Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Julia Virtanen

Kuormitustestaus mobiilipalvelun kustannusten optimoinnissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotuotanto

Insinöörityö

11.5.2020

Tekijä Otsikko Sivumäärä Aika	Julia Virtanen Kuormitustestaus mobiilipalvelun kustannusten optimoinnissa 37 sivua + 1 liite 11.5.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Yliopettaja Auvo Häkkinen QA Lead Tomi Juslin

Insinöörityön tavoitteena oli luoda kuormitustestit pilvipalvelinpohjaiselle mobiilisovellukselle. Kuormitustestien tuloksien perusteella pilvipalvelun laskentakapasiteetti konfiguroitaisiin uudestaan. Tavoitteena oli saada laskettua laskentakapasiteettia ja näin ollen säästää kustannuksissa.

Tuotannon data analysoitiin valitulta aikaväliltä ja analyysin perusteella valittiin testattavat toiminnot ja käytettävän kuorman määrä. Testiskriptit kirjoitettiin Neoloadia käyttäen. Kun kahdeksan eri vaihetta sisältävä testiskripti oli valmis, suoritettiin koeajo, jonka perusteella skriptiä vielä optimoitiin esimerkiksi lisäämällä aikaviiveitä ja tarkistuksia eri vaiheisiin. Näin varmistettiin skriptin toimivuus ja kuorman optimaalinen määrä. Tämän jälkeen suoritettiin referenssiajo, jonka tuloksiin varsinaisten testiajojen tuloksia verrattiin. Referenssiajon vaikutukset laskentakapasiteetin käyttöasteeseen oli tulkittavissa pilvilaskenta-alustan metriikoista. Näiden tulosten perusteella päätettiin konfiguraatioyhdistelmät, joita lähdettiin kokeilemaan. Testituloksista vertailtiin kutsujen vasteaikoja ja laskentakapasiteetin käyttöasteesta. Näin löydettiin optimaalisin konfiguraatioyhdistelmä ilman, että mobiilisovelluksen käytettävyys kärsii.

Tavoitteet saavutettiin. Testiskriptit saatiin onnistuneesti tehtyä. Referenssiajo myös osoitti, että konfiguroitavaa on ja laskentakapasiteettia on mahdollista laskea. Testattavat konfiguraatioyhdistelmät valittiin ja testattiin. Tehdyt testit ovat osoittivat, että muutoksia on mahdollista tehdä ja säästää merkittävästi pilvilaskenta-alustan tuomissa kustannuksissa.

Avainsanat	AWS, pilvilaskenta-alusta, kuormitustestaus, suorituskykytestaus

Author Title Number of Pages Date	Julia Virtanen Load Testing in Mobile Service Cost Optimization 37 pages + 1 appendice 10 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software engineering
Instructors	Auvo Häkkinen, Principal lecturer Tomi Juslin, QA Lead

The objective of the study was to create and execute load tests for a mobile app running in a cloud computing platform. The computing capacity would be configured by the results of the tests. The target was to scale the capacity down in order to optimize the expense of the platform.

The production data was analyzed from the chosen period. Based on the analysis user paths for the tests were chosen. The test scripts were written by using Neoload. When the scripts were ready, a test run was executed. By using the test results of that run, the test scripts were optimized regarding the user load. To make sure that the scripts were working as planned and the load corresponded to the production side, some delays and validations were added to the scripts. After this a reference run was executed. The results of the actual test runs will be compared to these results. The impacts that the reference run had on the utility ratio of the computing capacity could be seen from the metrics available from the platform. After analyzing these metrics, the configuration combinations used in the testing were decided on. The utility ratio of the computing capacity and response times are something to be compared between the test results. This is the way of finding out which configuration combination is the most optimized keeping the usability of the software as good as it was before the configuration.

The goals were met. The scripts were created successfully. The reference run showed that there is something to configure and it is possible to lower the computing capacity. Configuration combinations were decided and tested. The tests that were executed showed that there is a possibility to lower the computing capacity which should create significant savings to the costs created by the cloud computing platform.

Keywords	AWS, cloud computing platform, load testing, performance testing

Sisällys

Lyhenteet

AWS	Amazon Web Services on kokoelma etätietojenkäsittelyresurssien palveluja, jotka muodostavat yhdessä Amazon.comin kautta tarjottavan pilvilaskenta-alustan.
CDN	Content Delivery Network eli sisällönjakeluverkko. Koostuu eri palvelimista, jotka on hajautettu maantieteellisesti, jotta palvelut toimisivat pienemmällä viiveellä ja varmemmin.
CPU	Central Processing Unit eli suoritin tai prosessori.
ECS	Elastic Container Service, palvelimeton infrastruktuuri, joka mahdollistaa palvelun pyörittämisen.
GB	Gigabyte eli gigatavu.
GDPR	General Data Protection Regulation on vuonna 2018 voimaan tullut yleinen tietosuoja-asetus.
GiB	Gibitavu, yksi GiB on 1024^3 tavua.
I/O	Input/Output, tarkoittaa datan siirtoa joko tietokantaan tai sieltä ulos.
RDS	Relational Database Service, Amazonin tarjoama Aurora-tietokantapalvelu.
URL	Uniform Resource Locator, sisältää tiedon hakemiseen käytettävän protokollan ja palvelimen.
VPN	Virtual Private Network, eli virtuaalinen erillisverkko. Mahdollistaa yrityksen sisäverkon käyttämisen.

1 Johdanto

Opinnäytetyön tavoitteena oli konfiguroida AWS:n (Amazon Web Services) kapasiteetti optimaaliselle tasolle sovellukselle tehtävien kuormitustestien tulosten perusteella kustannusten laskemiseksi. AWS on kokoelma pilvipalveluita, jotka tarjoavat vaihtoehdon perinteiselle palvelinpohjaiselle ratkaisulle. Käytetyistä palveluista on olemassa eri kokoisia paketteja laskentakapasiteetin resurssien mukaan. Mitä enemmän resursseja ja isompi kapasiteetti, sitä kalliimpi hinta. Hinta määräytyy usein per tunti, joten palvelusta maksetaan käytön mukaan.

Alun perin Aktia Wallet-mobiilisovellus on kehitetty palvelinpohjaiseksi, josta se on myöhemmin siirretty AWS-alustalle. Ajan kuluessa on huomattu, että AWS:ään varatut resurssit ovat suuremmat kuin mitä sovelluksen käyttäjämäärät tällä hetkellä vaatisivat, joten tämänhetkinen alustarakenne ei ole kustannustehokas.

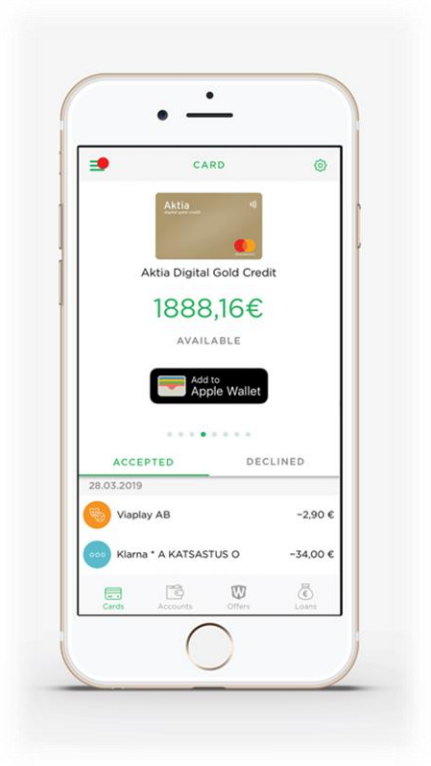
Työn tavoitteena oli hyödyntää kuormitustestausta AWS:n kulujen optimoinnin tueksi. Tämä tapahtui analysoimalla tuotannosta saatavaa dataa, jonka perusteella voitiin päätellä tämänhetkinen käyttöaste. Analyysista johdettujen tietojen perusteella luotiin kuormitustestiskriptit, joilla ilmennettiin käyttäjämääriä. Skriptien ajamisesta saatujen tulosten perusteella oli tarkoitus konfiguroida AWS uudestaan niin, että alustan laskentatehoa ja resursseja käytetään mahdollisimman tehokkaasti ilman, että vasteajat kasvavat liian suuriksi. Tarkoitus oli laskea AWS:n käytöstä aiheutuneita kuluja.

Opinnäytetyö tehtiin Aktia Pankki Oyj:lle. Aktia tarjoaa asiakkailleen muun muassa pankki-, varainhoito-, vakuutus- ja kiinteistönvälityspalveluita. Aktian omistavat suomalaiset Aktia- ja säästöpankkisäätiöt, yhteisöt, yritykset ja yksityishenkilöt. Konsernin maantieteelliseen toiminta-alueeseen kuuluvat Suomen rannikko, pääkaupunkiseutu ja sisämaan kasvukeskukset. Vuonna 1826 perustettu Aktia on vanhin Suomessa nykyisin toimivista talletuspankeista. Tällä hetkellä asiakkaita on noin 380 000 ja konttoreita 45. Digitaalisten kanavien sekä verkkopalvelujen kehittyessä konttorien määrä vähenee tulevaisuudessa. [1.]

Dokumentissa esitellään sovellus, jonka alustaratkaisua lähdettiin konfiguroimaan. Kerrotaan, miksi kuormitustestausta tehdään ja kuinka se suunniteltiin testattavan sovelluksen osalta. Lisäksi käydään läpi testiskriptin tekemisen eri vaiheet ja niiden ajosta saatavat tulokset konfiguraatioratkaisuineen. Lopuksi pohditaan, saavuttiko työ sille asetetut tavoitteet.

2 Aktia Wallet -mobiilisovellus

Aktia Wallet on digitaalinen lompakko, joka tarjoaa käyttäjälleen erilaisia välineitä taloutensa hallintaan. Se on mobiilisovellus, jonka avulla käyttäjä voi hallita Aktian myöntämiä kortteja sekä muuttaa mobiililaitteensa tai älykellonsa maksuvälineeksi (kuva 1). Tällä hetkellä Wallet-sovellusta on ladattu yli 150 000 kertaa [3, s.11].

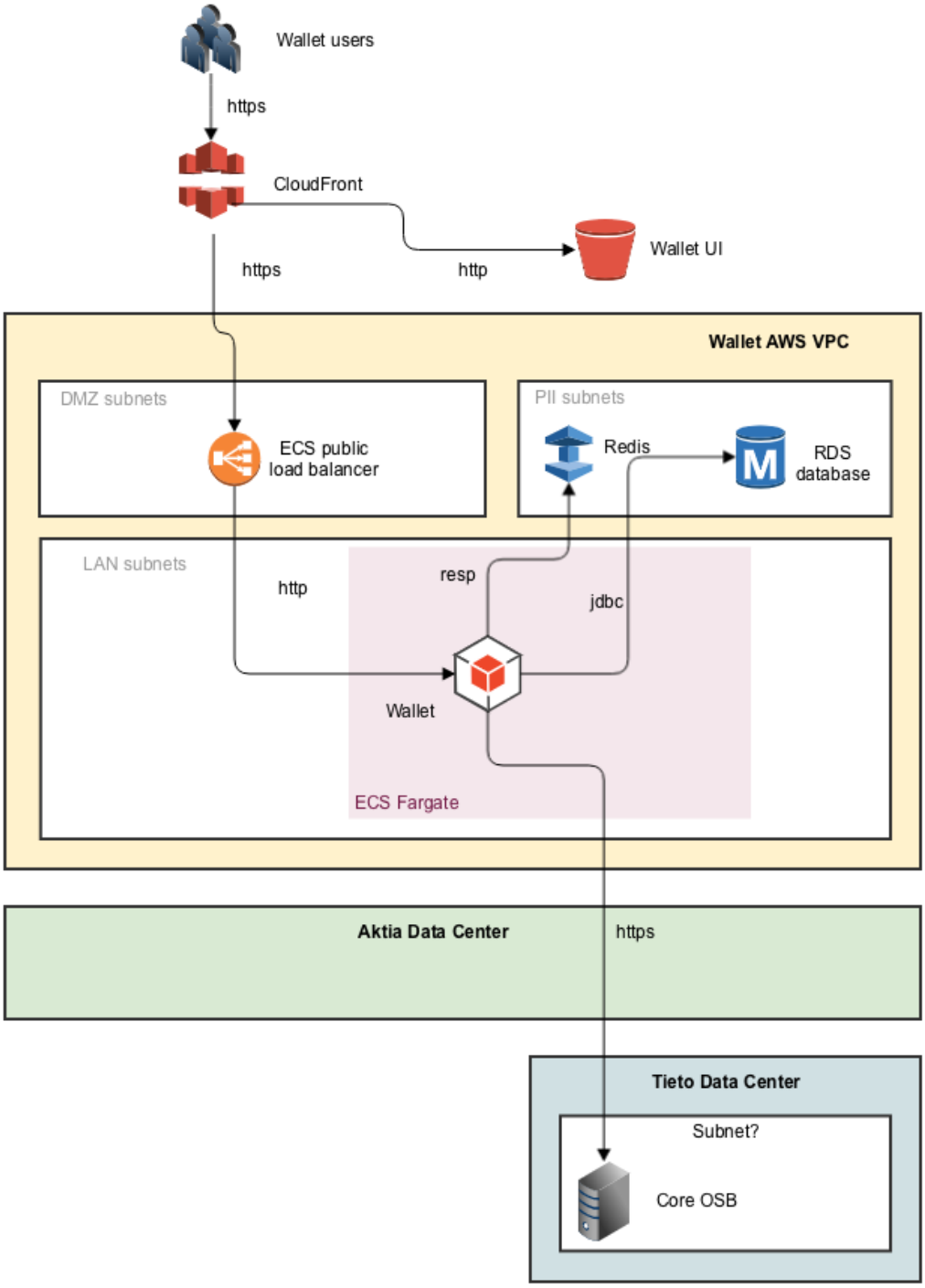


Kuva 1. Aktia Wallet on Aktian tarjoama maksukorttisovellus

Tällä hetkellä sovellus tukee Apple Payta, Google Payta, Garmin Payta ja Fitbit Payta. Nämä ovat kyseisten palveluntarjoajien kehittämiä sovelluksia, jotka mahdollistavat lähimaksamisen puhelinta tai älykelloa käyttäen. Walletin avulla käyttäjä pystyy lisäämään ja hallitsemaan Aktian maksukortteja lähimaksamisen osalta. Walletin avulla on myös mahdollista maksaa verkossa ja sovelluksissa. Maksamisen lisäksi käyttäjä pystyy hallitsemaan sovelluksen avulla Aktian fyysisiä pankki- ja luottokortteja, esimerkiksi muokkaamaan korttien rajoituksia, lukitsemaan kortteja sekä selailemaan maksutapahtumia.

