Tampereen yliopisto

Teemu Salonen

PALVELIMETTOMIEN WEB-SOVELLUSTEN KEHITTÄMINEN AWS-YMPÄRISTÖSSÄ

Katsaus palveluihin, hyötyihin ja haasteisiin

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2024

TIIVISTELMÄ

Teemu Salonen: Palvelimettomien web-sovellusten kehittäminen AWS-ympäristössä: Katsaus palveluihin, hyötyihin ja haasteisiin
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2024

Tämä tutkielma on kirjallisuuskatsaus, jonka tavoitteena on tutkia palvelimettoman arkkitehtuurin hyödyntämistä web-sovellusten kehityksessä Amazon Web Services (AWS) -pilvipalvelualustalla. Tutkielmassa esitellään yleisesti oleellisimpien AWS-palveluiden rooli palvelimettomien web-sovellusten kehityksessä. Tutkittavia palveluita ovat: Amazon S3, Amazon Cognito, Amazon API Gateway, AWS Lambda ja Amazon DynamoDB. Lisäksi tutkielmassa tarkastellaan palvelimettoman arkkitehtuurin hyötyjä ja haasteita web-sovelluskehityksessä.

Työn tutkimusaineistona käytettiin alan tieteellisiä artikkeleita, kirjallisuutta, konferenssijulkaisuja ja verkkosivustoja. AWS-palveluita koskevaa tietoa haettiin myös palveluntarjoajan omista tuoteselosteista ja dokumentaatioista.

Palvelimeton arkkitehtuuri on noussut viime vuosina merkittäväksi lähestymistavaksi web-sovellusten kehityksessä. Se perustuu pilvipalveluihin kirjoitettuihin funktioihin, joita kutsutaan tapahtumapohjaisesti verkon välityksellä. Palvelimettoman arkkitehtuurin osaksi lasketaan myös pilvipalveluntarjoajien tarjoamat taustapalvelut, joita voi hyödyntää sovelluksen toiminnallisuuksien, kuten tietokannan, toteutuksessa. Palvelimeton arkkitehtuurimalli poistaa organisaatioilta huolen palvelimien hankinnasta, skaalautuvuudesta ja ylläpidosta, sillä nämä tehtävät siirtyvät pilvipalveluntarjoajien vastuulle. Lisäksi organisaatiot maksavat vain käyttämistään resursseista, mikä tekee palvelimettomasta arkkitehtuurista kustannustehokkaan vaihtoehdon.

Tutkimus osoittaa, että palvelimeton arkkitehtuuri soveltuu hyvin web-sovellusten kehitykseen sen tapahtumapohjaisen luonteen ansiosta. AWS-palvelut tukevat laajasti palvelimettomien web-sovellusten kehitystä tarjoamalla kattavan valikoiman työkaluja sovellusten eri osa-alueille. Palvelut mahdollistavat interaktiivisten, nopeiden ja skaalautuvien web-sovellusten kehittämisen. Palvelimettomalla arkkitehtuurilla on useita hyötyjä web-sovelluskehityksessä, kuten automaattinen skaalautuvuus, kustannustehokkuus ja nopeampi sovelluskehitys. Kuitenkin tutkimuksessa havaitaan myös palvelimettoman mallin haasteet, kuten riippuvuus pilvipalveluntarjoajasta, sovellusten monimutkaistuminen, työkalujen rajallisuus ja latenssiongelmat. Organisaatioiden tulisi arvioida mallin hyötyjä ja haasteita huolellisesti harkitessaan palvelimettoman arkkitehtuurin ratkaisua web-sovelluksissaan.

Avainsanat: pilvipalvelut, palvelimeton arkkitehtuuri, web-sovelluskehitys, AWS

SISÄLLYSLUETTELO

1 Johdanto

Web-sovelluskehityksen ala on jatkuvassa muutoksessa, ja uusia innovatiivisia lähestymistapoja esiintyy jatkuvasti. Viime vuosien aikana pilvipalveluiden ja erityisesti palvelimettoman (engl. serverless) arkkitehtuurin rooli web-sovellusten rakentamisessa on kasvattanut suosiotaan. Datadogin (2023) raportin mukaan suurimmat pilvipalveluntarjoajat todistavat merkittävää kasvua palvelimettomien palveluiden käyttöönotossa. Suosituimpia pilvipalvelualustoja kehittäjien keskuudessa ovat Amazon Web Services (AWS), Microsoft Azure ja Google Cloud (Stack Overflow, 2023). Palvelimettoman arkkitehtuurin tarjoama kustannustehokkuus, skaalautuvuus ja kevyempi operatiivinen taakka ovat suurimpia syitä siirtyä käyttämään palvelimetonta mallia (Eismann ym., 2021).

Palvelimettoman arkkitehtuurin keskeinen piirre on, että kehittäjien ei tarvitse huolehtia infrastruktuuriresurssien hallinnasta. Sen sijaan pilvipalveluntarjoaja, kuten AWS, vastaa sovelluksen taustalla olevasta infrastruktuurista, mikä vapauttaa kehittäjät keskittymään yksinomaan sovelluskehitykseen ja liiketoimintatavoitteiden saavuttamiseen (Koschel ym., 2021).

Tämän kandidaatintutkielman tavoitteena on tutkia palvelimettoman arkkitehtuurin hyödyntämistä web-sovellusten kehityksessä Amazon Web Services -pilvipalvelualustalla. AWS valittiin alustaksi sen aseman vuoksi yhtenä suurimpana ja käytetyimpänä pilvipalvelualustana. Tutkielmassa tarkastellaan AWS-palveluita, jotka ovat olennaisia palvelimettomien web-sovellusten kehittämisessä. Näitä palveluita ovat Amazon S3, Amazon Cognito, Amazon API Gateway, AWS Lambda ja Amazon DynamoDB. Tutkielmassa ei syvennytä web-kehityksen tai AWS-palveluiden teknisiin yksityiskohtiin, vaan tarkoituksena on yleisesti arvioida palveluiden roolia palvelimettomien web-sovellusten toteutuksessa. Tavoitteena on myös analysoida palvelimettoman lähestymistavan tuomia hyötyjä ja mahdollisia haasteita web-sovelluskehityksen näkökulmasta.

Tutkimuksen tulokset osoittavat, että palvelimeton arkkitehtuuri soveltuu hyvin web-sovellusten kehitykseen sen tapahtumapohjaisuuden ansiosta. AWS-palvelut tukevat laajasti palvelimettomien web-sovellusten kehitystä tarjoamalla kattavan valikoiman työkaluja sovellusten eri osa-alueille. Palvelimettomalla arkkitehtuurilla on useita merkittäviä hyötyjä web-sovelluskehityksessä, kuten skaalautuvuus, kustannustehokkuus ja lyhyempi kehitysaika. Lisäksi malli mahdollistaa nopeamman sovellusten käyttöönoton ja helpomman ylläpidon. Tutkimuksessa havaitaan myös, että palvelimettoman arkkitehtuurin käyttöön liittyy joitakin haasteita, kuten riippuvuus palveluntarjoajasta, sovelluksen monimutkaistuminen ja mahdolliset latenssiongelmat.

Tutkielma on toteutettu kirjallisuuskatsauksena. Aineistona on käytetty alan tieteellisiä artikkeleita, konferenssijulkaisuja, kirjoja ja verkkosivustoja. AWS-palveluita tarkastellessa on käytetty lähteinä enemmän kirjoja ja verkkosivuja, sillä tutkimuksia palve-

luista oli rajallisesti. Palveluita koskevaa tietoa haettiin esimerkiksi AWS:n omista tuoteselosteista ja dokumentaatioista. Lähteiden haussa käytettyjä tietokantoja ovat ACM Digital Library, Andor, Google Scholar, IEEE Xplore ja SpringerLink. Haut on tehty englanniksi suomenkielisen aineiston vähyyden vuoksi. Hakuja on tehty eri lausekkeilla ja niiden yhdistelmillä. Hakulausekkeita on yhdistelty AND- ja OR-operaattoreilla. Käytettyjä hakulausekkeita ovat olleet esimerkiksi ”serverless”, ”serverless architecture”, ”Amazon Web Services”, ”AWS”, ”web” ja näiden yhdistelmät.

Tutkielma jakautuu kuuteen lukuun. Toisessa luvussa käsitellään palvelimettoman arkkitehtuurin perusteita ja sen mallia web-sovelluksissa. Kolmannessa luvussa tarkastellaan yleisesti Amazon Web Services -pilvipalvelualustaa ja sen tarjoamia palveluita palvelimettoman web-sovelluskehityksen näkökulmasta. Neljännessä luvussa tutkitaan viittä keskeistä AWS-palvelua ja niiden roolia palvelimettomissa web-sovelluksissa. Viidennessä luvussa arvioidaan palvelimettoman arkkitehtuurin hyötyjä ja haasteita. Lopuksi luvussa kuusi esitetään tutkielman yhteenveto, jossa tiivistetään tärkeimmät tutkielmassa havaitut asiat.

