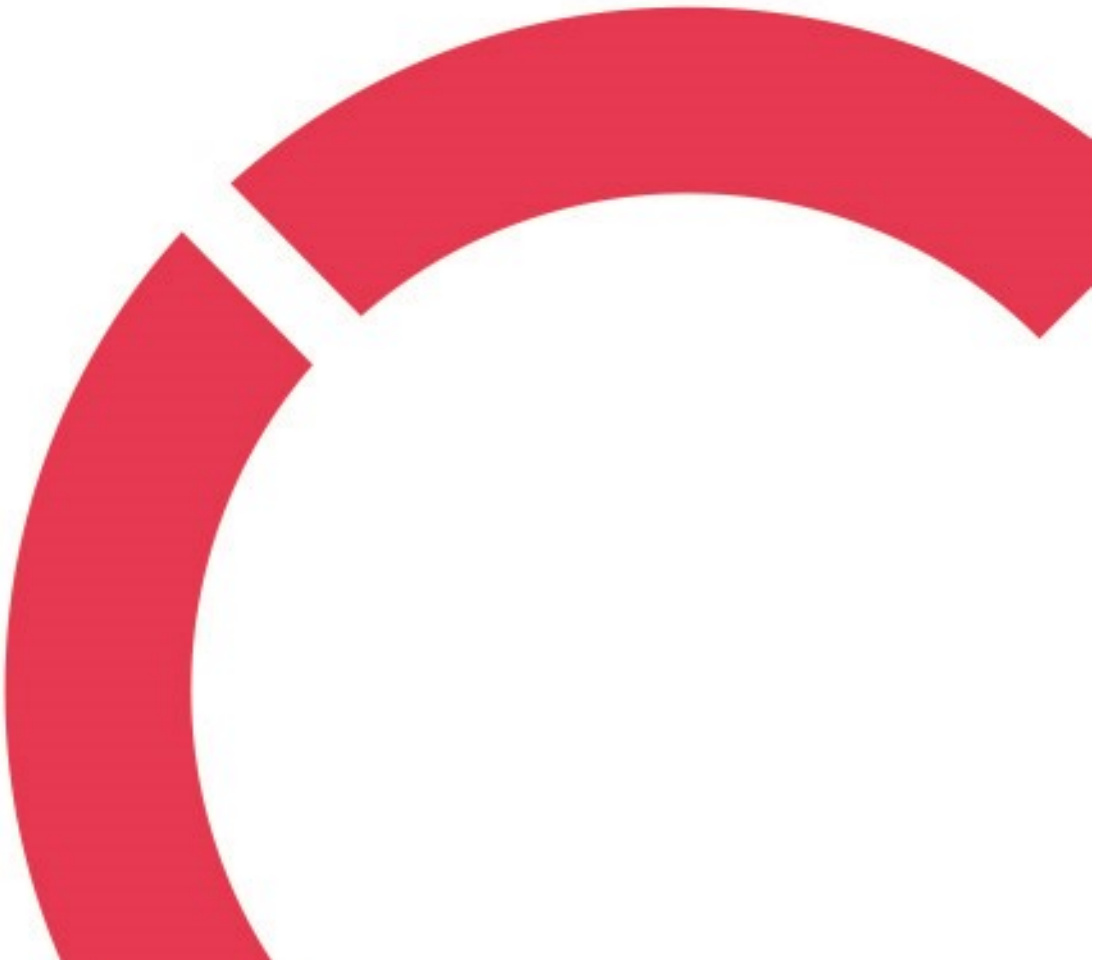
Taneli Laitakangas

SERVERLESS-PALVELUN LAATIMINEN MICROSOFT AZURE -YMPÄRISTÖÖN

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Lokakuu 2023



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Lokakuu 2023	**Tekijä/tekijät** Taneli Laitakangas
Koulutus Insinööri (AMK), tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
Työn nimi SERVERLESS-PALVELUN LAATIMINEN MICROSOFT AZURE YMPÄRISTÖÖN		
Työn ohjaaja Jari Isohanni		**Sivumäärä** 50
Työelämäohjaaja -		

Opinnäytetyössä tutkittiin serverless-palvelun luomisprosessia Azureen. Tarkoituksena oli käydä läpi pilvipalveluiden toimintaa ja keskeisiä tuotteita, jotka soveltuvat serverless-applikaation luomiseen. Vaikka työ sisältää kattavat ohjeet palvelun luomisesta Azureen, sen tarkoituksena oli mahdollistaa ohjeiden soveltaminen myös toisille pilvipalvelualustoille.

Teoriaosuudessa avattiin eri palvelumallien eroja ja tuotiin esille pilvipalvelujen ja serveless-palvelujen yhtäläisyyksiä. Työssä käsiteltiin sitä, miten ohjelmiston asetusten ja määritysten vastuu siirtyi palveluntarjoajan ja käyttäjän välillä palvelumallin mukaan. Opinnäytetyössä esiteltiin samanlaisia palvelutuotteita Google Cloudin, Amazon AWS:n ja Microsoft Azuren alustoilta. Keskeisimmät palvelut olivat eri tietokannat, varastotilat ja funktiot.

Käytännön osuus tarjoaa askel askeleelta seurattavan ohjeistuksen Azuren käyttöön, mukaan lukien SQL-tietokannan, funktioiden ja tietovaraston luomisen. Lopuksi esiteltiin, kuinka Reactilla luotu applikaatio julkaistiin Azuren ”Static Web app” -palvelussa ja kuinka julkaisuprosessi automatisoitiin GitHubin tarjoamilla Actioneilla.

Palvelun luomisprosessin jälkeen havaittiin, kuinka helpoksi ja vaivattomaksi ohjelmointi muuttui alkukonfiguroinnin jälkeen. Funktioita käytettiin kommunikointiin web-applikaation ja Azuren muiden palveluiden kanssa. Lukija saa käsityksen siitä, kuinka serverless-applikaatio luodaan moderneilla työkaluilla.

Asiasanat
Pilvipalvelut, Serverless, React, Azure

ABSTRACT

Centria University of Applied Sciences	Date October 2023	Author Taneli Laitakangas
Degree programme Bachelor of Engineering, Information and Communications Technology		
Name of thesis CREATING A SERVERLESS SERVICE IN THE MICROSOFT AZURE ENVIRONMENT		
Centria supervisor Jari Isohanni		**Pages** 50
Instructor representing commissioning institution or company -		

The aim of this thesis was to investigate the process of creating a serverless service in Azure. The intentation was to review the operation of cloud services and the key products suitable for building a serverless application. The work provides comprehensive instructions for creating a service in Azure. Its purpose was to facilitate the adaptation of these guidelines to other cloud platforms.

The theoretical section showed the differences between various cloud computing service models and highlighted the similarities between cloud services and serverless services. The work addressed how the responsibility for software settings shifted between the service provider and the user according to the service model. The thesis introduced similar service products from Google Cloud, Amazon AWS and Microsoft Azure platforms. The most important services were databases, storages and functions.

The practical section offers a step-by-step guide to using Azure. It covers the creation of SQL database, functions and blob storage. The section also demonstrated how an application created with React was published using Azure's "Static Web app" service. Finally, it showed how to deployment process was automated using GitHub Actions.

Following the creation process, it was observed that programming became more straightforward and effortless after the initial configuration. Functions were utilized to communicate between the web application and other services in Azure. By the end, the reader gain an unerstanding of how to create serverless application using modern programming tools.

Key words
Cloud services, Serverless, React, Azure

KÄSITTEIDEN MÄÄRITTELY

BLOB

Binary Large Object, eli binaarimuodossa olevaa dataa, kuten kuvia ja videoita.

CALLBACK-FUNKTIO

Callback-funktio on funktio, joka annetaan toiselle funktiolle argumenttina ja suoritetaan myöhemmässä vaiheessa.

CORS

CORS (Cross-Origin Resource Sharing) on turvallisuusprotokolla, joka mahdollistaa verkkoresurssien jakamisen eri alkuperän omaavien sivustojen kesken.

DEPLOYAUS

Deployaus viittaa prosessiin, jossa kehitetty ohjelmisto otetaan käyttöön tietyssä ympäristössä, kuten testaus- tai tuotantoympäristössä.

IDEMPOTENSSI

Idempotenssi tarkoittaa operaation ominaisuutta, jossa sen toistuva suorittaminen tuottaa saman lopputuloksen kuin sen suorittaminen kerran.

KONTTI

Kontti on kevyt, itsenäinen, suoritettava ohjelmistopaketti, joka sisältää kaiken tarvittavan ohjelmiston suorittamiseen. Kontit eristävät ohjelmiston muusta ympäristöstä varmistaen, että se toimii yhdenmukaisesti eri ympäristöissä.

NOSQL

NoSQL tarkoittaa "Not Only SQL" ja se viittaa tietokantajärjestelmiin, jotka eivät käytä perinteistä relaatiotietokanta-mallia.

NÄKÖISTIEDOSTO

Näköistiedosto eli ”image”, tarkoittaa tiedostoa, joka sisältää järjestelmän tai sovelluksen kopion ja kaikki siihen liittyvät resurssit.

RUNTIME

Runtime eli suoritusympäristö tarkoittaa ohjelmiston tai skriptin suorittamiseen tarvittavaa ympäristöä ja siihen liittyviä kirjastoja.

SOP

Same origin policy on selainten turvallisuustoimi, joka estää verkkosivustoja tekemästä pyyntöjä toisille sivustoille.

1 JOHDANTO

Opinnäytetyössäni perehdyn serverless-palvelun laatimiseen Microsoft Azuren ympäristöön. Serverless on termi, joka on viime vuosina herättänyt paljon huomiota pilvipalvelualalla, ja se edustaa uutta lähestymistapaa sovelluskehityksessä ja infrastruktuurin hallinnassa. Vaikka työssäni keskityn vahvasti serverlessiin ja Azureen, käsittelen myös pilvipalvelun toimintaa yleisemmällä tasolla, sillä serverless-arkkitehtuurin ymmärtäminen vaatii pilvipalveluiden peruskäsitteiden tuntemista. Selkeän rajan vetäminen serverlessin ja pilvipalvelun välille voi joissain tilanteissa olla haastavaa.

Tämän opinnäytetyön tavoitteena on antaa lukijalle kattava käsitys siitä, miten serverless-palvelu rakennetaan Azuren avulla. Pyrin suoriutumaan tästä ilman kustannuksia hyödyntämällä Azuren ensimmäiseksi vuodeksi tarjoamia ilmaisia tuotteita. Vaikka keskityn suurimmaksi osaksi Azureen, vertailen sitä muihin pilvipalveluntarjoajiin, jolloin lukija saa kattavamman ymmärryksen pilvipalvelutuotteista ja voi soveltaa oppimaansa tietoa myös muiden pilvipalvelualustojen parissa.

Valinta tehdä opinnäytetyö tästä aiheesta johtuu kahdesta selkeästä syystä: Ensinnäkin henkilökohtaisesti halusin syventää taitojani pilvipalveluiden parissa, koska aikaisempi kokemukseni aiheesta ei ollut kovin laaja ja olen varma, että juuri tämän aiheen valitseminen hyödyttää minua tulevaisuudessa työelämässä todella paljon. Toiseksi, vaikka serverless on noussut suureen suosioon maailmalla, suomenkieliset resurssit eivät ole vielä saavuttaneet samaa kattavuutta. Oli haastavaa löytää joihinkin pilvipalveluihin liittyvää aineistoa edes englanniksi.

Koska serverless on laaja aihe ja eri pilvipalveluntarjoajilla on monenlaisia tuotteita tarjolla, pyrin tässä työssä keskittymään erityisesti niihin tuotteisiin, jotka koen olennaisimpina serverless-sovelluksen kehittämiseen Azuressa. Lähteinä olen käyttänyt pääosin pilvipalveluntarjoajien virallista dokumentaatiota, mikä takaa ajantasaisen ja teknisesti paikkansapitävän tiedon. Lähteiden ajantasaisuus on erityisen tärkeää alan nopean kehityksen vuoksi sillä tuotteet, tekniikat ja palveluiden ulkoasut voivat muuttua merkittävästi lyhyessä ajassa.

2 YLEISTÄ PILVIPALVELUISTA

Pilvipalvelu mahdollistaa käyttäjilleen erilaisten tietotekniikan palveluiden, kuten palvelimien, laskentatehon ja tallennuskapasiteetin hyödyntämisen internetin kautta. Palveluiden sijaitseminen pilvessä mahdollistaa sen, että yrityksen ei tarvitse ylläpitää omista fyysistä palvelimista tai konesaleista. Pilvipalveluiden keskeisiä ominaispiirteitä ovat nopeus, skaalautuvuus ja kustannusten tehokas hallinta. (Microsoft Azure.)

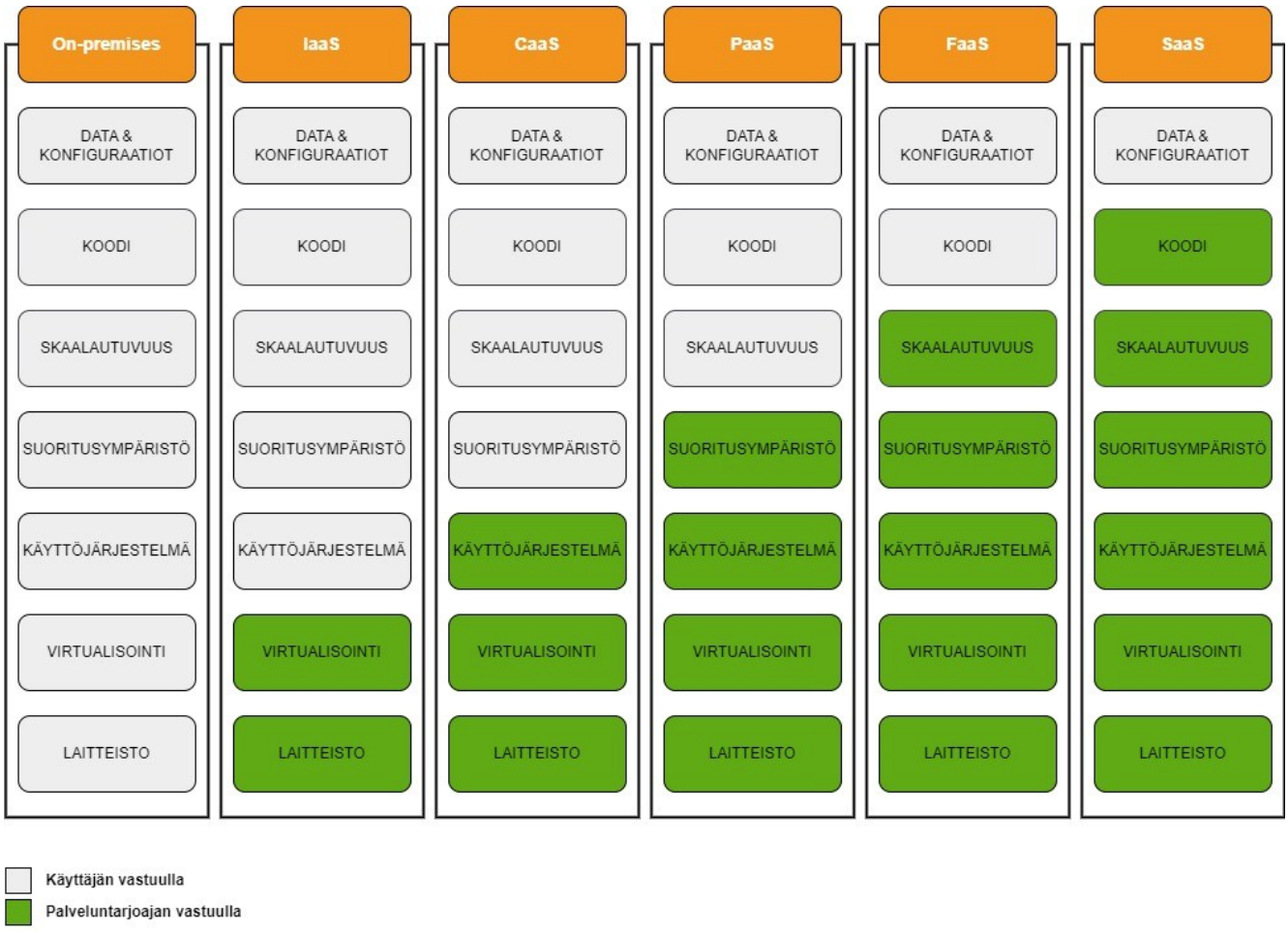
Erilaisten palveluiden käyttöönotto onnistuu tarvittaessa jopa minuuteissa ja sovelluskehittäjien tekemät muutokset voidaan julkaista sujuvasti. Skaalautuvuudella tarkoitetaan resurssien kuten laskentatehon lisäämistä tai vähentämistä tarpeen mukaan. Otetaan esimerkiksi kuvitteellinen tilanne, jossa nettikauppaa pyörittävän yrityksen sivuston kävijämäärä räjähtää Black Fridayn aikana. Tällöin pilvipalvelu havaitsee lisääntyneen kuormituksen ja varaa lisää automaattisesti laskentatehoa. Tämän ansiosta sivun käyttäjät eivät huomaa muutoksia sivuston suoritusnopeudessa ja kävijämäärän laskiessa pilvipalvelu vapauttaa ylimääräiset resurssit. (Amazon AWS. 2023.)

Kustannusten hallinnassa skaalautuvuus on avainasemassa. Yleinen pilvipalveluiden maksumalli on ”pay-as-you-go”-tyyppinen, joka perustuu käytön määrään. Lisäksi pilvipalveluiden käyttö vähentää konesalien ylläpitokustannuksia, sillä laitteiston ylläpito ja siihen liittyvät kustannukset ovat palveluntarjoajan vastuulla. (Amazon AWS 2023.)

2.1 Palvelumallit pilvipalveluarkkitehtuurissa

Pilvipalvelut jaetaan yleensä kolmeen eri palvelumalliin, jotka ovat SaaS (Software as a service), PaaS (Platform as a service) ja IaaS (Infrastructure as a service). Vaikka nämä kolme mallia ovat tunnetuimpia, on pilvipalveluissa myös muita malleja kuten FaaS (Function as a service) ja CaaS (Container as a service). Usein näitä pilvipalvelumalleja verrataan perinteiseen ”On-premises”-malliin, jossa organisaatio omistaa, ylläpitää ja hallinnoi koko infrastruktuuriaan. (PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different?)

Palvelumallit järjestetään yleensä sen mukaan, kuinka paljon käyttäjän on hallittava ja konfiguroitava eri asioita. Kuviossa 1 on esitelty eri palvelumallit järjestettynä vasemmalta oikealle käyttäjän vastuun mukaan.



KUVIO 1. Palvelumallit mukaillen (PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different?)

2.1.1 On-premises: Itse hallittava infrastruktuuri

”On-premises”-termillä tarkoitetaan yrityksen omissa tiloissa sijaitseva IT-infrastruktuuria. Siinä missä pilviratkaisut tarjoavat laitteistot ja ohjelmistot,”on-premises” -ratkaisussa yrityksen on huolehdittava itse kaikesta. Omien laitteistojen komponentteja tulee uusia, käyttöjärjestelmät on hankittava, laitteiston sähkölaskut maksettava ja tietoturvasta tulee olla itse vastuussa. Lisäksi isompi vastuu vaatii enemmän osaavaa IT-henkilökuntaa ja palkkakuluja voi syntyä enemmän.

