OAMK
OULUN AMMATTIKORKEAKOULU

Pipsa Korkiakoski

WEBSOCKET API:LTA SAAPUVAN DATAN KERÄYS AUTOMATISOIDUILLE TESTEILLE DOCKER-KONTISSA

WEBSOCKET API:LTA SAAPUVAN DATAN KERÄYS AUTOMATISOIDUILLE TESTEILLE DOCKER-KONTISSA

Pipsa Korkiakoski
Opinnäytetyö
Kevät 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä(t): Pipsa Korkiakoski
Opinnäytetyön nimi suomeksi: WebSocket API:lta saapuvan datan keräys automatisoiduille testeille Docker-kontissa
Opinnäytetyön nimi englanniksi: Collection of data from the WebSocket API for automated tests in the Docker container
Työn ohjaaja(t): Teemu Korpela
Työn valmistumislukukausi ja -vuosi: kevät, 2020
Sivumäärä: 35

Opinnäytetyön toimeksiantajana toimi OP-Palveluiden Pivo-mobiilisovellusprojekti. Mobiilisovelluksen taustajärjestelmässä käytetään REST API:a ja WebSocket API:a, joista REST-rajapinnalle oli jo aiemmin kehitetty automatisoidut yksikkötestit Ruby-ohjelmointikielellä Minitest-sovelluskehystä hyödyntäen, mutta WebSocket API:lta saapuvan datan vastaanottaminen ei ole ollut vielä mahdollista olemassa olevin puittein.

Opinnäytetyön lähtökohtana oli automatisoitujen testien kattavuuden parantaminen. Työssä kehitettiin ohjelma, joka kerää WebSocket API:lta saapuvaa dataa ja välittää datan olemassa oleville automaatiotesteille testeistä pyydettäessä.

Lopputuloksena saatiin kehitettyä Python-ohjelmointikielellä Docker-kontissa ajettava WebSocket API:lta dataa kerräävä ohjelma. Toimiessaan ohjelma välittää kerätyn datan rajapinnalle, josta automaatiotesteissä on REST-protokollaa hyödyntäen mahdollista hakea data testeihin käsiteltäväksi.

Asiasanat: testaus, yksikkötestaus, testausautomaatio, Docker-kontti, rajapinta

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Pipsa Korkiakoski
Title of thesis: Collection of data from the WebSocket API for automated tests in the Docker container
Supervisor(s): Teemu Korpela
Term and year when the thesis was submitted: spring, 2020
Pages: 35

The thesis was commissioned by OP-Palvelut. The project that the thesis was related was Pivo project. Pivo is a mobile application developed for mobile payments, which uses the REST API and the WebSocket API in the background system. Automated unit tests have already been developed for the REST interface, but it has not yet been possible to receive data from the WebSocket API under the current framework.

Existing unit tests have been developed using the Ruby programming language and the Minitest framework. The aim of the thesis was to develop a program that collects data from the WebSocket API and forwards it to an interface from which automated unit tests can retrieve data using the REST protocol. The result of the thesis was developed the program made using the Python programming language, which was possible to run in a Docker container.

Keywords: testing, unit testing, test automation, Docker container, interface

SISÄLLYS

TIIVISTELMÄ 3
ABSTRACT 4
SISÄLLYS 5
SANASTO 6
1 JOHDANTO 7
1.1 Lähtökohdat 8
1.2 Päämäärä 8
2 DEVOPSIN VAIKUTUKSET OP KETTERÄÄN 9
3 KONTTITEKNIIKKA 11
3.1 Kontti 11
3.2 Docker-tiedosto 12
3.3 Kontin tietoturva 14
4 AMAZON WEB SERVICES 16
4.1 Amazonin konttien hallintapalvelut 16
4.2 WebSocket API 17
5 WEBSOCKET API:N DATAN KERÄYS 18
5.1 WebSocket API:n kuuntelun säie 19
5.2 Dataa varten rakennettu rajapinta 20
5.3 Mywebsocket-luokan toiminta 22
5.4 Oman dataa välittävän rajapinnan käyttö automaatiotesteissä 23
6 OHJELMAN JA TESTIEN KONTITTAMINEN 25
6.1 Näköistiedostojen luominen 25
6.2 Monisäiliöisten palveluiden käynnistäminen rinnakkain 27
6.3 Kontitettujen testien ajaminen Jenkins-automaatiopalvelimella 28
7 YHTEENVETO 30
LÄHTEET 32

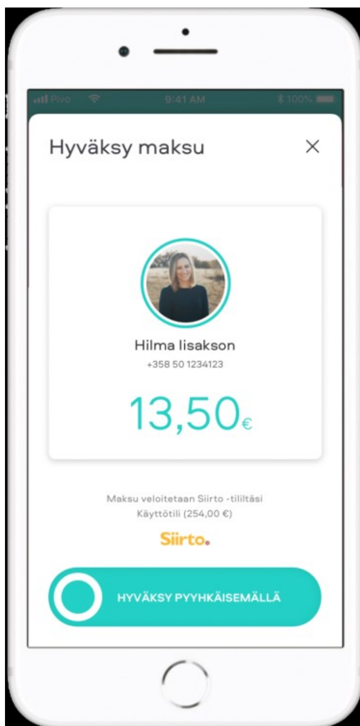
SANASTO

API	Application Programming Interface. Ohjelmistokehityksessä käytetty rajapinta.
AWS	Amazon Web Services. Amazonin kokoelma etätietojen käsittelyresurssien palveluita.
EC2	Amazon Elastic Compute Cloud. Pilvipalveluinfrastruktuuri.
REST	Representational State Transfer. Ohjelmointirajapintojen toteuttamiseen arkkitehtuurimalli.
S3	Amazon Simple Storage Service. Pilvitallennusalusta.

1 JOHDANTO

Aloitin OP-Palveluilla työskentelyn Pivo-mobiilisovellusprojektissa testaajana keväällä 2019. OP Ryhmällä on oma kehitys- ja teknologiayksikkönsä ja Pivo on OP:n yksi kehitys- ja teknologiayksikön mobiilimaksamiseen kehittämistä sovelluksista, jonka tekeminen aloitettiin vuonna 2012 (1). Pivoa voivat käyttää kaikkien suomalaisten pankkien asiakkaat. Pivolla voi siirtää rahaa kavereille, maksaa verkkokauppaostokset tai sitä voi käyttää maksuvälineenä kivijalkamyymälöissä. (2.) Kuvasta 1 saa käsityksen Pivon brändistä ja siinä on nähtävillä hyvin tyypillinen näkymä, kun kaverimaksua tai verkkokauppaostossa ollaan hyväksymässä.

