerate your application development and delivery*. Saatavilla: https://cloud.google.com/blog/topics/google-cloud-next/developer-productivity-announcements-at-next20-onair (Haettu 16.5.2021).

IDC 2020a, *Cloud Adoption and Opportunities Will Continue to Expand Leading to a $1 Trillion Market in 2024, According to IDC*. Saatavilla: https://www.idc.com/getdoc.jsp?containerId=prUS46934120 (Haettu 23.5.2021).

IDC 2020b, *IDC FutureScape: Worldwide IT Industry 2021 Predictions*. Saatavilla: https://www.idc.com/getdoc.jsp?containerId=US46942020 (Haettu 16.5.2021).

Johann Schleier-Smith, Sreekanti, V., Khandelwal, A., Carreira, J., Neeraja, J.Y., Raluca, A.P., Joseph, E.G., Stoica, I. & David, A.P. 2021, What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing, *Communications of the ACM,* **64**(5), s. 76. DOI 10.1145/3406011.

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J.E., Popa, R.A., Stoica, I. & Patterson, D.A. 2019, Cloud Programming Simplified: A Berkeley View on Serverless Computing, .

Kehoe, B. 2017, *The Serverless Spectrum*. Saatavilla: https://ben11kehoe.medium.com/the-serverless-spectrum-147b02cb2292 (Haettu 16.5.2021).

Miller, R. 2021, *Cloud infrastructure spending passed on-prem data centers in 2020*. Saatavilla: https://techcrunch.com/2021/03/19/cloud-infrastructure-spending-passed-on-prem-data-centers-in-2020/ (Haettu 16.5.2021).

Narang, S. 2021, *Introducing CORPS: The 5 Pillars for a Robust Cloud Architecture Framework*. Saatavilla: https://thenewstack.io/introducing-corps-the-five-pillars-for-a-robust-cloud-architecture-framework/ (Haettu 19.5.2021).

Onrego 2020, *IaaS, CaaS, PaaS, FaaS, SaaS – mitä mikäkin tarkoittaa?*. Saatavilla: https://onrego.fi/julkisen-pilven-palvelumallit-avattuna/ (Haettu 19.5.2021).

Sullivan, D. 2019, *Official Google Cloud Certified Associate Cloud Engineer Study Guide*, John Wiley & Sons, Inc., Indianapolis, Indiana.

Taibi, D., Spillner, J. & Wawruch, K. 2021, Serverless Computing-Where Are We Now, and Where Are We Heading? *IEEE Software,* **38**(1), s. 25-31. DOI 10.1109/MS.2020.3028708.

Venema, W. 2021, *Building serverless applications with google cloud run : a real-world guide to building production-ready services* , 1. p., O'Reilly Media, Sebastopol, California.