Vaikka pilvipalvelujen suosio on kasvanut yritysten keskuudessa, on ”on-premises”-ratkaisussa omat etunsa, joita pilvipalvelujen avulla on vaikea tai mahdoton toteuttaa. Yksi yleisimmistä syistä valintaan

on se, että yrityksellä on tällöin täysi kontrolli datasta. Kuvitellaan esimerkiksi tilanne, jossa valtion on hallittava salaiseksi luokiteltua dataa: tietojen vieminen kolmannen osapuolen ylläpitämään pilveen voi olla liian riskialtista. Tällöin tietoturvan hallinta ei ole täysin yrityksen omissa käsissä, ja yritys on riippuvainen pilvipalvelun toimivuudesta. Jos palvelussa on ongelmia, data voi olla saavuttamattomissa sillä hetkellä. (Cleo.)

2.1.2 IaaS: Infrastruktuuripalvelut pilvessä

IaaS (Infrastructure as a Service) on pilvipalvelumalli, joka voidaan nähdä seuraavana askeleena on-premises-mallista. Siinä osa vastuusta siirtyy palveluntarjoajalle, joka tarjoaa yritykselle tarvittavan infrastruktuurin, kuten palvelimet, verkon ja tallennustilan. Yrityksen täytyy kuitenkin itse huolehtia niiden hallinnasta, ylläpidosta ja suorittaa tarvittavat konfiguroinnit. Tyypillisesti pilvipalveluntarjoajat tarjoavat asiakkailleen web-pohjaisen hallintaportaalin, jonka kautta asiakkaat pystyvät hallitsemaan haluamiaan resursseja muutamalla klikkauksella. (What Is IaaS (Infrastructure as a Service)? 2023.)

Kun palveluntarjoaja huolehtii perus IT-infrastruktuurista, yritykset voivat keskittyä paremmin omien palveluidensa kehittämiseen. Pilvipalveluille tyypillinen skaalautuvuus pääsee oikeuksiinsa ja hyvän suorituskyvyn ylläpito onnistuu vaivattomammin perinteiseen on-premise-malliin verrattuna. (What Is IaaS (Infrastructure as a Service)? 2023.)

IaaS-mallin heikkoutena on, että se vaatii käyttäjältään teknistä osaamista, sillä se tarjoaa pohjimmiltaan vain alustan palveluiden pystyttämiseen. Lisäksi, kuten muidenkin pilvipalvelumallien kohdalla, tietoturva ei ole täysin käyttäjän käsissä, ja yritys on riippuvainen palveluntarjoajan teknisestä toimivuudesta ja palvelun saatavuudesta. (Rosencrance. 2021.)

2.1.3 CaaS: Konttiteknologian hallinta pilvessä

CaaS on palvelumalli, jossa palveluntarjoaja mahdollistaa ”containerien” eli konttien hallinnan, isännöinnin ja sijoittamisen käyttöympäristöön. Kontti on kevyt ja eristetty ympäristö, joka sisältää sovelluksen sekä kaikki sen tarvitsemat riippuvuudet, kuten kirjastot ja muut resurssit. Toisin kuin virtuaalikoneet, kontit jakavat saman käyttöjärjestelmän ytimen, mutta toimivat itsenäisessä ja eristetyssä ympäristössä. Kontit ovat suosittuja mikropalveluarkkitehtuurissa, sillä ne mahdollistavat eri palveluiden

eristämisen omiin kontteihinsa. Jokainen kontti voi sisältää oman ympäristönsä tarpeen mukaan. Kontit voidaan konfiguroida kommunikoimaan keskenään verkon yli, ja CaaS-palveluiden avulla niiden julkaisu voidaan suorittaa samanaikaisesti. Esimerkkejä CaaS palveluista ovat Amazon Elastic Container Service ja Azure Container Instances. (Buchanan.)

2.1.4 PaaS: Sovelluskehitysalustat pilvessä

PaaS eli (Platform as a service) on palvelumalli, joka tarjoaa alustan ohjelmistokehitystä varten. PaaS:issa palveluntarjoaja huolehtii muun muassa palvelimista, käyttöjärjestelmäistä, rajapinnoista ja tarjoaa kehityspaketteja eri ohjelmointikielille. Kehitystyö on suoraviivaista, sillä ohjelmistokehittäjän vastuulla on vain datan ja applikaation hallinta. (What is Platform as a Service (PaaS)?)

PaaS tarjoaa lisäksi erilaisia työkaluja, jotka helpottavat ja nopeuttavat kehitystyötä. Nämä työkalut ovat hallittavissa internetin yli, jolloin käyttäjän ei tarvitse erikseen asentaa ohjelmistoja. Palveluntarjoajan työkaluilla voi olla esimerkiksi mahdollista tarkastella lokeja ja applikaation tietoja liittyen sen toimintaan. Keskeisenä erona IaaS:iin on se, että IaaS tarjoaa alustan, mutta PaaS tarjoaa alustan lisäksi työkalut. (What is Platform as a Service (PaaS)?; Chai, Brush & Bigelow.)

2.1.5 FaaS: Funktiot palveluna pilvessä

FaaS:issa (Function as a Service) käyttäjän ei tarvitse huolehtia laitteistosta, virtualisoinnista tai edes web-serverin konfiguroinnista. Palveluntarjoaja huolehtii edellä mainituista asioista, jolloin käyttäjän on mahdollista keskittyä yksittäisten funktioiden toiminnan toteuttamiseen. Pilvipalveluiden funktioille on ominaista, että ne linkitetään haluttuihin tapahtumiin eli eventteihin, joten tietyn tapahtuman yhteydessä siihen liittyvä funktion suoritetaan. (What is FaaS?)

FaaS:issa näkyy selkeästi skaalautuvuus, sillä funktiot skaalautuvat kysynnän noustessa ylös ja alas kuin kuormitus vähenee. Funktioiden skaalaus toimii yksilöllisesti ja yhden funktion kuormitus, ei skaalaa muita funktioita automaattisesti. Laskutus tapahtuu suoritusajan mukaan ja ominaista on, että funktioiden laskutusta ei tapahtuu ilman funktioiden kutsumista. (What is serverless?)

2.1.6 SaaS: Ohjelmistot suoraan pilvestä

SaaS (Software as a Service) tarjoaa nimensä mukaisesti pilvessä toimivan sovelluksen, jonka käyttämiseen riittää selain sekä toimiva internetyhteys. Tässä palvelumallissa palveluntarjoaja huolehtii kaikista komponenteista kuten laitteistosta ja itse applikaatiosta. Tunnettuja SaaS-palveluita ovat esimerkiksi Gmail, Microsoft Office 365 ja Google Docs. (Red hat. 2022.)

SaaS-palvelut tarjoavat yleensä tilauspohjaisen maksuvaihtoehdon eikä suuria etukäteen tehtäviä maksuja tarvita. Käyttäjän on mahdollista saada nopeasti lisäresursseja käyttöön ottamalla kalliimman tilauksen käyttöön. Tilauspohjaisen sovelluksen kustannukset ovat helposti ennustettavissa ja hallitavissa. SaaS on suunniteltu tiettyä tehtävää varten ja siten se onkin kauimpana perinteisestä On-premises mallista. Kääntöpuolena SaaS-palvelumallissa on se, että sen integrointi ja sovelluksen siirtäminen toiseen palveluun on hankalaa. (What is SaaS? 2023.)

2.2 Serverless-arkkitehtuuri pilvipalveluissa

Serverless on pilvipalvelumalli, joka mahdollistaa sovelluskoodin suorittamisen ilman, että käyttäjän on huolehdittava palvelimista tai taustalla olevista infrastruktuurista. Näin ollen käyttäjä pystyy keskittymään tehokkaammin muihin ohjelmoinnin osa-alueisiin, mikä nopeutta ohjelmiston kehitysprosessia. Serverless tarjoaa backend-palveluita käyttäjille käytön perusteella ilman, että heidän tarvitsee huolehtia infrastruktuurista. Vaikka serverless-termistä voi saada kuvan, että taustalla ei ole lainkaan palvelimia, ovat ne kuitenkin olemassa, mutta käyttäjän ei tarvitse olla niistä tietoinen. (What is serverless?)

Ohjelmoijan näkökulmasta prosessi on yksinkertainen. Kirjoitetaan ohjelmistokoodi, määritellään konfiguraatiot ja sijoitetaan koodi valittuun palveluntarjoajan serverless-ympäristöön. Sijoittamisen jälkeen palveluntarjoaja huolehtii siitä, milloin ja kuinka paljon resursseja tarvitaan. Serverlessin keskeinen piirre on sen tapahtumapohjaisuus. Kun tietty tapahtuma ilmenee, tietty koodi tai sovellus aktivoidaan ja suoritetaan. Esimerkiksi, palveluntarjoajien funktiopohjaisten palveluiden kautta voidaan määritellä funktio, joka reagoi tietokannan muutokseen tai hallitsee muita pilvipalvelun SaaS-sovelluksia API:n kautta. Tällaisen funktion voi laukaista esimerkiksi HTTP-pyyntö tai ajastin, joka aktivoi koodin tietyin väliajoin. (Serverless on AWS.)

Tavoitteena on luoda funktio, joka sisältää suhteellisen yksinkertaisen koodin sillä funktioiden suoritusaika on syytä pitää pienenä. Useimmissa ympäristöissä on määritetty maksimiaika, jonka funktion suorittamiseen voi mennä tai muuten suoritus keskeytetään. Useimmissa ympäristöissä on määritetty maksimiaika. On tärkeää huomata, että funktiopohjaiset palvelut, kuten FaaS (Function as a Service), ovat vain yksi osa serverless-arkkitehtuuria eivätkä itsessään määrittele koko serverless-konseptia. Se on kuitenkin keskeisempiä serverlessin osia. (What is serverless? What is FaaS?)

Serverless-etuihin kuuluu pienet kustannukset, se yksinkertaistaa koodaamista ja konfigurointia. Sovelluksen sijoittaminen tuotantoympäristöön tapahtuu yleensä nopeasti serverless-arkkitehtuurissa. Yksinkertaisimmissa tapauksissa uusi versio sivustosta voidaan julkaista automaattisesti, kun ohjelmoija tekee koodimuutokset ja lähettää ne git-tietovarastoon. (Cloudfare. 2023.)

2.3 Keskeiset serverless-palvelutarjoajat ja palvelut

Serveless-arkkitehtuurin yleistyessä palveluntarjoajien määrä on myös kasvanut merkittävästi. Tunnetuimmat pilvipalvelun tarjoajat Google, Microsoft ja Amazon ovat myös johtavia serverless-palveluntarjoajia. Niistä nostan esille erityisesti keskeisimmät ja samankaltaiset palvelut, joilla voidaan rakentaa toimiva serverless-applikaatio kussakin pilvipalvelussa erikseen. Microsoft Azure käsitellään erikseen ja kattavammin seuraavassa luvussa.

2.3.1 Google Cloud Platform

Google Cloud Platform on Googlen tarjoama pilvipalvelu ja se on kolmanneksi suosituin heti Amazon Web Servicesin ja Microsoft Azuren jälkeen (Law. 2023). Google data centereitä löytyy Aasiasta, Australiasta, Euroopasta, Pohjois-Amerikasta ja Etelä-Amerikasta. Nämä alueet jaetaan vielä eri vyöhykkeisiin, jotka on nimetty kirjaimin. Esimerkiksi yhden Aasiassa sijaitsevan data centerin tunniste voi olla ”asia-east1-a”, jossa viimeinen kirjain on vyöhykkeen tunniste. Jokainen vyöhyke on oma eristetty toisista vyöhykkeistä. Google tarjoaa kilpailijoittensa lailla useita eri palveluita (yli 150), mutta serveless-näkökulmasta muun muassa seuraavat palvelut ovat kiinnostavia: Google Cloud Functions, Google Cloud Firestore, Google Cloud Storage ja Google Cloud Run. (Google Cloud overview.)

Google Cloud Functions on tyypillinen serverless-funktiopalvelu ja ne voidaan yhdistää helposti Googlen muihin palveluihin tai REST-rajapinnan avulla omiin mikropalveluihin tai kolmannen osapuolen ohjelmistoihin, jotka tarjoavat Webhook-integraatiomahdollisuuden. Cloud Functions on FaaS-palvelu, jonka laskutus perustuu funktioiden käyttöaikaan. Aika lasketaan 100 millisekunnin tarkkuudella ja toimettomista funktioista ei laskuteta laisinkaan. Google tarjoaa 2 miljoonan funktiokutsua jokaiselle käyttäjälle ilmaiseksi. (Cloud Functions.)

Cloud Firestore on serveless-dokumenttitietokanta. Se voidaan helposti integroida Googlen muihin palveluihin ja se skaalautuu käytön mukaan. Google mainostaa sitä suoraan serverless-tietokantana tyypillisineen skaalausominaisuuksineen. Firestorea voi kokeilla ilmaiseksi ja Google antaa 300 dollarin edestä krediittejä uusille käyttäjille käyttöön. (Firestore.)

Cloud Storage on tietovarasto, jonne on mahdollista tallentaa jäsentämätöntä dataa kuten esimerkiksi kuvia, tekstiä ja videoita. Google käyttää tietovarastossa olevista konteista termiä ”Bucket”, joka siis on eräänlainen kontti. Kontteihin siirretään data, jonka jälkeen se on ladattavissa, hallittavissa tai jaettavissa käyttäjille. Google tarjoaa 5 GB ilmaista tallennustilaa jokaiselle käyttäjälle. (Cloud Storage.)

Cloud Run on palvelu, joka mahdollistaa käyttäjien sijoittaa kontteja pilvipalveluun. Käyttäjät voivat ottaa kontteja käyttöön Cloud Runissa melkein millä tahansa ohjelmointikielellä. Ainoa ehto on, että niistä on pystyttävä tekemään näköistiedostoja, joista kontit luodaan. Lisäksi joillekin yleisimmille ohjelmointikielille on tarjolla mahdollisuus suorittaa sijoitus suoraan lähdekoodista. Tällöin sijoittamisprosessi on normaalia suoraviivaisempaa ja Cloud Run tekee kontituksen käyttäjille automaattisesti. (What is Cloud Run.)

Muitakin hyviä vaihtoehtoisia palveluita löytyy Google Cloudista. Tietokannaksi voidaan ottaa relaatiotietokanta eli Cloud SQL tai isännöinti voidaan suorittaa Google Cloud App enginen avulla. Mielenkiintoinen vaihtoehto serverless-applikaation luontiin on Googlen Firebase. Se on BaaS (backend as a service)-pohjainen ohjelmointialusta, joka mahdollistaa backend-palvelujen hallinnan, kuten esimerkiksi tietokannat, isännöinnin ja todennuksen. (Flutter.)

2.3.2 Amazon Web Services

Amazon Web Services eli AWS tarjoaa yli 200 pilvipalvelua ja se on tällä hetkellä (vuonna 2023) suosituin pilvipalvelujen tarjoaja. Google Cloudin tapaan AWS:n datacentereitä löytyy ympäri maailmaa useita. AWS tarjoaa useita palveluita, joilla voidaan rakentaa serverless-applikaatio. Näitä ovat esimerkiksi AWS Lambda, Amazon DynamoDB ja Amazon S3: Simple Storage Service. (AWS Cloud Products; Law, M. 2023.)

AWS Lambda on Amazonin FaaS-palvelu ja ne voi kytkeä yli 200 AWS:n palveluun tai SaaS-applikaatioihin. Googlen Funktioiden tavoin maksu perustuu käyttöön. Lambda-funktiot tukevat mm. natiivina seuraavia ajonaikaisia ympäristöjä: Node.js, Python, Java, .NET, Go ja Ruby. Lisäksi käyttäjän on mahdollista luoda omia kustomoituja ajonaikaisia ympäristöjä. Funktiot voidaan sijoittaa AWS:ään joko zip-tiedostona tai konttinäköistiedostona. (AWS Lambda; Lambda runtimes. 2023.)

Tietokannoista voidaan mainita erityisesti DynamoDB, joka on skaalautuva NoSQL-tietokanta. Se pystyy käsittelemään yli 10 biljoonaa tietokantapyyntöä päivittäin ja hetkellisesti yli 20 miljoonaa pyyntöä. DynamoDB:ssä on sisäänrakennettu tietoturva ja varmuuskopiointiominaisuus. Relaatiotietokantavaihtoehtona mielenkiintoinen vaihtoehto Amazon Aurora, joka on MySQL ja PostgreSQL yhteensopiva. Auroralle on myös kehitetty erikseen konfiguraatio nimeltä ”Aurora serverless”, jonka voi kytkeä päälle tietokantaa luodessa. Tällöin se skaalaa automaattisesti tietokannan tarpeen mukaan kapasiteettia. Kaiken kaikkiaan eri tietokanpalveluita on AWS:ssä useita. (Amazon Aurora; Database. 2023.)

Amazon S3 on oliovarasto (object storage), jossa data tallennetaan olioina ”bucketteihin”, jotka ovat AWS:n termi konteille. Olio koostuu tiedostosta ja siihen liittyvästä metadatasta, joka tarjoaa tietoa tiedoston ominaisuuksista ja tunnistamisesta. Yksittäinen olio voi olla enintään 5 teratavun kokoinen. Käytännön prosessi menee seuraavasti: Käyttäjä voi luoda bucketin haluamallaan nimellä ja valita AWS:n tarjoamista alueista sille sijainnin. Sen jälkeen haluttu data voidaan tallentaa buckettiin olioina. Jokaisella tallennetulla oliolla on uniikki avain, joka toimii tunnisteena. Lisäksi buckettiin ja olioihin voidaan konfiguroida pääsyvalvonta, jolla määritellään, kenellä on oikeus käyttää tallennettua dataa. (What is Amazon S3? 2023.)

3 AZURE ALUSTANA SERVERLESS-APPLIKAATION KEHITYKSESSÄ

Tavoitteenani on tässä luvussa pohjustaa lukijaa sen verran, että hänellä on tarvittava tieto Azuressa navigointiin, palveluiden etsimiseen ja hallitsemiseen. Lisäksi esittelen joukon keskeisiä palveluita, jotka soveltuvat hyvin serverless-applikaation luontiin. Läpikäytävistä palveluista tulee löytymään paljon yhtäläisyyksiä aikaisemmassa luvussa esiteltyihin Googlen ja Amazonin palveluihin. Tarkoituksena on, että lukija saa tarvittavan pohjatiedon ennen käytännön työtä.

3.1 Yleistä Azuresta

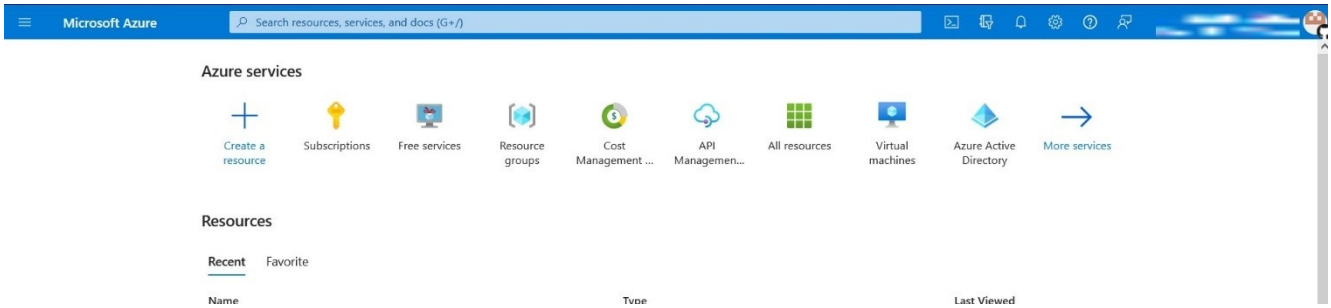
Azure on Microsoftin pilvipalvelualusta, joka koostuu yli 200 pilvipalvelutuotteesta. Käyttäjillä on pääsy kaikkiin näihin tuotteisiin tunnusten luonnin jälkeen. Maailmanlaajuisesti Azure on toiseksi suosituin pilvipalvelualusta heti Amazon AWS:n jälkeen. (Klint. 2023.) Solitan kotisivulla mainitaan heidän tutkimuksestaan, jonka mukaan Azure on Suomessa suurten yritysten suosituin pilvipalvelutarjoaja. Kyselytutkimuksessa 82 % vastaajista ilmoitti käyttävänsä Azurea. (Mäkilä. & Tuomisto. 2022.)