Työtehtävieni päämääränä Pivo-projektissa on ollut erityisesti ylläpitää ja kehittää automatisoituja REST-rajapintoja kutsuvia yksikkötestejä. Pivo-sovelluksen taustajärjestelmän REST API on tehty Amazon Web Services -pilvialustan API Gateway REST API:lla ja päätepisteitä testaavat testit on alun perin alettu kehittämään Ruby-ohjelmointikielellä käyttäen Minitest-sovelluskehystä. Minitest-sovelluskehys tarjoaa hyödyllisiä Ruby-ohjelmointikielen kirjastoja yksikkötestaukseen (3).



KUVA 1. Maksun hyväksynnän näkymä (4)

1.1 Lähtökohdat

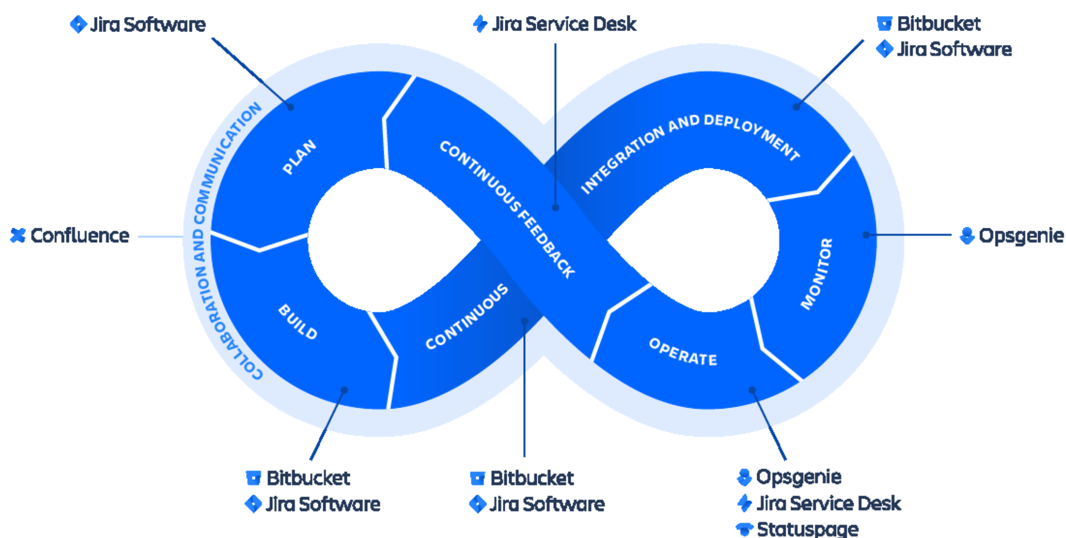
REST API:n lisäksi Pivon sovelluskehityksessä on otettu hiljattain käyttöön HTTP-pohjainen WebSocket-sovellusliittymä. WebSocket-rajapinnan testaamisen lisääminen automaatiotesteihin nykyisillä puitteilla osoittautui mahdottomaksi. Ruby-ohjelmointikielessä Minitest-sovelluskehystä käytettäessä suoritettavia säikeitä on mahdollista ajaa vain yksi kerrallaan (5). WebSocket-rajapinnasta saapuvan datan kuuntelu vaatii kuitenkin ajon taustalla omassa säikeessään, koska datan saapumista palvelimelta ei juurikaan voi ennakoida.

1.2 Päämäärä

Opinnäytetyön aihe syntyi ongelmasta automatisoida WebSocket-rajapintaa käyttävät testitapaukset nykyisten puitteiden avulla. Ongelman ratkaisemiseksi opinnäytetyön päämääränä oli kehittää taustalla ajettava sovellus, joka pystyy keräämään WebSocket-rajapinnasta saapuvaa dataa ja palauttaa automaatiotesteille vastauksena kutsuttaessa. Kriteeriksi taustalla ajettavan sovelluksen kehittämiselle määriteltiin otettavan huomioon sovelluksen ajettavuus Docker-kontissa. Lisäksi toivottiin, että sovelluskehityksen aikana otetaan huomioon olemassa olevien automaatiotestin vieminen konttiin ja näiden konttien mahdollisuus olla vuorovaikutuksissa toisiinsa.

2 DEVOPSIN VAIKUTUKSET OP KETTERÄÄN

Atlassian-organisaation sivuilla kerrotaan DevOpsin historian alkaneen vuoden 2007 jälkeen, kun it-alan toimintojen ylläpitäjät ja ohjelmistojen kehittäjäyhteisöt aloittivat toimintamallien parantamisesta keskustelun foorumeilla ja sovituissa tapaamisissa. Tuloksena syntyi teema, joka on nykyään tärkeä osa monessa ohjelmistokehityksessä (6). Atlassian-kotisivulta löytyneessä kuvassa (kuva 2) on DevOpsista tutun logon avulla hyvin kuvastettu jatkuvan kehittämisen ja toimituksen vaiheet. Kuvassa näkyy mainintoja Atlassianin omien työkaluista ja palveluista, joista JIRA Software ja Confluence ovatkin myös OP:llä käytössä projektinhallinnan ja dokumentaation hallinnan työkaluina.



KUVA 2. Atlassianin havainnollistama kuva ohjelmistokehityksen toimitusketjun ketteryydestä (6)

DevOps-ajattelutapa sai OP Ryhmässä aikaan mittavan toimintatapojen ja yrityskulttuurin muutoksen vuoden 2019 alussa ja muutosmatka sai organisaatiossa nimekseen OP Ketterä (7). Tavoitteena muutoksella on parantaa toiminnan tehokkuutta ja asiakaskokemusta (8). OP:lle DevOps tarkoittaa rutiininomaisten virheherkkien manuaalisten toistojen poistamista ja työn sujuvoittamista rakenta-

malla ohjelmistokehitykseen tuotteistettuja putkia, jotka mahdollistavat kehityksen jatkuvan integroinnin ja jatkuvan toimituksen. Ketterien menetelmien vaikutus näkyy loppuasiakkaille tuotannon päivitysten julkaisuiden nopeutumisena. (7.)