2 Palvelimeton arkkitehtuuri

Tässä luvussa käsitellään palvelimettoman arkkitehtuurin perusteita sekä sen mallia web-sovelluksissa. Aluksi tarkastellaan palvelimettoman arkkitehtuurin keskeisiä käsitteitä ja periaatteita. Tämän jälkeen syvennytään tarkemmin palvelimettomaan malliin web-sovelluksissa hyödyntäen esimerkkiä lemmikkiverkkokaupasta.

2.1 Palvelimettoman arkkitehtuurin perusteet

Palvelimeton arkkitehtuuri edustaa uutta lähestymistapaa web-sovelluskehityksessä perinteisen monoliittisen mallin sijaan. Vaikka mallin nimi onkin ”palvelimeton”, fyysiset palvelimet ovat edelleen olemassa. Pilvipalveluntarjoaja hallinnoi niitä dynaamisesti kysynnän mukaan, joten kehittäjien ei tarvitse olla niistä tietoisia. (Sewak & Singh, 2018.) Perinteisessä mallissa sovelluksen kehittäjät vastaavat itse sovelluksen infrastruktuurin ylläpidosta ja resurssien hallinnasta. Shafiei ja kumppanit (2022) korostavat, että palvelimettomassa arkkitehtuurimallissa päinvastoin sovelluksen ajoympäristö ja infrastruktuuri ovat piilossa sovelluskehittäjiltä. Kehittäjät voivat keskittyä ydintoiminnallisuuden toteuttamiseen, kun pilvipalveluntarjoajat huolehtivat infrastruktuurin hallinnasta ja ylläpidosta.

Palvelimeton arkkitehtuuri perustuu pääosin kehittäjien kirjoittamiin pilvipalveluissa suoritettaviin funktioihin, joita kutsutaan verkon välityksellä. Tämä Functions-as-a-Service (FaaS) -malli mahdollistaa ohjelmistojen kehittämisen ilman tarvetta rakentaa ja ylläpitää omaa ajoympäristön infrastruktuuria. (Li ym., 2022.) Esimerkki FaaS-palvelusta on Amazonin vuonna 2014 julkaisema AWS Lambda. Funktioita kutsutaan tapahtumapohjaisesti, minkä ansiosta resursseja käytetään vain suorituksen ajan, jonka jälkeen ne vapautetaan (Rajan, 2018). Esimerkiksi käyttäjän klikkaus käyttöliittymäelementtiin voi laukaista funktion suorituksen. FaaS on palvelimettoman laskennan käsite palvelimettomassa arkkitehtuurissa. Funktiot ovat tilattomia: ne alustetaan kutsusta kontteihin, ajetaan ja lopuksi kontit tuhotaan. (Koschel ym., 2021.)

Pilvipalveluntarjoajat tarjoavat myös Backend-as-a-Service (BaaS) -palveluita, jotka ovat täysimuotoisia palveluita taustaohjelmille, kuten tietokannalle. BaaS-palvelut ovat vastuussa tietyistä sovelluksen osa-alueista, kuten käyttäjienhallinnasta. Samoin kuin FaaS-palvelut, myös BaaS-palvelut vapauttavat kehittäjät resurssien hallinnasta, sillä palveluntarjoaja hoitaa automaattisesti resurssien käytön ja skaalaamisen. (Shafiei ym., 2022.) Esimerkki BaaS-palvelusta on Amazonin DynamoDB-tietokanta.

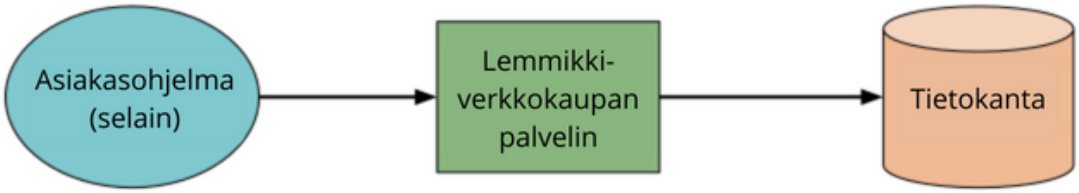
Shafiein ja kollegoiden (2022) mukaan palvelimeton arkkitehtuuri voidaan nähdä FaaS- ja BaaS-palveluiden yhdistelmänä. Sovelluksen tietojenkäsittely tapahtuu pilvipalvelualustalla FaaS-palvelun avulla, kun taas taustaohjelmissa, kuten autentikoinnissa tai tietokannan hallinnassa, voidaan hyödyntää pilvipalveluntarjoajan valmiita BaaS-ratkaisuja. Tapahtuman perusteella pyynnöt ohjataan FaaS-palveluun funktioiden suoritettaviksi. BaaS-palvelut voivat toimia usein suurelta osin itsenäisesti. Pilvipalveluntarjoajan

palvelut integroituvat yleisesti saumattomasti toistensa kanssa, mikä helpottaa niiden yhdistämistä osaksi sovellusarkkitehtuuria.

Palvelimettoman arkkitehtuurin oleellinen tunnus on se, että käyttäjät maksavat ainoistaan käyttämistään resursseista esimerkiksi pyyntöjen lukumäärän tai suorituksessa käytetyn ajan perusteella. Hassan ja kumppanit (2021) havainnollistavat maksumallia siten, että palvelimetonta mallia käytettäessä sovelluksen ollessa toimettomana, ei asiakkaalle synny lainkaan kustannuksia. Laskutus alkaa vasta silloin, kun sovellus on aktiivinen ja se hyödyntää pilvipalveluntarjoajan resursseja. Palvelimeton arkkitehtuuri poistaa perinteisen mallin ylläpitokustannukset, jotka ovat jatkuvia, vaikka sovelluksella ei tiettyinä ajankohtina olisikaan ollenkaan käyttöä.

2.2 Palvelimeton arkkitehtuuri web-sovelluksissa

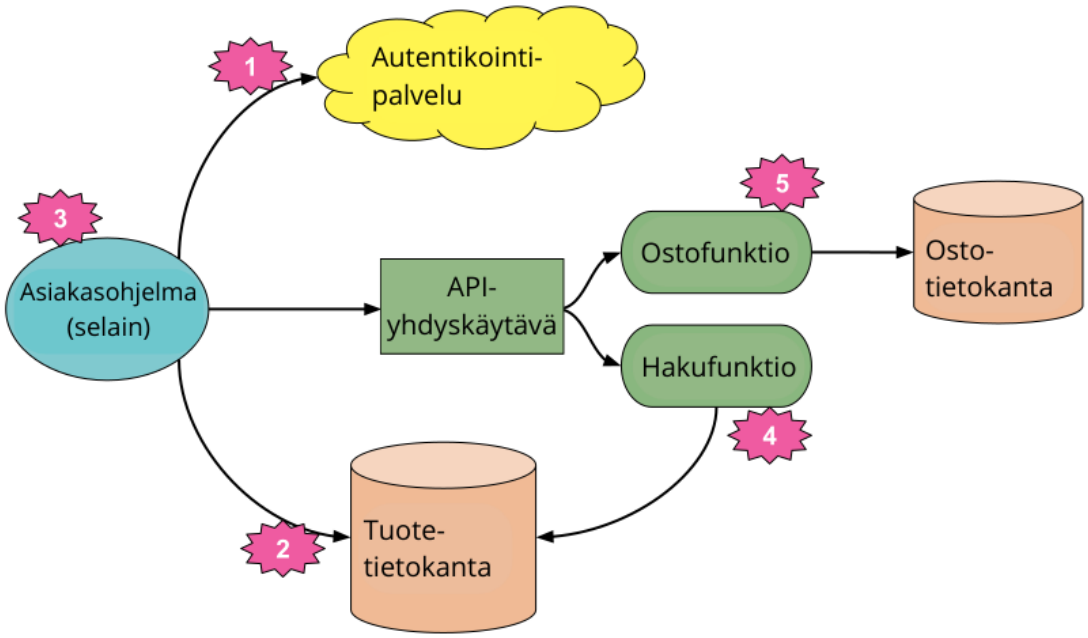
Perinteisessä kolmitasoisessa monoliittisessa web-sovellusarkkitehtuurissa on yleensä kolme pääkomponenttia: selaimen kautta käytettävä käyttöliittymä, palvelin ja tietokanta. Kuvassa 1 on esitetty yksinkertainen rakenne tällaisesta sovelluksesta käyttäen esimerkkinä lemmikkiverkkokauppaa. Tämä perinteinen monoliittinen web-sovellusarkkitehtuuri perustuu yhden palvelimen ylläpitoon, joka käsittelee suurinta osaa sovelluksen toiminnoista. Tämä tarkoittaa, että liiketoimintalogiikka, kuten tietokantakyselyt ja käyttäjien autentikointi, tapahtuvat samassa ympäristössä eli tässä tietyssä palvelimessa. Koska suurin osa toiminnoista on keskitetty palvelimelle, käyttöliittymän toteutus voi olla suhteellisen yksinkertainen. (Roberts, 2018.)