Walletiin on myös mahdollista hankkia täysin digitaalinen mobiililuottokortti, Aktia Digital Gold Credit. Mikään muu pankki Suomessa ei tarjoa vastaavaa palvelua [2]. Käyttäjän on mahdollista ladata sovellus ja hankkia digitaalinen luottokortti, vaikka ei olisi Aktian asiakas. Digikortti on perinteistä luottokorttia turvallisempi, sillä kortissa on jokaisen käyttökerran jälkeen vaihtuva CVC-turvakoodi. CVC (Card Validation Code) on kolmi- tai nelinumeroinen turvakoodi, joka perinteisistä maksukorteista löytyy yleensä kortin takaa. CVC-koodia käytetään varmentamiseen verkko-ostoksia tehdessä.

Alkuvuonna 2019 Wallet siirrettiin AWS:ään. Backend-instanssit on konfiguroitu pyörimään ECS Fargate -pohjaisesti. ECS (Elastic Container Service) on palvelimeton dynaaminen alusta, jolle sovelluksen backend-instanssit on pystytetty. Fargate on taas palvelun osa, jota käytetään näiden palvelujen käynnistämiseen. Lisäksi taustalla on Amazonin Aurora-tietokanta. Arkkitehtuurikaaviossa on selitetty palvelun rakennetta (kuva 2).



Kuva 2. Palvelun arkkitehtuurikaavio.

”Wallet users” kuvaa tuotteen loppukäyttäjää, joka käyttää palvelua omalla mobiililaitteellaan. CloudFront on Amazonin tarjoama CDN (Content Delivery Network) -sisällönjakeluverkko, joka välittää tietoa taustajärjestelmistä käyttäjälle. Sovelluksen palvelinpuoli

(backend) on pystyssä AWS-pilvipalvelussa. Ensimmäinen osa, ECS public load balancer, tasoittaa palveluun tullutta kuormaa jakamalla liikennettä mahdollisimman tasaisesti saatavilla olevien resurssien kesken. ESC Fargate Clusterilla sijaitsee sovelluksen taustapuolen instanssit. Kyseessä on palvelu, joka kommunikoi käyttäjälle näkyvän osuuden kanssa (engl. client) ja tarjoaa dataa, joka on haettu eri järjestelmistä. Redis on välimuisti (engl. cache), jossa säilytetään tiettyjä hakutuloksia. Näin ollen kaikkea dataa ei tarvitse aina hakea uudestaan. Tämä mahdollistaa liikennemäärän vähenemisen, jolloin palvelu toimii asiakkaan suuntaan tehokkaammin ja nopeammin. RDS (Relational Database Service) on palvelu, jossa Amazon Aurora -tietokannat sijaitsevat. Tietokannoissa säilytetään palvelun kannalta kriittistä dataa. ECS-klusterin tehtävät (engl. task) juttelevat sekä RDS:n että Rediksen kanssa. Nämä keltaisella pohjalla sijaitsevat AWS-palvelut ovat niitä, joiden kuluja oli työn aikana tarkoitus optimoida. Alimpana arkkitehtuurikaaviossa on Tiedon ja Aktian palvelimilla sijaitsevat Aktian taustajärjestelmät.

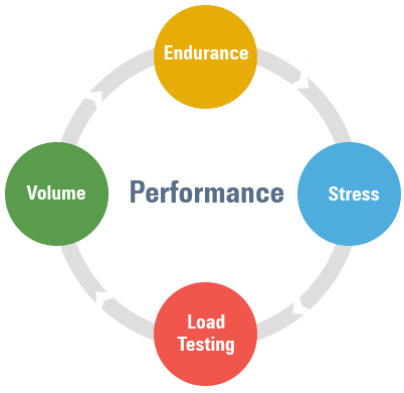
Sovelluksen kautta on mahdollista lähettää viestejä pankin asiakaspalveluun. Käyttäjän avatessa sovelluksen ja tarkastellessa hänelle tulleita viestejä, lähetetään kutsu viestien hakemiseksi. CloudFront ohjaa kutsun käyttäjän laitteelta ECS-kuormanjakajalle. Kuormanjakaja ohjaa kutsun eteenpäin sille ECS-tehtävälle, jolla on kapasiteettia sen suorittamiseksi. Kutsu käsitellään tehtävässä, jonka jälkeen kutsun mukaiset tiedot haetaan RDS-tietokannasta. Tietokannasta etsitään kyseisen käyttäjän viestit ja palautetaan ECS-tehtävälle. Tehtävä ohjaa palautuneet tiedot takaisin CloudFrontille, joka välittää haetut viestit käyttäjän laitteelle. Näin käyttäjä voi tarkastella sovelluksen kautta saamiaan viestejä.

3 Kuormitustestaus

Miksi kuormitustestausta kannattaa tehdä? Testauksen avulla voidaan todeta, että palvelu toimii hyvin suuremmankin käyttäjämäärän aikana hidastumatta tai kaatumatta. Tutkimukset ovat osoittaneet, että 47 % ihmisistä odottaa sivun tai sovelluksen aukeavan alle kahdessa sekunnissa ja 57 % sivujen käyttäjistä sulkevat sivun tai sovelluksen, jos sen aukeamiseen menee enemmän kuin kolme sekuntia [4]. Tämän vuoksi on tärkeää pitää huoli, etteivät suuret käyttäjämäärät vaikuta merkitsevästi sovelluksen käyttökokemukseen.

Suorituskyvyltään toimivan ja laadukkaan järjestelmän suunnittelu ja toteutus voi olla haastavaa. Suorituskykytestauksella testataan esimerkiksi, kuinka helposti sovellus kaatuu suuren käyttäjämäärän alla. Lisäksi voidaan selvittää, kuinka sivuston vasteaika muuttuu ja kuinka suuri kuorma vaikuttaa sivuston hidastumiseen. Myös suorituskykyä eniten rajoittavat pullonkaulat saadaan paikannettua testien avulla, joten ne voidaan korjata ennen tuotantoon vientiä. Testejä hyväksi käyttäen voidaan määritellä, mikä on suurin tuettu samanaikainen käyttäjämäärä sivustolla. Myös fyysisiä ominaisuuksia voidaan mitata ja optimoida testien avulla. Esimerkiksi tarvittavan laitteiston ja kaistanleveyden määrittelyssä on hyvä tietää sovelluksen kapasiteetti käsitellä suuria kuormia.

Suorituskykytestaus (engl. performance testing) jaetaan karkeasti neljään eri osa-alueeseen (kuva 5): kuormitustestaukseen (engl. load testing), rasitustestaukseen (engl. stress testing), kestävyystestaukseen (engl. endurance testing) ja piikkitestaukseen (engl. volume testing) [5].



Kuva 3. Suorituskykytestaus koostuu neljästä eri osa-alueesta.

Rasitustestauksessa luodaan järjestelmälle kuorma, joka on yli odotettujen ylärajojen. Tarkoituksena on ennaltaehkäistä järjestelmän kaatumista, kun osataan ennakoida, millä kuormalla järjestelmä mahdollisesti kaatuu. Kestävyystestauksessa tarkkaillaan erityisesti muistin käyttöä muistivuotojen varalta. Piikkitestauksessa järjestelmälle luodaan nopeita ja yllättäviä kuormapiikkejä, jotta saadaan selville, kuinka järjestelmä toimii nopean käyttäjämäärän nousun aikana ja kuinka se siitä palautuu. Kuormitustestauksessa selvitetään testattavan järjestelmän suorituskykyä tietyssä ajassa, kun kuormitus on kovaa, mutta ei kuitenkaan vastaavaa kuin rasitustestauksessa. Testien aikana seurataan vasteaikoja ja niiden kehitystä. Kuormitustestausta on mahdollista suorittaa manuaalisesti, mutta se on työlästä. Lisäksi testeissä simuloituja kuormia on vaikea toistaa, kun halutaan ajaa samankaltainen testi uudelleen. Tämän vuoksi kuormitustestaukseen on erilaisia välineitä, joilla voidaan luoda kuormaa sekä mitata vasteaikoja. [6.]

Muita haasteita kuormitustestauksessa on esimerkiksi testausympäristö. Aidosti tuotannonkaltaista testausympäristöä on vaikea järjestää, jolloin testausympäristössä voi esiintyä ongelmia, mitä tuotannossa ei esiinny ja päinvastoin. Lisäksi simuloitu kuorma on aina simuloitua kuormaa ja poikkeaa jossain määrin aidosta käyttäjäkuormasta. Ongelmat, joita testeistä paljastuu voivat olla uusia ja ennakoimattomia, jolloin niiden paikantaminen voi olla vaikeaa. Testeistä saatavat kaaviot ja vasteajat vaativat aina tulkintaa ja tulosten johtamista, jolloin inhimillisen virheen riski on läsnä. Testiskriptejä tehdessä ja tuloksia käsitellessä testaajan ammattitaito nousee esiin. Jos generoitu kuorma tai testin käyttötapaukset ovat liian yksinkertaisia, vääristyvät tulokset. Lisäksi pitää olla rohkeutta nähdä ja tuoda esille poikkeamat testituloksissa. Jos vain toivotaan hyviä tuloksia, on ne mahdollista luoda valikoiduilla testitapauksilla ja tulosten tulkinnalla. Tämän kaltainen testaaminen saattaa näyttää paperilla hyvältä, mutta ei palvele järjestelmän kehittäjiä tai käyttäjiä. Hyvä suunnittelu, keskeisten käyttötapausten tunnistaminen, testiskriptien muokkaaminen testauksen edetessä ja tuotannonkaltainen testidata ovat avainasemassa laadukasta kuormitustestausta tehdessä. [6.]

4 Valmistelu

Seuraavaksi esitellään työssä käytetyt ohjelmistot ja kerrotaan, kuinka dataa käsiteltiin ennen työn aloittamista.

4.1 Käytetyt ohjelmistot

Projektin aikana käytettiin erilaisia ohjelmistoja ja ohjelmia muun muassa testaukseen, versionhallintaan, lokien selaamiseen ja dokumentointiin. Seuraavaksi esitellään käytetyt ohjelmistot ja järjestelmät. Ohjelmistojen valinnassa ei juurikaan ollut vaihtoehtoja, sillä kuten monissa isoissa firmoissa, Aktia oli myös erikseen määritellyt ja ostanut lisenssit hyväksytyille ohjelmistoille.

AWS eli Amazon Web Services

AWS on kokoelma etätietojenkäsittelyresurssien palveluja, jotka muodostavat yhdessä Amazon.comin kautta tarjottavan pilvilaskenta-alustan. AWS antaa vaihtoehdon perinteiselle mallille, jossa palvelut pystytetään palvelimelle. Palvelimen sijaan palvelut pyörivät pilvessä, josta ostetaan palveluntarjoajalta haluttu määrä tilaa ja resursseja. AWS:n hyötyjä ovat muun muassa helppo skaalautuvuus, hinta, tietoturvallisuus ja monipuolinen konfiguroitavuus [7]. Vastaavanlaista palvelua tarjoavat muun muassa Alibaba, Windowa Azure ja Google Cloud Platform. Erona esimerkiksi AWS:n ja Alibaban välillä on muun muassa hinnoittelu. Kuukausitilauksella Alibaba on halvempi, mutta instanssipohjaisessa hinnoittelussa AWS vie voiton. Alibaba on käytetympi Aasian alueella, kun taas AWS on käytössä Pohjois-Amerikassa ja Euroopassa [8].

Splunk

Splunk on datankeruuohjelma, jonka avulla on mahdollisuus selata tuotannosta kerättävää käyttäjädataa ja virheilmoituksia. Ohjelma vastaanottaa järjestelmien ja sovellusten lähettämää tietoa reaaliaikaisesti, joka mahdollistaa sovelluksesta saatavan datan selailun viiveettä. Data myös tallennetaan, joten tietoa on mahdollista

hakea jopa kuukausien takaa. Splunk tallentaa myös laitetiedot. Tämä helpottaa esimerkiksi vianmääritystä. Jos joku tietty virhetilanne tuntuu toistuvan tuotannossa jatkuvasti, on Splunkista mahdollisuus katsoa, onko laite- ja sovellustiedoissa yhteneväisyyksiä. [9.] Splunkille on olemassa vaihtoehtoja, esimerkiksi Logdna tai Fluentd. Suurin osa näistä vaihtoehdoista olisi edullisempia kuin Splunk. Kuitenkin Splunk on ollut markkinoiden kärkeä laajan skaalautuvuuden ja konfiguroinnin osalta. [10.]