DevOps-toimintamalli tukee ketterää ohjelmistokehitystä, jossa pyritään automatisoimaan ohjelmistokehityksen palvelutoiminnot, kuten ohjelmistokehitys, testaus ja ylläpito. Ketterässä ohjelmistokehityksessä pyritään jatkuvaan integraatioon (continuous integration, CI) ja jatkuvaan toimitukseen (continuous deployment, CD). (9.) DevOps tulee sanoista Development eli kehitys ja Operations eli niin kutsutut operatiiviset toiminnot ja ylläpito (10).

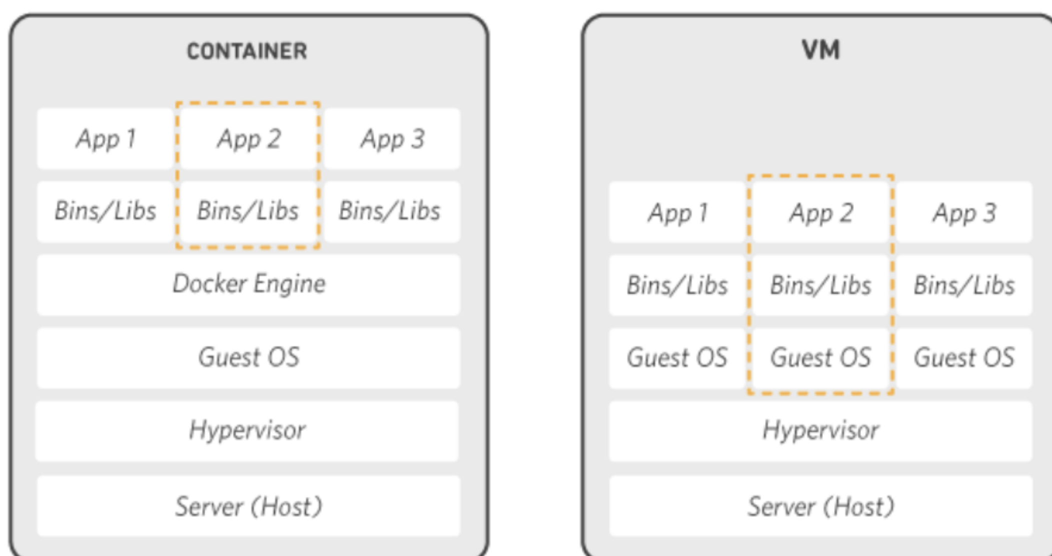
DevOps määrittelee OP:llä korkean automaatioasteen ja näkyy myös muutoksina työntekijäkokemuksien ja työn tekemisen kehittämisessä. DevOps vaatii sitoutumista koko organisaatiolta ja tuo työn tekemiseen läpinäkyvyyttä. (7.) Läpinäkyvyys mahdollistaa liiketoiminnan vaatimusten näkyvyyden kehittäjille ja kehittäjien on mahdollista avata ja perustella tekemistään liiketoiminnalle. Työtapojen parantelu, työntekijän osaamisen kehittämisen laajentaminen ja asioiden tekeminen viisaammin ja ketterämmin kuuluvat DevOps-ajatteluun. (11.)

3 KONTTITEKNIIKKA

Konttitekniikasta ja Docker-alustasta kerrottaessa elementtejä verrataan usein käyttäen esimerkkejä rahtilogistiikasta. Kontti on kuin rahtikontti ja alusta kuin kontteja kuljettava laiva. Rahtikonttien keksiminen vuonna 1950 ja niiden ottaminen käyttöön nopeutti ja helpotti tavaran kuljetusta ja siirtelyä, kun kuormaa ei enää tarvinnut välillä purkaa rekasta pois ja lastata uudelleen seuraavaan rekkaan ja niin edelleen vaan tavara oli mahdollista pakata kerran konttiin ja konttia oli sen jälkeen helppo siirtää vaikkapa laivasta rekkaan. Konttitekniikka säästää rahaa, vaivaa ja aikaa. Sama ilmiö allekirjoitetaan myös ohjelmistokehityksessä puhuttaessa konttitekniikasta. (12.)

3.1 Kontti

Kontti mahdollistaa sovelluksien paketoinnin ja ajon. Kontteja ajetaan yhteisen alustan päällä ja tyypillinen alustan valinta konteille on Docker-ohjelmisto (13). Kuvassa 3 on vertailtu virtuaalikoneen ja kontin ero. Alustan lisäksi konttien käsittelyyn tarvitaan vielä hallintaohjelmisto. Niistä tyypillisesti käytettyjä ovat esimerkiksi Googlen Kubernetes, kotimainen Kontena sekä Amazonin AWS ECS. (13.) Pivo-projektissa käytetään viimeisimmäksi mainittua.



KUVA 3. Kontin ja virtuaalikoneen erot (15)

Konttitekniikan avulla on mahdollista ajaa rinnakkain sovelluksia yhdessä virtualikoneessa eristettyinä muista konteista ja sovelluksista. Sovelluskehityksen ja tuotannon yhteistoiminta nopeutuu sekä jatkuva julkaisu on mahdollista konttitekniikan avulla. Myös testattavuus paranevat, kun ohjelmistoversion julkaisun päivittäminen eri ympäristöihin ja takaisin onnistuu kehittäjältä sekunneissa. Skaalautuvuus ja suorituskyky paranee, kun sovellus voidaan pilkkoa yksittäin ylläpidettäviin mikropalveluihin ja sitä kautta jakaa palvelut useampaan eri konttiin. (13.)

Pilkkominen mikropalveluihin mahdollistaa päivittämisen pala kerrallaan ja ongelmatilanteissa helpottaa paluuta vanhaan versioon eli tekemään niin kutsun takaisin vedon (roll-back) (12). Docker-kontti tehostaa resurssien käyttöä ja vaatii vähemmän tilaa palvelinvirtualisointiin verrattuna, koska Docker-kontti tarvitsee vain suoritettavan ohjelman komponentit ja se hyödyntää alustansa käyttöjärjestelmää yhdessä muiden konttien kanssa (14).