Azure tarjoaa monia pilvipalveluita ilmaiseksi kaikille käyttäjille. Lisäksi uudet käyttäjät saavat käyttöönsä ensimmäisen kuukauden ajaksi 200 dollarin edestä Azure-krediittejä ja tietyt tuotteet ovat ilmaisia ensimmäisen vuoden ajan. On kuitenkin huomioitava, että ilmaisissa palveluissa on käyttörajoituksia, kuten montako pyyntöä voidaan tehdä määrätyssä ajassa tai kuinka paljon dataa voidaan käyttää. (Build in the cloud with an Azure free account. 2023.)

3.2 Navigointi Azuressa

Azuren portaalin kautta käyttäjät voivat navigoida ja hallita sovelluksia. Portaali sisältää kaikki Azuren tarjoamat pilvipalvelutuotteet, kuten esimerkiksi tietokannat, virtuaalikoneet, virtuaaliverkot ja Azure-funktiot. Portaalin ulkoasu on muokattavissa ja käyttäjän on mahdollista valita mitä palveluita hänelle näytetään oletuksena. Myös käyttöoikeusasetukset vaikuttavat siihen, mihin tietyllä käyttäjällä tai ryhmällä on pääsy portaalin kautta. (Microsoft Azure portal. 2023.)

Käyttäjän kirjauduttua Azureen, portaalin kotinäkymä näyttää oletuksena käytetyimmät palvelut. Azuren kaikki pilvipalvelutuotteet saa näkyviin mm. klikkaamalla ”Create a resource” tai ”More services” -painiketta kotinäkymässä. Myös vasemmassa yläkulmassa sijaitsee valikkonappi, josta avautuu erillinen sivupalkki, jonka navigointi on mahdollista. Jos käyttäjä on tietoinen etsimänsä palvelun nimestä, voi hän hyödyntää yläpalkissa olevaa hakukenttää. (KUVA 1.)



KUVA 1. Kotinäkymä Azure-portaalissa.

Uusien pilvipalvelutuotteiden löytämisen lisäksi tärkeää on löytää portaalista jo käytössä olevat palvelut. Kotinäkymää alas skrollatessa paljastuu osio nimeltään ”Navigate” ja sen alta löytyy palvelut: ”Subscriptions”, ”Resource groups”, ”All resources” ja ”Dashboard” (KUVA 2).

KUVA 2. Kotinäkymän navigaatio-osio.

”Subscription”-näkymästä löytyy olemassa olevat tilaukset ja niihin kuuluvat resurssit. ”Resource groups” jaottelee luodut resurssit resurssiryhmien perusteella ja ”All resources” listaa kaikki käyttäjän luomat resurssit. ”Dashboard”-näkymä on muokattavissa ja siihen voidaan kiinnittää itselle oleellisimmat palvelut. Oletuksena myös siellä on nähtävissä kaikki resurssit. ”Dashboard”-näkymiä voi tehdä useita. Käyttäjä voi luoda esimerkiksi itselleen oman näkymän ja yleisen näkymän muille organisaatiossa oleville käyttäjille. (What is the Azure portal? 2023.)

Kolmantena nostan esille resurssin nimeltä ”Free services” eli ilmaiset palvelut, joka löytyy esimerkiksi hakukentän avulla. Se listaa kaikki ilmaiset pilvipalvelut lisätietoineen. Sen kautta on myös kätevä luoda uudet resurssit, jolloin asetukset asetetaan oletuksena vastaamaan ilmaisten palveluiden vaatimuksia. Näin on mahdollista välttää ei-toivotut laskutukset. (Create free services. 2023.)

3.3 Keskeisimmät palvelut serverless-applikaation luonnissa

Serverless-applikaation luomisessa keskityn pääasiassa kolmeen keskeiseen palveluresurssiin: Yksi niistä on isännöintipalvelu, joka mahdollistaa applikaation julkaisun ja käyttöönottoon liittyvät asiat. Tietokanta ja datan varastointiin tarvittavat palvelut ovat oleellisia lähes jokaisessa applikaatiossa. Erityisen keskeinen rooli työssäni on Azuren funktioilla ja ne toimivat serverless-applikaationi selkärankana. Ne yhdistävät muut Azuren palvelut sivustooni.

3.3.1 Azure Functions

Azure Functions on pilvipalvelutuote, joka tarjoaa serverless-pohjaisen ympäristön funktioiden kirjoittamiseen. Funktioiden toiminta perustuu tapahtumiin, jotka laukaisevat funktioiden suoritusprosessin. Azure Functions -dokumentaatiossa puhutaan konseptista nimeltä ”triggers and bindings”. Trigger on jokin tapahtuma, joka määrittää sen, miten ja milloin tietty funktio kutsutaan. Jokaisella funktiolla voi olla vain yksi triggeri, mutta bindauksia voi olla useita samassa funktiossa. (About triggers and bindings. 2023.)

Funktioiden yhdistämisestä toiseen resurssiin käytetään termiä nimeltä ”binding”. Bindaus eli yhdistäminen mahdollistaa sen, että funktioita voidaan käyttää vaivattomasti monenlaisessa eri skenaariossa. Niillä on esimerkiksi mahdollista rakentaa kokonainen web-API, suorittaa ajastettuja tehtäviä tai yhdistää ne tietokantaan, jolloin funktio suoritetaan tietyn tietokantatapahtuman yhteydessä. (Azure Functions scenarios. 2023.)

Yhdistäminen voidaan määritellä sisääntuloon (input bindings), ulostuloon (output bindings) tai molempiin. Yleensä sisääntuloon yhdistetään, kun halutaan hakea tietoa esimerkiksi tietokannasta ja ulostuloon, kun halutaan tehdä muutoksia johonkin resurssiin. Tapahtumat ovat aina sisääntuloon yhdistetty. Vaikka sisään- ja ulostuloon yhdistämisen käsitteet saattavat tuntua aluksi haastavilta, ne ovat

melko loogisia, kun ajatellaan miten data liikkuu funktiossa. Funktio saa sisään tulevan datan parametrina ja ulostuleva data on funktion palauttama data. (About triggers and bindings. 2023.)

Yksittäisen funktion asetukset määritellään function.json nimiseen tiedostoon (KUVA 3). Tiedostossa on ”bindings” niminen attribuutti, joka saa taulukkona kaikki bindaukset. Bindingsin tarvitsemat attribuutit voivat vaihdella riippuen valitusta tyypistä, mutta vähintään on määriteltävä tyyppi, nimi ja suunta. Esimerkiksi jos tahdomme funktion palauttavan HTTP-vastauksen, tyypiksi laitetaan ”http”, nimeksi haluttu muuttujan nimi kuten ”response” ja sunnaksi ”out”. (About triggers and bindings. 2023.)

KUVA 3. HTTP-trigger -mallipohjalla generoitu function.json tiedoston sisältö.

Itse funktion koodi on usein hyvin yksinkertainen ja looginen. Ensimmäinen parametri on tyypiltään ”context”-olio ja funktio saa sen suoritusympäristöltä funktion kutsun yhteydessä. Sen avulla on mahdollista muun muassa lukea bindausten dataa, kirjoittaa lokeja, lukea HTTP-pyynnön dataa ja muokata HTTP-vastausta. Context on pakollinen parametri, mutta sen lisäksi voi määritellä lisäparametreja, joilla pääsee suoraan käsiksi function.json tiedostossa määriteltyihin bindauksiin. Lisäparametrit tulee

antaa funktiolle samassa järjestyksessä kuin ne ovat function.json tiedostossa. (Azure Functions Node.js developer guide. 2023.)

Funktioiden tunnistautumistavaksi on useita vaihtoehtoja. Jos valitaan ”anonymous”, funktioita voidaan kutsua ilman API-avainta. Vaihtoehtoisesti, ”functions”-tunnistautumistavassa vaaditaan funktiospesifinen API-avain. Jos arvoksi asetetaan ”admin”, kutsussa on käytettävä admin-avainta. Yleistä on, että API-avain vaaditaan, jolloin se annetaan arvoksi parametrille nimeltä ”code”. Tyypillisesti funktion URL muodostuu seuraavalla tavalla: https://<APPLIKAATION_NIMI>.azurewebsites.net/api/<FUNCTION_NIMI>?code=<API_AVAIN>.

API-avain ei kuitenkaan välttämättä ole riittävä suojatoimi tuotantoversioon ja tällöin tulisi käyttää toisenlaista tunnistautumistapaa. Yksi vaihtoehto on käyttää Webhookia ja tällöin avain saadaan sen kautta. Webhook saadaan käyttöön antamalla function.json tiedostoon attribuutti ”webHookType” jolle on mahdollista antaa arvoksi joko ”GitHub”, ”Slacks” tai ”genericJson”. GitHubin ja Slackin taustalla on heidän oma logiikkansa ja genericJson nimensä mukaisesti on geneerinen ratkaisu eikä sillä ole palveluntarjoajien asettamia rajoituksia. (Azure Functions HTTP trigger. 2023.)

Azure Functions -dokumentaatio listaa hyviä käytäntöjä, joita tulisi noudattaa funktioita luodessa. Funktioiden suoritusaika kannattaa pitää mahdollisimman pienenä, jolloin yllättäviltä aikakatkaisuun liittyviltä virheiltä voidaan välttyä. Käyttömukavuuden lisäksi nopeat funktiot ovat kustannuksiltaan alhaisemmat. Isot ja hitaat funktiot on syytä jakaa useampaan pienempään funktioon, jotka toimivat yhtenäisesti. Funktioita kirjoittaessa on hyvä ajatella, että poikkeustilanteisiin voi törmätä koska vain. Tällöin ne voidaan ohjelmoida tavalla, että funktioiden suoritusta voidaan jatkaa toisella kertaa, vaikka edellisellä kerralla olisi kohdattu jokin poikkeustilanne. Kolmantena kirjoitusohjeena mainitaan, että funktioiden tulisi olla mahdollisimman tilattomia ja idempotenssisia, kuin se vain on suinkin mahdollista. Tällä on suotuisia vaikutuksia datan eheyteen ja järjestelmän toimivuuteen. (Improve the performance and reliability of Azure Functions. 2023.)

3.3.2 Static web app

Staattiset web-sovellukset ovat sovelluksia, jotka voidaan lähettää käyttäjän selaimeen ilman tarvetta siirtää tiedostoja erikseen palvelimen kautta käyttäjälle. Näitä sovelluksia voidaan käyttää suoraan selaimessa ilman palvelimella tapahtuvaa prosessointia, koska niiden kaikki tarvittavat tiedostot kuten

HTML, CSS ja JavaScript ovat jo valmiina ja saatavilla selaimessa. Tällä voidaan parantaa sivuston latautumisnopeutta ja suorituskykyä, koska ylimääräinen liikenne palvelimen ja selaimelle välillä vähenee. (What is Azure Static Web Apps? 2023.)

Azuressa oleva Static web app -niminen palvelu mahdollistaa vaivattoman web-applikaation julkaisuprosessin Azureen. Julkaisu tapahtuu automaattisesti koodimuutosten perusteella, kun Static web app seuraa tietyn tietovaraston (englanniksi repository) muutoksia esimerkiksi GitHubissa. Tyypillisiä muutoksia ovat työnnöt (git push) ja vetopyynnöt (pull request). (What is Azure Static Web Apps? 2023.)

Static web app -palveluresurssin luonnin yhteydessä Azure luo satunnaisesti generoidun verkkotunnuksen ja tarjoaa sille tarvittavan SSL-sertifikaatin. Verkkotunnus on myös vaihdettavissa omavalintaiseen tunnukseen ja se on mahdollista joko Azure DNS -palvelun tai jonkin ulkopuolisen DNS-palveluntarjoajan kautta. (Custom domains with Azure Static Web Apps. 2023.)

Static web app -palveluun on sisäänrakennettu serverless-rajapinta, johon on mahdollista integroida Azuren palveluja, kuten Azuren funktiot. Rajapinta mahdollistaa integroitujen Azure-palvelujen kutsumisen ilman ylimääräisiä CORS-konfiguraatioita ja suora pääsy käyttäjän todennukseen liittyviin tietoihin. Tällöin ei ole tarvetta toteuttaa erillistä funktiota todennukseen hallintaan liittyen. (What is Azure Static Web Apps; Accessing user information in Azure Static Web Apps. 2023)

Stati web app -palvelun asetukset koostuvat kolmesta eri kokonaisuudesta: Applikaatiokonfiguraatio määritellään staticwebapp.config.json-nimiseen tiedostoon ja se on vastuussa applikaation perusasetuksista kuten tietoturvasta, reitityksestä ja omien otsakkeiden määrittelystä. Sovelluksen rakennus- ja käyttöönottoprosessiin liittyvät konfiguraatio määritellään YAML-tiedostoon, joita esimerkiksi GitHub Actions käyttää hyväksi. Applikaatioasetukset kuten tietokantojen yhdistämiseen käytetyt merkkijonot voidaan määrittää Azure-portaalin kautta. (Configuration overview. 2022.)

Static web app -resurssista on valittavana, joko ilmainen tai maksullinen isännöintivaihtoehto. Ilmainen vaihtoehto on tarkoitettu henkilökohtaisille projekteille ja se sisältää lähes samat ominaisuudet kuin maksullinen vaihtoehto. (Azure Static Web Apps hosting plans. 2023.)

3.3.3 Blob Storage

Blob Storage on Azuren tarjoama oliovarasto jäsentämättömälle datalle ja se on suunniteltu suuren datamäärän varastointiin. Azure tarjoaman REST-rajapinnan kautta Blob Storageen yhteyden ottaminen onnistuu vaivattomasti. Tähän löytyy valmiit kirjastot suosituille ohjelmointikielille, kuten esimerkiksi Nodelle, Pythonille, Go:lle ja .NET:lle. Blob Storage koostuu kolmesta tärkeästä resurssista: Storage accountista eli varastointitilistä, ”containerista” eli kontista ja ”Blobista” eli varastoidusta resurssista. Varastointitilistä on kaiken alkupiste ja sen alta löytyy niin kontit, kuin kaikki Blob storagessa oleva data. Varastointitilejä voi olla useita vaikka sana ”tili” saattaa antaa kuvan, että niiden määrä olisi rajattu vain yhteen. Varastotili tarjoaa nimiavaruuden, joten tilin nimen täytyy olla täysin uniikki ja sisältää vain merkkejä, jotka on sallittu osana URI:a. Sen avulla päästään käsiksi varastossa olevaan dataan HTTP-pyyntöjä käyttäen. Varastointitilejä on erityyppisiä ja niiden ominaisuudet eroavat toisistaan. Tilin tyyppi valitaan tilin luontihetkellä eikä sitä voi muuttaa jälkikäteen. Tyyppi vaikuttaa myös hinnoitteluun, joten on tärkeää tehdä valinta oman projektin tarpeiden mukaan. Esimerkkinä tyypeistä voidaan mainita ”Standard general purpose v2”, joka on suositeltu valinta useimpia skenaariota varten ja ”Premium block blobs”, joka sopii tapauksiin, joissa transaktioiden määrät ovat korkeita tai käyttöviiveen minimointi on tärkeää palvelulle. (Storage account overview. 2023.)

Kontin voi karkeasti ottaen ajatella kansioksi tai hakemistoksi, joka sisältää tiedostoja. Varastointitilillä voi olla teoriassa loputon määrä eri nimisiä kontteja ja yksi kontti voi sisältää loputtoman määrän Blobeja. Kontin nimen täytyy olla tilin tavoin uniikki ja merkeiksi kelpaavat vain kirjaimet, numerot ja väliviiva, koska tilin tavoin kontin nimi on osana lopullista URI:a. Blob on Blob Storagen resurssi, joka voi olla tarkemmalta tyypiltään, joko ”Block blob”, ”Append blob” tai ”Page blob”. Blobin tyyppi päätetään sen luontihetkellä eikä sitä pysty muuttamaan jälkikäteen. (Understanding block blobs, append blobs, and page blobs. 2023.)

Block blobit on suunniteltu suuren datamäärän varastointiin ja Block blob koostuu nimensä mukaisesti blockeista eli pienemmistä osista. Yksi blob voi sisältää enimmillään jopa 50 000 blockia. Blockien koko voi vaihdella blobissa ja maksimikoko määräytyy valitun palveluversion pohjalta. Esimerkiksi versio ”2019-12-12” ja sitä myöhemmät sallivat maksimissaan 400 MiB blockin koon. Kokonaisen blobin maksimaalinen koko saadaan siis laskutoimituksella 50 000 kertaa 400 MiB, joka on noin 190,7 TiB (tebitavua). Blob, jonka koko on pienempi kuin version määräämä yksittäisen blobin maksimi koko voidaan lähettää Blob storageen kokonaisena, mutta blobin ylittäessä tämän rajan, se joudutaan lähettämään useammassa osassa. Tämä on mahdollista, koska jokaisella blockilla on ”block ID”, joka

toimii tunnisteena ja sen avulla blockit kyetään yhdistämään oikeaan Blobiin. (Understanding block blobs, append blobs, and page blobs. 2023.)

Page blobit on suunniteltu erityisesti satunnaisia ja usein tapahtuvia luku- ja kirjoitusoperaatioita varten. Page blobit koostuvat 512 tavun kokoisista sivuista ja blobin maksimikoko voi olla noin 8 teratavua. Sivut mahdollistavat kirjoittamisen ja lukemisen satunnaisiin tavualueisiin. Tämä ominaisuus tekee Page blobeista erityisen hyödyllisiä, kun tarvitaan säilytystilaa tietorakenteille, jotka käyttävät indeksejä tai joissa on paljon tyhjää tilaa, kuten käyttöjärjestelmän levyt, virtuaalikoneiden datalevyt ja tietokannat. Esimerkiksi Azure SQL-tietokanta käyttää hyödyksi Page blobeja. (Overview of Azure page blobs. 2023.)

Append blob on nimensä mukaisesti blob, johon liitetään uutta dataa. Esimerkiksi yksi yleinen käyttökohde on lokitiedosto. Koska lokitiedosto on tyypiltään Append block, sen perään voidaan liittää uudet lokitekstit ilman, että koko tiedosto on luettava tai olemassa olevia tekstejä on muokattava. Jos sama toteutettaisiin Block blobin avulla, koko tiedosto olisi luettava tai korvattava uudella datalla. (Smikar software. 2023.)