Kuva 1. Perinteisen kolmitasoisen monoliittisen web-sovelluksen yksinkertaistettu arkkitehtuuri (mukaillen Roberts, 2018).

Vaikka perinteinen monoliittinen malli on usein riittävä ja toimiva, etenkin sovelluksen kasvaessa se voi aiheuttaa haasteita. Koschel ja kumppanit (2021) nostavat esiin palvelimien hankinnan ja ylläpidon kustannukset sekä skaalautuvuuden vaikeudet. Esimerkiksi lemmikkiverkkokaupan käyttäjämäärän kasvaessa palvelimen resurssit voivat osoittautua liian rajallisiksi, mikä puolestaan johtaa hidastuvaan suorituskykyyn ja heikentyneeseen käyttäjäkokemukseen. Palvelimen skaalaaminen vastaamaan kasvavaa käyttäjämäärää voi myös olla sekä kallista että aikaa vievää.

Palvelimeton arkkitehtuuri tarjoaa perinteiselle monoliittiselle mallille vastakkaisen, hajautetun lähestymistavan. Sovellus jaetaan pienempiin ja itsenäisiin komponentteihin: funktioihin ja tiettyihin tehtäviin erikoistuneisiin palveluihin. Nämä komponentit mahdollistavat sovelluksen eri toiminnallisuuksien suorittamisen erillisissä pilvipalveluntarjoajan palveluissa. Kuvassa 2 on esitetty yksinkertaistettu palvelimetonta arkkitehtuuria hyödyntävä versio aiemmasta kuvan 1 lemmikkiverkkokaupasta. Kuvassa 2 esitetty API-yhdyskäytävä (engl. API gateway) toimii porttina ja ohjelmointirajapintana sovelluksen käyttöliittymän ja taustapalveluiden välillä. Ostotapahtumista ja hakutapahtumista vastaavat FaaS-palvelun funktiot reagoivat tämän API-yhdyskäytävän kautta saapuviin tapahtumiin, kuten käyttäjän klikkauksiin käyttöliittymässä. Käyttäjän vuorovaikuttaessa käyttöliittymän kanssa pyynnöt ohjataan API-yhdyskäytävän kautta funktioille. Funktiot suorittavat toimenpiteen ja palauttavat tuloksen käytävän kautta takaisin käyttöliittymään käyttäjän nähtäväksi. Kuvassa 2 näkyvät myös autentikoinnista vastaava BaaS-palvelu, sekä kaksi erillistä tietokantaa tuotteita ja ostotapahtumia varten. Tässä palvelimetonta arkkitehtuuria hyödyntävässä versiossa autentikointi suoritetaan erillisesti käyttöliittymän ja autentikointipalvelun välillä eikä yhteyttä API-yhdyskäytävän välityksellä taustapalveluihin tarvita. Tietokannat voivat olla pilvipalveluntarjoajan tarjoamia BaaS-vaihtoehtoja. Tuotetietokanta on käytettävissä sekä käyttöliittymän että funktioiden kautta, kun taas ostoja koskeva tietokanta on saatavilla ainoastaan ostofunktion kautta. (Roberts, 2018.)



Kuva 2. Palvelimettoman web-sovelluksen yksinkertaistettu arkkitehtuuri (mukaillen Roberts, 2018).

Kuvassa 1 esitetyn monoliittisen version tapauksessa suurin osa toiminnallisuuksista toteutettiin yhden keskitetyn palvelimen kautta. Sen sijaan kuvassa 2 esitetyssä palvelimettomassa versiossa ei ole keskitettyä hallinnoijaa toiminnoille. Palvelimettomassa versiossa käytetään koreografiaa, jossa jokainen komponentti toteuttaa tietyn roolin ja toimii arkkitehtuurillisesti tietoisesti. (Roberts, 2018.) Tämä palvelimeton versio tarjoaa automaattisen skaalautuvuuden, ja laitteiston ylläpidosta vastaa pilvipalveluntarjoaja. Näin sovellus kykenee vastaamaan nopeastikin kasvaviin käyttäjämääriin, ja kehittäjät voivat keskittyä lisätoiminnallisuuksien toteuttamiseen palvelimen ylläpidon ja konfiguroinnin sijaan. Robertsin (2018) mukaan kuitenkin palvelimettomalla mallilla on omat kompromissinsa verrattuna monoliittiseen malliin. Siinä on enemmän komponentteja, jotka vaativat monitorointia, mikä monimutkaistaa sovelluksen rakennetta. Palvelimettoman arkkitehtuurin hyötyjä ja haasteita tarkastellaan tarkemmin luvussa viisi.

3 Amazon Web Services (AWS)

Tässä luvussa tarkastellaan yleisesti Amazon Web Services (AWS) -pilvipalvelualustaa ja sen perusteita. Lisäksi luvussa tarkastellaan esimerkkiarkkitehtuurikaavion avulla, kuinka AWS-ympäristön palveluita voidaan hyödyntää palvelimettomien web-sovellusten toteutuksessa.

3.1 AWS:n yleiskatsaus ja perusteet

AWS on maailman johtava pilvipalvelualusta, joka tarjoaa laajan valikoiman palveluita web-sovellusten eri toiminnallisuuksien toteuttamiseen. Vuoden 2023 Stack Overflow -sivuston kehittäjäkyselyn mukaan AWS on selkeästi suosituin pilvipalvelualusta, ja sen käyttöaste on lähes kaksinkertainen verrattuna seuraavaksi suosituimpaan vaihtoehtoon. Kyselyn mukaan AWS:n suurimpia kilpailijoita ovat Microsoft Azure ja Google Cloud. AWS on tarjonnut infrastruktuuripilvipalveluita organisaatioille vuodesta 2006 lähtien. Tänä päivänä sadat tuhannet yritykset yli 190 maasta hyödyntävät AWS-palveluita eri tehtävissä. (AWS Whitepaper, 2024.) Sewakin ja Singhin (2018) mukaan esimerkiksi yksi maailman suurin suoratoistopalvelu, Netflix, käyttää AWS-työkaluja palveluidensa toteuttamiseen ja ylläpitoon.

AWS tarjoaa palveluita esimerkiksi web-sovellusten koodin suorittamiseen (AWS Lambda), tietojen hallintaan (DynamoDB), pyyntöjen ohjaamiseen (API Gateway), staattisen datan tallennukseen ja jakeluun (S3), sekä autentikointiin (Cognito). AWS Lambdan kanssa yhteensopivia AWS-palveluita on yli 200, ja ne tarjoavat automaattisen skaalautuvuuden ja korkean saatavuuden. AWS mahdollistaa kehittäjille monipuoliset työkalut sovellusten rakentamiseen ja laajentamiseen ilman tarvetta huolehtia infrastruktuurin ylläpidosta ja resurssien hallinnasta. (AWS, 2024a.)

AWS-palveluiden hinnoittelu perustuu todelliseen käyttöön. Hinnoittelu tapahtuu käytettyjen resurssien mukaan, eikä irtisanomismaksuja ole. Palveluiden laskutus on jaettu kolmeen pääkategoriaan: laskenta, tiedon talletus ja tiedon siirto. Laskenta hinnoitellaan ajan, kuten sekuntien, mukaan, kun taas tiedon talletuksen ja siirron osalta hinnoittelu perustuu yleensä gigatavujen määrään. Usein tiedon siirto on ilmaista AWS-palveluiden kesken tietyn alueen, kuten Euroopan, sisällä, mutta alueiden välillä siirto voi olla kalliimpaa. Amazonin palveluita on mahdollista testata ja käyttää myös ilmaiseksi tietyn ajanjakson ajan tai tiettyyn datarajaan asti. Näin käyttäjillä on mahdollista tutustua palveluihin ja testata niiden sopivuutta omiin tarpeisiinsa. Koska AWS-palveluiden hinnoittelu perustuu vain kulutettuihin resursseihin, ovat kulut usein alhaisemmat kuin jos palveluita ylläpitäisi itse. (AWS Whitepaper, 2023.) Kuitenkin koska palveluiden hinnoittelu koostuu useista eri osa-alueista, kuten siirretyn datan määrästä, voi lopullisten kustannusten arvioiminen olla usein hankalaa. Hinnoittelu voi myös vaihdella ajanjaksojen välillä paljon riippuen käytön määrästä ja käyttökohteista.