3.2 Docker-tiedosto

Docker-tiedosto on tekstitiedosto, jota kutsutaan nimellä *Dockerfile.* Docker-tiedosto mahdollistaa kontin näköistiedoston suorittamisen ja sisältää Docker-näköistiedoston rakentamisen käskyt avainsanoin vaihe vaiheelta (16). Kuvassa 4 näkyy yksinkertainen esimerkki Docker-tiedostosta. Docker-moottorille annetut komennot on kirjoitettu isoilla kirjaimella ja ne suoritetaan konttia rakennettaessa järjestyksessä rivi riviltä.

Dockerfile

```
1  FROM python:2.7-alpine
2  COPY . /app
3  WORKDIR /app
4  RUN pip install -r requirements.txt
5  RUN pip install flask
6
7  RUN pip install websocket-client
8  ENTRYPOINT ["python"]
9  CMD ["app.py"]
10
```

KUVA 4. Dockerfile-esimerkki

Docker-tiedosto aloitetaan FROM-komennolla, jolla määritellään näköistiedoston pohjana käytettävä näköistiedosto. Kuvassa 4 nähdään, että näköistiedoston pohjaksi on määritelty Alpine Linux -projektiin pohjautuvaa näköistiedosto, jossa Python-ohjelmointikielen kääntäjän kirjastojen versio on 2.7. Alpine-pohjaisen näköistiedoston käyttämistä suositellaan, kun halutaan rakentaa mahdollisimman kevyt näköistiedosto (17).

Kuvassa 4 näkyvän Docker-tiedoston seuraava komento COPY kopioi rakennettavan kansion kansioon nimeltä app ja asettaa sen työympäristöksi komennolla WORKDIR. Kuvassa komennettujen RUN-käskyjen avulla asennetaan PIP-työkalulla requirements-tiedostossa määritellyt kirjastot sekä seuraavilla riveillä yksittäin flask- ja websocket-client-nimiset kirjastot.

Kuvassa 4 ENTRYPOINT-osioon on kirjattu ”python”, joka määrittelee kontin sovelluksen oletustilaksi python-ohjelmointikielen kääntäjän, jonka jälkeen CMD-komennossa voi määritellä pythonilla käännettävän tiedoston nimen.

Docker-näköistiedosto rakennetaan komennolla ”docker build” ja se luo näköistiedoston Docker-tiedostosta ja kontekstista (16). Kuvassa 5 näkyy näköistiedoston rakentamisen viimeiset vaiheet, kun Docker-ohjelmalle on annettu build-komento ja se on suorittanut Docker-tiedoston komennot vaihe vaiheelta. Viimeisillä rivillä kerrotaan, että näköistiedosto on valmis id:llä ”cfdec7eb8f2c”. Kontin voi näköistiedoston rakentamisen jälkeen käynnistää komennolla ”docker run <image_id>”. Komento käynnistää kontin, jossa näköistiedostoa ajetaan.

```
 ---> d4e3993210db
Step 6/8 : RUN pip install websocket-client
 ---> Running in a443ba663d96
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020.
hon as Python 2.7 is no longer maintained. A future version of pip will d
.7. More details about Python 2 support in pip, can be found at https://p
elopment/release-process/#python-2-support
Collecting websocket-client
  Downloading websocket_client-0.57.0-py2.py3-none-any.whl (200 kB)
Requirement already satisfied: six in /usr/local/lib/python2.7/site-packa
nt) (1.14.0)
Installing collected packages: websocket-client
Successfully installed websocket-client-0.57.0
Removing intermediate container a443ba663d96
 ---> fb0783beb5fe
Step 7/8 : ENTRYPOINT ["python"]
 ---> Running in 9c73210e14fc
Removing intermediate container 9c73210e14fc
 ---> 36d43ec3d4c9
Step 8/8 : CMD ["app.py"]
 ---> Running in 98f333cf43b9
Removing intermediate container 98f333cf43b9
 ---> cfdec7eb8f2c
Successfully built cfdec7eb8f2c
```

KUVA 5. Kuvankaappaus kontin luonnista Docker-tiedoston mukaisesti

3.3 Kontin tietoturva

Konttiteknologian kontit toimivat erillään toisistaan. Teknologia tarjoaa siten hyvin eristetyn ympäristön ohjelmistojen toimittamiselle. Eristäytyminen vähentää konttien haavoittuvuutta, jonka vuoksi tekniikkaa voidaan pitää jopa turvallisempana vaihtoehtona verrattuna perinteisempiin ratkaisuihin. Vaikka kontit toimivat erillään toisistaan, niillä on yhteinen käyttöjärjestelmä, joka haavoittuessaan voi vaarantaa kaikki sen päällä toimivat kontit. (18.)

Ensimmäisenä konttia rakentaessa tietoturvaan voi kiinnittää huomiota valittaessa näköistiedostona käytettävää pohjatiedostoa. Opinnäytetyön projektissa pohjaksi on määritelty käytettävän Alpine -pohjaista näköistiedosta, koska se on mahdollisimman riisuttu, jolloin se tuo mahdollisimman vähän tietoturvaa heikentäviä asioita mukanaan rakennettavaan kontin näköistiedostoon (19).

Haavoittuneiden kirjastojen käyttäminen kontissa voi mahdollistaa myös hyökkääjän pääsyn kirjaston kautta aina konttien isäntään asti, joka altistaa näin ollen myös muut kontit hyökkäykselle (18). Digiarjessa-blogissa Tero Niemistö on listannut tietoturvaa parantavia asioita. Turvallisuuteen voi vaikuttaa asettamalla

kontille ”vain luku”-tilan konttia käynnistäessä. Turvallisuutta voi lisätä myös määrittelemällä erillisen verkon konttien välille ja harkitsemalla konttien välisen verkkoliikenteen rajoittamista. Lisäksi konttien käyttöoikeuksia voi rajoittaa käyttäjätunnuksilla sekä sallituilla käyttäjäryhmillä. (19.)