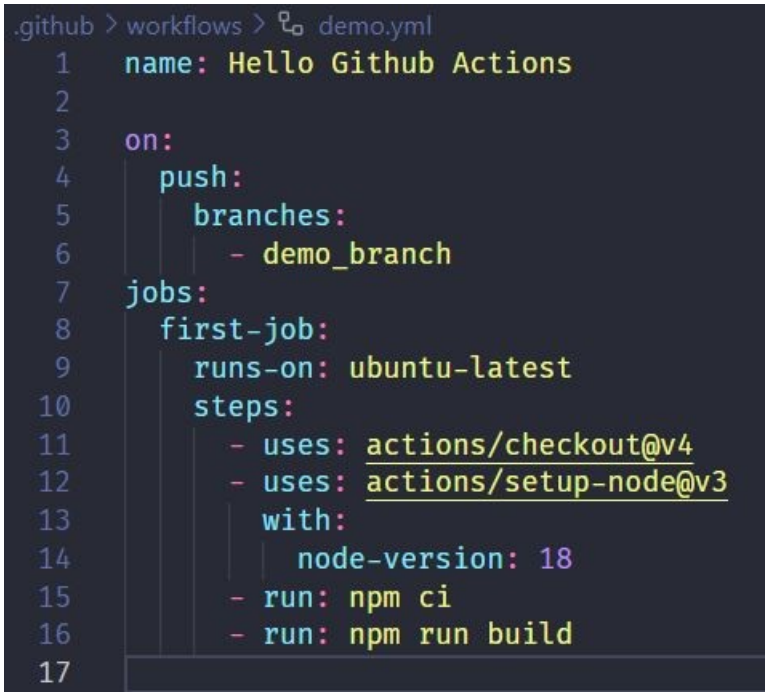
3.3.4 GitHub Actions

Actions on GitHubin tarjoama CI/CD (Continuous Integariton and Continous deployment) ratkaisu. Sen avulla on mahdollista muodostaa prosessi (kutsutaan workflowiksi), joka koostuu erilaisista automaattisista toiminnoista kuten käännöksestä, testeistä ja applikaatioon julkaisusta, kun esimerkiksi uudet muutokset yhdistetään tiettyyn haaraan GitHub-tietovarastossa. Actionsien käyttö ei rajoitu pelkkiin koodiin liittyviin muutoksiin, vaan ne voi yhdistää moniin eri tietovarastossa tapahtuviin muutoksiin.(Understanding GitHub Actions.)

Prosessi koostuu yhdestä tai useammasta työstä eli ”jobista”. Jokainen työ taas sisältää sarjan eri toimintoja, kuten skriptien suorituksia ja pakettien asennuksia. Kaikki vaiheet suoritetaan aina samassa järjestyksessä. Työt suoritetaan oletuksena samanaikaisesti, mutta käyttäjän on mahdollista määrittää erinäisiä riippuvuuksia töiden kesken. Tällöin esimerkiksi yksi töistä odottaa, kunnes toinen työ on suoritettu loppuun. Prosessi suoritetaan virtuaalipalvelimilla, joita kutsutaan ”runnereiksi” ja yksi runneri pystyy suorittamaan yhden työn kerralla. Runnerin käyttöjärjestelmänä voi olla joko Windows, macOs tai Ubuntu Linux. (Understanding GitHub Actions. 2023.)

”Workflow”-prosessi määritellään YAML-tiedostoon ja se sijoitetaan tietovarastoon ”github/workflow”-nimiseen kansioon. YAML-tiedoston ”name”-attribuutti on vapaaehtoinen ja sillä on mahdollista asettaa nimi ”workflowille”. Attribuutilla ”on” määritetään milloin ”workflow” suoritetaan. Se on pakollinen ja siihen on määritettävä vähintään yksi tapahtuma eli triggeri, joka käynnistää suorituksen. ”Jobs” on myös pakollinen ja sille on määriteltävä runneri eli ympäristö missä työt suoritetaan. Lisäksi se tarvitsee yhden tai useamamman stepin eli työvaiheen. Steppeihin voidaan määrittää. ”Uses”-attribuutilla voidaan ottaa käyttöön toimintoja, joilla voidaan suorittaa valmiiksi määriteltyjä tehtäviä, jotka voivat koostuvat useasta monimutkaisesta komennosta. ”Run” attribuutilla voidaan määritellä yksittäinen komento (Workflow syntax for GitHub Actions. 2023.)

Kuva 4 näyttää toimivan workflow-tiedoston, jonka työvaiheet suoritetaan, kun ”demo_branch”-niminen haara työnnetään GitHubiin. Se asentaa Node version 18 ja suorittaa kaksi ”npm”-komentoa.



KUVA 4. Workflow-tiedoston sisältö.

4 SERVERLESS-APPLIKAATION LUONTI AZUREEN

Tavoitteenani tässä luvussa on kuvata kuinka applikaation luonti Azuren pilvipalveluun onnistuu. Otan käyttöön pilvipalvelutuotteita, joita esittelin edellisessä luvussa. Tämän opinnäytetyön tavoitteena on luoda selkeähkö ohjeistus serverlessin ja Azuren käyttöön. Palveluiden valintoihin vaikuttivat myös henkilökohtaiset oppimistavoitteeni, joten palveluiden valinnat eivät välttämättä ole kaikissa kohdissa optimaalisimpia tapoja rakentaa applikaatio Azureen. Esimerkiksi halusin kehittää SQL-osaamistani, vaikka dokumenttitietokanta olisi voinut olla otollisempi valinta näin yksinkertaiseen tietokannan rakentamiseen. Syytä on mainita, että ohjelmoinnissa on yleinen tapa käyttää muuttujien ja resurssien nimeämiseen englantia, joten käytän sitä käytännöntyössäni.

4.1 Sovelluksen suunnittelu

Luodaan web-applikaatio, jossa käyttäjän on mahdollista selata eri maiden tietoja. Jokaisesta maasta haluamme seuraavat tiedot: väkiluku, pääkaupunki, pinta-ala ja kuva lipusta. Lisäksi maiden pääkaupungeista on mahdollista saada tarkentavia tietoja kuten pääkaupungin väkiluku ja sijaintitietoja. Maasta ja pääkaupungeista tehdään kaksi erillistä SQL-taulua.

Web-sivulla listataan tietokannassa olevat maat. Tehdään jokaisesta taulukon rivistä klikattava, jolloin käyttäjän on mahdollista saada lisätietoja maasta ja sen pääkaupungista. Tässä näkymässä käyttäjällä on mahdollisuus poistaa tai muuttaa maahan liittyviä tietoja. Maiden lisääminen tapahtuu erillisestä näkymästä, joka aukeaa pääsivulla olevasta napista. Maiden tiedot lisätään HTML-lomakkeen avulla ja kuva lipusta lisätään joko raahaamalla ja pudottamalla (drag & drop) jpg-tyyppinen kuva tai selaamalla tiedostoja Windowsista.

Nyt saatavilla on tarvittavat tiedot API:n suunnitteluun. Tehdään päätös, että kaikki liikenne kulkee Azuren funktioiden kautta, vaikka esimerkiksi pystyisimme kommunikoimaan Blob storagen kanssa suoraan web-applikaation kautta. Applikaation toiminta on suoraviivainen, kun funktiot hoitavat HTTP-pyyntöihin vastaamisen. Maita ja kaupunkeja on siis pystyttävä lisäämään, poistamaan ja muokkaamaan HTTP-pyyntöjä avulla. Lisäksi lippujen kuvien hakeminen, poistaminen ja lisääminen tapahtuu HTTP-pyynnöillä.

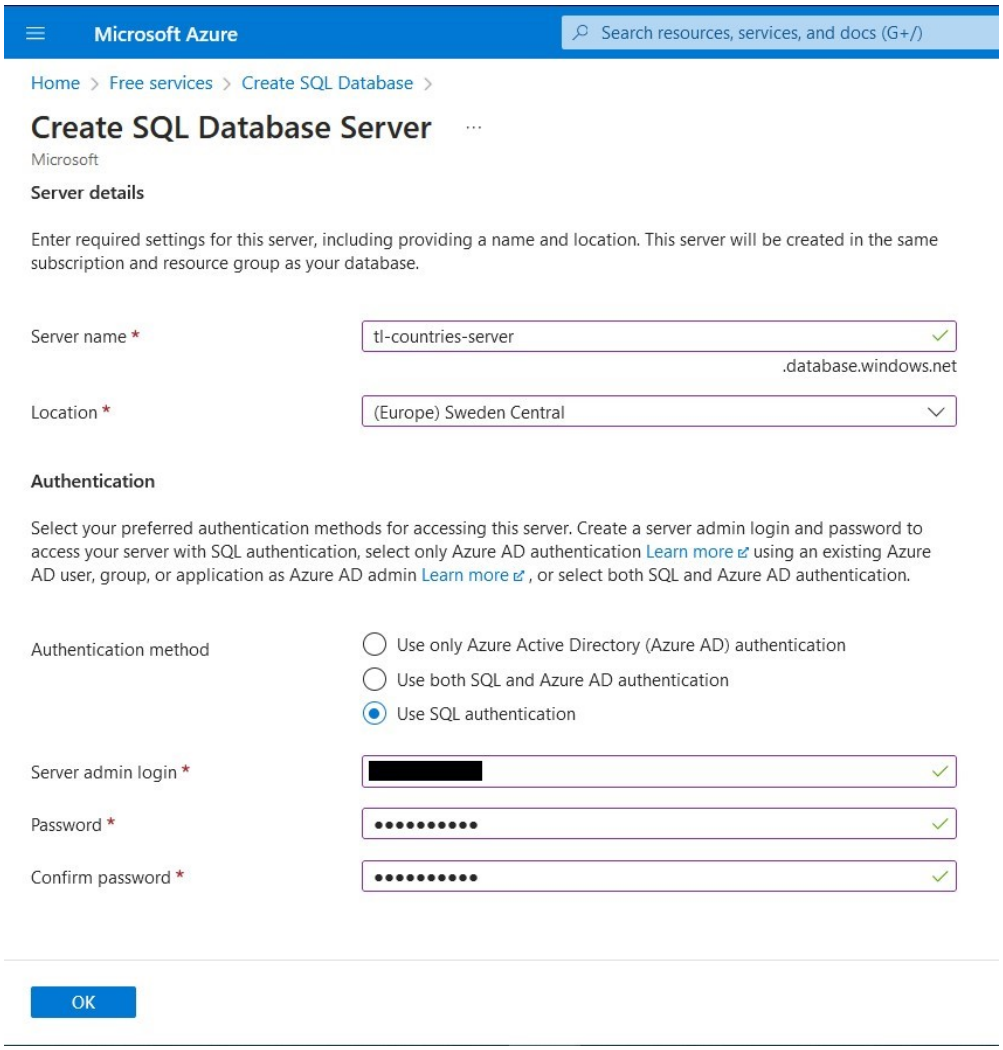
4.2 Tietokannan luonti

Etsitään Azuresta ”Free services”-niminen palvelu, josta löytyy Azure SQL Database. Sen alapuolelta klikataan ”Create”-nappia, jolloin tietokannan luontinäkymä ilmestyy ja tietokannan perustietojen täyttäminen on mahdollista.

”Project Details”-nimisen osion alta löytyy ”subscriptions” ja ”resource group” -kentät. Valitaan olemassa oleva tilaus ”subscriptions”-kenttään. ”Resource group”-kentän alapuolelta painetaan ”create new”-nappia ja luodaan uusi resurssiryhmä. Käytän helposti tunnistettavaa nimeä ”countries”. Asetan kaikki applikaatioomme liittyvät resurssit tähän resurssiryhmään.

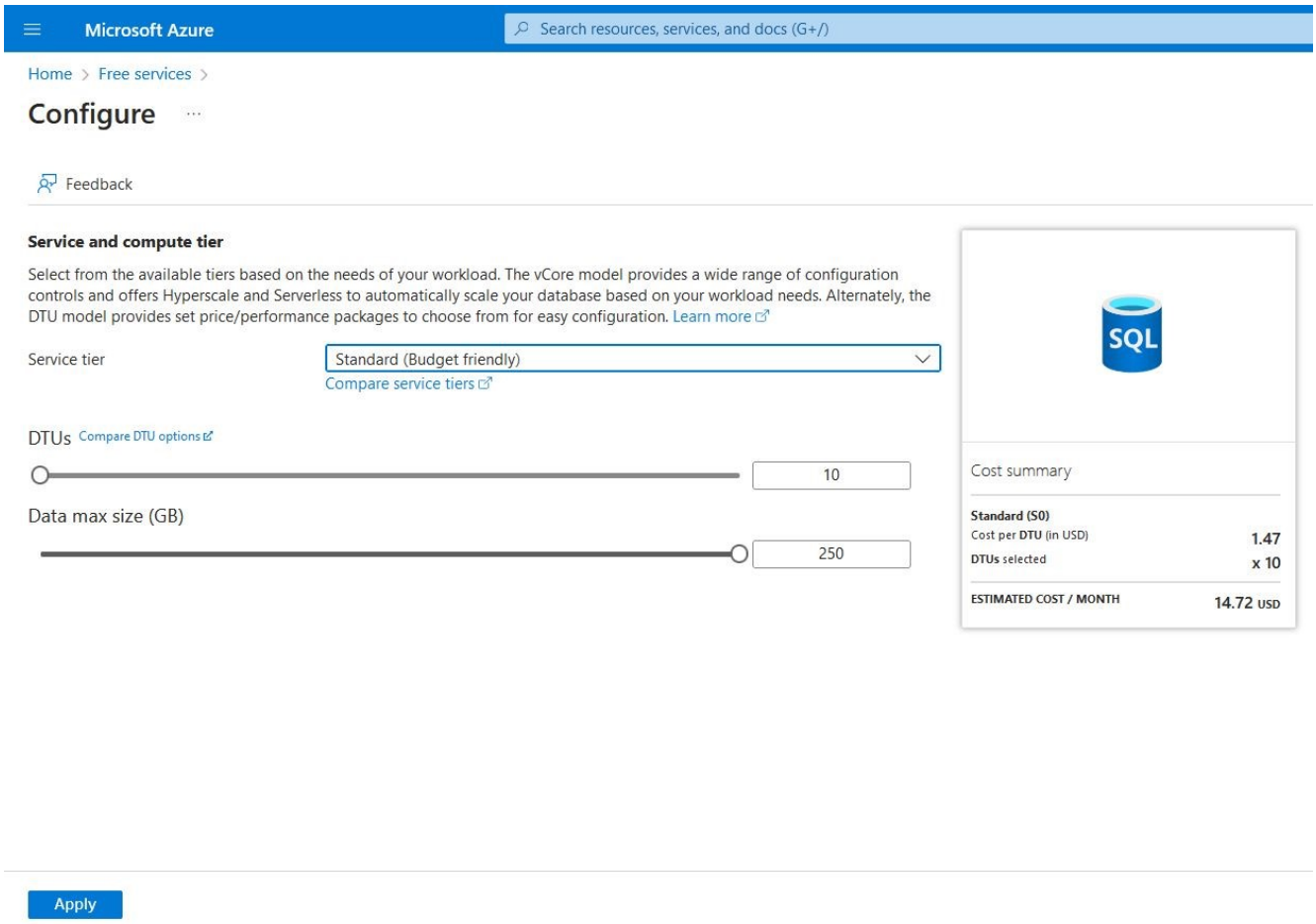
Nimetään tietokanta globaalisti uniikilla ja kuvaavalla nimellä. Azure kertoo, jos nimi on saatavilla. Nimen löydyttyä voimme siirtyä SQL-serverin asetuksiin. Klikataan ”Creat new”-linkkiä palvelinkentän alapuolelta.

Palvelimen nimen tulee myös olla uniikki ja sijainniksi kannattaa valita paikka, joka on mahdollisimman lähellä loppukäyttäjiä. Tässä tapauksessa käyttäjänä on vain minä itse, joten valitsen itseäni lähimpänä olevan sijainnin eli Ruotsin. Todennustavoista perintein SQL-todennus on nopea ja yksinkertainen toteuttaa, joten päätän aloittaa sillä. Asetuksia on mahdollista vielä jälkikäteenkin muuttaa. Syötetään tunnukset ja luodaan SQL-palvelin painamalla OK-nappia. (KUVA 5.)



KUVA 5. SQL-tietokantapalvelimen luontinäkymä

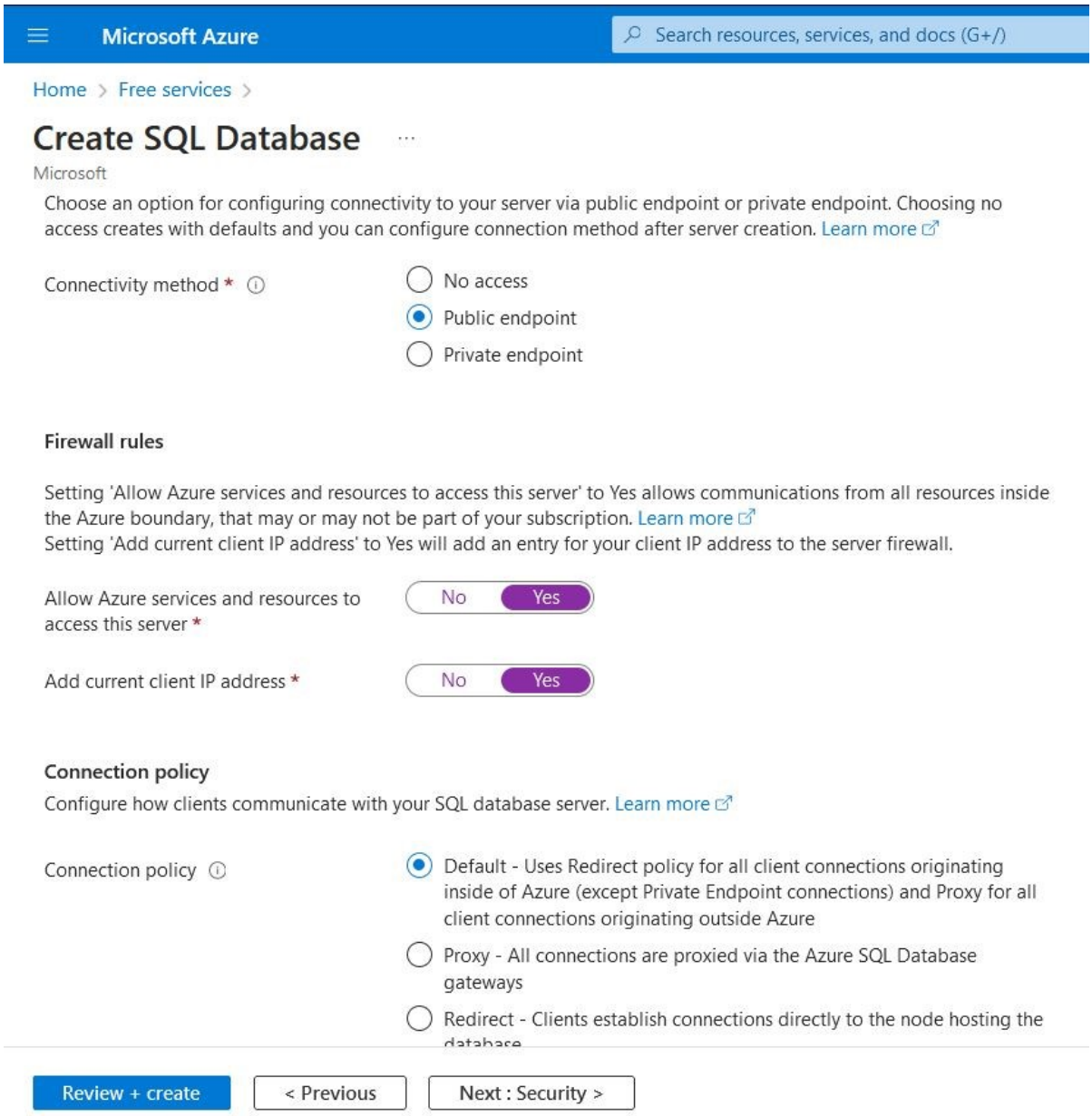
Perustiedot näkymässä muokkaamme vielä ” Compute + storage”-asetuksia. ”Configure database”-linkki avaa tietokannan konfigurointinäkymän. Valitaan asetukselle ”Service tier” arvoksi ”Standard (Budget friendly)” -valinta. Tässä kohtaa kannattaa varmistaa, että asetukset vastaavat ”free tierin” vaatimuksia. Ne saattavat muuttua useinkin, joten kannattaa olla tarkkana. Vuonna 2023 ilmaisuuden vaatimuksena olivat, että DTUs arvo on 10 ja datan maksimi koon arvo on 250GB. (KUVA 6; Try Azure SQL Database free with Azure free account. 2023)