3.2 AWS:n hyödyntäminen palvelimettomien web-sovellusten rakentamisessa

AWS tarjoaa laajan valikoiman palveluita, jotka mahdollistavat web-sovellusten eri toiminnallisuuksien toteuttamisen. On mahdollista rakentaa sovellus täysin Amazon Web Servicesin palveluiden varaan, mikä tarjoaa kokonaisvaltaisen ja integroidun kehitysympäristön. Tämä kattaa kaiken laskennasta tietojen tallennukseen ja käyttäjienhallintaan. AWS antaa kehittäjille joustavuutta arkkitehtuurin eri vaihtoehtojen suhteen samalla poistaen huolen ympäristön hallinnasta ja skaalaamisesta.

Kuva 3 esittää esimerkkiarkkitehtuurin yksinkertaiselle palvelimettomalle web-sovellukselle, joka on rakennettu pelkästään AWS-palveluita hyödyntämällä. Käyttöliittymä on käyttäjän vuorovaikutuskohta sovelluksen kanssa. Kuvassa 3 Amazon S3 -palvelu vastaa käyttöliittymän staattisen sisällön, kuten HTML- ja JavaScript-tiedostojen säilytyksestä ja jakelusta käyttäjille heidän avatessaan sovelluksen. Käyttöliittymästä lähetetään tapahtumien, kuten käyttäjän klikkauksien, perusteella pyyntöjä Amazon API Gateway -palvelulle, joka toimii rajapintana käyttöliittymän ja AWS Lambda -palvelun välillä. API Gateway ohjaa saapuvat pyynnöt AWS Lambda -funktioille. Lambda-funktiot ovat Lambda-palveluun ladattuja pieniä, itsenäisiä koodinpalasia, jotka suorittavat tiettyjä toiminnallisuuksia tapahtumien mukaan. Näitä funktioita voidaan käyttää esimerkiksi liiketoimintalogiikan toteuttamiseen, kuten tietokantakyselyjen suorittamiseen. Tässä esimerkissä on käytetty tietokantana Amazon DynamoDB -palvelua. Amazon Cognito -palvelun vastuulla on käyttäjien rekisteröinti ja autentikointi sovellukseen. Autentikointi voidaan tehdä suoraan käyttöliittymän ja Cogniton välillä, eikä yhteyttä taustapalveluihin tarvita. Luvussa 4 tutkitaan tarkemmin kuvan 3 AWS-palveluita ja niiden rooleja palvelimettomissa web-sovelluksissa.

Kuva 3. Yksinkertainen arkkitehtuuri AWS-palveluilla rakennetulle palvelimettomalle web-sovellukselle (mukaillen AWS, 2024b).

4 Palvelimettomien web-sovellusten kehittäminen AWS-palveluilla

Tässä luvussa tutkitaan AWS-palveluita, jotka ovat keskeisiä yksinkertaisten palvelimettomien web-sovellusten rakentamisessa. Luvussa tutkitaan yleisesti näiden palveluiden ominaisuuksia ja niiden roolia palvelimetonta arkkitehtuuria hyödyntävissä web-sovelluksissa. Esimerkkinä käytettävän tarkasteltavan sovelluksen rakenteena toimii kuvan 3 mukainen yksinkertainen arkkitehtuuri. Sovelluksen käyttöliittymäosaan kuuluvat palvelut Amazon S3 ja Amazon Cognito, kun taas taustapalveluihin kuuluvat palvelut Amazon API Gateway, AWS Lambda ja Amazon DynamoDB.

4.1 Amazon S3: käyttöliittymätiedostojen isännöinti ja jakelu

Amazon Simple Storage Service (S3) -palvelu on olennainen osa palvelimettomien web-sovellusten käyttöliittymän toteutusta. Sovellusten käyttöliittymän staattiset resurssit, kuten HTML-, JavaScript- ja CSS-tiedostot tallennetaan ja jaetaan S3-palvelun avulla. Amazon S3 on skaalautuva tallennuspalvelu, joka sopii erinomaisesti staattisen datan säilytykseen ja jakeluun. S3 vastaa käyttöliittymän tiedostoista, kun taas AWS Lambda hoitaa taustapalveluiden tapahtumapohjaisen koodin suorituksen. Tiedostot ladataan S3-palvelussa ”ämpäreihin” objekteina. S3 on myös konfiguroitava mahdollistamaan verkkosivustojen isännöinti. (Zanon, 2017.) Amazon S3 tarjoaa myös salauksen ja autentikoinnin talletetulle datalle, ja sen avulla voidaan määrittää, kuka pääsee käsiksi mihinkin dataan. Lisäksi S3 mahdollistaa tiedostojen varmuuskopioinnin ja palauttamisen virhetilanteissa. (Kulkarni ym., 2012.)

Vaikka Amazon S3 -palvelua käytetään vain staattisten resurssien tallentamiseen ja jakeluun, on palvelua käyttämällä mahdollista rakentaa dynaamisia web-sovelluksia hyödyntäen eri JavaScript-kirjastoja ja -kehyksiä. Suosituimpia valintoja kehittäjien keskuudessa ovat React, Angular ja Vue.js (Stack Overflow, 2023). Näiden kirjastojen tai kehyksien avulla toteutettu käyttöliittymä ohjelmoidaan kommunikoimaan taustapalveluiden, kuten AWS Lambdan, kanssa Amazon API Gateway -palvelun läpi. Tämä mahdollistaa selkeän rajapinnan käyttöliittymän ja taustajärjestelmän välillä.

Kulkarnin ja kumppanien (2012) mukaan käyttöliittymän tiedostoja voidaan jakaa suoraan käyttäjille S3-palvelun kautta. Kuitenkin, jotta käyttöliittymän staattisia resursseja voidaan toimittaa käyttäjille nopeammin ja tehokkaammin, voidaan hyödyntää sisällönjakeluverkkoa (engl. content delivery network). Amazonin sisällönjakeluverkkopalvelu Amazon CloudFront voi välittää sovelluksen tiedostot käyttäjille lähempänä sijaitsevilta palvelimilta, mikä parantaa latausnopeutta ja käyttökokemusta merkittävästi (Zanon, 2017). Pattersonin (2019) mukaan Amazon CloudFront tarjoaa myös lisätoimintoja hallintaan ja reititykseen.

Amazon S3 on suunniteltu erittäin luotettavaksi jopa 99,999999999 prosentin varmuudella. Palvelua käyttävät miljoonat sovellukset ympäri maailmaa. S3:n hinnoittelu

perustuu tallennetun datan määrään sekä tietojen jakelun ja tehtyjen pyyntöjen lukumäärään. Tiedostojen jakelussa mahdollisesti käytettävän CloudFrontin hinnoittelu perustuu yhtäläisesti pyyntöjen ja toteutettujen jakeluiden lukumäärään. (AWS Whitepaper, 2023.)

Kaiken kaikkiaan Amazon S3 -palvelu tarjoaa tehokkaan ja skaalautuvan ratkaisun käyttöliittymätiedostojen säilytykseen ja jakeluun palvelimettomissa web-sovelluksissa. Resurssit ladataan S3-ämpäriin ja jaetaan käyttäjille joko suoraan S3:n tai mahdollisesti Amazon CloudFront -palvelun kautta. Näiden teknologioiden yhdistelmä mahdollistaa sulavan käyttökokemuksen ja nopean latausajan, mikä on keskeistä modernin web-sovelluksen menestyksen kannalta.

4.2 Amazon Cognito: käyttäjien identiteetin hallinta

Käyttäjäidentiteetin hallinta ja autentikointi muodostavat olennaisen osan modernien web-sovellusten kehitystä. Amazon Cognito -palvelu nousee keskeiseksi osaksi tarjoamalla kattavan ratkaisun näihin kysymyksiin. Sen avulla voi rekisteröidä ja autentikoida palvelun käyttäjiä ilman erillistä käyttäjätietokantaa. (Sbarski ym., 2022.) Singhin ja kollegoiden (2023) mukaan Cognito mahdollistaa myös ulkopuolisten palveluiden, kuten Googlen tai Facebookin, käyttämisen kirjautumisen helpottamiseksi ja käyttäjäkokemuksen parantamiseksi. Cognito tekee lisäksi käyttäjien tunnistamisesta turvallista. Palvelu tarjoaa edistyneitä turvaominaisuuksia, kuten riskipohjaisen mukautuvan tunnistautumisen, vaarantuneiden tunnusten seurannan ja työkaluja bottikäyttäjien havaitsemiseen (Amazon Cognito, 2024). Cognito on automaattisesti skaalautuva palvelu, joten sovelluksella voi olla muutamista jopa miljooniin käyttäjiä (AWS Whitepaper, 2024).

Cognitoon luodaan käyttäjäallas, joka mahdollistaa rekisteröinnin ja kirjautumisen sovellukseen. Käyttöliittymä voidaan yhdistää suoraan Amazon Cognitoon ilman AWS Lambdaa. Näin ollen autentikointipyynnöt voidaan tehdä yksinkertaisesti suoraan käyttöliittymän ja Cogniton välillä. (Zanon, 2017.) Lisäksi Cognito voidaan integroida moneen muuhun AWS-palveluun, mikä mahdollistaa hienovaraisen pääsynhallinnan eri palveluiden resursseihin (AWS Documentation, 2024). Näin vain auktorisoidut käyttäjät voivat käyttää sovelluksen resursseja, mikä parantaa sovelluksen tietoturvaa ja yksityisyyttä.