4 AMAZON WEB SERVICES

Opinnäytetyön päämääränä oli kehittää dataa keräävä sovellus Docker-kontissa suoritettavaksi palveluksi. Lisäksi olemassa olevat automatisoitavat taustajärjestelmää testaavat testit haluttiin saattaa kontissa ajettavaksi palveluksi. Opinnäytetyön aikana kehitystyö tehtiin paikallisesti Docker Desktop -ohjelman avulla omalla tietokoneella. Olennaista automatisoinnin kannalta on kuitenkin se, että molemmat palvelut ovat valmistuessaan mahdollista siirtää pilvipalveluiden ympäristöön. Tämä on mahdollista Pivo-projektissa käytössä olevien Amazon Web Services -palveluita hyödyntäen.

Amazon Web Services eli tutummin AWS on skaalautuva pilvilaskenta-alusta, joka tarjoaa erilaisia työvälineitä ja palveluita verkkopalvelujen ja pilviympäristöjen rakentamiseen (20). Palveluiden ja resurssien saatavuus riippuu tilille määritellystä laskenta-alustan alueesta. Tietty palvelu tietyllä alueella ei välttämättä ole käytettävissä toisella alueella. Alueet on jaettu sijainnin mukaan maantieteellisesti ja resurssit ovat täysin erillään toisistaan. (21.) Amazonin IAM-palvelu on yksi poikkeuksista ja on käytössä yhtenäisesti alueesta riippumatta. IAM-palvelulla eli identiteetin ja käyttäjien oikeuksien hallintapalvelulla voidaan määritellä, kenellä on pääsy mihinkin AWS-tilin resursseihin ja oikeus hallita milläkin tavalla mitäkin AWS-palveluita ja niiden sisältöä. (22) Määrittelemällä käyttäjäryhmiä ja rooleja voidaan helposti määritellä yhteisiä oikeuksia tietyille käyttäjäryhmille tiettyihin palvelun tasoihin (23; 24).

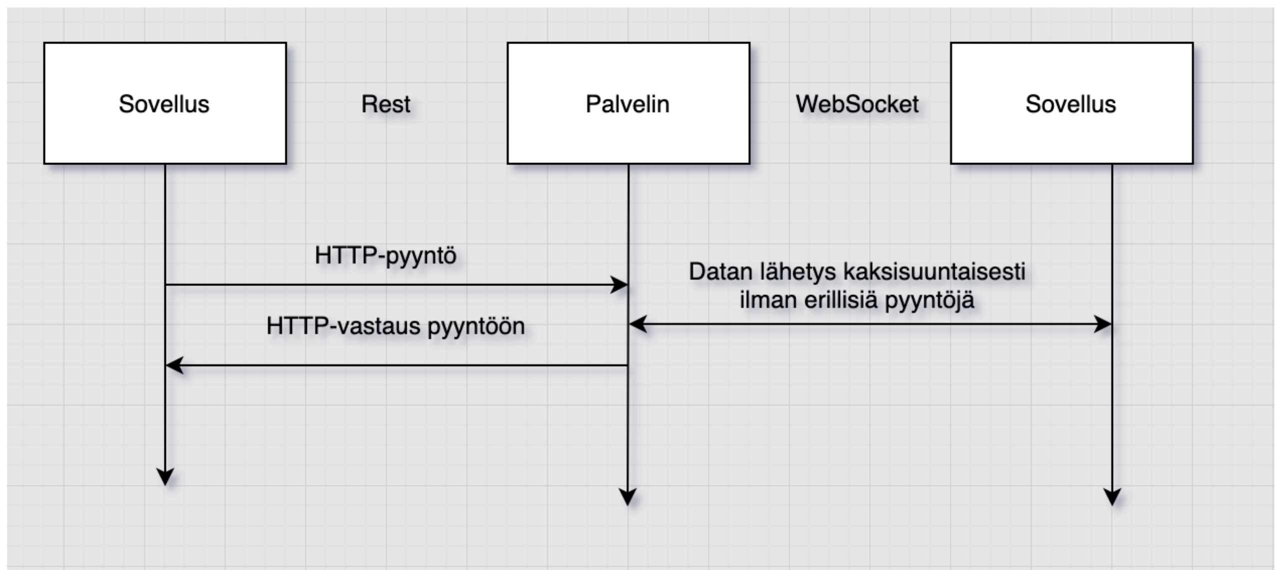
4.1 Amazonin konttien hallintapalvelut

Amazon Elastic Container Service eli ECS on skaalautuva ja täysin hallittu konttien orkestrointipalvelu. ECS on integroitavissa muihin Amazonin palveluihin, kuten käyttöoikeuksien IAM-hallintapalveluun ja CloudWatch-palveluun. Amazon Elastic Container Registry eli ECR-palvelulla voidaan hallita Docker-konttien rekisteriä Amazonin pilvipalveluissa. (25.) Amazonin elastisessa konttirekisterissä sijaitsevat Docker-konttien näköistiedostot. Rekisteri on luotettava ja skaalautuva

ja rekisterin käyttöoikeudet pohjautuvat resurssipohjaisiin käyttöoikeuksiin. Kehittäjät voivat tuoda, viedä ja hallita konttien näköistiedostoja Docker Command Line Interface -työkalun eli CLI:n kautta. (26.)

4.2 WebSocket API

WebSocket API on Amazon API Gateway -palvelun sovellusliittymä, joka mahdollistaa kaksisuuntaisen keskustelun asiakassovelluksen ja palvelimen välillä. WebSocket on kokoelma reittejä, jotka on integroitu taustaohjelman HTTP-päätepisteisiin, Lambda-toimintoihin ja muihin AWS-palveluihin. WebSocket-sovellusliittymän käyttäminen parantaa asiakassovelluksen ja palvelimen vuorovaikutusta, koska sen avulla palvelin voi lähettää dataa sovellukselle ilman sovelluksen tekemää pyyntöä. (27.) Amazon API Gateway on REST-, HTTP- ja WebSocket-sovellusliittymien luomiseen, julkaisemiseen, ylläpitämiseen ja seurantaan tarkoitettu AWS-palvelu (28). Kuva 6 havainnollistaa perinteisen REST API:n ja WebSocket API:n eron yksinkertaisimmillaan.