Neoload

Kuormitustestien ajamiseen käytettiin ohjelmistoa nimeltä Neoload. Neoload on Neotysin kehittämä ohjelma muun muassa suorituskyvyn ja rajapintojen testaukseen ja testien automatisointiin sekä testauksen suunnitteluun ja organisointiin. Neoload mahdollistaa testiskriptien luomisen käyttöliittymää käyttäen. Ohjelmistolla on mahdollisuus nauhoittaa, kun testattavaa järjestelmää käytetään halutulla tavalla. Neoload muodostaa tästä skriptin, jota käytetään kuormitustestejä ajaessa. Vaihtoehtona on myös kirjoittaa skripti niin sanotusti käsin, jolloin erilaisia muuttuja, parametrimäärittelyitä ja operaatioita tekemällä testiskripti muodostetaan. Neoload on tällä hetkellä markkinoiden automatisoiduin suoritustestaukseen kehitetty työkalu. Vaihtoja ohjelmistolle on useita, mutta ohjelmistojen ominaisuudet ja yhteensopivuus vaihtelevat runsaasti. Työtä tehdessä Neoloadista oli käytössä ohjelmistoversio 7.1, joka ei tällä hetkellä ole enää uusin versioiden 7.2 ja 7.3 julkaisun jälkeen. [11.]

Versionhallinta ja dokumentointi

Dokumentaatio ja versionhallinta on yrityksessä toteutettu Atlassianin konfiguroiduilla tuotteilla. Atlassian on toimija, jolla on monia ohjelmistokehitykseen sopivia tuotteita, joita yrityksen on mahdollisuus konfiguroida juuri omiin tarpeisiin sopiviksi. Tällaisia ovat muun muassa Bitbucket-pohjainen Git ja Confluensesta konfiguroitu Aktia Wiki. Ohjelmistoissa on hyvä se, että myös Atlassianin projektinhallintaan tarkoitettu ohjelma Jira on käytössä. Näin ollen dokumentaatio ja versionhallinta voidaan helposti yhdistää projektinhallintaohjelmistoon esimerkiksi linkittämällä dokumentteja tiketteihin tai scrum-boardille.

Testausympäristö

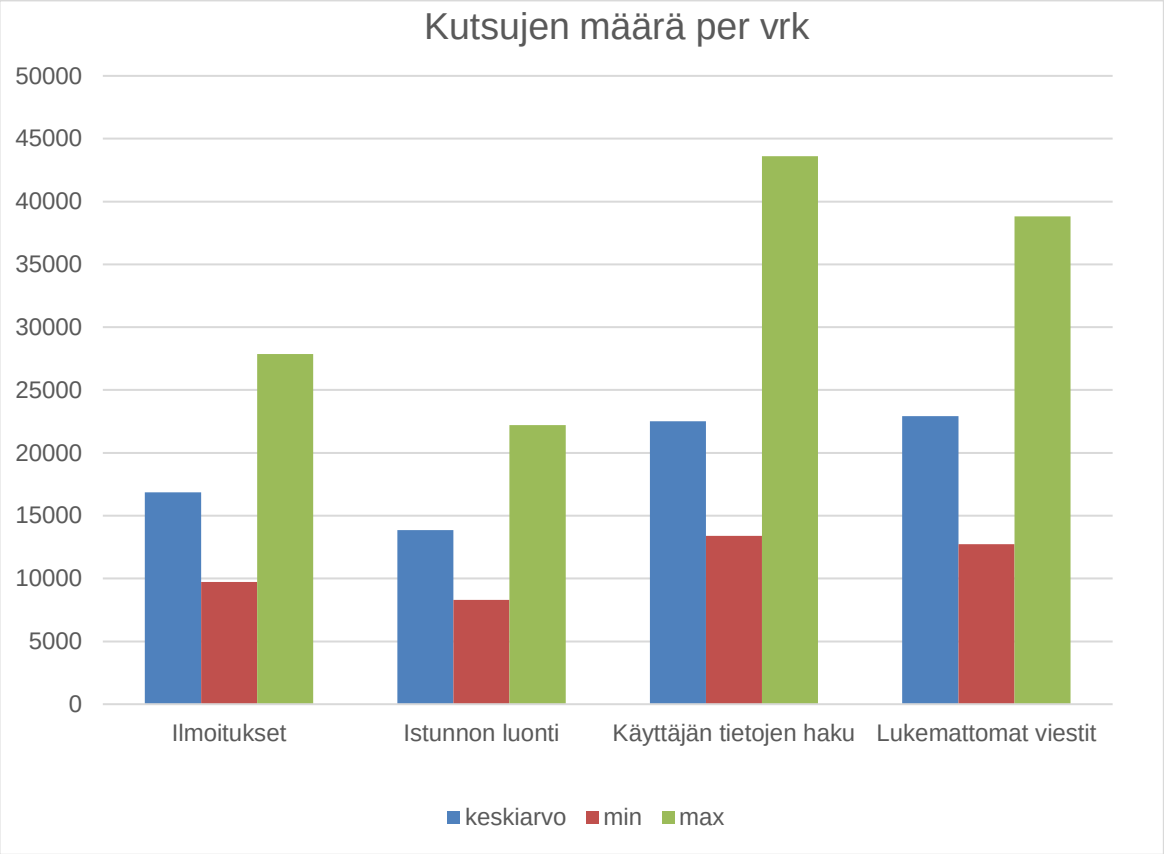
Testaus suoritettiin Aktian järjestelmätestausympäristössä (edempänä SYS). SYS on tuotannonkaltainen ympäristö, jossa ohjelmistosta on testattavana uusin versio, joka on integraatiotestattu.

4.2 Datan analysointi

Walletilla on tällä hetkellä noin 36 000 kuukausittaista uniikkia käyttäjää, jotka aloittavat yli 400 000 istuntoa kuukausittain. Päivittäin sovellusta käyttää noin 10 000 henkilöä. Jotta kuormitustestaus saataisiin rakennettua mahdollisimman järkevästi, on tärkeä tietää, mitä kutsuja ja ominaisuuksia käyttäjät käyttävät eniten. Kun tämä on selvitetty, on helpompi optimoida testien teko ja tehdä testit käyttäen vain niitä ominaisuuksia, joiden osalta kuormitus on suurempaa. Tätä varten haettiin Splunkin avulla lokeja tuotannosta, joista pystyttiin katsomaan käyttäjämääriä ja käytettyjä toiminnallisuuksia halutulta aikaväliltä.

Tarkasteltavaksi ajanjaksoksi valikoitui 20.11.2019-21.2.2020. Ajanjakso haluttiin rajata niin, että mukaan tulisi päiviä, jolloin käyttäjämäärä olisi poikkeava. Tällaisia päiviä oli muun muassa Black Friday (29.11.2019) sekä joulun ja uuden vuoden pyhät. Tuotannosta haettiin 50 käytetyintä toimintoa (engl. endpoint). Toiminnolla tarkoitetaan tiettyä kohtaa järjestelmässä, johon käyttäjä päätyy suoritettuaan tietyn sarjan komentoja. Näistä toiminnoista päätettiin jättää analyysin ulkopuolelle kutsut, jotka käyttävät vahvasti kolmannen osapuolen tarjoamaa palvelua. Lisäksi osa tiettyjä taustajärjestelmiä käyttävistä kutsuista oli jätettävä pois. Koska kuormitustestit oli tarkoitus ajaa järjestelmätestausympäristössä, ei kaikki tuotannossa toimivat järjestelmät ole testattavissa, tämä myös rajasi analysoitavien toimintojen määrää. Valitut toiminnot käyttäjämäärineen analysoitiin (taulukko 1).

Taulukko 1. Kutsujen määrät vuorokaudessa.



Käyttäjämääriä tarkastellessa oli huomattavissa tiettyjä yhtenäisyyksiä. Lähes poikkeuksetta eniten käyttäjiä oli 19.12., joka oli torstai ennen joulun pyhiä. Tällöin käyttäjiä saattoi olla 20-30% enemmän kuin normaalisti kiireisinä päivinä. Lisäksi käyttäjäpiikkejä oli huomattavissa kuun puolivälissä ja vaihteessa, sillä nämä ovat yleisiä palkkapäiviä. Hiljaisin päivä oli taasen jouluaatto, 24.12., jolloin käyttäjiä oli selkeästi vähemmän kuin muina päivänä. Joidenkin toimintojen kohdalla käyttäjämäärä tipahti alle puoleen keskimääräisestä käyttäjämäärästä. Vuorokausitasolla vaihtelua oli runsaasti. Arkisin käyttäjäkuorma pysyi välillä klo 9-20 suhteellisen tasaisena, jonka jälkeen se tippui noin kahden tunnin aikana noin 15%:iin tästä. Viikonloppuisin ja pyhäpäivinä sama kaava oli nähtävissä, tosin silloin kovemman kuormituksen aikaikkuna oli muutaman tunnin pienempi kuormituksen alkaessa tunnin pari myöhemmin ja loputtua aikaisemmin.

Eniten käyttäjäpiikkejä havaittiin käyttäjän tietojen haussa. Keskimäärin kutsua haettiin yli 22 000 kertaa vuorokaudessa. Vaihtelua päivien välillä oli kuitenkin huomattavasti.

Korkeimmillaan kutsuja tehtiin yli 43 000 vuorokaudessa. Tällöin laskennallisesti jokainen käyttäjä on vähintään kerran käyttänyt toimintoa vuorokauden aikana. Hiljaisimpana päivänä alle 13 000 kutsua on tehty.

Käyttäjän rekisteröityessä käyttäjäksi tai kirjautuessa sisään luodaan uusi istunto. Istunnon luomiseen käytetty kutsu on käyttäjämääriltään hyvin samanlainen. Kokonaismäärällisesti kutsuja on tehty vähemmän keskiarvon ollessa 13800 kutsua vuorokaudessa, mutta vaihtelevuus kutsujen määrän suhteen on sama vaihteluvälin ollessa 8300-22200 kutsua vuorokaudessa.

Tämän lisäksi tarkasteltiin lukemattomien asiakasviestien hakemiseen käytetyn kutsun käyttäjämääriä. Kyseessä on sovelluksen eniten kantaa kuormittava kutsu. Tämä on yksi syy, miksi kutsun toimivuutta on tärkeä testata suurien käyttäjämäärien aikana. Keskimäärin lukemattomia viestejä haetaan lähes 23 000 kertaa vuorokaudessa. Valitulla aikavälillä kutsuja tehtiin kiireisempänä päivänä 38 800, kun taas hiljaisimpana päivänä alle 13 000. Kutsujen määrän vaihteluväli on jälleen suuri.

Neljäs kutsuista palauttaa tiedon, mitä ilmoituksia käyttäjälle näytetään tämän kirjoutuessa sisään. Keskiarvo kutsujen määrälle on noin 16 800 kutsua ja vaihteluväli hieman alle 10 000 kutsusta lähes 28 000 kutsuun vuorokaudessa.

5 Skriptin tekeminen

Seuraavaksi kerrotaan skriptin kirjoittamisesta. Lisäksi kerrotaan myös, millaista suunnittelutyötä tuli tehdä ennen varsinaisen testiskriptin tekemistä sekä millaista työtä skripti vaati, kun tekninen toteutus oli valmis.

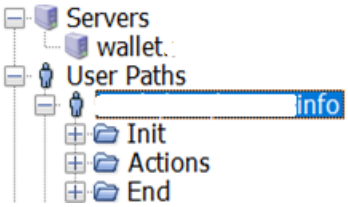
5.1 Suunnittelu

Kun tuotannon data oli analysoitu ja oli selvitetty, mitkä käyttötapaukset toistuivat tuotannossa useimmiten, oli kyseisistä käyttötapauksista luotava skriptit. Skriptien avulla simuloitaisiin käyttötapauksia. Kun skriptit ajettaisiin Neoloadilla, saataisiin luotua haluttu kuorma, ja näin testattua sovelluksen kestävyyttä.

Sovellukselle oli jo olemassa joitain vanhoja kuormitustestausskriptejä. Näitä skriptejä käytettiin pohjana, kun alettiin luoda uusia testiskriptejä. Skriptit luotiin syöttämällä parametreja ja muuttujia Neoloadiin. Parametreilla määriteltiin testikäyttäjä, jolle istunto aloitettiin. Niiden avulla varmistettiin myös, että testeissä käytetään oikeaa versiota sovelluksesta. Näiden tietojen avulla ohjelma loi testiskriptin. Osa skriptistä piti kuitenkin kirjoittaa käsin.

Aluksi oli testattava, että Neoloadilla saadaan yhteys oikeille palvelimille ja palveluihin. Koska kyse on pankkipalveluissa, on ne suojattu erityisen hyvin ulkopuolisilta, kuten arvata saattaa. Tämän vuoksi oli järkevintä ensin selvittää, että oikeat palomuuriavaukset oli tehty, jotta työ saataisiin tehtyä. Tätä varten valittiin yksi kutsu, jota pystyi käyttämään ilman, että käyttäjää autentikoitaisiin. Aivan ensin tuli määrittää palvelinasetukset. Asetuksista pystyy päättämään, missä ympäristössä sijaitsevaa palvelua halutaan testata. Asetuksiin siis määritettiin osoite, jonne palvelu on pystytetty sekä portti, jota käytetään.

Tämän jälkeen oli määritettävä käyttötapaus (engl. user path). Neoloadissa käyttötapaus koostuu kolmesta vaiheesta (kuva 4).



Kuva 4. Yhden käyttötapauksen eteneminen Neoloadissa.