Amazon Cogniton hinnoittelu perustuu aktiivisten käyttäjien määrään kuukaudessa. Palvelu on ilmainen, jos aktiivisia käyttäjiä on alle 50 000 kuukaudessa. Rajan ylittyessä palvelu laskuttaa jokaisesta rajan ylittävästä käyttäjästä erikseen. Aktiiviseksi lasketaan käyttäjä, joka on käyttänyt palvelua esimerkiksi rekisteröitymällä, kirjautumalla tai vaihtamalla salasanan. (Amazon Cognito, 2024.)

Kokonaisuudessaan Amazon Cognito -palvelu tarjoaa luotettavan ja helppokäyttöisen ratkaisun web-sovellusten käyttäjäidentiteetin hallintaan ja autentikointiin. Se vapauttaa kehittäjät rakentamasta omia käyttäjähallintaratkaisuja, integroituu saumattomasti muihin AWS-palveluihin ja tarjoaa joustavan hinnoittelumallin.

4.3 Amazon API Gateway: skaalautuvat ohjelmointirajapinnat

Palvelimettomia web-sovelluksia rakennettaessa AWS-ympäristössä nousee Amazon API Gateway -palvelu keskeiseen rooliin. Palvelun avulla voi rakentaa ohjelmointirajapintoja (engl. application programming interface, API) käyttöliittymän ja taustapalveluiden välille. Sen avulla voi rakentaa web-sovelluksille tyypillisiä REST-rajapintoja. (AWS Documentation, 2024.) Anandin ja kollegoiden (2023) mukaan Amazon API Gateway -palvelun avulla kehittäjät voivat rakentaa, julkaista, ylläpitää, monitoroida ja turvata rajapintoja. Muiden AWS-palveluiden tapaan API Gateway on myös automaattisesti skaalautuva. AWS-palveluiden dokumentaatiossa (2024) kerrotaan, että palvelu kykenee prosessoimaan satojatuhansia samanaikaisia rajapintakutsuja. API Gatewayn hinnoittelu määräytyy kutsujen lukumäärän ja siirretyn datan määrän perusteella.

API Gateway on oleellinen osa palvelimetonta web-sovellusta, sillä se ohjaa käyttöliittymästä tulevat HTTP-pyynnöt oikeiden Lambda-funktioiden suoritettavaksi. API Gateway tekee välikappaleena mahdolliseksi Lambda-funktioiden kutsun ja suorituksen, kun käyttäjä vuorovaikuttaa käyttöliittymän kanssa. Lambda-funktiot suorittavat jonkin toimenpiteen, kuten tietokantakyselyn, ja palauttavat mahdollisen tuloksen API Gatewayn kautta takaisin käyttöliittymään. Tämä mahdollistaa käyttöliittymän nopean päivityksen ja reagoinnin käyttäjän toimiin.

Yhteenvetona Amazon API Gateway -palvelu on olennainen osa palvelimettomien web-sovellusten infrastruktuuria AWS-ympäristössä. tarjoten monipuolisen ja tehokkaan työkalun ohjelmointirajapintojen hallintaan ja integrointiin. Se tarjoaa nopean, helppokäyttöisen ja luotettavan tavan rakentaa ja hallita ohjelmointirajapintoja, samalla varmistaen skaalautuvuuden, turvallisuuden ja integroitavuuden.

4.4 AWS Lambda: tapahtumapohjainen tietojenkäsittely

AWS Lambda on palvelimettoman arkkitehtuurin kannalta merkittävä palvelu, joka mahdollistaa koodin suorittamisen pilvessä ilman, että kehittäjien tarvitsee huolehtia ympäristöstä. Lambda-funktioiden koodi käynnistetään automaattisesti suoritukseen tapahtuman, kuten käyttäjän klikkauksen, perusteella. Tämä tekee Lambda-funktioista erittäin joustavia, sillä ne reagoivat dynaamisesti tuleviin pyyntöihin. AWS Lambda skaalautuu automaattisesti saapuvien pyyntöjen mukaan. (AWS Documentation, 2024.)

Singhin ja kumppaneiden (2023) mukaan AWS Lambda tukee monia ohjelmointikieliä, mikä tekee siitä helposti lähestyttävän palvelun eri taustat omaaville kehittäjille. Tuettuja kieliä ovat muun muassa JavaScript, Java ja Python (AWS Documentation, 2024). Tämä monipuolisuus mahdollistaa kehittäjille tuttujen ohjelmointikielten käytön Lambda-funktioiden toteuttamisessa, eikä aikaa kulu uuden kielen opettelemiseen.

AWS Lambda on keskeinen osa palvelimettomien web-sovellusten kehitystä, sillä se vastaa itse toiminnallisuuden suorittamisesta. Lambda-funktiot integroituvat saumatto-

masti muiden AWS-palveluiden kanssa. Funktiot voidaan konfiguroida esimerkiksi vastaamaan muiden palveluiden tapahtumiin. (Patterson, 2019.) Oleellisimmat tapahtumat web-sovelluksissa ovat API Gatewayn kautta saapuvat kutsut, jotka laukaisevat Lambda-funktioita suoritukseen.

Lambdan käytön hinnoittelu perustuu kutsuttujen funktioiden määrään, niiden suoritusaikaan ja suorituksessa käytettyyn muistiin. Suoritusaika lasketaan funktion suorituksen aloituksesta sen loppumiseen pyöristettynä lähimpään millisekuntiin. AWS Lambda tarjoaa kuukausittain miljoona pyyntöä ilmaiseksi, jonka ylittyessä jokainen pyyntö maksaa erikseen. Lisäksi kuukauden ensimmäiset 400 000 gigatavusekuntia ovat ilmaisia 3,2 miljoonaan laskentasekuntiin asti, jonka ylittyessä jokainen käytetty gigatavusekunti maksaa erikseen. (AWS Whitepaper, 2023.)

Tiivistäen AWS Lambda -palvelu tarjoaa tehokkaan ratkaisun palvelimettomien web-sovellusten tietojenkäsittelytarpeisiin. Sen dynaaminen skaalautuvuus, monipuoliset integraatiomahdollisuudet ja joustava hinnoittelumalli tekevät siitä keskeisen osan modernin pilvipohjaisen web-kehityksen työkalupakkia.

4.5 Amazon DynamoDB: nopea NoSQL-tietokanta

Palvelimettomat web-sovellukset tarvitsevat luotettavan ja suorituskykyisen tietokannan, joka kykenee käsittelemään dynaamista dataa reaaliajassa. Amazon DynamoDB on nopea NoSQL-pilvitietokantapalvelu, joka integroituu saumattomasti muiden AWS-palveluiden, kuten AWS Lambdan, kanssa mahdollistaen välittömän reagoinnin tapahtumiin (Sbarski ym., 2022). DynamoDB on lisäksi tehokas, luotettava ja skaalautuva tietokanta, jota käytettäessä ei kehittäjien tarvitse huolehtia konfiguroinnista tai resurssien kasvattamisesta (Anand ym., 2023). Palvelu tarjoaa vain millisekuntien viiveen kaikilla skaaloilla ja pystyy käsittelemään yli 20 miljoonaa pyyntöä sekunnissa (AWS Whitepaper, 2024). DynamoDB lupaa 99,999 prosentin saatavuuden ja mahdollistaa tietokantataulujen varmuuskopioinnin. Taulun voi palauttaa mihin tahansa tilaan ja ajanhetkeen edellisen 35 päivän ajalta esimerkiksi tahattoman poisto-operaation seurauksena. (Amazon DynamoDB, 2024.) AWS-dokumentaation (2024) mukaan DynamoDB tarjoaa lisäksi sisäänrakennetun salauksen datalle, mikä poistaa kehittäjiltä vaivan arkaluontoisen datan salauksesta.