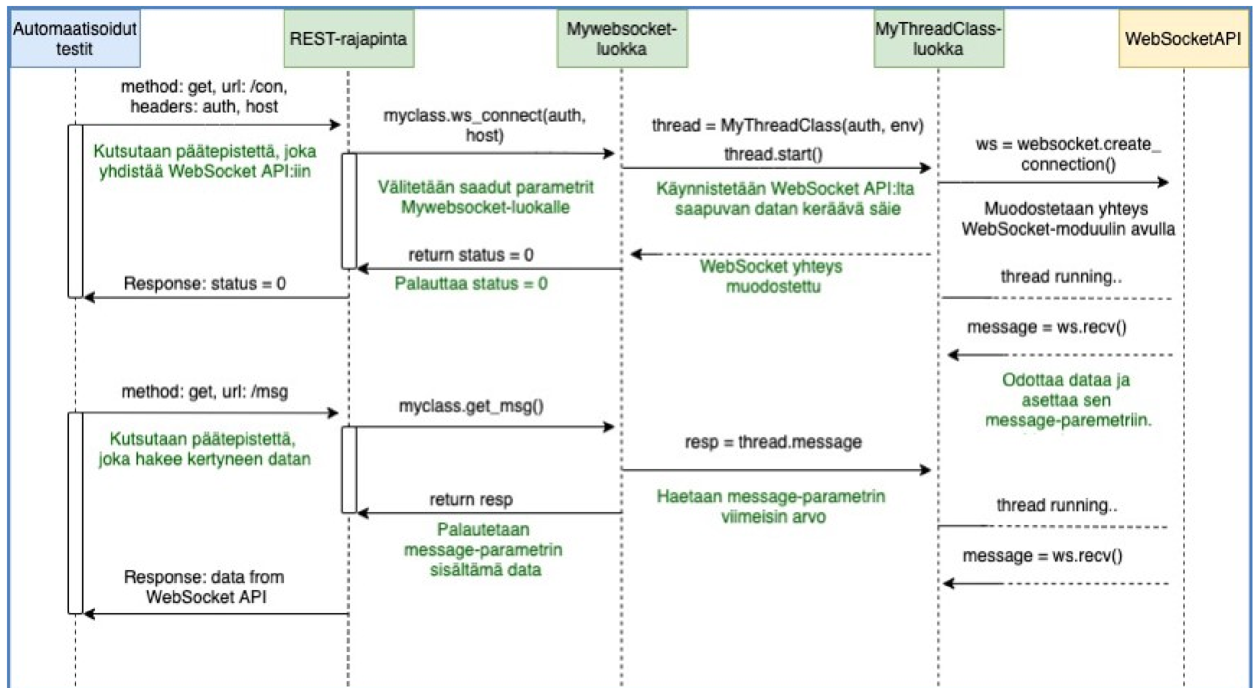
KUVA 6. REST- ja WebSocket-protokollien erot

5 WEBSOCKET API:N DATAN KERÄYS

Pivo-sovelluksen taustajärjestelmä on pitkään käyttänyt REST API:a saadakseen asiakkaan dataa taustajärjestelmistä asiakkaan sovellukselle. REST API:a testaavat automatisoidut testit on tehty Ruby-ohjelmointikielellä käyttäen yksikkötestaukseen kehitettyä Minitest-sovelluskehystä. Sovelluksen toiminnallisuuden parantamiseksi otettiin REST API:n lisäksi käyttöön AWS:n tarjoama WebSocket API, joka sallii palvelimen lähettää dataa sovellukselle ilman erillistä pyyntöä. WebSocket API:lta arvaamatta saapuvaa dataa ei pysty Ruby-ohjelmointikielellä Minitest-sovelluskehyksessä kuuntelemaan taustalla, joten opinnäytetyön päämääränä oli kehittää ratkaisu datan vastaanottamiseksi.

Työ toteutettiin rakentamalla ohjelma, jolla on oma REST-rajapinta ja WebSocket API:a kuunteleva toiminnallisuus. Ohjelma kerää saapuvan datan ja välittää sen rajapinnan kautta saataville. Tämän jälkeen automatisoiduista testeistä pystytään REST-protokollaa hyödyntäen kutsumaan datankeräysohjelman omasta rajapinnasta WebSocket API:lta mahdollisesti saapunutta ja kerättyä dataa.

Kuvan 7 sekvenssikaaviosta näkee kehitetyn dataa keräävän ohjelman suhteen automaatiotesteihin ja WebSocket API:iin kokonaisuudessaan. Kuten kaaviosta käy ilmi, automaatiotesteistä on tehtävä ensimmäisenä REST-pyyntö omalle dataa keräävälle rajapinnalle con-päätepisteeseen, joka puolestaan muodostaa yhteyden WebSocket API:iin Mywebsocket-luokan kautta ja käynnistää saapuvan datan kuuntelun säikeen MyThreadClass-luokan instanssina. Yhdistämisen jälkeen dataa voi kysellä msg-päätepistettä REST-pyyntönä kutsumalla aina tarvittaessa. Tämä pyyntö suorittaa get_msg()-metodin, joka hakee säikeessä asetetun message-parametrin viimeisimmän arvon ja palauttaa sen automaatiotesteille. Seuraavien alaotsikoiden alla on selitetty toiminnallisuus yksityiskohtaisemmin lähdekoodin avulla.



KUVA 7. Sekvenssikaavio dataa keräävän ohjelman toiminnasta

5.1 WebSocket API:n kuuntelun säie

WebSocket API:lta saapuvan datan kuuntelu on tehty Python-ohjelmointikielellä käyttäen Python-kirjastojen WebSocket-moduulia. Asiakassovelluksille suunnattuja esimerkkejä WebSocket-datan vastaanottamiseksi löytyi internetistä monella eri ohjelmointikielellä ja kirjastoilla tehtynä. Tärkeintä oli löytää projektiin sopiva WebSocket-moduuli, jonka WebSocket-yhteyden luomiseen tarkoitetulla metodilla pystyi välittämään HTTP-protokollan mukaisen otsikkotiedon WebSocket API:lle. Selvittelyn tuloksena löytyi Python-ohjelmointikielelle tarkoitettu WebSocket-moduuli, jonka create_connection()-metodiin pystyi asettamaan otsikkotiedon parametriksi. Python-kirjastojen WebSocket-moduuli on matalan tason sovellusliittymä WebSocket API:lle (29). Saapuvan datan kuuntelu tapahtuu omassa säikeessään.