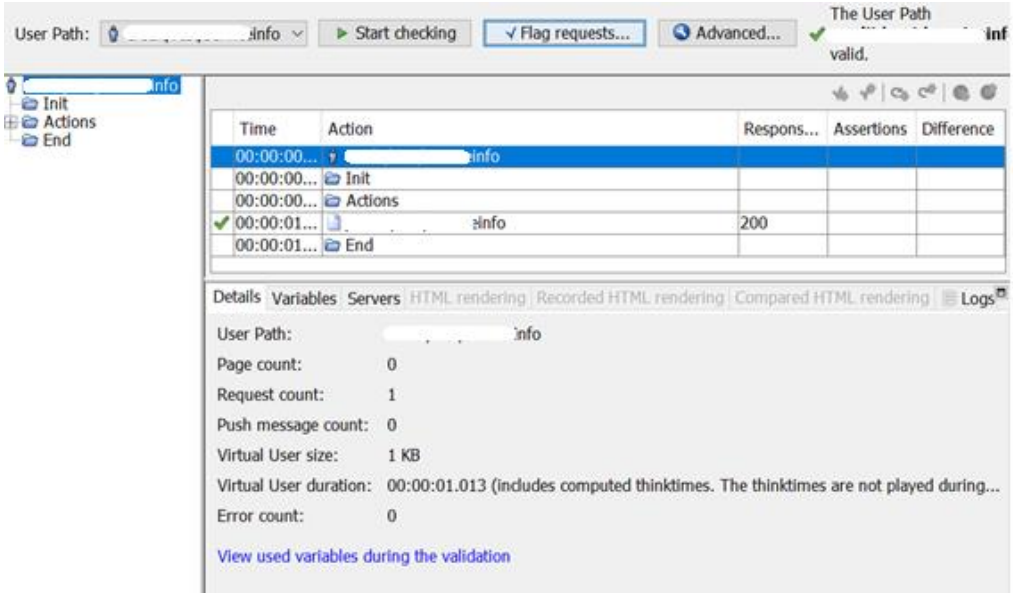
Ensimmäisen Init-vaiheen kohdalla tehdään asiat, jotka aloittavat istunnon. Eli jatkoa ajatellen tässä vaiheessa luotaisiin istunto ja autentikoitaisiin eli tunnistettaisiin käyttäjä valtuustunnuksen (engl. token) ja PIN-koodin avulla. Seuraavassa Actions-vaiheessa tehtäisiin varsinaiset toimintoon tarvittavat kutsut. Kun nämä on saatu tehtyä, siirrytään viimeiseen End-vaiheeseen, jossa istunto lopetettaisiin. Kun nämä kaikki vaiheet on käyty läpi, on haluttu käyttötapaus suoritettu kerran alusta loppuun.

Koska ensimmäiseksi käyttötapaukseksi valittiin sellainen toiminto, joka ei vaadi käyttäjän tunnistautumista, voitiin aloitusvaihe jättää pois. Koska kyse oli yksittäisestä tietoja hakevasta kutsusta, ei varsinaista istuntoa aloitettu ollenkaan. Näin ollen Init- ja End-vaiheet jätettiin kokonaan tyhjäksi määrittelyissä ja kaikki toimintoon liittyvät määrittelyt tehtiin toimintavaiheessa. Ensin oli määriteltävä toiminnon tyyppi. Kyseisessä tapauksessa käytettiin manuaalista määrittelyä. Tämä tarkoittaa, että toiminto ja kutsut määritellään käsin, ja toiminto myös käynnistetään manuaalisti. Vaihtoehtona olisi esimerkiksi toiminnon käynnistyminen toisen käyttötapauksen päätteeksi toisesta käyttötapauksesta käytettävän linkin tai uudelleenohjauksen avulla.

Käyttötapauksessa haettiin palvelun tiedot. Kutsu määriteltiin GET-kutsuksi. Kutsun osoitteeksi asetettiin aiemmin määritelty palvelimen osoite, johon lisättiin aiemmin määritelty portti ja kutsu kokonaisuudessaan. Tämän lisäksi palvelin tuli valita vielä erikseen listasta, jossa näkyvät kaikki määritellyt palvelimet, tässä tapauksessa vain se yksi aiemmin määritelty. Path-kohtaan määriteltiin kutsun polku, joka on sama, kuin mitä tuotannon dataa analysoidessa valikoitui. Tässä ensimmäisessä tapauksessa käytettiin kutsua /info. Kun toimintavaiheen määrittelyt oli saatu kuntoon, oli tehty kaikki valmistelut, mitä ensimmäisen käyttötapauksen ajamiseksi tarvitaan. Tämän jälkeen käyttötapaus validoitiin, eli sen toiminta testattiin ajamalla yksi yksittäinen kutsu käyttötapauksen mukaisesti.

Tämä säästää aikaa ja kuormitusta koneelta ja palvelimilta verrattuna vaihtoehtoon, että käyttötapauksen toimivuutta testattaisiin ajamalla se varsinaisessa testiajossa.

Info -käyttötapaus meni heti ensimmäisellä yrittämällä läpi. Validoinnin raportista (kuva 5) on nähtävillä, että yhteys palvelimelle on muodostettu, ja valittu kutsu on palauttanut onnistuneesti 200.



Kuva 5. Käyttötapauksen validoinnin tulokset.

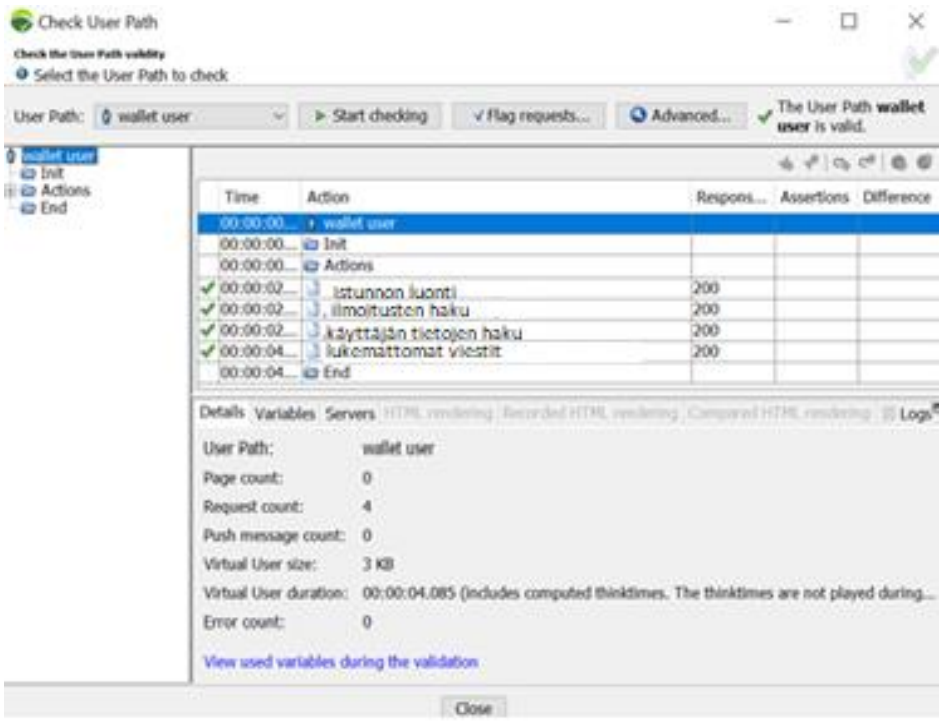
Raportista näkee myös, että kutsu on lähetetty kerran ja että vastauksen saamiseen on mennyt 1,013 s. Vastauksen saamiseen kulunut aika on normaalia suurempi, sillä kyseinen validointi suoritettiin etätyöskentelypäivänä. Näin ollen kutsujen tekeminen ja vastausten saaminen on hieman hitaampaa kuin toimiston sisäverkosta käsin tehtynä. Voitiin siis todeta, että oikeat asetukset oli saatu tehtyä, jotta skriptien kirjoittaminen voitiin aloittaa.

5.2 Skriptin luominen

Kun oli saatu varmistettua, että yhteys palvelimelle toimii, ja tietoja saadaan haettua, oli aika kokeilla kutsua, jonka toteuttaminen vaatisi käyttäjän autentikointia. Ensimmäisenä kokeiltiin istunnon luomiseen käytettyä POST-kutsua, joka luo valtuustunnuksen mukaan

määritellylle käyttäjälle istunnon. Tätä varten kutsuun oli luotava otsakkeita (engl. html header), eli kutsun määrittteleviä muuttujia. Käyttäjän autentikointi tapahtui PIN-koodin ja valtuustunnuksen avulla. Testikäyttäjillä oli kaikilla sama PIN-koodi, joka päätettiin kovakoodata testiskriptiin. Tämä tarkoittaa myös sitä, että jatkossa kaikille testikäyttäjille pitää määritellä sama kyseinen PIN-koodi. Käyttäjän yksilöivä valtuustunnus valittiin listasta, joka oli luotu testikäyttöä varten. Tämän lisäksi määriteltiin käytettävän sovelluksen versio sekä missä muodossa vastaus haluttiin. Kun otsakkeet ja parametrit oli määritelty oikein, kutsu meni hyvin läpi.

Tämän jälkeen haluttiin luoda testit muille tuotannon datasta johdetuille kutsuille. Aluksi päätettiin ketjuttaa kaikki kutsut samaan käyttötapaukseen, jotta yhden istunnon kautta saataisiin ajettua kaikki kutsut. Myöhemmin nämä olisi mahdollista erottaa omiksi käyttötapauksiseen, jos haluttaisiin ajaa yksittäisiä toimintoja kokonaisen ketjun sijasta. (Kuva 6.)

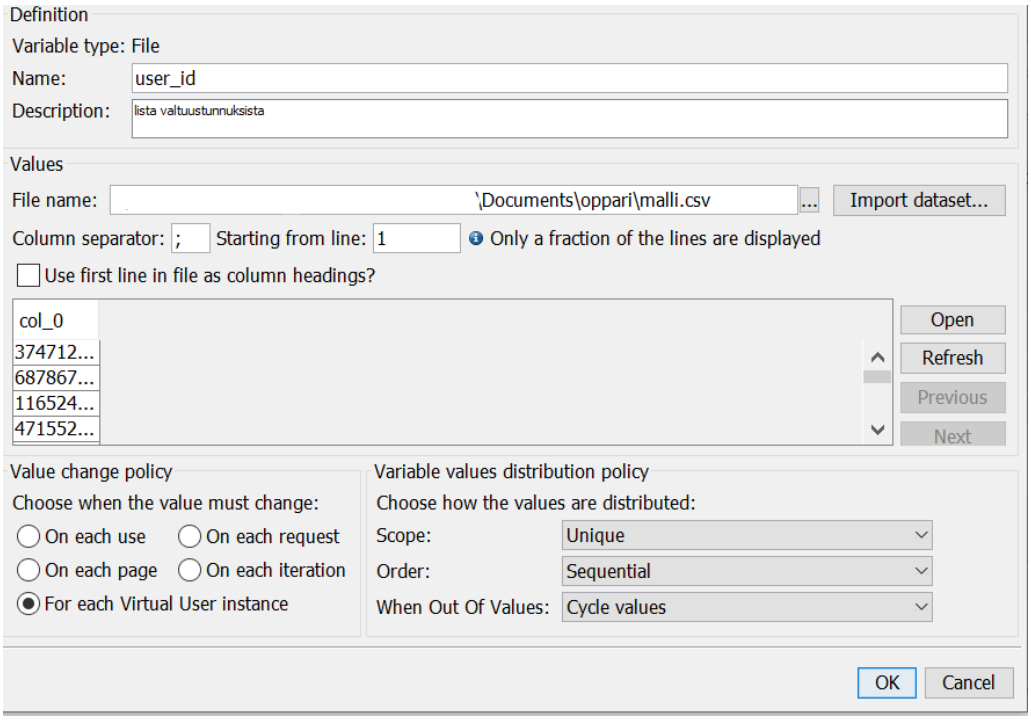


Kuva 6. Ketjutettujen kutsujen validaatioajo.

Loput valituista kutsuista oli GET-kutsuja, jotka hakivat pyydettyjä tietoja. Kutsujen parametrisisältö oli sama. Ainoastaan itse kutsu vaihdettiin. Käyttötapauksen tarkistamisessa

Neoload ajaa yhden testikäyttäjän osalta käyttötapauksen yhden kerran läpi varmistaen, että kutsut menevät läpi ja palauttavat vastauksen. Tarkistus käynnistetään klikkaamalla ”Start checking” -painiketta. Kaikki ajettavat kutsut ovat palauttaneet vastauksena 200, mikä tarkoittaa, että kaikki kutsut on ajettu onnistuneesti läpi. Kutsuista on osa peitetty tietoturvasyistä, ja näkyviin on jätetty vain polun loppuosa.

Tämän jälkeen oli luotava skriptiin silmukka, joka mahdollistaisi istunnon luomisen uudella käyttäjällä aina, kun toiminto on ajettu läpi. Vanhoissa testiskripteissä oli olemassa lista valtuustunnuksia testikäyttöön. Tarkoitus oli siis luoda skriptiin silmukka, joka vaihtaa aina valtuustunnuksen uuteen istunnon luomisen yhteydessä (kuva 7).



Kuva 7. Listamuuttujan määrittely.

Ensimmäisenä oli luotava muuttuja, joka sisältää kyseisen listan. Luotiin muuttuja user_id, johon ladattiin csv-tiedostona oleva lista valtuustunnuksista. Muuttujan asetuksissa määritettiin, miltä riviltä listaa lähdetään lukemaan ja missä vaiheessa lähdetään siirtymään listassa eteenpäin. Listaa lähdettiin lukemaan ensimmäiseltä riviltä ja yhden käyttötapauksen lopussa siirryttiin listassa seuraavaan alkioon.