DynamoDB-tietokannan perusrakenne on taulu, joka kuitenkin eroaa paljon relaatiotietokantojen tauluista. DynamoDB-taulu sisältää tietueita ja jokainen tietue koostuu joukosta ominaisuus-arvo-pareja. Voidaan kuvitella, että jokainen tietue vastaa yhtä riviä taulukossa, ja jokaisella rivillä on joukko erilaisia ominaisuuksia, joilla on niihin liittyvät arvot. Ominaisuuden nimi toimii kuin kolumnin otsikko, määritellen, millaista tietoa kyseisessä sarakkeessa on. DynamoDB-tauluja kutsutaan joskus tietokantakaaviottomiksi (engl. schemaless), koska toisin kuin relaatiotietokannoissa, taulun kaikilla tietueilla ei

tarvitse olla samaa joukkoa ominaisuus-arvo-pareja. (Chawathe, 2019.) Petrovskan ja Ajdarin (2019) mukaan palveluun voi tallentaa myös dokumentteja. Heidän arvionsa mukaan DynamoDB on erinomainen tietokantavaihtoehto palvelimettomaan web-sovellukseen, koska se tarjoaa automaattisen skaalautuvuuden eikä juurikaan vaadi ylläpitoa. DynamoDB:n avulla datan varmuuskopiointi on myös helppoa. Palvelu sopii hyvin web-sovelluksille tyypillisen JSON (JavaScript Object Notation) -datan tallentamiseen. DynamoDB veloittaa käyttäjiä datan talletuksesta, lukemisesta, kirjoittamisesta ja mahdollisista valituista lisäominaisuuksista (AWS Whitepaper, 2023).

Kaiken kaikkiaan DynamoDB on helppokäyttöinen ja luotettava NoSQL-tietokantapalvelu, joka tarjoaa erinomaisen suorituskyvyn, skaalautuvuuden ja turvallisuuden palvelimettomien web-sovellusten tietojen hallintaan. Sen avulla kehittäjät voivat luoda joustavia ja tehokkaita sovelluksia ilman monimutkaisen tietokannan konfigurointia ja ylläpitoa.

5 Palvelimettoman arkkitehtuurin hyödyt ja haasteet

Tässä luvussa analysoidaan palvelimettoman arkkitehtuurin hyötyjä ja haasteita web-sovelluskehityksessä. Ensiksi tutkitaan palvelimettoman arkkitehtuurin tuomia etuja, kuten kustannussäästöjä, skaalautuvuuden parantumista ja kehityssyklin nopeutumista. Sen jälkeen tutkitaan mallin haasteita, kuten kylmäkäynnistysongelmaa, riippuvuutta palveluntarjoajasta ja monimutkaisuutta suurissa sovelluksissa.

5.1 Hyödyt web-sovelluskehityksessä

Palvelimettoman arkkitehtuurin käyttö web-sovelluskehityksessä tarjoaa lukuisia merkittäviä hyötyjä. Eismannin ja kollegoiden (2021) mukaan suurin hyöty on kustannustehokkuus. Palvelimettoman sovelluksen infrastruktuuria ei tarvitse rakentaa alusta alkaen itse, mikä alentaa järjestelmän käyttöönoton kustannuksia. Koska palvelimettomassa arkkitehtuurissa resursseja käytetään ainoastaan tarvittaessa ja asiakkaat maksavat vain suoritetuista toiminnoista, kustannukset ovat usein alhaisemmat kuin perinteisissä palvelinratkaisuissa. Perinteisessä mallissa on maksettava jatkuvasti palvelimen ylläpidosta, huollosta ja päivittämisestä, vaikka käyttöä ei juuri olisikaan. Tämä voi johtaa tarpeettomaan resurssien tuhlaamiseen ja ylimääräisiin kustannuksiin. Sen sijaan palvelimeton malli tarjoaa mahdollisuuden maksaa ainoastaan todellisesta käytöstä, mikä voi tehdä siitä edullisemman vaihtoehdon tilanteissa, joissa palvelimen kuormitus vaihtelee suuresti. Koschelin ja kumppanien (2021) mukaan kustannuksiin on mahdollista myös itse vaikuttaa. Mitä optimaalisemmin pilvipalveluntarjoajan palveluun ladattu koodi on kirjoitettu, sitä nopeammin se suoritetaan. Tämä laskee suoritusaikaa ja pienentää kustannuksia. Lisäksi palvelimettomassa arkkitehtuurissa käytön mukaan kertyvät toimintakulut vastaavat enemmän oikeita liiketoimintaprosesseja (van Eyk ym., 2018). Liiketoiminnan kannattavuuden arviointi on helpompaa, kun laskutus perustuu suoritettuihin toimintoihin, eikä ennalta hankittuihin resursseihin ja ylläpitokustannuksiin, jotka eivät välttämättä vastaa todellista kysyntää.

Toinen merkittävä etu palvelimettomassa ratkaisussa on skaalautuvuus. Asiakkaiden ei tarvitse huolehtia itse resurssien lisäämisestä, sillä pilvipalveluntarjoaja tarjoaa tarvittavat resurssit automaattisesti (Shafiei ym., 2022). Automaattinen skaalautuvuus mahdollistaa nopean käyttäjämäärän kasvun. Zanonin (2017) mukaan palvelimettomassa mallissa skaalautuvuus on erityisen nopeaa: lisää resursseja on saatavilla käyttöön millisekunneissa. Näin ollen palvelimetonta arkkitehtuuria käyttävä web-sovellus pystyy reagoimaan välittömästi suuriinkin käyttäjämäärien kasvuihin esimerkiksi mainoskampanjan aikana. Sovellukselle tarjottavat resurssit skaalautuvat välittömästi vastaamaan kysyntää, mikä varmistaa sovelluksen paremman käyttäjäkokemuksen ja vähentää mahdollisia käyttökatkoja ruuhkatilanteissa.

Palvelimettoman arkkitehtuurin helppokäyttöisyys ja sen mahdollistama nopea sovellusten kehitys ovat etuja, jotka tulee huomioida. Palvelimettomassa mallissa kehittäjät

voivat kehitysprosessin alusta alkaen keskittyä toiminnallisuuden toteuttamiseen infrastruktuuriin liittyvien tehtävien sijaan (Eismann ym., 2021). AWS-palveluiden hyödyntäminen vähentää kirjoitettavan koodin määrää ja tehostaa työtä. AWS-palvelut voivat poistaa tarpeen kehittää kokonaisia sovelluksen osa-alueita, kuten autentikoinnin tai tietokannan. Esimerkiksi Amazon Cognito huolehtii käyttäjien rekisteröinnistä, kirjautumisesta ja salasanojen hallinnasta. Kun kirjoitetun koodin määrä vähenee, niin käy myös ylläpidettävälle koodille. Tämä konkretisoituu kehittäjien tehokkuudessa, kun heidän ei tarvitse ylläpitää monimutkaisia järjestelmiä, vaan he voivat keskittyä ainoastaan oman toiminnallisen koodinsa ylläpitoon ja päivittämiseen. (Sbarski ym., 2022.)

Palvelimettomassa arkkitehtuurissa sovelluksen julkaisun jälkeinen ylläpito helpottuu merkittävästi. Zanonin (2017) mukaan palvelimettomassa mallissa ei tarvitse käyttää aikaa laitteistovikoihin, järjestelmäpäivityksiin tai verkko-ongelmiin, mikä vähentää tarvittavia ylläpitotunteja. Kehittäjien tarvitsee keskittyä ainoastaan toiminnallisen koodin ylläpitoon ja päivittämiseen tarpeen mukaan. Tämä säästää aikaa ja resursseja sekä mahdollistaa keskittymisen olennaiseen: sovelluksen jatkuvaan kehittämiseen ja käyttäjäpalautteen huomioimiseen. Palvelimettoman arkkitehtuurin myötä saavutetaan siis pitkäaikainen ja tehokas ylläpito.

Kun hyödynnetään palvelimetonta arkkitehtuuria, voidaan laitteisto- ja suoritusriskien katsoa vähentyvän. Perinteisessä mallissa sovelluksen kaikki osa-alueet ovat kehittäjien vastuulla, mikä saattaa johtaa pidempään virheiden ratkaisuaikaan, sillä kehittäjät eivät välttämättä ole kaikilla osa-alueilla asiantuntijoita. Palvelimettomia ratkaisuja käytettäessä kehittäjät eivät ole vastuussa yhtä monesta teknologiasta, mikä voi vähentää virheiden riskiä ja nopeuttaa niiden korjausta. Lisäksi pilvipalveluntarjoajien palveluissa ongelmia esiintyy harvemmin, ja niiden kestot ovat lyhyempiä, mikä parantaa sovellusten saatavuutta ja luotettavuutta. (Roberts & Chapin, 2017.)

Nopeampi kehitys ja matalammat kustannukset mahdollistavat palvelimettomien sovellusten nopeamman saattamisen markkinoille ja asiakkaiden käyttöön, mikä on olennaista liiketoiminnan kannalta. Palvelimettomat ratkaisut tarjoavat organisaatioille mahdollisuuden kokeilla erilaisia päivityksiä ja aloittaa uusia projekteja pienillä riskeillä ja kustannuksilla. (Roberts, 2018.) Vaikka kokeilut epäonnistuisivat, ei organisaatioille koidu hankaluuksia tai suuria kustannuksia esimerkiksi tarpeettomien palvelimien uudelleenkäytöstä tai poistamisesta, sillä konkreettista infrastruktuuria ei missään vaiheessa alustettu. Kokeiluihin hankitut palvelut voidaan tarvittaessa myös helposti lopettaa. Jos kokeilut osoittautuvat menestyksekkäiksi ja niitä päätetään jatkaa, tämäkin on vaivatonta palvelimettomassa mallissa, sillä lisäresursseja ei tarvitse hankkia. Palvelimettoman arkkitehtuurin joustavuus ja kustannustehokkuus ovat keskeisiä tekijöitä, jotka vauhdittavat liiketoimintaa ja edistävät innovaatioita.