Säiettä kutsuttaessa luodaan yhteys WebSocket API:iin create_connection()-metodilla. Metodin parametrina välitetään WebSocket API:n osoite ja HTTP-protokollan mukainen otsikko. Parametrit saadaan asiakassovellukselta ja ne sijoitetaan säiettä kutsuttaessa kuvassa 8 näkyviin env- ja auth-parametreihin. Auth-

parametri sisältää käyttöoikeuden varmistavan JWT-avaimen. Osoite, joka on kuvassa nimellä dest_url, koostetaan asiakassovellukselta välitetyn env-parametrin avulla. Parametri kertoo käytettävän ympäristön isäntäverkon sijainnin.

Säikeen ollessa käynnissä asetetaan ja päivitetään WebSocket API:lta saapunut data message-parametriin. Saapuvaa dataa kuunnellaan WebSocket-moduulin recv()-metodin avulla.

```
mythreadclass.py
1  import threading
2  import time
3  import websocket
4  import json
5
6
7  class MyThreadClass(threading.Thread):
8      def __init__(self, auth = "", env = "", message = {}):
9          threading.Thread.__init__(self)
10         self.message = message
11         self.auth = auth
12         self.env = env
13         dest_url = "wss://" + self.env +"/v1"
14         self.ws = websocket.create_connection(dest_url, header={'Authorization': self.auth})
15
16     def run(self):
17         while True:
18             set_message(self)
19
20 def set_message(self):
21     #Lataa dataa websocketilta ja palauttaa pyydettaessa apille
22     msg = json.loads(self.ws.recv())
23     print(msg)
24     self.message = msg
25
```

KUVA 8. Kuvankaappaus säikeen muodostavasta lähdekoodista

5.2 Dataa varten rakennettu rajapinta

Varsinainen ohjelma voidaan käynnistää ajamalla projektin app.py-tiedosto Python-ohjelmointikielen kääntäjällä. Ohjelma käynnistää pienen web-palvelimen, jonka avulla ohjelma voi tarjoilla yksinkertaisen REST-rajapinnan, jonka kautta automaatiotesteillä on mahdollista pyytää yhdistämistä WebSocket API:iin ja kysellä WebSocket API:lta saapunutta dataa ohjelmalta. Yksinkertaisen REST-rajapinnan rakentamiseksi Python-ohjelmointikielellä valikoitui nopeasti Flask-so-

velluskehys, koska se tarjosi REST API:n rakentamiseen sopivat työkalut Resource- ja Api-luokkien laajennuksena, joiden avulla rajapinnan reitittäminen ja resurssien luominen on mahdollista.

Tiedoston suorittaminen luo tiedoston alussa määritellyn pienen palvelimen ilmentymän Flask-sovelluskehyksen avulla, joka näkyy kuvassa 9 app-nimisenä olion määrittelemisenä. Toisena sovellusta käännettäessä luodaan api-niminen luokan ilmentymä Api-luokasta. Tämän instanssin avulla määritellään yksinkertaiselle rajapinnalle kaksi URL-reittiä, joita on mahdollista kutsua HTTP-protokollan GET-menetelmän avulla. Kolmantena luodaan oman MyWebSocket-luokan ilmentymä nimeltä myclass, joka huolehtii WebSocket API:n kuuntelun säikeen käynnistämisen.

Kummallekin URL-reitin määrittävälle resurssille on oma luokka, joilla molemmilla on oma get()-metodi. Kuvan 9 mukaisesti ConnectMyWSAPI-luokan get()-metodia kutsuttaessa parametreina on välitettävä HTTP-protokollan otsikon Authorization-kentässä JWT-avain ja host-kentässä isäntäverkon osoite. Parametrien puuttuessa rajapinta palauttaa paluuarvona viestin: ”Authorization missing”. JWT-avain sijoitetaan auth-nimiseen muuttujaan ja isäntäverkon osoite sijoitetaan env-muuttujaan.

Tämän jälkeen WebSocket API:iin muodostetaan yhteys kutsumalla myclass-luokan connect_ws()-metodia, jonka parametreina JWT-avain ja isäntäverkon osoite viedään myclass-luokan välityksellä WebSocket API:n dataa keräävälle säikeelle. MessageMyWSAPI-luokan get()-metodia kutsuttaessa myclass-olion get_msg()-metodikutsu palauttaa WebSocket API:lta kerätyn datan.

```python
app.py
1  from flask import Flask
2  from flask_restful import Resource, Api
3  from flask import jsonify
4  from flask import request
5
6
7
8  class TimeoutError(Exception):
9      pass
10
11 def _sig_alarm(sig, tb):
12     raise TimeoutError("timeout")
13
14 app = Flask(__name__)
15 api = Api(app)
16 myclass = Mywebsocket()
17
18 class ConnectMyWSAPI(Resource):
19     def get(self):
20         if request.headers['Authorization'] != '':
21             auth = request.headers['Authorization']
22             env = request.headers['host']
23
24             resp = myclass.connect_ws(auth, env)
25             return resp
26
27         else:
28             return "Authorization missing"
29
30 class MessageMyWSAPI(Resource):
31     def get(self):
32         resp = myclass.get_msg()
33         return resp
34
35
36 api.add_resource(ConnectMyWSAPI, '/con')
37 api.add_resource(MessageMyWSAPI, '/msg')
38
39 if __name__ == '__main__':
40     app.run(debug=True, host='0.0.0.0')
41
42
43
```

KUVA 9. Ajettavan sovelluksen varsinainen lähdekoodi

5.3 Mywebsocket-luokan toiminta

Mywebsocket-luokan connect_ws()-metodin kutsuminen käynnistää WebSocket-dataa keräävän säikeen ja palauttaa rajapintaa kutsuvalle onnistuneen