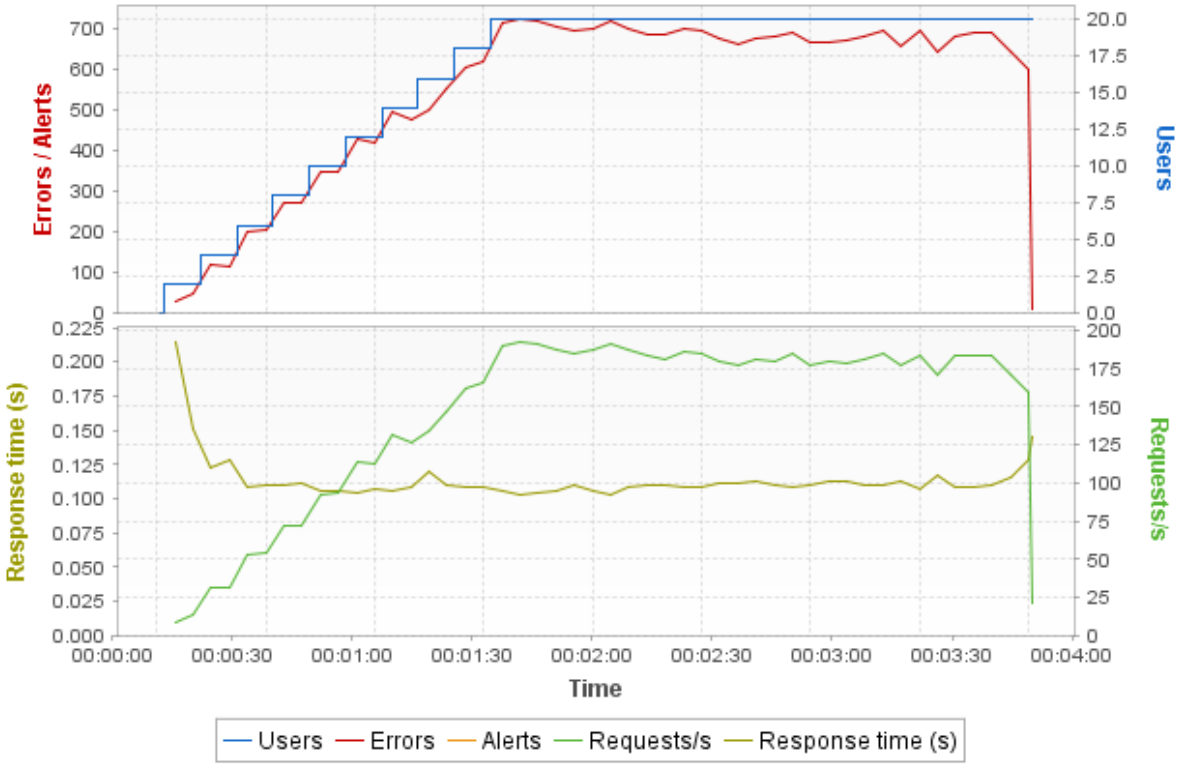
Jokaisessa kutsussa kyseistä muuttujaa tarvittiin kahdessa eri kohdassa; muuttujan arvo toimi sekä käyttäjän että istunnon tunnistamisessa. Listamuuttuja käytti yhtä listan alkiota ja siirtyi seuraavaan alkioon vasta seuraavassa iteraatiossa. Näin ollen samaa user_id-listamuuttujaa voitiin käyttää molempien tarvittavien parametrien määrittelyyn.

Itse listamuuttujan luominen ja sen asetusten määrittäminen oli helppo ja yksikertainen juttu. Päänvaivaa toi muuttujan käyttäminen käyttötapauksen ensimmäisessä POST-kutsussa. Kyseinen toiminto luotaisiin aloitusvaiheeseen, jolloin se aloittaa koko istunnon. Ilman tämän kutsun toimimista ei voisi muitakaan toimintoja ajaa onnistuneesti. Kyseisen kutsun parametrit lähetetään JSON-muotoisena. Ohjelman korjaavista ja avustavista toiminnoista sekä ahkerasta tiedon hausta huolimatta oikean syntaksin löytyminen paljastui yllättävän vaikeaksi. Lopulta selvisi, että parametrina lähetettävän listamuuttujan olisi myös oltava lainausmerkeissä, jotta kutsu toimii oikein.

Kun tämä saatiin toimimaan, menivät muutkin kutsut toivotusti läpi. Tämän jälkeen tarkistettiin Splunkista, miltä kutsujen jättämät lokit näyttävät. Tässä vaiheessa huomattiin, että GET-kutsujen URL:ssä on nähtävissä lähetetyt parametrit, eli käyttäjän ja istunnon identifioivat tunnukset ja PIN-koodi. Vaikka testikäytössä tämä ei haittakaan, haluttiin testeistä mahdollisimman samankaltaiset tuotannon kanssa. Tämän vuoksi oli parametrit saatava pois URL:stä.

Alkuvaiheessa skriptien tekoa oli nämä määritelty kutsun parametreiksi Neoloadin käyttöliittymän kautta. Tämän määrittelyn mukaan parametrit kuljetetaan eteenpäin URL:ssä, eikä tätä ollut mahdollista muuttaa ohjelman kautta. Niinpä ainoa vaihtoehto oli siirtää parametrit kutsun runko-osaan (engl. html body). Tätä ei kuitenkaan voinut enää muokata kutsun luomisen jälkeen, vaan ne oli tehtävä kutsua luodessa. Kutsut oli siis luotava alusta asti uudestaan, ja luonnin yhteydessä oli määriteltävä käyttäjän yksilöivä tunnus (user_id) ja PIN-koodi. Tämä oli kuitenkin yksinkertainen vaihe, sillä yhden GET-kutsun luonnin jälkeen kutsu oli mahdollista kahdentaa, ja muokata vain kutsun polun loppuosio oikeaksi. Parametrien siirtäminen otsakkeista runkoon oli itsessään nopeahko vaihe. Kauemmin kesti ymmärtää, ettei kutsuja pysty muokkaamaan, vaan ne pitää luoda kokonaan uudestaan.

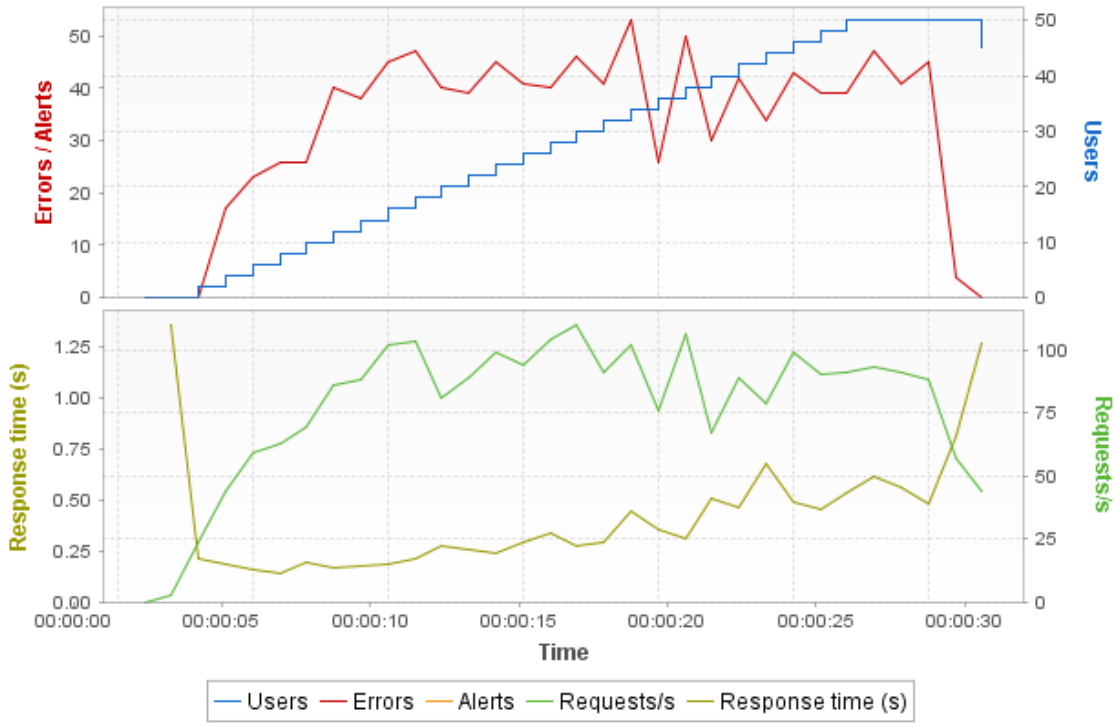
Kun testien kutsut oli saatu kuntoon, pystyttiin tekemään ensimmäisiä koeajoja. Ensimmäinen ajo kesti pari minuuttia. Testin käynnistyessä virtuaalikäyttäjiä oli 5. Aina 30 sekunnin jälkeen Neoload lisäsi ajoon uudet 5 käyttäjää. Näin ollen 20 testikäyttäjää saatiin testiajoon 2 minuutin kuluessa. Kun ajo oli suoritettu loppuun, Neoload kokosi siitä raportin. Raportista huomattiin, että ajon virheprosentti (engl. error rate) oli huomattavan suuri, 74,9 % (kuva 8).



Kuva 8. Ensimmäisen varsinaisen testiajon tulokset. Punainen kuvaaja kertoo virheiden määrän.

Kun Splunkista ryhdyttiin selvittämään, mikä korkean prosentin aiheuttaa, huomattiin, että moni kutsu palauttaa 403, eli virhe autentikoinnissa. Tämän jälkeen kävi ilmi, että kuukausi takaperin oli tehty GDPR:ään (General Data Protection Regulation) perustuva tietokantojen putsaus, jossa viimeisen vuoden aikana käyttämätön data putsattiin tietokannoista. Tämä operaatio oli koskenut myös testiympäristöjä, joten testikäyttäjät, joita ei ollut vuoteen käytetty, oli poistettu kokonaan. Oli siis kokeiltava käyttötapauksen ajamista jokaisella käyttäjällä ja poistettava käsin userid-muuttujalistasta näiden käyttäjien valtuustunnukset. Käytännössä valtuustunnuksien tarkastus toteutettiin ajamalla kolme 15-20 testikäyttäjän testiajoa. Jokaisen ajon jälkeen Splunkista tarkastettiin, mitkä kutsut

ovat palauttaneet koodin 403, ja tarkastettiin, millä valtuustunnuksella kyseinen kutsu oli tehty. Tämän jälkeen kyseinen valtuustunnus poistettiin listasta. Kun kaikki valtuustunnukset oli käyty läpi, tehtiin vielä yksi ajo, jossa testattiin, että kaikki listaan jääneet valtuustunnukset ovat toimivia. Kuitenkin listan siistimisen jälkeen valtuustunnuksia jäi jäljelle vain kolme kappaletta. Päädyttiin siis luomaan 50 uutta testikäyttäjää. Kun nämä oli tehty, Neoloadiin ladattiin uusi lista valtuustunnuksista muuttujan asetuksista lähdetiedostoksi ja tehtiin uusi testiajo. Ajossa huomattiin, että kaikki käyttäjät ovat tällä hetkellä toimivia. Kuitenkin virheprosentti oli edelleen liian iso, 42,2% (kuva 10), ja oli lähdettävä selvittämään, mikä tämän aiheuttaa.



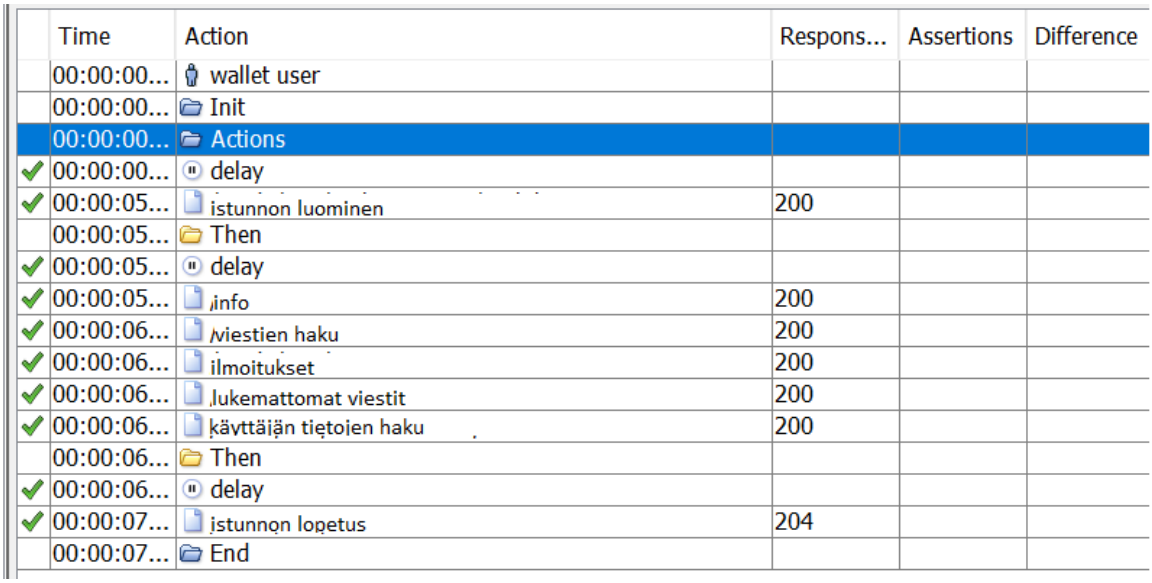
Kuva 8. Käyttäjälistan muokkauksen jälkeen tehty testiajo.

Jälleen Splunkista tarkasteltaessa huomattiin, että käyttäjän tietoja ja lukemattomia viestejä hakevat kutsut epäonnistuvat lähes poikkeuksetta. Tarkemman tutkimisen jälkeen huomattiin, että kyseiset kutsut jäävät niin sanotusti orvoksi istunnosta. Ongelma paikannettiin virheeksi skriptissä istuntojen hallinnan suhteen. Ongelma ratkaistiin tallentamalla ensimmäisestä POST-kutsusta palautuva istuntotunnus muuttujaan, joka periytyi aina onnistuneen toiminnon suorittamisen jälkeen käyttötapauksen seuraavalle toiminnolle.