5.2 Haasteet web-sovelluskehityksessä

Vaikka palvelimettoman arkkitehtuurin käytöllä on lukuisia etuja, se tuo mukanaan myös haasteita, jotka organisaatioiden on otettava huomioon. Yksi keskeisistä haasteista on riippuvuus pilvipalveluntarjoajasta. Organisaatiot ovat riippuvaisia pilvipalveluntarjoajan hinnoittelusta, palveluiden saatavuudesta ja toimitusvarmuudesta (Zanon, 2017). Tulevaisuuden suunnittelu ja budjetointi voi vaikeutua, koska ne perustuvat toisen yrityksen tarjontaan. Usein puhutaan toimittajaloukusta (engl. vendor lock-in), kun asiakkaan sovellus on riippuvainen pilvipalveluntarjoajan toimittamasta infrastruktuurista ja palveluista (Patil ym., 2021). Tämä voi tehdä siirtymisen toiselle palveluntarjoajalle vaikeaksi tai jopa mahdottomaksi, mikä rajoittaa organisaation joustavuutta ja mahdollisuutta vastata muuttuviin tarpeisiin ja markkinatilanteisiin. Koschel ja kumppanit (2021) mainitsevat yhdeksi ratkaisuksi toimittajaloukkuun monipilvilähestymistavan, jossa sovelluksen kehitys suunnitellaan siten, että suurin osa siitä kehitetään sisäisesti eikä näin ollen olla riippuvaisia palveluntarjoajasta. Tämä lähestymistapa nostaa kehityskustannuksia, mutta vastineeksi vähentää sovelluksen mahdollisen tulevan siirtymän kustannuksia. Mahdollisen toimittajaloukun vuoksi organisaatioiden on tärkeää arvioida huolellisesti valitun pilvipalveluntarjoajan tarjoama tuki, palvelut ja hintataso tulevaisuudessa ennen sitoutumista asiakkaaksi.

Palvelimettoman arkkitehtuurin toinen merkittävä haaste on sovelluksen monimutkaistuminen hajautetun luonteen vuoksi. Palvelimettoman web-sovelluksen arkkitehtuuri voi olla monimutkaisempi ja vaikeammin hallittava monen eri palvelun ja funktion vuoksi (Hassan ym., 2021). Kun sovellus koostuu useista hajautetuista komponenteista, sen ymmärtäminen ja opettelu voi olla kehittäjille haastavampaa verrattuna perinteisiin monoliittisiin sovelluksiin, jossa kaikki toiminnallisuus on samassa paikassa. Kehittäjien on tärkeää pysyä ajan tasalla palveluntarjoajien jatkuvasti päivittyvistä ympäristöistä ja tarjolla olevista palveluista. Mahdollisesti sovelluksessa voi olla samaan aikaan käytössä useita palveluita eri pilvipalveluntarjoajilta, jolloin kaikkien näiden ympäristöjen hallitseminen on tärkeää. Van Eykin ja kumppanien (2018) mukaan nopeasti kehittyvät palvelimettomat rajapinnat, kirjastot ja kehykset muodostavat esteen sovellusten elinkaaren hallinnalle sekä palveluiden löytämiselle ja välittämiselle. Tämä jatkuvasti muuttuva palveluiden ja rajapintojen kirjo luo haasteita kehittäjille, jotka pyrkivät ymmärtämään ja hallitsemaan näitä teknologioita.

Työkalujen puutteellisuudet ovat eräs palvelimettoman arkkitehtuurin ongelma. Sovelluksen vianmääritys voi olla monimutkaista pilvipalveluntarjoajien tarjoamien työkalujen vähyyden ja heikkouden vuoksi (Patil ym., 2021). Mallinnustyökalujen riittämättömyys ja virheenetsintä- ja testaustyökalujen puutteet vaikeuttavat kehittäjien työtä entisestään (Shafiei ym., 2022). Hassanin ja kollegoiden (2021) mukaan palvelimettomiin palveluihin tarvitaan lisää monitorointityökaluja, jotta kehittäjät voisivat seurata, miten

funktiot toimivat käytännössä. He mainitsevat myös, että vaikka palveluntarjoajat ovat julkaisseet yhä enemmän työkaluja käytettäviksi, niissä on edelleen paljon parannettavaa. Näiden seikkojen myötä on selvää, että tukityökalujen riittämättömyys ja heikkous voivat olla merkittäviä esteitä palvelimettoman mallin käyttöönotolle.

Yksi keskeisimmistä palvelimettoman arkkitehtuurin ongelmista on latenssi. Sovellusten toiminnallisuus eli funktiot suoritetaan konteissa. Kun saapuu pyyntö tietyn toiminnon suorittamiseksi, järjestelmä ensin tarkastaa, onko sopivaa konttia jo käynnissä. Mikäli ei ole, on tarpeen käynnistää uusi kontti, mikä luonnollisesti vie enemmän aikaa kuin tilanne, jossa kontti on jo valmiiksi käynnissä eli lämmin. Tätä tilannetta kutsutaan kylmäkäynnistykseksi (engl. cold start). (Patil ym., 2021.) Kylmäkäynnistystilanteessa sovelluksen toimintojen suorittaminen voi kestää huomattavasti kauemmin, mikä voi merkittävästi heikentää käyttäjäkokemusta. Zanonin (2017) mukaan on mahdollista pystyttää toinen palvelu, joka tekee pyyntöjä pääpalveluun tietyin väliajoin pitääkseen sen lämpimänä ja näin välttää kylmäkäynnistystilanteita. Tämä lisää kuitenkin kustannuksia. Vaikka latenssiongelmat ovat yhä ajantasaisia, ne ovat vähentyneet ajan myötä. Esimerkiksi Amazon on julkaissut varatun kapasiteetin ominaisuuksia, jotka tarjoavat esilämmitettyjä funktioinstansseja kylmäkäynnistysten vähentämiseksi (Eismann ym., 2021). Tämä osoittaa, että palvelimettoman arkkitehtuurin kehitys ja parannukset jatkuvat, mikä luo toivoa latenssiin liittyvien ongelmien lieventymisestä tulevaisuudessa.

Palvelimettoman arkkitehtuurimallin ratkaisua harkittaessa on otettava huomioon tietoturvakysymykset. Koska funktiot ja data ovat pilvipalveluntarjoajan jaetulla alustalla, ne täytyy eristää ei-toivotuilta käyttäjiltä. Lisäksi palvelut kommunikoivat ja jakavat dataa muiden samassa ympäristössä olevien tai ulkopuolisten palveluiden kanssa. Näin ollen kaikki nämä palvelut tulee olla toteutettu huolellisesti ja turvallisesti, jotta tietoturvariskejä ei syntyisi. (Hassan ym., 2021.) Pilvipalveluntarjoajien palveluiden tietoturvan on oltava kunnossa, jotta ulkopuoliset eivät pääse käsiksi muiden dataan jaetulla alustalla. Koska palvelimettomassa arkkitehtuurissa sovelluksen rakenne on hajautettu moneen eri palveluun, riski, että jossakin palvelussa on tietoturvaongelma, kasvaa. Lisäksi sovelluksen hyökkäyspinta-ala on suurempi, etenkin jos sen toteutuksessa hyödynnetään monia eri pilvipalvelualustoja ja niiden palveluita. (Koschel ym., 2021.) On tärkeää, että kaikki nämä tietoturvanäkökohdat otetaan huomioon palvelimettoman arkkitehtuuria hyödyntävän sovelluksen suunnittelussa ja toteutuksessa.

6 Yhteenveto

Tässä kandidaatintutkielmassa tutkittiin palvelimettoman arkkitehtuurin käyttöä web-sovellusten kehittämisessä Amazon Web Services (AWS) -pilvipalvelualustalla. Tutkielmassa analysoitiin olennaisia AWS-palveluita ja niiden roolia palvelimettomien web-sovellusten rakentamisessa. Lisäksi tutkielmassa tarkasteltiin palvelimettoman arkkitehtuurin tuomia hyötyjä ja haasteita web-sovelluskehityksessä.