KUVA 6. Tietokannan konfigurointi

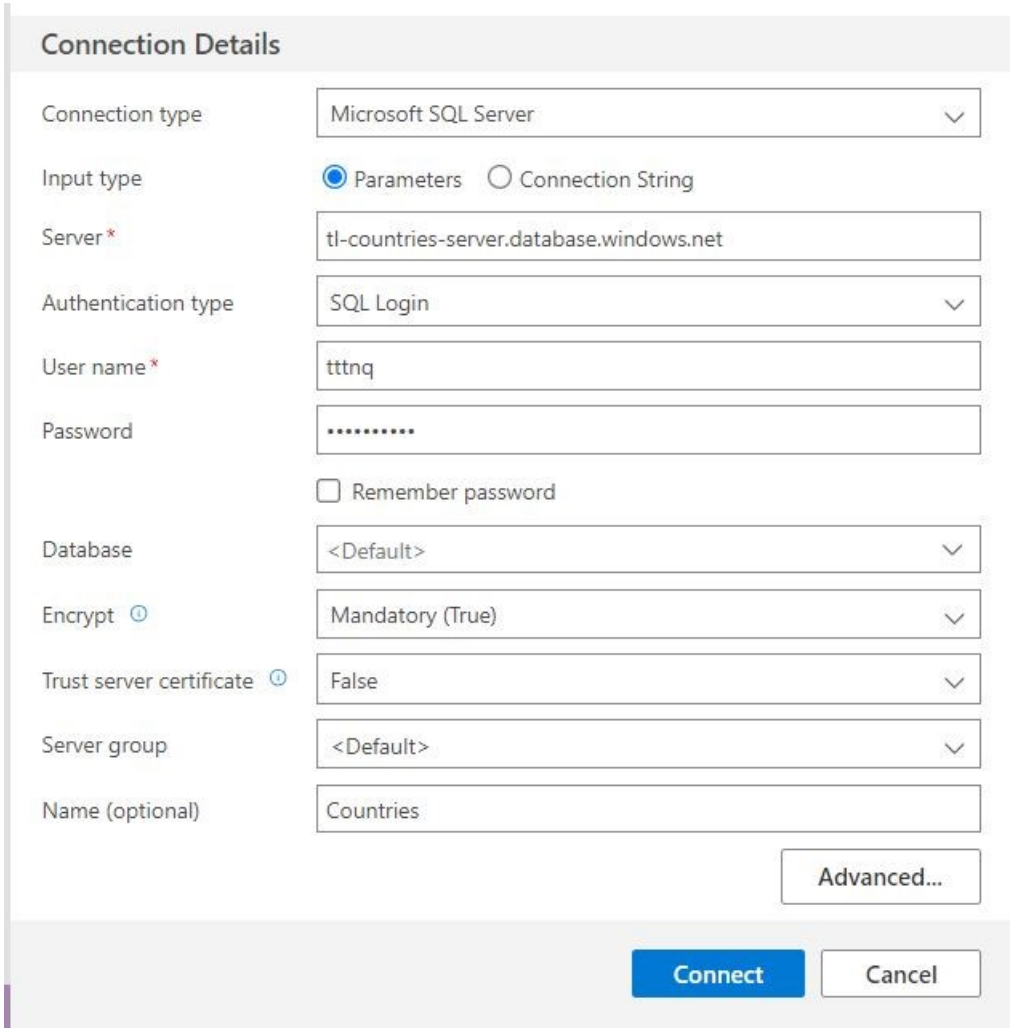
Siirrytään verkkoasetusten konfigurointiin painamalla ”Network”-nappia näkymän alaosasta. Muutetaan yhdistysmenetelmäksi ”Public endpoint”. Tämän jälkeen muutetaan palomuuriasetuksia siten, että hyväksymme toisten Azuren palvelutuotteiden pääsyn SQL-serverillemme. Lisätään vielä nykyisen klientin IP-osoite asettamalla sitä koskeva asetus päälle. Muut asetukset voimme jättää oletusarvoiksi verkkoasetusnäkymässä.

Loput asetukset kuten tietoturva voidaan jättää oletusarvojen varaan. Painetaan ”Review + create” -nappia (KUVA 7). Tässä tarkistusnäkymässä näkyvät kaikki asetukset yhdelle sivulle koottuna. Huomioitavaa on se, että asetukset vastaavat ”Free tierin” vaatimuksia. Asetukset ollessa kunnossa, voidaan jatkaa ja luoda SQL-serveri ja -tietokanta klikkaamalla ”Create”-nappia. Luonnin valmistuttua, voimme siirtyä luomaamme resurssiin ”Go to resource”-nappia painamalla tai etsimällä sen Azure Portalista.



KUVA 7. SQL-serverin verkkoasetusten konfigurointi

Nyt tietokanta on Azuressa, joten SQL-taulujen luonti voidaan aloittaa, kuten tämän luvun alussa suunniteltiin. Datan luominen tietokantaan on mahdollista monella tavalla, mutta valitsen aputyökaluksi Microsoftin ohjelmiston nimeltä Azure Data Studio. Azure Data Studiossa luodaan uusi yhteys ja täytetään yhdistämiseen tarvittavat tiedot. Palvelimen nimen löytää Azuresta SQL-tietokannan yleiskatsausnäkymästä (overview). Kopioidaan se ja liitetään Data Studioon. Seuraavaksi vaihdetaan tunnistautumistyypiksi SQL-kirjautuminen ja syötetään tunnukset, jotka luotiin aikaisemmin. (KUVA 8.)



KUVA 8. Azure Data Studio yhdistämisasetukset

Yhdistämisen jälkeen luodaan taulut maita ja pääkaupunkeja varten. Azure Data Studiolla on mahdollista tehdä SQL-kyselyitä, joten taulujen luominen onnistuu kuvien 9 ja 10 mukaisesti.

```
1   CREATE TABLE Country (
2       CountryId VARCHAR(255) PRIMARY KEY,
3       CountryName VARCHAR(255) NOT NULL,
4       Capital VARCHAR(255) NOT NULL,
5       Population INT,
6       Area INT
7   );
```

KUVA 9. SQL-käsky, joka luo taulukoiden maita varten

```sql
CREATE TABLE Capital (
    CityId VARCHAR(255) PRIMARY KEY,
    CountryId VARCHAR(255) NOT NULL,
    CityName VARCHAR(255) NOT NULL,
    Population INT,
    Latitude FLOAT,
    Longitude FLOAT,
    FOREIGN KEY (CountryId) REFERENCES Country(CountryId)
);
```

KUVA 10. SQL-käsky, joka luo taulukon pääkaupunkeja varten

Taulujen luonnin jälkeen asetetaan hieman testidataa tauluihin ”INSERT INTO” käskyllä. Käytetään Azure SQL-serverin tarjoamaa ”NEWID()”-funktiota, joka generoi uniikin tunnisteen. (KUVA 11.)

```sql
INSERT INTO dbo.Country (CountryId, CountryName, Capital, Population, Area)
VALUES
    (NEWID(), 'Finland', 'Helsinki', 5559521, 338472),
    (NEWID(), 'Sweden', 'Stockholm', 10529000, 450295),
    (NEWID(), 'United States', 'Washington, D.C.', 331449281, 9826630),
    (NEWID(), 'Ukraine', 'Kyiv', 45553047, 603628),
    (NEWID(), 'Japan', 'Tokyo', 124490000, 377975),
    (NEWID(), 'China', 'Beijing', 1411778724, 9596961),
    (NEWID(), 'India', 'New Delhi', 1393409038, 3287590),
    (NEWID(), 'Germany', 'Berlin', 84359000, 357588),
    (NEWID(), 'Brazil', 'Brasília', 202656788, 8514877),
    (NEWID(), 'Canada', 'Ottawa', 38527000, 9984670);
```

KUVA 11. Esimerkki maiden lisäämisestä.

Samalla periaatteella voidaan asettaa dataa taulukkoon, joka sisältää pääkaupunkien tiedot. Lopuksi testataan, että tietokanta sisältää halutun datan. Kuva 12 näyttää SQL-käskyn, joka hakee molemmista tauluista nimi- ja asukasmäärätietoja.

```
1   SELECT C.CountryName, C.Population, Cap.CityName, Cap.Population
2   FROM Country AS C
3   LEFT JOIN Capital AS Cap
4   ON C.CountryId = Cap.CountryId;
```

Results Messages

	CountryName	Population	CityName	Population
1	China	1411778724	Beijing	19612368
2	Finland	5559521	Helsinki	664921
3	India	1393409038	New Delhi	249998
4	Sweden	10529000	Stockholm	978770
5	Ukraine	45553047	Kyiv	2952301
6	Brazil	202656788	Brasília	2482210
7	Germany	84359000	Berlin	3574830
8	Japan	124490000	Tokyo	13995469
9	United States	331449281	Washington, D.C.	712816
10	Canada	38527000	Ottawa	934243

KUVA 12. SQL-pyyntö, joka palauttaa nimi- ja asukastietoja

4.3 Varastotilan määritys

Kuvia varten Azuresta löytyy ”Azure Blob Storage”-niminen resurssi. Luodaan uusi varastotila applikaatiota varten, jonne voidaan tallentaa kuvat maiden lipuista. Varastoa luodessa asetus, joka tulee ottaa huomioon, on redundanssi, jonka arvoksi valitaan LRS (locally redudant stroage). LRS on ”free tier”-vaatimus ja kuvauksen mukaan sitä suositellaan ei-kriittisiä skenaarioita varten, joten se on sopiva valinta applikaatiolle. Muut asetukset voidaan jättää oletuksille ja suorittaa resurssin luonti.

Onnistuneen resurssin luonnin jälkeen siirrytään tutkimaan sitä tarkemmin. Skrollataan alaspäin vasemmalla olevaa valikkoa ”Data storage”-osioon saakka ja klikataan siellä olevaa ”containers”-valintaa. Tässä konttinäkymästä voidaan luoda uusi kontti klikkaamalla yllä olevaa ”container”-nappia. Tehdään kontti lippuja varten antamalla sille nimi ”flags”. Navigoidaan flags-konttiin ja siirretään sinne tietokannassa olevien maiden liput. Googlettamalla löytyy paljon vapaasti käytettäviä lippujen kuvia.

4.4 API:n luonti

Tietokannassa on tarpeeksi testidataa, jotta voidaan aloittaa Azuren funktioiden yhdistämisen tietokantaan. Haetaan portaalista jälleen ”Free services”-resurssi ja skrollataan näkymää alas, kunnes löytyy resurssi ”Azure functions”.

Luodaan uusi ”Azure functions”-resurssi ja täytetään tiedot samaan tapaan, kuin tietokantaresurssia luodessa. Perustiedotnäkymässä valitaan sama tilaus ja resurssiryhmä, kuin tietokannalle. Suoritusympäristöksi valitaan Node.js ja versioksi 18 LTS. Sijainniksi valitaan taas lähin vaihtoehto eli Ruotsi. Muut asetukset jätetään oletuksille.

Seuraavaksi muutetaan varastoasetuksia (storage). Sieltä löytyy asetus nimeltä ”Storage account”. Tehdään uusi varasto, jonka jälkeen nimetään se tai käytetään Azuren oletuksena ehdottamaa valmiiksi nimettyä varastoa. Azure functions tarvitsee varaston mm. logien ja triggereiden hallitsemista varten (Storage considerations. 2023).

”Monitoring”-asetuksista otetaan ”application insight”-niminen asetus pois päältä lisäkulujen välttämiseksi. Muut asetukset voidaan jättää koskemattomiksi ja siirtyä ”tarkastus ja luonti” -vaiheeseen. Tarkastetaan asetukset ja luodaan resurssi. (KUVA 13.)

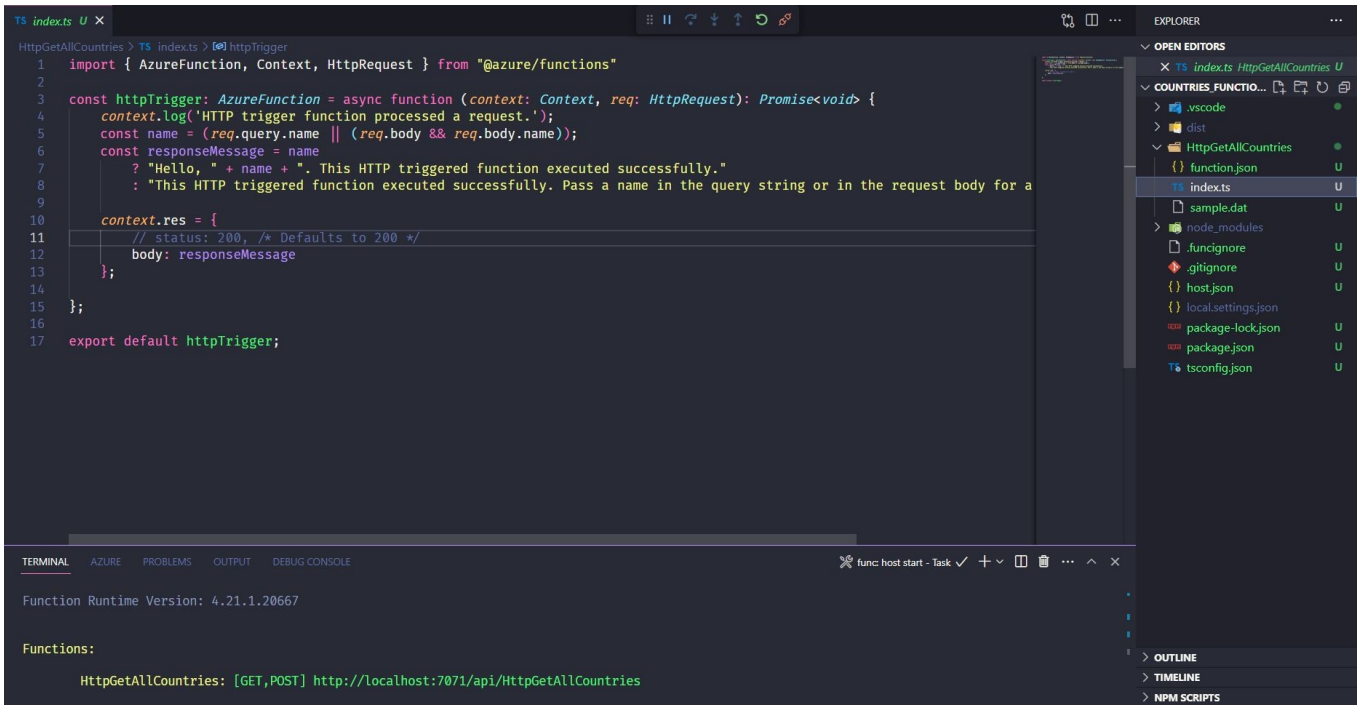
KUVA 13. Yhteenveto ”Azure Function”-applikaation asetuksista.

Siirrytään seuraavaksi tarkastelemaan vasta luotua resurssia. Yleiskatsausnäkymän alaosassa Azure esittelee kolme eri tapaa funktioiden luontiin: Portaalin, VS Coden tai komentorivin kautta. Valitaan VS Coden, koska sen avulla JavaScriptin ohjelmointi on luontevaa ja funktioiden sijoittaminen Azureen on yksinkertaista.

Käynnistetään VS Code ja ladataan ”Azure Functions”-niminen lisäosa. Lisäosan avulla saadaan yhteys Azure-tiliin, pystytään hallitsemaan ja sijoittamaan funktiot Azureen. Seuraavaksi tehdään ensimmäinen funktio komentopaletin kautta painamalla Ctrl + shift + P ja kirjoittamalla ilmestyvään tekstipalkkiin ”Azure functions: create function”.

Ohjattu funktioiden luonti tapahtuu vaiheittain. Ensimmäisessä vaiheessa annetaan projektin sijainti, toisessa vaiheessa valitaan ohjelmointikieli, jolla funktio halutaan ohjelmoida. Valitaan TypeScript, jolloin jatkokysymyksenä kysytään mikä malli otetaan käyttöön. Valitaan V3-mallin, sillä tätä opinnäytetyötä tehdessä V4-malli on vielä ”preview” eli esikatselutilassa. Seuraavassa vaiheessa valitaan mallipohja funktiolle. Suunnitteluun perustuen, tarvitaan HTTP-pyyntöihin perustuvia triggereitä, joten valinta on ”HTTP trigger” -mallipohja. Annetaan funktiolle nimi ”httpGetAllCountries”. Viimeisessä vaiheessa valitaan tunnistautumistavaksi funktiopohjainen tunnistautuminen.

Funktion luontiprosessin jälkeen voidaan kokeilla sen testaamista paikallisesti. Jotta funktioita voidaan suorittaa omalla koneella, on ladattava ”Azure Function Core Tools” niminen paketti. Asennuksen jälkeen pitäisi olla mahdollista käynnistää funktiopalvelin paikallisesti aloittamalla debuggaus VS Codessa, mikä onnistuu esimerkiksi F5-paniketta painamalla. Palvelimen käynnistyttyä terminaalissa näkyy linkki, joka voidaan avata selaimessa. Selaimeen pitäisi tulostua oletusteksti, joka on määritelty funktion koodissa. (KUVA 14.)