Näin varmistuttiin siitä, etteivät istunnot mene sekaisin ja käyttötapauksen viimeiset kutsut jää orvoksi. Tämän lisäksi lisättiin vielä käyttötapauksen viimeiseksi toiminnoksi DELETE-kutsu, joka lopettaa käynnissä olevan istunnon. Kun nämä muutokset oli tehty, saatiin skripti ajettua läpi 0 % virheprosentilla.

Haasteet skriptien kanssa osoittivat, että vaikka Neoloadilla tehtävä käyttötapauksen validointi osoittaisi skriptin olevan hyvä ja toimiva, voi siellä olla virheitä, jotka vääristävät testituloksia. Tultiin siis siihen tulokseen, että olisi hyvä lisätä jonkinlaista validointia käyttötapaukseen. Testiasiakkaiden tiedoista päätettiin tuoda asiakkaan puhelinnumerot ja henkilöturvatunnukset, ja yhdistää lista valtuustunnuslistaan. Nyt listassa oli kolme saraketta; jokaisella rivillä oli omassa sarakkeessaan testikäyttäjän valtuustunnus, puhelinnumero ja henkilöturvatunnus. Kun ensimmäinen POST-kutsu tehtiin tietyllä valtuustunnuksella, haettiin saman rivin toisesta ja kolmannesta sarakkeesta puhelinnumero ja henkilöturvatunnus. Samalla Neoloadiin asetettiin kummallekin tiedolle oma muuttuja. Muuttujien asetuksissa määriteltiin, että muuttujien arvot haettiin vastauksen rungosta. Lisäksi määriteltiin, minkä tietyn merkkijonon jälkeen haluttu arvo esiintyy ja mihin tallennettava arvo loppuu.

Näiden tietojen perusteella Neoload etsi vastauksesta halutun tiedon ja tallensi sen muuttujaan. Tätä muuttujaa verrattiin listan vastaavassa sarakkeessa olevaan arvoon. Jos tiedot täsmäsivät, istunto oli luotu oikealle asiakkaalle, ja käyttötapausta voitaisiin jatkaa eteenpäin. Jos tiedot eivät kuitenkaan täsmänneet, oli istunnon luonnissa mennyt jotain pieleen ja istunto lopetettiin. Samanlainen tarkastus lisättiin käyttäjän tietoja hakevan kutsun perään. Nyt kun molemmat tarkastukset oli saatu tehtyä ja testiajo tuli takaisin 0 % virheprosentilla, voitiin todeta, että testiskripti oli vihdoin valmis (kuva 11).



	Time	Action	Respons...	Assertions	Difference
	00:00:00...	wallet user			
	00:00:00...	Init			
	00:00:00...	Actions			
✓	00:00:00...	delay			
✓	00:00:05...	istunnon luominen	200		
	00:00:05...	Then			
✓	00:00:05...	delay			
✓	00:00:05...	/info	200		
✓	00:00:06...	/viestien haku	200		
✓	00:00:06...	ilmoitukset	200		
✓	00:00:06...	/lukemattomat viestit	200		
✓	00:00:06...	/käyttäjän tietojen haku	200		
	00:00:06...	Then			
✓	00:00:06...	delay			
✓	00:00:07...	/istunnon lopetus	204		
	00:00:07...	End			

Kuva 11. Valmiin skriptin käyttötapauksen validoinnin tulokset.

Tarkastusajossa on näkyvillä, kuinka skripti käydään läpi vaihe vaiheelta. Init-vaiheessa tehdyn istunnon aloituksen jälkeen tarkastetaan ”if-then-else”-operaation avulla, että istunnon aloitus palauttaa oikean käyttäjän tiedot. Tämän jälkeen on ajettu muut viisi toimintoa onnistuneesti näiden palauttaessa arvon 200. Viimeinen kutsu lopettaa istunnon ja palauttaa arvon 204 onnistuneen lopetuksen jälkeen.

5.3 Skriptin optimointi

Kun testiskripti oli teknisesti valmis, oli vielä optimoitava se vastaamaan tuotannon kuormaa. Kuormaa ei haluttu laskea aivan yhtä alas kuin tuotannossa, vaan haluttiin jättää hieman pelivaraa asettamalla ajettava kuorma hieman tuotantoa korkeammaksi. Oli siis laskettava, paljonko kuormaa tulisi testeissä ajaa, jotta se simuloisi sovellukseen tuotannossa kohdistunutta kuormitusta. Tätä varten haettiin Splunkista yhden arkipäivän lokit testeissä käytettyjen toimintojen osalta (taulukko 2).

Taulukko 2. Tuotannon lokeista tuotu käyttäjämäärä yhdeltä arkipäivältä.

aika	ilmoitukset	istunnon aloitus	käyttäjän tietojen haku	info	lukemattomat viestit
0:00	284	232	409	189	383
1:00	183	148	272	110	234
2:00	129	108	184	87	169
3:00	114	89	154	55	136
4:00	123	96	156	72	143
5:00	208	180	272	128	252
6:00	423	359	568	252	533
7:00	570	492	797	355	791
8:00	833	717	1158	550	1066
9:00	869	767	1270	618	1229
10:00	1094	922	1568	739	1511
11:00	1109	1004	1720	829	1588
12:00	2729	1998	4522	1950	3943
13:00	3422	2471	5908	2353	4970
14:00	3361	2433	5892	2270	4667
15:00	3174	2351	5119	2117	4503
16:00	1700	1383	2524	1071	2256
17:00	1563	1321	2311	959	2070
18:00	1413	1215	2077	916	1982
19:00	1144	1032	1759	795	1632
20:00	1051	912	1545	717	1443
21:00	914	772	1316	606	1225
22:00	769	619	1065	516	1058
23:00	525	426	746	350	754
ka/h	1154,33	918,63	1804,67	775,17	1605,75

Data eroteltiin näyttämään toteutuneiden kutsujen määrän tunnissa. Näistä arvoista laskettiin keskiarvo sekä maksimiarvo. Otanta on datan tarkastelujakson kiireisimmältä päivältä, 19.12. Käytetyimmässä kutsussa, lukemattomien viestien haussa, keskiarvo oli 1606 kutsua tuntia kohden, korkeimpien käyttäjämäärien ollessa lähes 5000 kutsua. Testiskriptin yhdessä käyttötapauksessa ajetaan kaikki toiminnot kerran, eli kutsutaan jokaista toimintoa yhden kerran, joten oli otettava selvää, montako kertaa käyttötapaukset ajetaan nykyisellä asetuksella tunnin aikana.

Tätä varten tehtiin kokeeksi yksi 10 minuutin testiajo. Ajossa oli aluksi kaksi käyttäjää, ja siihen lisättiin kaksi käyttäjää 10 sekunnin välein käyttäjien maksimimäärän ollessa 50. Tämä saavutettiin nykyasetuksilla 4 minuutissa, jonka jälkeen kuorma pidettiin samana

6 minuuttia. Tämän ajon aikana käyttötapauksia suoritettiin alusta loppuun lähes 26 000 kappaletta. Tästä voidaan todeta, että kyseisillä määrittelyillä tunnin kuormitustestauksen aikana käyttötapauksia toteutettaisiin alusta loppuun noin 156 000 kappaletta eli noin 30-kertainen määrä tuotannon maksimikuormitukseen verrattuna. Ajettavaa kuormaa tulisi vielä pienentää, jotta se vastaisi tuotantoa.

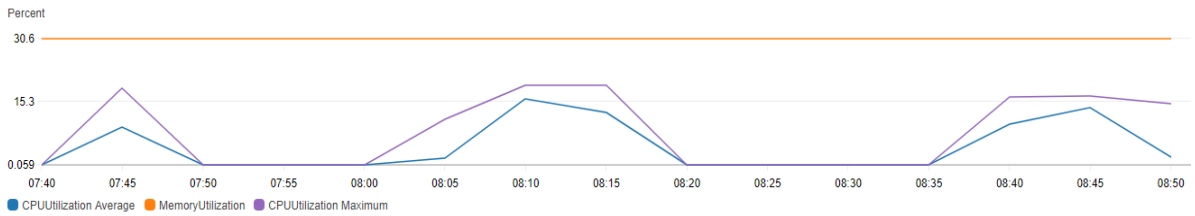
Tämä toteutettiin lisäämällä ennen istunnon luovaa kutsua kahden sekunnin viive. Tämän jälkeen suoritettiin uusi 10 minuutin ajo. Muutaman testiajon jälkeen päädyttiin ratkaisuun, jossa käyttötapauksen alussa on satunnaisesti arvottu 5-20 sekunnin viive. Tällä ratkaisulla käyttötapausten määrä tunnissa oli noin 10 000 kappaletta. Käyttötapausten määrä oli edelleen noin kaksinkertainen verrattuna tuotannossa esiintyneisiin maksimikuormiin. Jos viiveen alarajaa olisi nostettu 10 sekuntiin, olisi riski, ettei päällekkäisiä käyttötapauksia ajettaisi niin paljoa kuin olisi optimaalista. Tämän vuoksi kuormitustestaus päätettiin toteuttaa kyseisellä käyttäjämäärällä ja skriptillä. Optimoidun ja valmiin skriptin 10 minuutin testiajon tulokset on nähtävillä liitteessä 1.

6 Testaus ja konfigurointi

Edempänä käydään läpi kuormitustesteistä saadut tulokset ja analysoidaan ne. Tämän jälkeen kerrotaan AWS:n palvelujen konfiguroinnista ja niiden vaikutuksista kuluihin ja palvelun toimintaan.

6.1 AWS:n lähtökohdat

Aluksi tutkittiin AWS-palveluiden nykyistä kulurakennetta. AWS:n kulurakennemetriikoista oli mahdollista saada tietoon kyseisen palvelun kuukausittaiset kokonaiskulut kyseisen ympäristön osalta, mutta ei pelkästään tietyn sovelluksen osalta. ECS-tehtävistä aiheutuneet kulut perustuvat palvelun todelliseen käyttöön, joten resursseja uudelleenkonfiguroimalla ei olisi mahdollista luoda taloudellisia säästöjä. Tämän vuoksi ECS:n osalta keskityttiin tutkimaan, miten palvelun resursseja voitaisiin hyödyntää tehokkaammin. Tähän käytettiin apuna muun muassa AWS:stä saatavia metriikoita ECS-klustereiden muistinkäytön ja CPU:n käytön osalta (kuva 9).



Kuva 9. Testattavan sovelluksen metriikat ECS:n muistinkäytön ja CPU:n käytön osalta.

Edellä esitettyjen metriikoiden perusteella oli mahdollista päätellä, kuinka suuri osa laskentaresursseista olisi sen hetkisillä konfiguraatioilla käytössä. Kuvaajat kuvaavat ECS:n CPU:n (Central Processing Unit) keski- ja maksimiarvoja sekä muistinkäytön keskiarvoa. Kuvassa näkyvät kolme kuvaajan nousua selittyvät kyseisinä ajankohtina suoritetuilla lyhyehköillä testiajoilla, jotka aiheuttivat hetkittäisen muistinkäytön nousun.

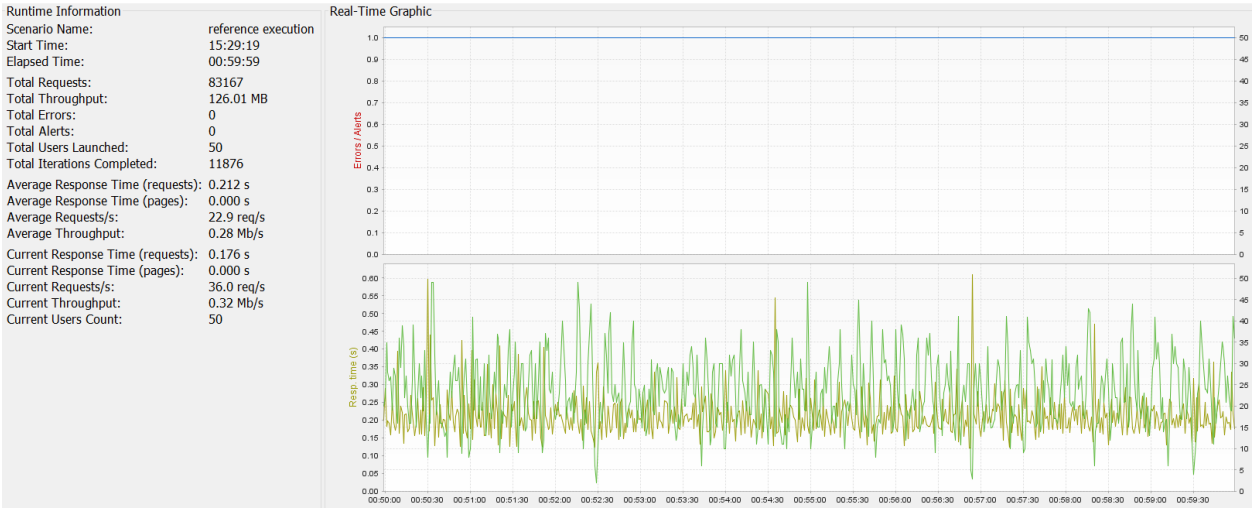
Tietokannan instanssien osalta kulujen arviointi oli hieman helpompaa. RDS-tietokantapalvelun kulut ovat instanssin kokoon perustuvia, eli laskutus menee suoraan kantainstanssien käytön mukaisesti. Tiedossa oli sovelluksen käyttämät instanssiresurssit

ennen ja jälkeen konfiguroinnin. Kun tiedossa oli laskutusmalli ja hinta, pystyi testitulosten perusteella arvioimaan konfiguroinnin tuoman säästön.