Palvelimeton arkkitehtuuri soveltuu erinomaisesti web-sovellusten arkkitehtuuriksi sen tapahtumapohjaisuuden ansiosta. Erityisesti sovellusten koon kasvaessa palvelimeton malli voi olla hyvä vaihtoehto perinteisen monoliittisen mallin sijaan. AWS tukee kattavasti palvelimettomien sovellusten kehitystä tarjoamalla laajan valikoiman palveluita sovellusten eri osa-alueille tietokannoista käyttäjänhallintaan. Näiden palveluiden integrointi keskenään on sujuvaa, ja niiden hinnoittelu perustuu todelliseen resurssien käyttöön. AWS-palveluita hyödyntämällä on mahdollista rakentaa interaktiivisia, tehokkaita ja automaattisesti skaalautuvia web-sovelluksia, jotka vaativat minimaalista ylläpitoa.

On kuitenkin tärkeää harkita, missä tilanteissa palveluiden hyödyntäminen on järkevää. Pienemmissä sovelluksissa, joissa liikennemäärät ovat ennustettavissa ja toiminnallisuudet yksinkertaisia, perinteinen monoliittinen palvelinarkkitehtuuri voi olla yksinkertaisempi vaihtoehto. Jos taas suunnitellaan alusta alkaen suurempaa ja monimutkaisempaa web-sovellusta, palvelimeton malli voi olla erinomainen valinta. Voi olla perusteltua aluksi rakentaa sovellus perinteisellä mallilla ja sen kasvaessa siirtyä tarpeen mukaan asteittain palvelimettomaan malliin yksi toiminnallisuus tai palvelu kerrallaan.

On myös olennaista pitää mielessä, että pilvipalveluntarjoajat, kuten AWS, luonnollisesti markkinoivat omia palveluitaan korostaen niiden etuja. Palveluissa voi ilmetä piilotettuja kustannuksia ja rajoituksia. Lisäksi kokonaiskustannuksia on vaikea arvioida etukäteen hinnoittelun perustuessa todelliseen käyttöön. Siksi itsenäinen tutkimus palveluista on olennaista, jotta organisaatiot voivat vertailla eri vaihtoehtoja ja valita tarpeisiinsa parhaiten sopivan ratkaisun. AWS on yksi lukuisista pilvipalveluntarjoajista, jotka tarjoavat palvelimettomia palvelukokonaisuuksia. Muita tunnettuja vaihtoehtoja ovat esimerkiksi Microsoft Azure ja Google Cloud.

Kaiken kaikkiaan palvelimettomalla arkkitehtuurilla on useita merkittäviä etuja web-sovelluskehityksessä. Se tarjoaa automaattisen skaalautuvuuden kysynnän mukaan, säästää kustannuksia resurssien tehokkaan käytön ansiosta ja nopeuttaa kehitysprosessia valmiiden palveluiden ansiosta. Kehittäjät voivat keskittyä sovellusten ydintoimintoihin infrastruktuurin hallinnan sijaan. On kuitenkin olennaista tunnistaa ja ottaa huomioon tämän mallin haasteet, kuten riippuvuus pilvipalveluntarjoajasta, monimutkaisempi sovellusten hallinta, työkalujen puutteellisuudet ja mahdolliset latenssiongelmat. Organisaatioiden tulisi arvioida näitä etuja ja haasteita huolellisesti ja valita arkkitehtuuriratkaisu, joka par-

haiten vastaa niiden liiketoiminnallisia tarpeita ja tavoitteita. Palvelimettoman arkkitehtuurin yleistyminen on merkki jatkuvasta kehityksestä ja innovaatiosta web-sovelluskehityksessä. Jatkotutkimus sen hyötyihin ja haasteisiin, sekä pilvipalvelualustojen palveluihin voi tarjota arvokasta tietoa palvelimettoman mallin käytön optimoimiseksi web-sovelluksissa. Tulevaisuudessa voidaan odottaa palvelimettoman arkkitehtuurin käytön kasvavan, kun pilvipalveluntarjoajat kehittävät entistä monipuolisempia ja tehokkaampia ratkaisuja vastatakseen kasvavaan kysyntään ja tarpeisiin.

Lähdeluettelo

Amazon Cognito. (2024). *Amazon Cognito*. https://aws.amazon.com/cognito/ (Haettu 13.5.2024)

Amazon DynamoDB. (2024). *Amazon DynamoDB*. https://aws.amazon.com/dynamodb/ (Haettu 13.5.2024)

Anand, H., Biradar, S., Prajapat, Shiparad, D. (2023). Deployment of a Serverless Web Application using AWS services. *American Journal of Science & Engineering*, 3(4). https://doi.org/10.15864/ajse.3401

AWS. (2024a). *Serverless on AWS*. https://aws.amazon.com/serverless/ (Haettu 13.5.2024)

AWS. (2024b). *Serverless Web Application.* https://aws.amazon.com/serverless/build-a-web-app/ (Haettu 13.5.2024)

AWS Documentation. (2024). *AWS Documentation*. https://docs.aws.amazon.com/ (Haettu 13.5.2024)

AWS Whitepaper. (24.2.2023). *How AWS Pricing Works.* https://docs.aws.amazon.com/pdfs/whitepapers/latest/how-aws-pricing-works/how-aws-pricing-works.pdf

AWS Whitepaper. (1.3.2024). *Overview of Amazon Web Services*. https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-overview/aws-overview.pdf

Chawathe, S. S. (2019). Data Modeling for a NoSQL Database Service. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 0234–0240. https://doi.org/10.1109/UEMCON47517.2019.8992924

Datadog. (2023). *The state of serverless*. https://www.datadoghq.com/state-of-serverless/ (Haettu 13.5.2024)

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), 32–39. https://doi.org/10.1109/MS.2020.3023302

Hassan, H. B., Barakat, S. A. & Sarhan, Q. I. (2021). Survey on serverless computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 10, artikkeli 39. https://doi.org/10.1186/s13677-021-00253-7

Koschel, A., Klassen, S., Jdiya, K., Schaaf, M. & Astrova, I. (2021). Cloud Computing: Serverless. *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*. 1–7. https://doi.org/10.1109/IISA52424.2021.9555534

Kulkarni, G., Sutar, R. & Gambhir, J. (2012). Cloud Computing-Storage as Service. *International Journal of Engineering Research and Applications (IJERA)*, 2(1), 945–950. https://www.researchgate.net/publication/234166321_Cloud_Computing-Storage_as_Service

Li, Y., Lin, Y., Wang, Y., Ye, K. & Xu, C. (2022). Serverless Computing: State-of-the-Art, Challenges and Opportunities. *IEEE Transactions on Services Computing*, 16(2), 1522–1539. https://doi.org/10.1109/TSC.2022.3166553

Patil, R., Chaudhery, T. S., Qureshi, M. A., Sawant, V. & Dalvi, H. (2021). Serverless Computing and the Emergence of Function-as-a-Service. *2021 International Conferen-ce on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 764–769. https://doi.org/10.1109/RTEICT52294.2021.9573962

Patterson, S. (2019). *Learn AWS Serverless Computing*. Packt Publishing.

Petrovska, J. & Ajdari, J. (2019). Amazon's role in the field of cloud relational and NoSQL databases: a comparison between Amazon Aurora and DynamoDB. *3rd International Scientific Conference on Business and Economics (ISCBE)*, 214–224. https://www.researchgate.net/publication/337155638_ISCBE2019_Conference_Proceedings_FINAL

Rajan, A. (2018). Serverless Architecture - A Revolution in Cloud Computing. *2018 Tenth International Conference on Advanced Computing (ICoAC)*, 88–93. https://doi.org/10.1109/ICoAC44903.2018.8939081

Roberts, M. (22.5.2018). *Serverless Architectures*. https://martinfowler.com/articles/serverless.html

Roberts, M. & Chapin, J. (2017). *What Is Serverless?* O'Reilly Media, Inc.

Sbarski, P., Cui, Y. & Nair, A. (2022). *Serverless Architectures on AWS*. Second Editi-on. Manning Publications.

Sewak, M. & Singh, S. (2018). Winning in the Era of Serverless Computing and Function as a Service. *2018 3rd International Conference for Convergence in Technology (I2CT)*, 1–5. https://doi.org/10.1109/I2CT.2018.8529465

Shafiei, H., Khonsari, A. & Mousavi, P. (2022). Serverless Computing: A Survey of Opportunities, Challenges, and Applications. *ACM Computing Surveys*, 54(11s), artikkeli 239. https://doi.org/10.1145/3510611

Singh, V., Acharya, C., Jain, A., Gupta, K. & Wajari, K. (2023). Creating Serverless Web Applications with AWS. *International Journal for Research in Applied Science and Engineering Technology*, 11(11), 2008–2011. https://doi.org/10.22214/ijraset.2023.56985

Stack Overflow. (2023). *2023 Developer Survey*. https://survey.stackoverflow.co/2023/ (Haettu 13.5.2024)

van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uță, A. & Iosup, A. (2018). Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing*, 22(5), 8–17. https://doi.org/10.1109/MIC.2018.053681358

Zanon, D. (2017). *Building Serverless Web Applications*. Packt Publishing.