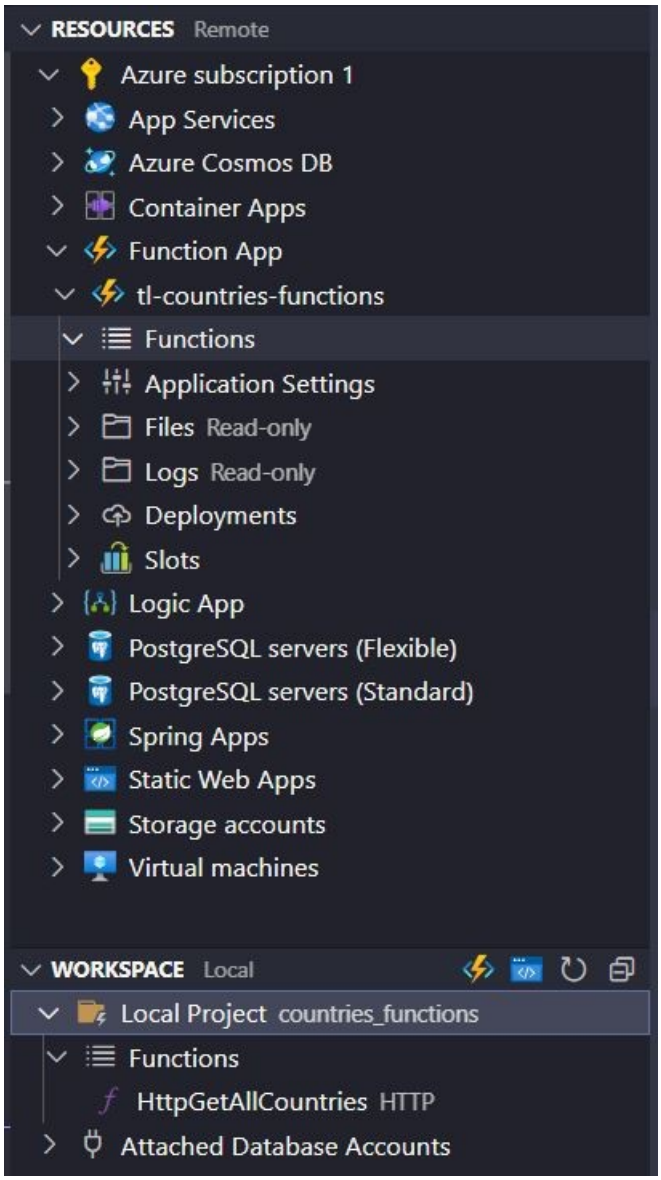


KUVA 14. Funktion koodi VS Code -ohjelmassa.

Funktion koodi sijaitsee index.ts-tiedostossa. Tiedosto löytyy kansiosta, joka on nimetty funktion mukaan eli tässä tapauksessa ”HttpGetAllCountries”-kansio. Toinen tärkeä samassa kansiossa oleva tiedosto on function.json. Se sisältää funktion konfiguraatioasetukset, kuten bindaukset.

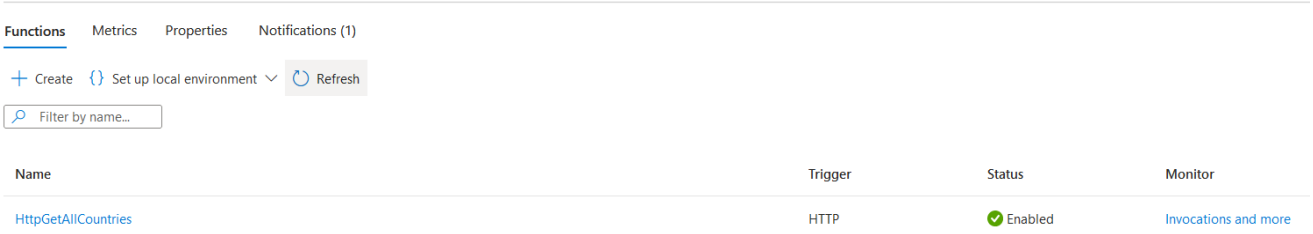
4.4.1 Funktioiden sijoitus Azureen

VS Coden sivupalkista siirrytään Azure-näkymään, jolloin näkyviin tulevat resurssit ja työtila. Resurssiosiosta löytyy sisäänkirjautumispainike, jota painamalla aukeaa selaimeen Microsoftin kirjautumissivusto. Suoritetaan kirjautuminen, jonka jälkeen VS Codessa pitäisi näkyä tilaus ja sen alla eri resursseja, kuten ”Azure Function app”. (KUVA 15.)



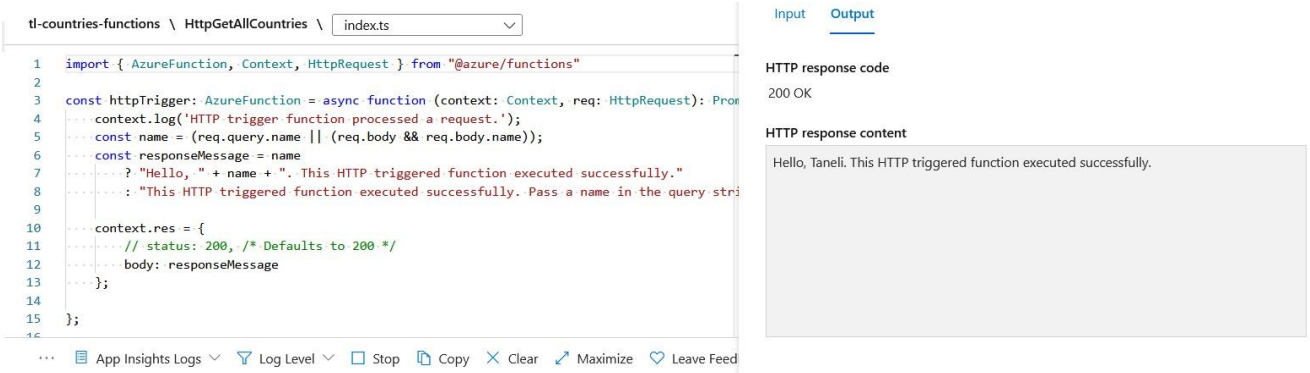
KUVA 15. Resurssit VS Codessa kirjautumisen jälkeen

”Workspace”-osio sisältää paikalliset funktiot, kuten juuri luotu ”HttpGetAllCountries”-funktio. ”Workspace”-tekstin vieressä on ”Azure functions”-kuvake, jota klikkaamalla voimme suorittaa funktioiden sijoittamisen Azureen valitsemalla valikosta vaihtoehdon ”Deploy to Function App…”. VS Code varoittelee vielä, että sijoitusprosessi yli kirjoittaa Azuressa mahdollisesti jo olevat funktiot. Hoidetaan kaikki funktioiden sijoitus Azureen VS Coden avulla, jolloin ei tarvitse välittää varoituksesta. Funktion pitäisi nyt näkyä Azure-portaalissa. Siirrytään ”Azure Functions”-applikaatioon portaalissa, jolloin yleisnäkymäsivun alaosassa pitäisi näkyä funktio. (KUVA 16.)



KUVA 16. Funktiot listattuna Azuressa

Kuvassa näkyy luotu ”HttpGetAllCountries”-funktio ja sitä klikkaamalla aukeaa uusi näkymä. Sieltä löytyy valinta nimeltä ”Code + Test”, jossa voidaan testata funktion toiminnallisuutta. Painetaan ”Test/Run”-nimistä nappia, jolloin sivupalkki aukeaa. Sivupalkissa voidaan valita HTTP-metodi, avaimen, asettaa parametreja ja otsakkeita sekä lähettää dataa HTTP-bodyssä. Index.ts-tiedoston esimerkkikoodi tulostaa erilaisen tekstin, mikäli HTTP-POST-pyynnön mukana lähetetään ”name”-parametri tai HTTP-bodyssa on ”name”-attribuutti. Kun arvo on annettu, painetaan suorita-nappia, jolloin pitäisi tulostua teksti, johon on sisällytetty annettu arvo. (KUVA 17.)



KUVA 17. Testaus suoritettu antamalla oma nimi arvoksi

4.4.2 Tietokannan yhdistäminen

Siirrytään takaisin koodin pariin VS Codessa ja muutetaan funktio toimimaan nimensä mukaisesti eli palauttamaan kaikki maat tietokannasta. Määritellään SQL-tietokantaa varten muutokset function.json-tiedostoon. Lisätään ”bindings”-taulukon loppuun objekti, jossa on seuraavat attribuutit: ”name”, ”type”, ”direction”, ”commandText”, ”commandType” ja ”connectionStringSetting”. Name-attribuutin arvo on nimi, jota käytetään parametrina funktiossa. Tyypin täytyy olla SQL, koska SQL-tietokanta yhdistetään funktioon. ”Direction” eli suunta on tässä tapauksessa IN eli input, jotta saadaan tietokannan data funktioon parametrina eli sisään funktioon. ”CommandText” on T-SQL-komento ja ”CommandType” on tyyppiä ”text”, kun tehdään SQL-kysely (Functions documentation. 2023). ”SqlConnectionString” on ympäristömuuttuja, joka sisältää merkkijonon, jolla yhdistäminen tietokantaan onnistuu. Kuva 18 näyttää asetukset, joiden avulla voidaan olla yhteys tietokantaan.

```
{
  "bindings": [
    {
      "authLevel": "function",
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": [
        "get"
      ]
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "name": "countries",
      "type": "sql",
      "direction": "in",
      "commandText": "SELECT [CountryId], [CountryName] FROM dbo.Country",
      "commandType": "Text",
      "connectionStringSetting": "SqlConnectionString"
    }
  ],
  "scriptFile": "../dist/HttpGetAllCountries/index.js"
}
```

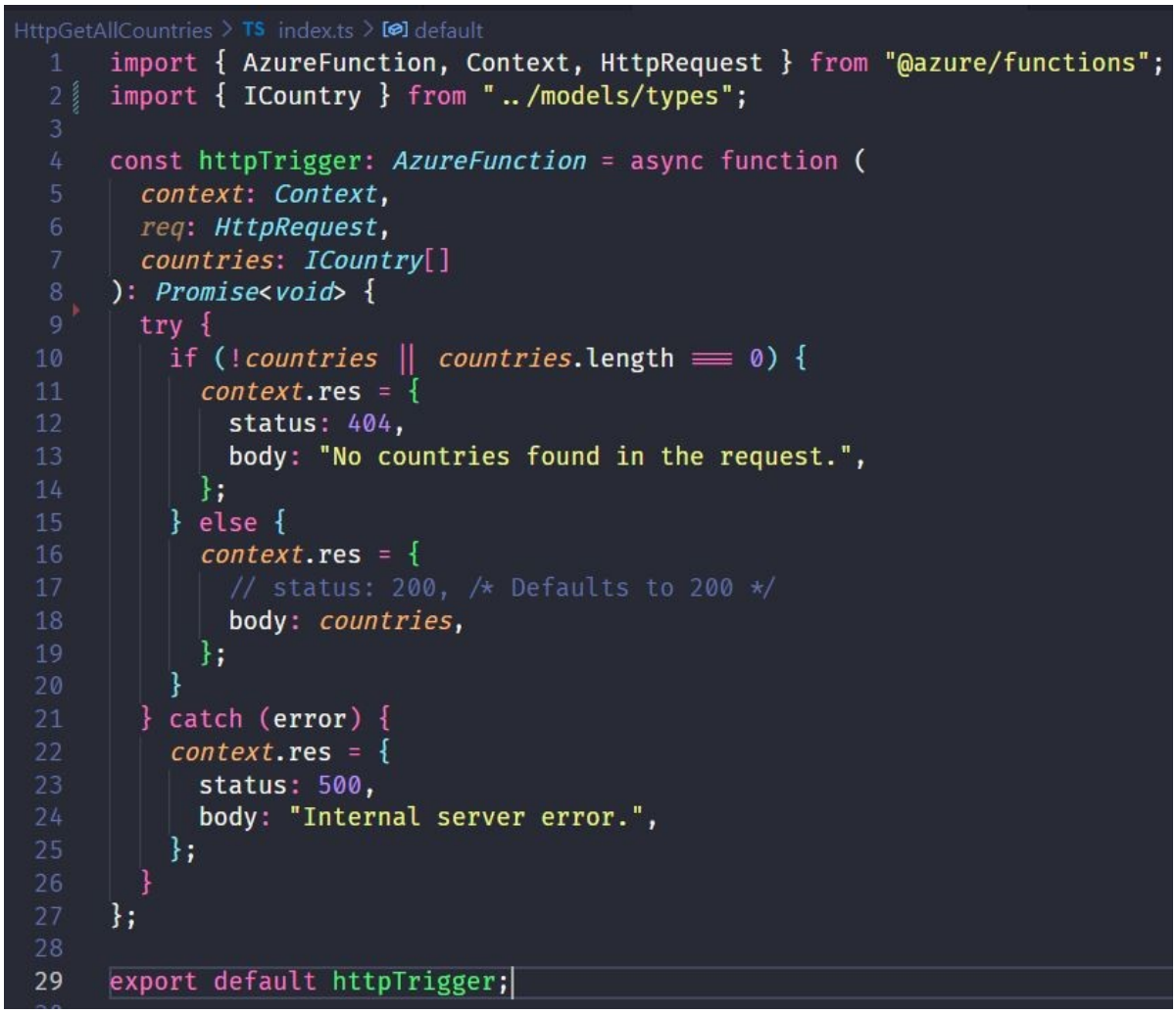
KUVA 18. Function.json-tiedoston valmis konfiguraatio

Valmiina olevasta httpTrigger-bindingista voidaan ottaa metodeista pois POST-metodi sillä funktio tulee käsittelemään vain GET-pyyntöjä. Muut asetukset voidaan pitää ennallaan. Muokataan seuraavaksi host.json-tiedostoa siten, että asetetaan ”extensionBundle”-objektin versiokentän arvoksi ”[4.*, 5.0.0)”. Muutoksen jälkeen SQL-tyyppinen bindingin pitäisi toimia ilman ongelmia. Jos versiota ei muuteta, on funktioiden ajamisen jälkeen seuraava virhe todennäköinen: ” The binding type(s) 'sql' were not found in the configured extension bundle”.

Jotta voidaan testata funktiota paikallisesti, on määritettävä ”SqlConnectionString”-ympäristömuuttuja local.settings.json-tiedostoon. Kyseinen tiedosto sisältää nimensä mukaisesti asetukset, joita käytetään funktioiden suorittamiseen paikallisesti. (Develop and debug locally. 2023.)

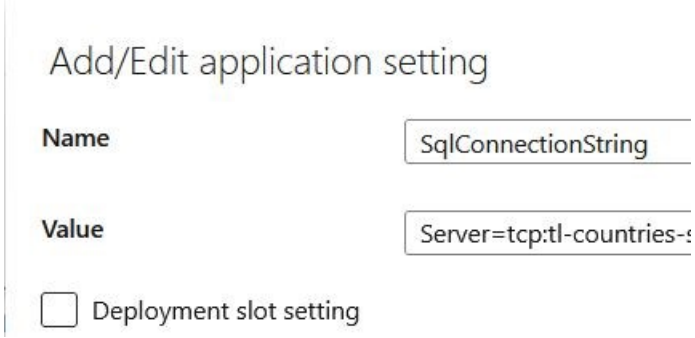
Yhdistämiseen tarvittava merkkijono löytyy tietokannan asetuksista Azure-portaalissa. Kopioidaan sivulla näkyvä SQL-tunnistautumiseen tarkoitettu merkkijono. Merkkijonossa on parametri nimeltä ”password” jonka perään lisätään SQL-salasana. Liitetään merkkijono local.settings.json-tiedoston ”SqlConnectionString”-kentän arvoksi.

Asetukset ovat nyt kunnossa ja funktion koodin voidaan viimeistellä. Lisätään funktion parametriksi function.json-tiedoston mukaisesti määritelty SQL-bindingi, jonka nimi-kentälle annoimme arvon ”countries”. ”Countries”-parametri sisältää kaikki maat, joten voidaan palauttaa se HTTP-vastauksena bodyssä käyttäjälle. Kuva 19 näyttää koodin kokonaisuudessaan.



KUVA 19. Funktion koodi, joka palauttaa tietokannasta kaikki maat

Testataan funktiota paikallisesti käynnistämällä debuggeri. Avattaan funktio selaimessa, jolloin kaikkien maiden tiedot pitäisi tulostua sivustolle. Seuraavaksi voidaan sijoittaa funktio Azureen. Azuressa funktio ei kuitenkaan toimi tässä vaiheessa ja palauttaa virhekoodin 500. Tämä johtuu siitä, että ”SqlConnectionString”-ympäristömuuttujaa ei ole määritelty funktioapplikaatioon Azuressa. Navigoidaan Azuressa funktioapplikaatioon ja sieltä konfiguraatioasetuksiin. Lisätään uusi applikaatioasetus (ympäristömuuttuja) ja annetaan nimeksi ”SqlConnectionString”. Arvoksi annetaan sama yhdistämismerkkijono, jota käytettiin paikallisesti. Tallennetaan asetukset ja testaan uudestaan funktion toimintaa. Nyt funktio toimii oikein ja tulostaa maadatan. (KUVA 20.)



KUVA 20. Ympäristömuuttujan lisääminen funktionapplikaatioon Azuressa

Luodaan loput funktiot samalla periaatteella, kuin ensimmäinen funktio. Tehdään tyypiltään HTTP-trigger-funktio, joka palauttaa tietyn maan id:n perusteella. Annetaan sille nimeksi ”HttpGetCountry”. Muokataan function.json-tiedostossa asetuksia siten, että metodina on pelkkä GET. Lisätään uusi sql-bindingi. Edelliseen funktioon, joka palautti kaikki maat, erona on ”parameter”-kenttä, jolle annetaan arvoksi ”@Id = {CountryId}”. @Id on paikkamerkkinä (placeholder) komentotekstissä ja se korvataan URL:in ”CountryId”-parametrin arvolla. Esimerkiksi URL:issa http://localhost:7071/api/Http-GetCountry?CountryId=2 ”CountryId”-parametrin arvo on 2 ja se korvaa seuraavassa SQL-komennossa @Id-paikkamerkin: "SELECT * FROM dbo.Country WHERE CountryId = @Id". Komento palauttaa maan, jonka ”CountryId” on 2. (KUVA 21.) Muokataan vielä funktion index.ts-tiedostoa lisäämällä siihen ”country”-parametri ja palautetaan se HTTP-vastauksen mukana. (Azure SQL input. 2023.)

KUVA 21. ”HttpGetCountry”-funktion lopulliset asetukset

Maatietojen hakemisen lisäksi tarvitaan funktiot, jotka muokkaavat tietokannassa olevia tietoja. Toteutetaan mahdollisuus lisätä uusia maita tietokantaan. Lisääminen tehdään perinteisesti POST-metodilla. Bindingsien suunnat voidaan päätellä maalaisjärjellä: Triggeri on tässäkin tapauksessa HTTP-pyyntö eli suuntana on ”in”. Funktio palauttaa HTTP-vastauksen, kun koodi on suoritettu eli se on ”out”. Tarvitaan taas sql-tyyppinen binding ja sen suunnaksi tulee aiemmista funktioista poiketen ”out”, koska funktion HTTP-vastauksen tulee sisältää tarvittavat tiedot maan luomiseen. Nämä tiedot lähetetään vastauksen mukana pyynnön tekijälle (käyttäjälle) ja tietokantaan.