Työn alkaessa järjestelmätestausympäristön RDS:n kaksi instanssia olivat molemmat kokoa XL. Tuotannossa instanssien koot ovat tähän mennessä olleet 2XL. Kyseiset instanssit maksavat tällä hetkellä $1.16/instanssi/tunti [13]. Järjestelmätestausympäristössä kuorma on ollut niin vähäistä, ettei nähty mielekkääksi nostaa instansseja tuotannon tasolle edes kuormitustestauksen ajaksi, sillä nykyisen instanssin kapasiteetti oli riittävä testausta ajatellen. ECS:n suhteen järjestelmätestausympäristössä pyörii normaalisti kaksi tehtävää. Referenssiajon ajaksi ne nostettiin tuotantoa vastaavaan määrään, eli neljään tehtävään.

6.2 Referenssiajo ja testaussuunnitelma

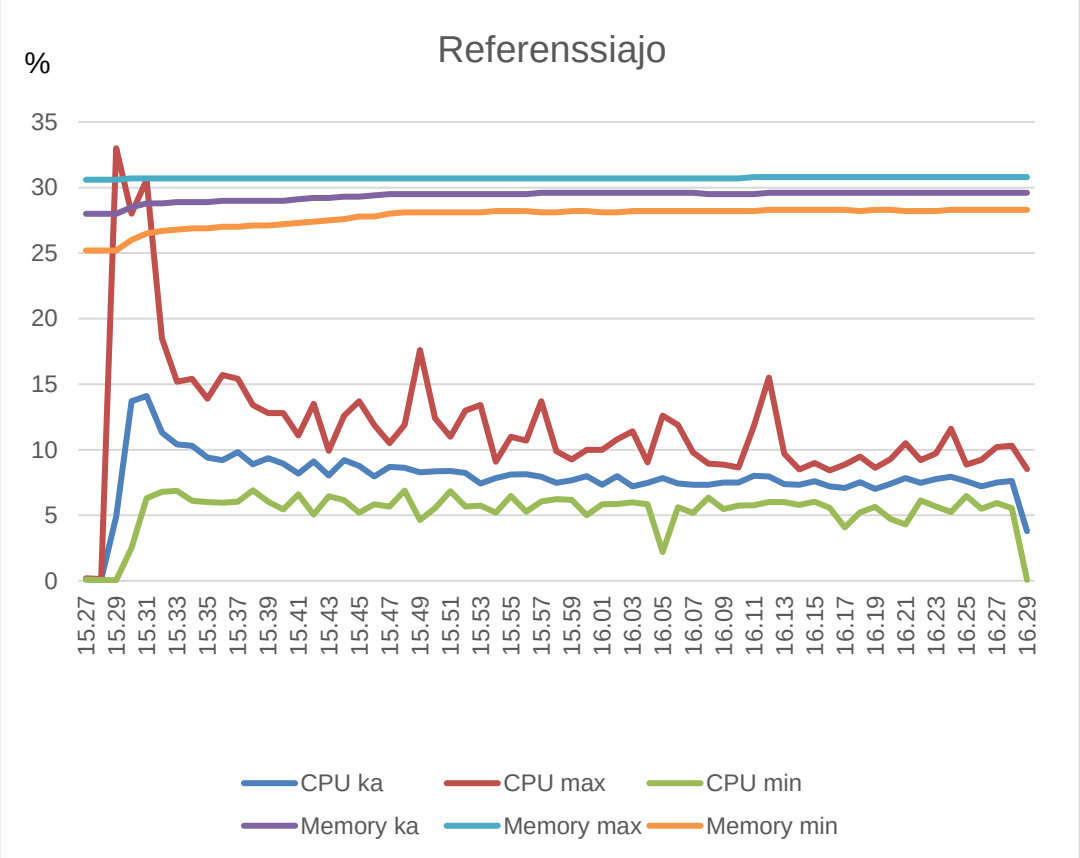
Ensimmäiseksi oli suoritettava referenssiajo. Ajo suoritettiin skriptille lasketulla optimikuormalla AWS konfiguroituna mahdollisimman tuotannonkaltaiseksi. Ensimmäinen referenssiajo suoritettiin suunnitellulla testiskriptillä (kuva 10).



Kuva 10. Referenssiajon tulokset.

Referenssiajo kesti tunnin ja sen aikana ehdittiin käydä käyttötapaus läpi lähes 11900 kertaa, kun kutsuja lähetettiin yli 83 000 kappaletta. Keskimääräinen vasteaika kutsuun oli 0,212 sekuntia. Testiskriptissä olevat viiveet eivät vaikuttaneet kutsujen vasteaikoihin.

Volyymi referenssiajossa oli tuotantoa huomattavasti korkeammalla. AWS:sta saatavien metriikoiden avulla voitiin arvioida, kuinka suuri osa laskentakapasiteetista oli ajon aikana käytössä (kuva 11).



Kuva 11. ECS:n metriikat referenssiajon osalta.

Kaavan pystyakselilla on käytetty kapasiteetti prosenteissa ja vaaka-akselilla kellonajat. Kaavasta voidaan tulkita, että vaikka kuormituksen ollessa kovempaa ja kapasiteetin pienempi kuin tuotannossa tällä hetkellä, käytetään silti vain huomattavan pientä osaa laskentakapasiteetista. Alun nousun jälkeen kaikkien testikäyttäjien ollessa käytössä ECS-klusterit käyttävät ainoastaan noin 15 % laskentakapasiteetista. RDS:n kohdalla instanssien tehon käyttöaste on vielä matalampi, noin 11 %.

Referenssiajon tulosten perusteella suunniteltiin testattavat konfiguraatioasetukset. Päätettiin kokeilla eri variaatioita ECS-tehtävien ja RDS:n instanssien koossa (taulukko 3).

Taulukko 3. Suunnitelma testattavista konfiguraatioasetuksista.

ECS-tehtävien määrä	RDC:n instanssikoko
4	**XL**
2	XL
1	XL
4	L
2	L
1	L
1-4 (automaattinen skaalaus)	*XL*
1-4 (automaattinen skaalaus)	*L*

Taulukossa esitetään millä instanssikoon ja tehtävien lukumäärän yhdistelmällä AWS konfiguroidaan ennen testiajoa. Instanssikoossa L tarkoittaa isoa ja XL erittäin isoa. Eroa näillä instansseilla on, että XL:n muistikapasiteetti on 30,5 GiB (gibibyte) ja CPU:ta 4 kun L-kokoisessa instanssissa taas vastaavasti muistia 15,25 GiB ja CPU:ta 2. XL:n kapasiteetti on siis kaksinkertainen verrattuna L-kokoisen kapasiteettiin [11]. Ylin lihavoitu rivi kertoo, millä määrittelyillä referenssiajo ajettiin. Kaksi alinta kursivoitua riviä ovat sellaisia, jotka ajateltiin ajaa, jos aikaa jää. Koska automaattisen skaalauksen konfiguroiminen on haastavampaa kuin staattisen, ne päätettiin jättää myöhempää testausta varten. Testisuunnitelmaa muutetaan tarpeen mukaan testien edetessä.

6.3 Testiajot ja tulosten analysointi

Seuraavaksi lähdettiin muokkaamaan AWS:n konfiguraatiota testisuunnitelmassa esitetyllä tavalla. Konfiguraation jälkeen ajettiin referenssiajoa vastaava testiajo, jonka jälkeen analysoitiin tulokset. Tulosten analysoinneissa tarkkailttiin kutsujen vasteaikaa sekä AWS:stä saatavia metriikoita. Metriikoiden avulla voidaan tulkita, kuinka suuri CPU:n käyttöaste on ECS:n ja RDS:n osalta. Jos vaikutti siltä, että laskentatehoa voi laskea joko RDS:n tai ECS:n osalta, tehtiin uusi konfiguraatio ja testiajo.

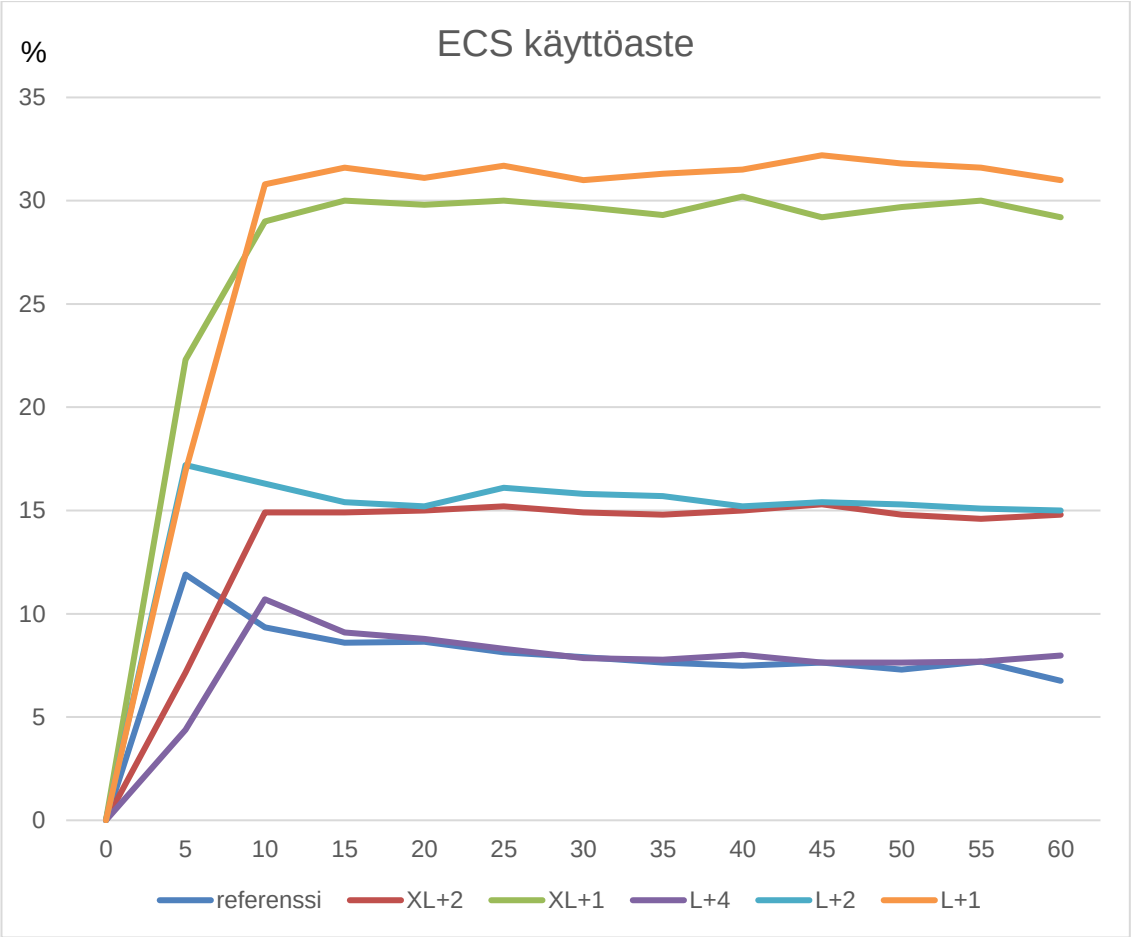
Testaus toteutettiin aiemmin esitellyn suunnitelman mukaisesti. Testiajot sujuivat ongelmitta ja kaikki konfiguraatiot saatiin testattua. Kutsujen vastausajoissa ei ollut huomattavia eroja eri konfiguraatioiden välillä (taulukko 4).

Taulukko 4. Vasteajat eri testiajojen mukaan eriteltynä.

vasteaika (s)	referenssi	XL+2	XL+1	L+4	L+2	L+1
istunnon luonti	0,687	0,740 (+0,053)	0,762 (+0,075)	0,606 (-0,081)	0,611 (-0,076)	0,682 (-0,005)
info	0,119	0,111 (-0,008)	0,108 (-0,011)	0,103 (-0,016)	0,105 (-0,014)	0,110 (-0,009)
viestit	0,156	0,157 (+0,001)	0,154 (-0,003)	0,143 (-0,013)	0,144 (-0,012)	0,161 (+0,005)
lukemattomat viestit	0,136	0,133 (-0,003)	0,130 (-0,006)	0,121 (-0,015)	0,122 (-0,014)	0,132 (-0,004)
käyttäjän tietojen haku	0,13	0,123 (-0,007)	0,119 (-0,011)	0,113 (-0,017)	0,114 (-0,016)	0,121 (-0,009)
ilmoitukset	0,128	0,121 (-0,007)	0,119 (-0,009)	0,113 (-0,015)	0,105 (-0,023)	0,11 (-0,018)
lopetus	0,125	0,116 (-0,009)	0,113 (-0,012)	0,108 (-0,017)	0,110 (-0,015)	0,115 (-0,010)

Taulukossa on esitelty eri kutsujen vasteaikojen keskiarvot eri testiajojen aikana. Sulkuihin on myös merkitty vasteajan muutos referenssiajon vasteaikoihin. Ylimmällä rivillä on esitetty eri testiajoissa käytetyt konfiguraatioyhdistelmät. XL tai L kuvaa RDS-instanssien kokoa ja sitä seuraava luku ECS-tehtävien lukumäärää. Kuten aiemmin todettiin, referenssiajo suoritettiin konfiguraatioyhdistelmällä RDS:n koko XL ja ECS-tehtävien määrä 4. Taulukosta on huomattavissa, että erot vasteajoissa eri konfiguraatioiden välillä olivat korkeintaan 0,085 sekuntia.

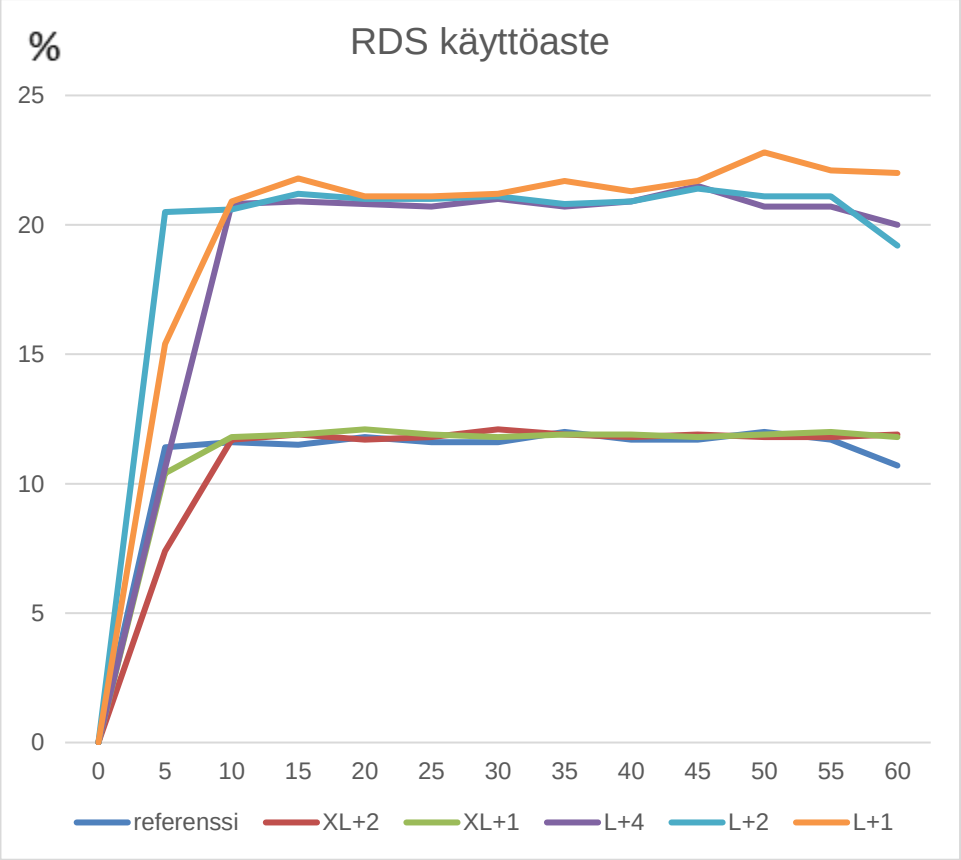
Vasteaikojen lisäksi tarkasteltiin ECS:n (kuva 12) ja RDS:n (kuva 13) laskentakapasiteetin käyttöastetta.



Kuva 11. ECS:n laskentakapasiteetin käyttöaste testiajojen aikana.

Kaavan pystyakselilla on laskentakapasiteetin käyttöaste prosentteina. Vaaka-akselilla on testiajon eteneminen minuuteissa. Kaavassa on esitetty, kuinka monta prosenttia laskentakapasiteetista on käytössä testiajojen aikana. Tällä hetkellä tuotannon tehtävien määrä on neljä kappaletta. Kuvaajasta on tulkittavissa, kuinka ECS-tehtävien määrän laskeminen neljästä yhteen nostaisi laskentakapasiteetin käyttöasteen noin 8 %:sta noin 30 %:iin. Yhdellä tehtävälläkin laskentakapasiteetista käytetään ainoastaan kolmannes. Kuitenkin voidaan ajatella, että palvelun toiminnan takaamiseksi kaksi tehtävää on minimi. Näin taataan saavutettavuus, jos toiselle tehtävälle kävisi jotain. Tultiin siis lopputulokseen, että vaikka yhdellä tehtävällä pärjäisi laskentakapasiteetin puolesta, ei se olisi palvelun laadun kannalta järkevää.

ECS-hinnoittelu on määritelty kahden eri osan perusteella, CPU per tunti ja GB (gigabyte) per tunti. Molemmat määräytyvät käytön mukaan, joten se ei muutu, vaikka konfiguraatiota muuttaisi. Tämän perusteella voitaneen todeta, ettei konfiguraatiomuutoksilla ole merkitystä palvelun kustannuksiin. Kuitenkin tehtävien määrää laskemalla saataisiin käytössä olevia laskentaresursseja hyödynnettyä tehokkaammin. Kuvaajasta on nähtävillä, että tehtävien määrän puolittaminen kaksinkertaisti laskentaresurssien käyttöasteen.



Kuva 12. RDS:n laskentakapasiteetin käyttöaste testiajojen aikana.

Kuvassa 13 on taas esitetty tietokantojen instanssien laskentakapasiteetin käyttöaste testien aikana. Jälleen pystyakselilla on laskentakapasiteetin käyttöaste prosentteina, vaaka-akselilla on testiajon eteneminen minuuteissa. Testeissä käytettiin XL- ja L-kokoisia instansseja. Tuotannon instanssikoko on 2XL. Kaaviossa on huomattavissa, että testikonfiguraatioista isommalla XL-koolla laskentakapasiteetin käyttöaste on noin 12 %.

Pienemmällä L-koon instanssilla käyttöaste nousee hieman yli 20 %:n. Kuitenkin pienemmälläkin instanssikoolla kuormitustestejä tehdessä laskentakapasiteetista oli käytössä vain neljännes.

Tällä hetkellä tuotannossa olevat 2XL-kokoiset instanssit maksavat $1.16/instanssi/tunti. Vastaava kustannus L-kokoisille instansseille olisi $0,29/instanssi/tunti [13]. Jos siis instanssikoko laskettaisiin L-kokoiseen, säästettäisiin instanssien kustannuksissa arviolta noin 75 %.

7 Yhteenveto

Insinöörityön tavoitteena oli luoda kuormitustestit sovellukselle ja niiden tulosten mukaan löytää konfiguraatio, jolla AWS-palveluita saataisiin kustannustehokkaammaksi. Tavoitteet saavutettiin. Kuormitustestit luotiin ja ajettiin sekä konfiguraatioyhdistelmät testattiin. Testien avulla löydettiin optimaalinen konfiguraatioyhdistelmä, mutta varsinaisia konfiguroinnteja ei viety tuotantoon asti.

Haasteita työn tekemiseen toivat uusi testaustyökalu, jolla skriptien luonti vei odotettua enemmän aikaa. Myös maailmanlaajuisesta koronapandemiasta aiheutuneet poikkeusolosuhteet vaikuttivat työn tekemiseen. Koska lähikontakteja tuli välttää, eikä samassa tilassa saanut olla yli 10 ihmistä samaan aikaan yritykset, mukaan lukien Aktia, määräsivät työntekijänsä työskentelemään täysin etänä. Ohjeistuksen tullessa voimaan työstä oli vielä valtaosa tekemättä, joten skriptien kirjoitus ja testaus tuli tehdä kotoa käsin VPN:n (Virtual Private Network avulla. Tämä myös hidasti huomattavasti skriptien ajamista. Yhden kolmen minuutin testiajon suorittamiseen ja testin tulosraportin kokoaminen saattoi kestää yli puoli tuntia. Koska testiskriptin toimivuuden testaaminen oli äärimmäisen hidasta, oli myös skriptin kehittäminen ja optimoiminen aikaa vievää. Tämän vuoksi varsinaiset testiajot päästiin aloittamaan suunniteltua myöhemmin, joten konfigurointien optimoimisenkin aloitus viivästyi.

Referenssiajo ja siitä saadut metriikat osoittivat, että konfigurointeja on mahdollista tehdä ja laskentatehoa laskea. Referenssiajossa huomattiin, että resurssien todellinen käyttöaste on tietokannan osalta korkeintaan 15 % ja taustajärjestelmän osalta korkeintaan 30 %. Varsinaisten testien jälkeen huomattiin, että palvelu toimii hyvin pienemmilläkin resursseilla ja vasteajat pysyvät kohtuullisina. Tultiin siihen tulokseen, että laskentakapasiteettia pystyy molempien palveluiden kohdalla laskemaan. Kuitenkin ECS:n osalta kustannukset määräytyvät käytön mukaisesti, jolloin konfiguraatiomuutokset eivät vaikuta kustannuksiin, mutta mahdollistaisi kuitenkin resurssien tehokkaamman käytön. RDS:n kustannukset taasen määräytyvät instanssikoon käytön mukaisesti, ei todellisen käyttöasteen. Tämän uudelleenkonfiguroinnin tuomat säästöt olisivat arvioilta 75 % luokkaa RDS:n osalta.

Tällä hetkellä tuotannossa RDS:n instanssikokoa on laskettu alkuperäisestä 2XL:n instanssikoosta kuormitustestauksessakin käytettyyn XL-kokoon. Testien jälkeen voidaan todeta, että tuotannon instanssikokoa voisi edelleen laskea turvallisesti vieläkin pienempään L-kokoon. ECS:n osalta tuotantoon ei vielä ole tehty muutoksia.

Kuormitustestaus toimi hyvin laskentaresurssien optimaalisen tason arvioinnissa. Testit oli kohtuullisella vaivalla muokattavissa vastaamaan tuotannossa esiintyvää kuormaa. Testiajoista oli saatavilla selkeät raportit, joista oli helppo tarkastaa uudelleenkonfiguroinnin vaikutus vasteaikoihin. Lisäksi AWS:stä saatavat selkeät metriikat tekivät resurssien käyttöasteen arvioinnista vaivatonta. Kuormitustestauksen selkeitä etuja ovat helppous testien toistettavuudessa ja testiskriptien muokkauksessa. Yhteenvetona voidaan siis todeta, että kuormitustestaus soveltuu hyvin resurssien käyttöasteen ja kustannusten arviointiin.

Jatkoa ajatellen oli hyvä, että testiskriptit saatiin tehtyä toimiviksi. Testattavalle mobiilisovellukselle ei työn aloittamishetkellä ollut säännöllisesti ajettavia kuormitustestejä, joten oli positiivista, että jotain testejä saatiin tehtyä. Tulevaisuudessa käyttötapauksia voisi luoda enemmän ja näin muokata kuormitustestipaketista monipuolisempaa. Kuormitustestejä olisi jatkossa myös hyvä ajaa, vaikka konfiguraatio olisi saatu jo suoritettua.

Tulevaisuudessa suunnitellaan myös tuotannon laskentaresurssien konfigurointia. Kun uusi konfiguraatio on todettu järjestelmätestausympäristössä toimivaksi eikä mobiilisovelluksessa ole merkkejä sen hidastumisesta, voidaan alkaa suunnitella tuotannon konfiguraatiota.

Lisäksi insinöörityön tekeminen on tuonut tekijälleen uutta osaamista ja syventänyt sitä tietotaitoa, mikä on jo testauksen suhteen olemassa. Kuormitustestaus ja testiskriptien tekeminen on oma maailmansa, joka poikkeaa huomattavasti kehityksen aikaisesta testauksesta. Myös uuden ohjelmiston opettelu on tuonut uusia haasteita ja sitä myötä oppeja tekijälleen. Tämän lisäksi myös osaaminen pilvilaskenta-alustojen käytöstä, toiminnasta ja luomista mahdollisuuksista on kasvanut. Pilvilaskenta-alustojen käyttö on yleistymässä koko ajan, ja osaamista niiden suhteen arvostetaan työmarkkinoilla. Lisäksi sovelluksen muu testaaminen tulee jatkossa olemaan helpompaa, kun ymmärtää enemmän, miten palvelu itsessään toimii.

Lähteet

1 Aktia, ”Tietoa Aktiasta”, verkkolähde, <https://www.aktia.fi/fi/tietoa-aktiasta> Luettu 18.2.2020.

2 Markus Lehtiniitty, ”Aktia toi tarjolle täysin digitaalisen luottokortin”, verkkolähde <https://mobiili.fi/2019/01/30/aktia-toi-tarjolle-taysin-digitaalisen-luottokortin-tavallista-turvallisempi-lahimaksut-android-puhelimella-ilman-ylarajaa/> Luettu 18.2.2020.

3 Aktian yhtiökokouksen pöytäkirja, 2019, verkkolähde <https://www.aktia.com/documents/10560/887037/yhtiökokous_2019_esitys.pdf/7da4359a-2b70-43af-adb4-2440ade827bc> Luettu 18.2.2020.

4 Harva marketing, ”Sivunopeus tuo näkyvyyttä ja isää asiakassidonnaisuutta”, verkkolähde <https://www.harvamarketing.fi/blog/sivunopeus-tuo-nakyvyytta-ja-lisaa-asiakassidonnaisuutta> Luettu 27.2.2020.

5 Sogeti, ”Suorituskyky ja kuormitustestaus”, verkkolähde <https://www.sogeti.fi/palvelumme/testauspalvelumme/suorituskyky--ja-kuormitustestaus/> Luettu 27.2.2020.

6 ConformiQ, Kuormitustestaus, verkkolähde <http://users.jyu.fi/~kolli/testaus2006/materiaali/6.3_Kuormitustestaus_v1.pdf> Luettu 27.2.2020.

7 Jay Chapel, ”AWS vs Alibaba Cloud Pricing: A Comparison of Compute Options”, verkkolähde <https://medium.com/@jaychapel/aws-vs-alibaba-cloud-pricing-a-comparison-of-compute-options-c7ea928fd9c9>, luettu 11.5.2020.

8 Amazon Web Services, “About AWS”, verkkolähde <https://aws.amazon.com/about-aws/> Luettu 9.3.2020.

9 E2 Software, ”Splunk”, verkkolähde <https://e2-software.com/splunk/> Luettu 25.2.2020.

10 Thu Ngyen, ”5 Splunk alternatives”, verkkolähde <https://logdna.com/5-splunk-alternatives-for-logging-their-benefits-shortcomings-and-which-one-to-choose/>, luettu 11.5.2020.

11 Neotys, ”Neoload”, verkkolähde <https://www.neotys.com/neoload/overview> Luettu 25.2.2020.

12 AppStore, ”Aktia Wallet”, verkkolähde <https://apps.apple.com/fi/app/aktia-wallet/id1154413529?l=fi> Luettu 25.2.2020.

13 Amazon Web Services “AWS Pricing”, verkkolähde <https://aws.amazon.com/rds/instance-types/> Luettu 8.5.2020.

Valmiin skriptin testitulokset