Poistaminen on myös suoraviivaista, mutta tässä kohtaa on hyvä paikka hyödyntää Azure SQL-serverin tarjoamaa ”stored procedure” eli tallennettuja menettelytapojen toimintoa. Se on tietokantaan tallennettu kokoelma SQL-lauseita, jonka avulla voidaan suorittaa useita SQL-käskyjä yhdellä tietokantakutsulla. (Ravikiran. 2022.) Tietokannassa olevilla tauluilla on suhde, maa- ja pääkaupunkitaulu on yhdistetty ”CountryId”-attribuutin arvolla. Jos ensin yritetään poistaa maa, saadaan mahdollisesti virhe, sillä maataulun ”CountryId” on pääkaupunkitaulun vierasavain. Kaupunki on poistettava ennen kuin

pystytään poistamaan haluttu maa. Voi olla myös tilanne, jossa on maa muttei pääkaupunkia yhdistettynä siihen. Tallennetun menettelymallin avulla voidaan tarkistaa löytyykö hakutermillä pääkaupunkia ja jos löytyy, suoritetaan poisto (KUVA 22).

```
CREATE PROCEDURE dbo.deleteCountryAndCapital
    @CountryId VARCHAR(255)
AS
BEGIN
    -- Tarkistetaan löytyykö pääkaupunki
    IF EXISTS (SELECT 1 FROM dbo.Capital WHERE CountryId = @CountryId)
    BEGIN
        -- Poistetaan pääkaupunki
        DELETE FROM dbo.Capital WHERE CountryId = @CountryId;
    END
    -- Tarkistetaan löytyykö maa
    IF EXISTS (SELECT 1 FROM dbo.Country WHERE CountryId = @CountryId)
    BEGIN
        -- Poistetaan maa
        DELETE FROM dbo.Country WHERE CountryId = @CountryId;
    END
END;
```

KUVA 22. Tallennetun menettelymallin sisältö

Kun menettelymalli on tallennettuna, sen kutsuminen onnistuu helposti funktion kautta: ”commandText” saa arvon ”procedure”, ”parameters”-attribuutti saa arvon ”CountryId” ja ”commandText” saa arvokseen tietokantaan tallennetun menettelymallin nimen (KUVA 23),

KUVA 23. Konfiguraatio tallennetun menettelytavan kutsumiseen

Blob storagen yhdistämisen voi hoitaa esimerkiksi luomalla yksinkertaisen ”httpTrigger”-tyyppisen funktion. Function.json-tiedoston sisältöä ei tarvitse paljoa muokata sillä riittää, että triggerinä on http, metodina ”GET” ja funktion palauttaa HTTP-tyyppisen vastauksen.

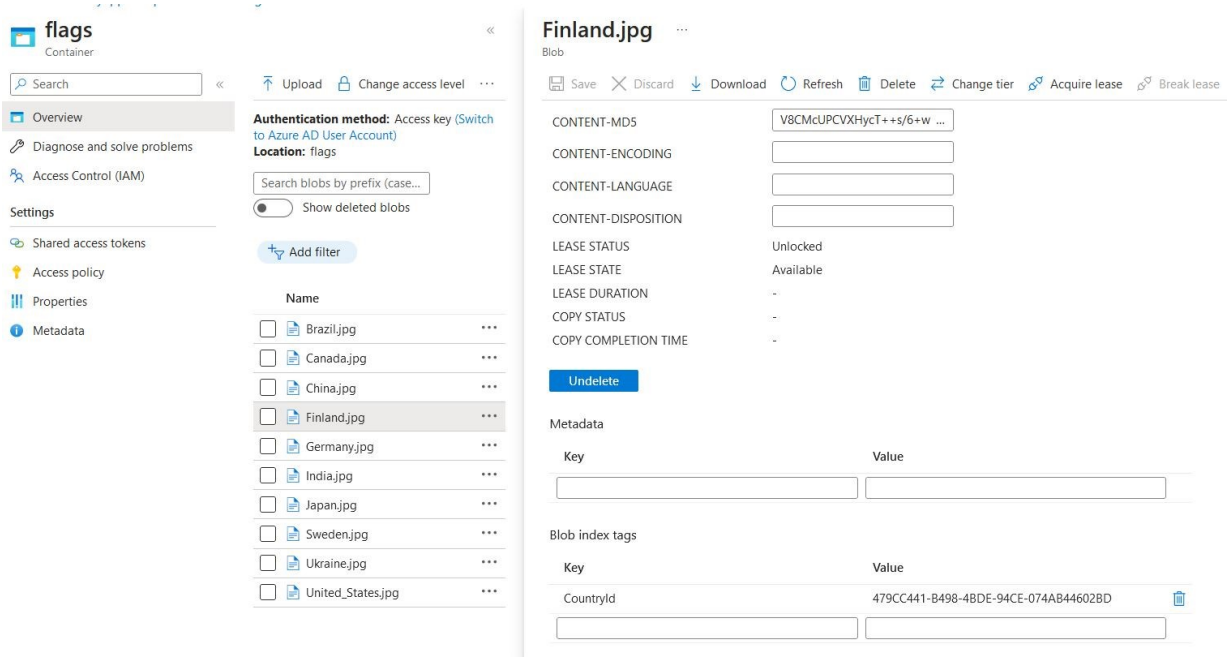
Funktion yhdistämisen Blob storageen voi tehdä SAS-URI:n avulla. Sen voi generoida halutun Blob storagen kontin asetuksesta nimeltä ”Shared access toksens”. Tietoturvallisesti paras ratkaisu on luoda SAS, joka on voimassa mahdollisimman vähän aikaa ja mahdollisimman pienillä käyttöoikeuksilla. Valitaan kehitysajaksi luku-, poisto- ja luontioikeudet sekä määritetään erääntymispäivä kuukauden päähän. Protokolaksi kannattaa valita pelkkä HTTPS, sillä lähetämme SAS-avaimen HTTP-pyyntöjen mukana. Lopulta generoitu SAS-URI laitetaan Azure Functions -applikaation ympäristömuuttujaksi, jotta siihen pääsee kiinni funktiossa.

Funktioapplikaatioprojektissa asennettaan ”Azure Blob storage”-kirjasto käskyllä ”npm install @azure/storage-blob”. Nyt on mahdollista luoda uusi ”BlobServiceClient”-luokka ja antaa sille parametrina SAS-URI ympäristömuuttujan avulla. Kontin saa haettua metodilla ”getCointainerClient”, kun annetaan sille parametrina kontin nimi. (KUVA 24.)

```
lib > TS getContainerClient.ts > ...
1  import { BlobServiceClient } from "@azure/storage-blob";
2
3  const SAS_URL = process.env["AzureWebJobsSecretStorageSas"];
4
5  const getContainerClient = () ⇒
6    new BlobServiceClient(SAS_URL).getContainerClient("flags");
7
8  export default getContainerClient;
```

KUVA 24. Konttiin yhdistäminen SAS:in avulla

Lipun etsiminen voidaan toteuttaa käyttämällä tageja hyödyksi. Jokaiselle blobille voi antaa tageja avain-arvo-pareina. Voidaan esimerkiksi antaa lipulle tagiksi sama ”CountyId” arvo, joka on annettu tietokannassa vastaavalle maalle. Tagin voi määritellä Azuresta, kun avaa kontissa olevan Blobin tiedot. (KUVA 25.)



KUVA 25. Tagi, jolla on avaimena CountryId ja arvona tietokanassa oleva id

Tagia vastaavan maa voidaan hakea ”containerClientin” tarjoamalla metodilla ”findBlobsByTags”. Kyseiselle metodille annetaan parametri muodossa ”avain=arvo”. Metodi palauttaa kaikki hakua vastaavat blobit ja ne voidaan iteroida läpi ”next”-metodilla. (Use blob index tags to manage and find data with TypeScript. 2023.) Tässä tapauksessa attribuutin ”CountryId” arvo pitäisi olla uniikki, jolloin yksi Blob pitäisi löytyä. Blobin tietojen löydyttyä voidaan käyttää sen nimeä Blobin URL:in hakemiseen. Se onnistuu ”getBlobClient”-metodilla kuvan 26 mukaisesti. (Get URL for container or blob with TypeScript. 2023.)

HttpGetFlag > TS index.ts > ...

```typescript
import { AzureFunction, Context, HttpRequest } from "@azure/functions";
import getContainerClient from "../lib/getContainerClient";

const getFlagUrl = async (countryId: string) => {
  if (!countryId.length) return "";
  const containerClient = getContainerClient();
  const iter = containerClient.findBlobsByTags(`CountryId='${countryId}'`);

  let blobItem = await iter.next();
  blobItem.value;
  while (!blobItem.done) {
    const blobURL = containerClient.getBlobClient(blobItem.value.name).url;
    if (blobURL) {
      return blobURL;
    }
    blobItem = await iter.next();
  }
  return "";
};

const httpTrigger: AzureFunction = async function (
  context: Context,
  req: HttpRequest
): Promise<void> {
  try {
    const flagUrl = await getFlagUrl(req.query.CountryId);
    if (!flagUrl.length) {
      context.res = {
        status: 404,
        body: "No flag found by id",
      };
    } else
      context.res = {
        body: flagUrl,
      };
  } catch (error) {
    context.res = {
      status: 500,
      body: "Internal server error.",
    };
  }
};

export default httpTrigger;

```

KUVA 26. Funktio, joka hakee tagin avulla kuvan URL:in ja palauttaa sen HTTP-vastauksena

Ennen kuin siirrytään frontendin pariin, lisätään tiedostoon ”Host”-attribuutti, jolle voidaan määrittää CORS. Sallitaan kaikista osoitteista tulevat pyynnöt, jotta ei törmätä ongelmiin, kun suoritetaan sekä funktio- että frontend-applikaatiota lokaalisti. Saman alkuperän käytäntöön liittyviä ongelmia esiintyy mitä luultavammin, koska applikaatiolla on eri portit. (Code and test Azure Functions locally. 2023.)

4.5 Frontendin luonti

Nyt kasassa on kaikki tarvittava käyttöliittymän rakentamiseen. Suurin osa ohjelman konfiguraatiosta sijaitsee Azure Functions -applikaatiossa, joten frontendin ohjelmoinnin pitäisi olla suoraviivaista. Azuren ansiosta voidaan hyödyntää serverless-arkkitehtuurin etuja, kuten automaattista skaalausta ja resurssien tehokasta käyttöä, jonka pitäisi tehdä sovelluksesta tehokkaan, joustavan ja helposti kehitettävän.

4.5.1 Projektin alustus ja sijoitus Azureen

Aloitetaan luomalla React-projekti paikallisesti halutulla koontityökalulla. Esimerkiksi ”Vite”-niminen koontityökalu on hyvä valinta serverless-applikaatiota varten. Projektin alustus käy helposti ja nopeasti NPM-paketinhallintatyökalun avulla. (Next Generation Frontend Tooling.)

Syötetään komentoriville komento ”npm create vite@latest countries-app” ja valitaan viitekehykseksi React ja variantiksi Typescript. Tässä vaiheessa kannattaa ottaa Git-versionhallinta mukaan projektiin ”git init”-komennolla sillä Vite ei erikseen alusta gittiä projektiin.

Katsotaan seuraavaksi, kuinka voidaan hyödyntää Azuren funktioita projektissa. Asennetaan muutama NPM-paketti helpottamaan applikaation tilan hallintaa ja kyselyjen tekemistä. Asennetaan Redux, jolla hoidetaan tilanhallinta ja Axios HTTP-kyselyitä varten. Projektin struktuuriksi valitaan seuraava lähestymistapa: ”services”-niminen kansio, jonne sijoitetaan kaikki HTTP-kyselyt. Kansioon ”features” sijoitetaan Redux ”slicet” eli tilanhallintaan liittyvä logiikka. ”Models”-kansioon laitetaan TypeScripin tyyppitiedostot, joita tarvitaan useassa paikassa applikaatiota. Oleellisimmat ovat country.ts- ja capital.ts-tiedostot, joissa sijaitsee tietokannassa olevia tauluja vastaavat tyypit. Poikkeuksellisesti aloitan

TypeScriptin tarjoaman ”interface”-attribuuttien nimet isolla alkukirjaimella, jos ne liittyvät tietokannasta saataviin tietoihin, koska aiemmin käytettiin Azure funktioissa ja tietokannassa isoja-alkukirjaimia (KUVA 27).

KUVA 27. Maa-interface, joka vastaa tietokanassa olevaa maataulua

Vastaavanlainen interface tehdään myös pääkaupunkia varten. Nyt HTTP-kyselyjen luonti on helppoa. Jatketaan maan käyttämistä esimerkkinä ja katsotaan tarkemmin, kuinka tehdä GET- ja POST-tyyppiset HTTP-pyynnöt. Alla oleva kuva näyttää kokonaisuudessaan koodin, jolla pyyntöjen tekeminen onnistuu. Ensimmäinen funktio palauttaa kaikki maat, toinen funktio on yhden maan tietojen pyytämistä varten ja ”createCountry”-funktio huolehtii uuden maan luomisesta. (KUVA 28.)

KUVA 28. Axios-kirjastolla toteutetut HTTP-pyynnöt

URL on funktion tarkka osoite, jonka loppuosa muodostuu seuraavasti: api/funktion nimi + mahdolliset parametrit + funktion avaimen merkkijono. Yksi tapa määrittää URL on käyttää hyväksi tietoa, että suoritetaanko applikaatiota kehitys- vai tuotantoympäristössä. Esimerkiksi kehitysympäristössä käytetään funktioiden localhost-osoitetta ja tuotannossa Azuressa olevan funktioapplikaation osoitetta. Käytetään koodissa itse määriteltyä ”getUrl”-nimistä apufunktiota, joka muodostaa URL:in parametreineen. (Env Variables and Modes.)

Seuraavaksi toteutetaan Reduxilla tapa kutsua näitä funktioita. Luodaan ”features”-kansioon tiedosto nimeltä ”countrySlice.ts”. ”Slice” on mahdollista luoda ”createSlice” funktiolla, joka saa parametreiksi nimen, alustavat arvot ja reducer-funktion, jonka avulla tilaa päivitetään.
Redux toolkit tarjoaa kuitenkin promisejen hallintaan oman funktion nimeltään ”createAsyncThunk”. Se kykenee muodostamaan toimintoja eli actioneja, promisien palauttamien tilojen pohjalta. Promiset pystyvät palauttamaan kolme eri tilaa: hylätty (rejected), täytetty (fullfilled) ja odottava (pending). ”CreateAsyncThunkille” annetaan parametriksi nimi ja vakiintunut tapa on antaa nimeksi ”slicen nimi/actionin nimi” eli esimerkiksi ”country/fetch”, jossa country on ”slicen” nimi ja ”fetch” on action. Toiseksi parametriksi annetaan callback-funktio, joka tekee API-kutsun ja palauttaa promisen. (KUVA 29.)

```
export const fetchContent = createAsyncThunk(
  "country/fetchContent",
  async () => {
    const res = await getAllCountries();
    return res;
  }
);
export const fetchCountryById = createAsyncThunk(
  "country/fetchCountryById",
  async (countryId: string) => {
    const res = await getCountry(countryId);
    return res;
  }
);
```

KUVA 29. Esimerkki ”createAsyncThunkin” käytöstä

Yllä kuvatut funktiot eivät vielä määrittele sitä, kuinka promisejen tiloihin tulisi vastata vaan on vielä lisättävä ”createSlicen” parametreihin uusi attribuutti nimeltä ”extraReducer”, jonka arvona on call-back-funktio. Callbackista saadaan ”builder”-olio, jolla on ”addCase”-niminen metodi, jolle annetaan aiemmin ”createAsyncThunkilla” tehdyt funktiot. Jos tahdomme huolehtia kaikista promisen tiloista, on meidän määritettävä kolme eri ”addCasea”. Jokaisen ”addCasen” kautta pääsee käsiksi tilaan ja actioniin, joiden toiminta on sama kuin normaalien reducerien yhteydessä. Alla oleva kuva näyttää oikean tavan. (KUVA 30.)

```
export const countrySlice = createSlice({
  name: "country",
  initialState,
  reducers: {
    selectCountryId: (state, action) => {
      state.selectedCountry.id = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder.addCase(fetchContent.pending, (state) => {
      state.loading = true;
    });
    builder.addCase(fetchContent.fulfilled, (state, action) => {
      state.loading = false;
      state.countries = action.payload;
    });
    builder.addCase(fetchContent.rejected, (state, action) => {
      state.loading = false;
      state.error = action.error.message ?? "Error";
    });
```

KUVA 30. extraReducersien määrittäminen

Valmiit ”slicen” avulla luodut reducerit on vielä liitettävä ”redux storeen”. Se onnistuu ”redux/toolkit”-kirjaston ”configureStore”-funktiolla. Yksinkertaisimmillaan se tehdään määrittämällä ”reducer”-attribuutti, johon liitetään ”slicen” reducerit (KUVA 31). Lopuksi ”redux/toolkit”-kirjaston ”provider” niminen komponentti asetetaan applikaation komponenttipuun ympärille ja annetaan sille määritetty ”store” parametrina. (Chima. 2022.)

KUVA 31. Redux-store, johon reducerit liitettynä

Redux-storen ja slicen konfiguraation jälkeen tilan hallinta on todella suoraviivaista. Demonstroidaan tätä näyttämällä applikaation aloituspisteen koodi. ”LoadingCountryData”-nimisellä Redux-”selectorilla” saadaan tietoon, jos promisen on vielä ”pending”-tilassa. Käytetään tätä tietoa hyväksi ja näytetään teksti ”Loading…” kunnes tila vaihtuu. ”Dispatch”-metodilla saadaan lähetettyä tieto tilan muutoksesta Redux-varastolle, jolloin reducer-funktio suorittaa tilan päivityksen. Annetaan sille parametriksi aiemmin tehty ”fetchContent”-funktio. Suoritetaan ”dispatch” Reactin tarjoaman ”useEffect-hookin” yhteydessä. (KUVA 32.)

```
const App = () => {
  const loadingCountryData = useAppSelector((v) => v.country.loading);
  const dispatch = useAppDispatch();

  useEffect(() => {
    dispatch(fetchContent());
  }, []);

  return (
    <>
      {loadingCountryData ? (
        <Box>Loading ... </Box>
      ) : (
        <Box>
          <CountryList />
          <Button onClick={() => dispatch(toggleCreateCountry())}>
            Create Country
          </Button>
        </Box>
      )}
      <ViewDetails />
      <EditCountry />
      <CreateCountry />
      <CreateCapital />
    </>
  );
};

export default App;
```

KUVA 32. App.tsx tiedoston sisältämä koodi

Yllä olevassa koodissa näkyy, kuinka ”toggleCreateCountry”-funktion kutsu tapahtuu klikkaustapahtuman yhteydessä. Tämän jälkeen ”CreateCountry”-niminen komponentti saa ”selectorin” avulla päivitetyn Redux-varaston tilan itselleen. Muutoksen avulla on mahdollista hallita, että näytetäänkö vai piilotetaanko komponentti käyttäjältä. Samalla varaston tilan muokkaamisperiaatteella voidaan toteuttaa esimerkiksi uuden maan luominen. Hienoin ominaisuus applikaatiossa on se, että maiden lisäksi pääkaupunkien ja lippujen luonti onnistuu täysin samalla logiikalla. Näin applikaation jatkokehitys on sulavaa.

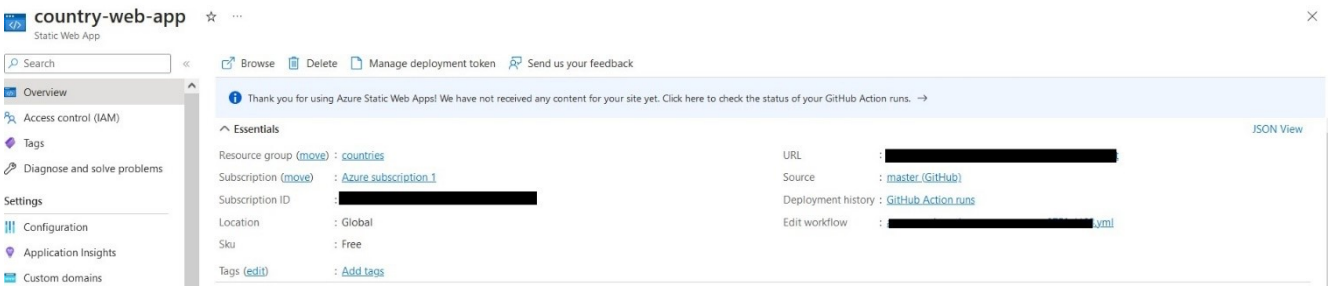
4.5.2 Static web app

Nyt viimeistään on aika siirtää koodi GitHubiin, kun applikaatio kommunikoi onnistuneesti Azuren funktioiden kanssa lokaalisti. Tässä kohtaa oletetaan, että käyttäjällä on perustieto GitHubin käytöstä

ja kuinka luodaan uusi tietovarasto (repository), joten käydään prosessin läpi vain pääpiirteittäin. Luodaan yksityinen tietovaraston nimellä ”country-app” ja lisätään ”origini” eli alkuperä GitHubin antamien ohjeiden mukaisesti. Nyt ensimmäinen versio on valmiina ja on aika siirtää applikaatio kaikkien saataville ”Azure static web app”-resurssin avulla.

Luodaan Azure-portaalista ”Static web app”-niminen resurssi valitsemalla sama tilaus ja resurssiryhmä, kuin muille luoduille resursseille. Resurssin luomiseen liittyvistä asetuksista valitaan GitHub lähteeksi ja kirjaudutaan sisään GitHub-tunnuksilla. Tämän jälkeen pystytään hakemaan aiemmin luotu tietovarasto ja siihen liittyvät tiedot. Valitaan vielä käännösasetuksista esiasetukseksi ”React”. Muut asetukset voidaan jättää oletustilaan, sillä niitä on mahdollista muokata jälkikäteen. Siirrytään esikatselunäkymään ja tarkistuksen jälkeen luodaan resurssi loppuun.

Applikaation automaattinen sijoitus Azureen mahdollisesti epäonnistuu tässä kohtaan, jos etukäteen ei olla määritelty kaikkia tarvittavia asetuksia oikeiksi. Tämä voidaan tarkistaa menemällä vasta luodun ”Static web”-applikaation yleisnäkymään ja sieltä painamalla ”Browse”-nappia. Napin painalus avaa uuden välilehden, jossa applikaation pitäisi sijaita. Sijoituksen epäonnistuessa sivustolla näkyy kuitenkin vain teksti, joka kertoo applikaation olevan käytössä, mutta odottaa sisältöä. (KUVA 33.)

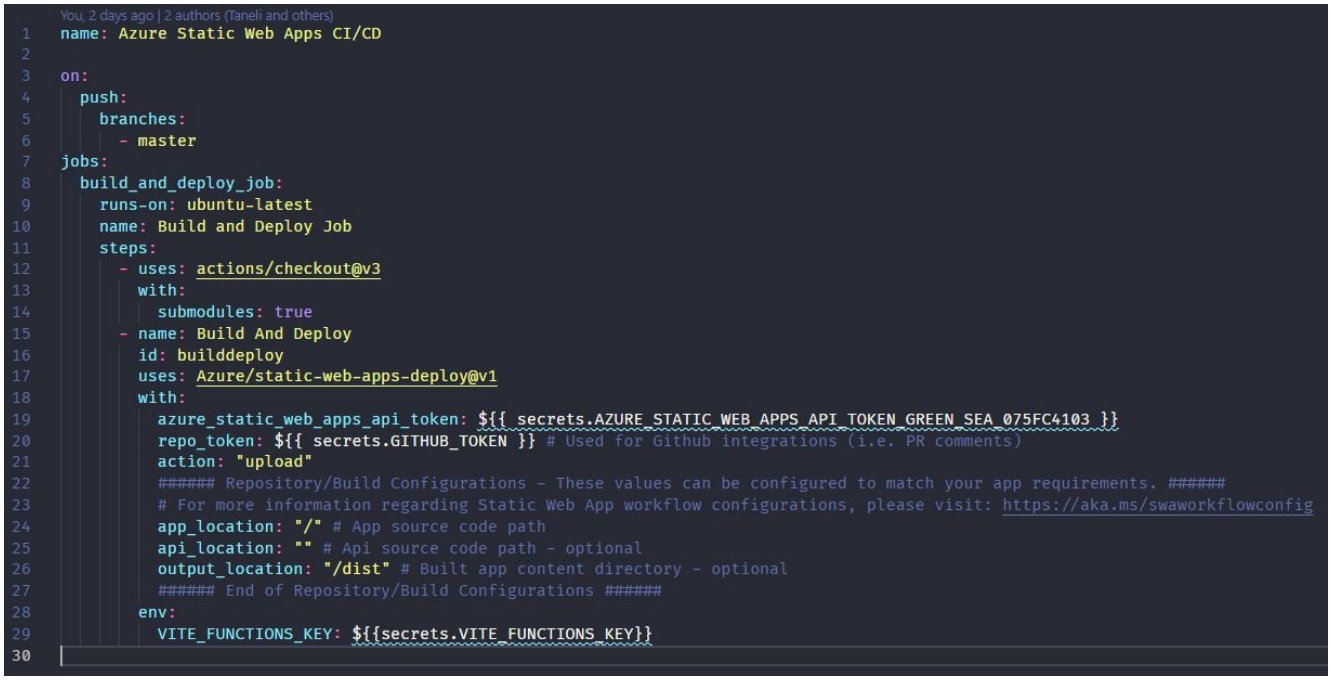


KUVA 33. ”Static web”-applikaation yleisnäkymä

Käydään katsomassa mitä muutoksia ”static web”-applikaation luonti teki tietovarastoon GitHubissa. Tietovarastoa tarkastellessa pitäisi sieltä löytyä yksi uusi. yml-tyyppinen tiedosto kansiosta ”.github/workflows/”. Kyseinen tiedosto määrittelee pitkälti sen mitä tapahtuu, kun pusketaan uudet koodit tietovarastoon, joten tutkitaan sitä hieman tarkemmin.

”Workflow”-tiedosto on jo valmiiksi konfiguroitu lähes oikein ja siitä voidaan poistaa ylimääräisiä osia. Tehdään skriptistä hieman yksinkertaisempi siten, että ”workflow” suoritetaan aina, kun muutos

pusketaan haaraan nimeltä ”master”. ”Pull requestiin” liittyviä asetuksia ei tarvita, joten ne voi halutessaan poistaa. Töiden nimiä on mahdollista muuttaa. Edellä mainittujen vapaaehtoisten toimien lisäksi on tehtävä tärkein muutos tiedostoon eli ”build_and_debloy” työssä olevalle ”output_location”-attribuutille on annettava polku, missä sijaitsee applikaation tuotantoon optimoidut tiedostot. Vitessä tiedostojen oletussijainti on kansiossa ”/dist”. Lopuksi lisätään vielä tarvittavat ympäristömuuttujat ”build_and_debloy” työn loppuun, esimerkiksi Azure funktioiden avaimen voi asettaa ympäristömuuttujaksi. (KUVA 34; Build configuration for Azure Static Web Apps. 2023.)



```
You, 2 days ago | 2 authors (Taneli and others)
name: Azure Static Web Apps CI/CD

on:
  push:
    branches:
      - master
jobs:
  build_and_deploy_job:
    runs-on: ubuntu-latest
    name: Build and Deploy Job
    steps:
      - uses: actions/checkout@v3
        with:
          submodules: true
      - name: Build And Deploy
        id: builddeploy
        uses: Azure/static-web-apps-deploy@v1
        with:
          azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN_GREEN_SEA_075FC4103 }}
          repo_token: ${{ secrets.GITHUB_TOKEN }} # Used for Github integrations (i.e. PR comments)
          action: "upload"
          ###### Repository/Build Configurations - These values can be configured to match your app requirements. ######
          # For more information regarding Static Web App workflow configurations, please visit: https://aka.ms/swaworkflowconfig
          app_location: "/" # App source code path
          api_location: "" # Api source code path - optional
          output_location: "/dist" # Built app content directory - optional
          ###### End of Repository/Build Configurations ######
        env:
          VITE_FUNCTIONS_KEY: ${{secrets.VITE_FUNCTIONS_KEY}}
```

KUVA 34. Valmis ”workflow”-tiedosto

Nyt sivuston julkaisun pitäisi onnistua, kun pusketaan muutokset ”master”-haaraan GitHubissa. Navigoidaan nettisivulle sen jälkeen, kun Actionit on suoritettu GitHubissa. Tässä kohtaa todennäköisesti saadaan virhe, joka kertoo jälleen SOP-virheestä. Korjataan tämä ”Azure functions”-resurssin asetuksista avaamalla CORS:ia koskevat asetukset. Lisätään sallituksi alkuperäksi nettisivun osoite ja tallennetaan muutokset, jonka jälkeen sivun pitäisi toimia normaalisti. Jos jotain ongelmia esiintyy, kannattaa katsoa funktioapplikaation asetukset Azuresta, että siellä on samat ympäristömuuttujat asetettuna, kuin local.settings.json-tiedostossa oli funktioita suorittaessa lokaalisti onnistuneesti.
Serverless-palvelu toimii nyt sulavasti ja kehitystyötä voi tästä pisteestä jatkaa vaivattomasti ilman, että tarvitsee huolehtia jatkuvasti asetusten ja konfiguraatioiden lisäämisistä.

5 JOHTOPÄÄTÖKSET

Opinnäytetyöni tavoitteet olivat kehittää taitojani yleisesti serverless-arkkitehtuurin ja Azuren parissa. Tärkeässä osassa oli myös laatia serverlessiin liittyvä kattava suomenkielinen ohjeistus, jota lukija pystyy seuraamaan ilman kustannuksia.

Mielestäni aiheeni oli kohtuullisen haastava, sillä aikaisempi kokemukseni pilvipalveluihin liittyen oli vähäistä ja valitsin välillä työssäni vaikeampia toteutustapoja, kuin olisi ollut tarve maksimoidakseni oppimiseni. Haasteen toi myös se, että joistakin asioista dokumentaation määrä oli hyvin vähäistä edes englanniksi, saati sitten suomeksi. Aiheen laajuus ja täten oleellisimpiin asioihin keskittyminen toi myös omat haasteensa. Kaikilla kolmella pilvipalveluntarjoajalla on satoja palveluita tarjolla, joten oikeiden tuotteiden valinta projektiin vaatii paneutumista eri vaihtoehtoihin. Uusia tuotteita ja uusia versioita olemassa olevista tuotteista julkaistaan nopeaan tahtiin, joten pilvipalveluihin liittyvää osaamista tulee päivittää jatkuvasti.

Parhaiten työssäni onnistuin oman tietotaitoni kehittämisessä, joka oli myös ykkösprioriteettini. Vaikka pääpaino oli Azuressa, sain hyvän käsityksen myös Google Cloudin ja Amazon AWS:n palvelutuotteista. Uskon saaneeni erittäin hyvät pohjatiedot, jotta kykenisin luomaan kyseisiin palveluihin serverless-applikaation suhteellisen vaivattomasti. Uskon myös, että työni ohjaa lukijan oikealle polulle, jos hän on enemmän kiinnostunut Googlen tai Amazonin pilvipalvelusta.

Oli myös hienoa nähdä käytännön työni valmiina, sillä tietyn pisteen ylittyessä ohjelmointi muuttui vaivattomaksi ja nopeaksi. Ohjelman jatkokehitys tulisi olemaan mielekästä. Vauhtiin päästessä projektia voisi laajentaa lähes loputtomiin. Seuraavaksi applikaation kehityksessä kehittäisin tietoturvaan liittyviä asioita, sillä työtäni tehdessä näin järkeväksi valita kehitysprosessiin riittävän tietoturvan, mutta ohjelman julkaisu vaatisi parannuksia tietoturvaan. Ohjelmaan monitorointia tulisi parantaa ottamalla käyttöön eri tuotteita, jotka analysoivat lokeja ja järjestelmän toimivuutta. Oma verkkotunnus myös toisi ennalta annetun verkkotunnuksen sijaan ammattimaisemman kuvan sivustosta.

LÄHTEET

About triggers and bindings. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings. Viitattu 24.7.2023.

Accessing user information in Azure Static Web Apps. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/static-web-apps/user-information. Viitattu 12.10.2023.

Amazon Aurora. 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/rds/aurora/. Viitattu 25.10.2023.

Amazon AWS. 2023. What is cloud computing? Saatavissa: https://aws.amazon.com/what-is-cloud-computing/. Viitattu 1.4.2023.

AWS Cloud Products. 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/products/. Viitattu 25.10.2023.

AWS Lambda. 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/lambda/. Viitattu 11.7.2023.

Azure Functions HTTP trigger. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger. Viitattu 19.10.2023.

Azure Functions Node.js developer guide. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference-node. Viitattu 19.10.2023.

Azure Functions scenarios. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios. Viitattu 19.10.2023.

Azure SQL input. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-azure-sql-input. Viitattu 11.8.2023.

Azure Static Web Apps hosting plans. 2023. Microsoft Azure. https://learn.microsoft.com/en-us/azure/static-web-apps/plans. Viitattu 12.10.2023.

Buchanan, I. What is containers as a service? Saatavissa: https://www.atlassian.com/microservices/cloud-computing/containers-as-a-service. Viitattu: 3.4.2023.

Build configuration for Azure Static Web Apps. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/static-web-apps/build-configuration. Viitattu 24.10.2023.

Build in the cloud with an Azure free account. 2023. Microsoft Azure. Saatavissa: https://azure.microsoft.com/en-us/free/. Viitattu 12.7.2023.

Chai, W., Brush, K. & Bigelow, S. What is PaaS? Platform as a service definition and guide. Saatavissa: https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS. Viitattu 25.10.2023.

Chima, I. 2022. What is createAsyncThunk in redux? Saatavissa: https://dev.to/ifeanyichima/what-is-createasyncthunk-in-redux--mhe. Viitattu 24.10.2023.

Cleo. On Premise vs. Cloud: Key Differences, Benefits and Risks. Saatavissa: https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud. Viitattu 13.4.2023

Cloudfare. 2023. What is serverless computing? Saatavissa: https://www.cloudflare.com/learning/serverless/what-is-serverless/. Viitattu 25.10.2023.

Cloud Functions. Google. Saatavissa: https://cloud.google.com/functions. Viitattu: 11.7.2023.

Cloud Storage. Google. Saatavissa: https://cloud.google.com/storage. Viitattu 11.7.2023.

Code and test Azure Functions locally. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-local. Viitattu 3.10.2023

Configuration overview. 2022. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/static-web-apps/configuration-overview. Viitattu 12.10.2023.

Create free services. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/cost-management-billing/manage/create-free-services. Viitattu 24.7.2023.

Custom domains with Azure Static Web Apps. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/static-web-apps/custom-domain. Viitattu 12.10.2023.

Database. 2023. Amazon AWS. Saatavissa: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/database.html. Viitattu 12.7.2023.

Develop and debug locally. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-local. Viitattu 5.8.2023.

Env Variables and Modes. Vite. Saatavissa: https://vitejs.dev/guide/env-and-mode.html. Viitattu 24.10.2023.

Firestore. Google. Saatavissa: https://cloud.google.com/firestore. Viitattu: 11.7.2023.

Flutter. Firebase. Saatavissa: https://docs.flutter.dev/data-and-backend/firebase. Viitattu 11.7.2023.

Functions documentation. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/. Viitattu 4.8.2023.

Get URL for container or blob with TypeScript. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-get-url-typescript. Viitattu 24.10.2023.

Google Cloud overview. Google. Saatavissa: https://cloud.google.com/docs/overview. Viitattu: 8.7.2023.

Improve the performance and reliability of Azure Functions. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/performance-reliability. Viitattu 19.10.2023.

Klint, L. 2023. What is Azure? Microsoft's cloud platform explained. Saatavissa: https://www.pluralsight.com/resources/blog/cloud/what-is-microsoft-azure. Viitattu 25.10.2023.

Lambda runtimes. 2023. Amazon AWS. Saatavissa: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html. Viitattu 11.7.2023.

Law, M. 2023. Top 10 biggest cloud providers in the world in 2023. Saatavissa: https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023. Viitattu 11.7.2023.

Microsoft Azure. What is cloud computing? Saatavissa: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing. Viitattu 1.4.2023.

Microsoft Azure portal. 2023. Microsoft Azure. Saatavissa: https://azure.microsoft.com/en-us/get-started/azure-portal. Viitattu 25.10.2023.

Mäkilä, M. & Tuomisto, J. 2022. Azure is the most used cloud service in Finland – What makes Solita the best Azure consultancy? Saatavissa: https://data.solita.fi/azure-is-the-most-used-cloud-service-in-finland-what-makes-solita-the-best-azure-consultancy/. Viitattu 26.10.2023.

Next Generation Frontend Tooling. Vite. Saatavissa: https://vitejs.dev/. Viitattu 16.8.2023.

Overview of Azure page blobs. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-pageblob-overview. Viitattu 25.9.2023.

PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different? Google. Saatavissa: https://cloud.google.com/learn/paas-vs-iaas-vs-saas . Viitattu: 3.4.2023.

Ravikiran, A. 2022. Stored Procedure in SQL: Benefits And How to Create It. Saatavissa: https://www.simplilearn.com/tutorials/sql-tutorial/stored-procedure-in-sql. Viitattu 24.10.2023.

Red Hat. 2022. What is SaaS? Saatavissa: https://www.redhat.com/en/topics/cloud-computing/what-is-saas. Viitattu: 4.4.2023.

Rosencrance, L. SaaS vs. IaaS vs. PaaS: Differences, Pros, Cons and Examples. 2021. Saatavissa: https://www.techtarget.com/whatis/SaaS-IaaS-PaaS-Comparing-Cloud-Service-Models. Viitattu 20.4.2023.

Serverless on AWS. 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/serverless/. Viitattu 24.5.2023.

Smikar software. 2023. What Are Append Blobs. Saatavissa: https://www.smikar.com/what-are-append-blobs/. Viitattu 26.10.2023.

Storage account overview. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-overview. Viitattu 20.9.2023.

Storage considerations. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-functions/storage-considerations?tabs=azure-cli. Viitattu 1.8.2023.

Try Azure SQL Database free with Azure free account. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-sql/database/free-sql-db-free-account-how-to-deploy. Viitattu 19.10.2023.

Understanding block blobs, append blobs, and page blobs. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs. Viitattu 21.9.2023.

Understanding GitHub Actions. 2023. GitHub. Saatavissa: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Viitattu 26.10.2023.

Use blob index tags to manage and find data with TypeScript. 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-tags-typescript. Viitattu 24.10.2023.

What is Amazon S3? 2023. Amazon AWS. Saatavissa: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. Viitattu 12.7.2023.

What is Azure Static Web Apps? 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/static-web-apps/overview. Viitattu 12.10.2023.

What is Cloud Run. Google. Saatavissa: https://cloud.google.com/run/docs/overview/what-is-cloud-run. Viitattu 11.7.2023.

What is FaaS? IBM. Saatavissa: https://www.ibm.com/topics/faas. Viitattu 3.4.2023.

What Is IaaS (Infrastructure as a Service)? 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/what-is/iaas/. Viitattu: 20.4.2023.

What is Platform as a Service (PaaS)? Google. Saatavissa: https://cloud.google.com/learn/what-is-paas. Viitattu 4.4.2023.

What is SaaS? 2023. Amazon AWS. Saatavissa: https://aws.amazon.com/what-is/saas/. Viitattu 4.4.2023.

What is serverless? IBM. Saatavissa: https://www.ibm.com/topics/serverless. Viitattu 24.5.2023.

What is the Azure portal? 2023. Microsoft Azure. Saatavissa: https://learn.microsoft.com/en-us/azure/azure-portal/azure-portal-overview. Viitattu 24.7.2023.

Workflow syntax for GitHub Actions. 2023. GitHub. Saatavissa: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions. Viitattu 29.9.2023.