WebSocket-yhteyden luonnin jälkeen resp-nimisen muuttujan, joka näkyy kuvassa 10. Resp-muuttujaan sijoitetaan JSON-muodossa vastaus, jossa status-nimisen muuttujan arvona on nolla, millä halutaan informoida, että yhteyden luonti onnistui. Get_msg()-metodi palauttaa säikeen message-muuttujan arvon rajapinnalle.

```
mywebsocket.py
import websocket
from mythreadclass import MyThreadClass
import sys

class Mywebsocket:

    def __init__(self, thread = ""):
        self.thread = thread

    def connect_ws(self, auth, env):
        #Luodaan ja kaynnistetaan thread websocketin kuunteluun
        print("Thread created")
        self.thread = MyThreadClass(auth, env)
        print("Thread started")
        self.thread.start()
        resp = {'status': 0}
        return resp

    def get_msg(self):
        resp = self.thread.message
        return resp
```

KUVA 10. Mywebsocket-luokan lähdekoodi kuvankaappauksena

5.4 Oman dataa välittävän rajapinnan käyttö automaatiotesteissä

Aiemmin kehitetyissä yksikkötesteissä on käytetty RestClient-moduulia, joka tarjoaa HTTP- ja REST-resurssit rajapintakutsuille. WebSocket API:lta saapuvan datan välittämiseen automaatiotesteille rakennettu ohjelma tarjoaa REST-rajapinnan, josta automaatiotesteissä voidaan RestClient-moduulia käyttäen kutsua execute()-metodilla rajapinnan päätepisteitä. Metodikutsussa parametrina välitetään HTTP-metodina ”GET” ja osoitteena rajapinnan päätepistettä osoittava osoite. Lisäksi otsikossa annetaan tarvittaessa JWT-avain ja isäntäverkon osoite.

Ensimmäisenä automaatiotesteissä on suoritettava REST-pyyntö dataa välittävälle ohjelmalle rajapinnan päätepisteeseen ”con”. Kutsu huolehtii WebSocket-

yhteyden muodostamisesta. Vastauksena saadaan metodin suorittamisen jälkeen rajapinnalta: ”status = 0”. WebSocket-dataa pyydettäessä automaatiotesteissä kutsutaan rajapinnan msg-päätepistettä ja paluuarvona saadaan WebSocket API:lta kerätty data JSON-muotoisena vastauksena.

6 OHJELMAN JA TESTIEN KONTITTAMINEN

Uusien WebSocket-dataa käsittelevien testitapausten myötä automaatiotestit ovat riippuvaisia WebSocket-dataa keräävän ohjelman aktiivisuudesta. Siksi on tärkeää, että molemmat palvelut ovat käynnissä ja saatavilla samanaikaisesti. Kun paketoidaan dataa keräävä ohjelma ja automatisoitavat taustajärjestelmän REST-rajapintaa testaavat yksikkötestit Docker-konttien näköistiedostoiksi, on molempien palveluiden, testien ja WebSocket-dataa keräävän ohjelman, ajaminen mahdollista Docker Compose -työkalun avulla saman aikaisesti.

6.1 Näköistiedostojen luominen

Dataa keräävän ohjelman näköistiedoston luomista varten projektille on määriteltävä Dockerfile-tiedosto, jota käytettiinkin jo aiemmin esimerkkinä opinnäytetyön luvussa 3.2. Ohjelman kontin näköistiedosto rakennetaan ”docker build”-komennolla ja merkitään wsapp-nimiseksi näköistiedostoksi. Ruby-ohjelmointikielellä kehitetyille automaatiotesteille koostettiin myös oma Dockerfile-tiedosto, jonka näköistiedoston pohjaksi valikoitui myös aiempien kokemuksien perusteella kevyt Alpine Linux -projektiin pohjautuvaa näköistiedosto Ruby-kirjastojen versiolla 2.3.0. Automaatiotestien Dockerfile-tiedosto näkyy kuvassa 11. RUN-käskyihin määriteltiin asennettaviksi automaatiotestien kääntämiseen tarvittavat Ruby-ohjelmointikielen kirjastot, joita testien kehittämisessä on käytetty. COPY-komennolla kopioidaan projektin Gemfile- ja Gemfile.lock-tiedostot. Kopioinnin jälkeen Gemfile-tiedostoon määritellyt riippuvuudet asennetaan ”bundle install”-käskyllä.

```
Dockerfile
1  FROM ruby:2.3.0-alpine
2
3  RUN apk update
4  RUN apk add build-base nodejs postgresql-dev tzdata
5  RUN apk add --no-cache ca-certificates && \
6      update-ca-certificates
7
8  RUN mkdir /myapp
9  WORKDIR /myapp
10
11 RUN gem install minitest -v 5.8.4
12 RUN gem install bundler -v 1.11
13 RUN gem install minitest-junit
14 RUN gem install oauth2
15 RUN gem install openssl
16 RUN gem install rake -v '10.5.0'
17
18 COPY . /myapp
19 COPY Gemfile /myapp/Gemfile
20 COPY Gemfile.lock /myapp/Gemfile.lock
21
22
23 RUN bundle install
24
25
26 WORKDIR /myapp
27
28 ENTRYPOINT ["ruby"]
29 CMD ["pivo2_minitest.rb"]
30
```

KUVA 21. Automaatiotesteille määritelty Dockerfile-tiedosto

Automaatiotestien kontin näköistiedosto rakennetaan ”docker build”-käskyllä ja merkitään awstest-nimiseksi näköistiedostoksi. Kun molempien konttien näköistiedostot on luotu, ne löytyvät paikalliselle tietokoneelle tallennettuina ”docker images”-komennolla listattuna:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wsapp	latest	d50ff92f8cbe	4 minutes ago	199MB
awstest	latest	eac4815b0204	7 minutes ago	502MB
python	2.7-alpine	8579e446340f	6 days ago	71.1MB
ruby.	2.3.0-alpine	248bc6f3140f	4 years ago	125MB

6.2 Monisäiliöisten palveluiden käynnistäminen rinnakkain

Docker Compose -työkalun avulla voi käynnistää monisäiliöisen eli useaa konttia käyttävän palvelukokonaisuuden yhdellä käskyllä. Compose-työkalua varten määritellään YAML-tiedosto nimeltä ”docker-compose.yml”, johon määritellään palvelukokokonaisuuteen kuuluvien konttien näköistiedostot. Palveluiden kokonaisuus eli tarvittavat kontit käynnistetään ”docker-compose up”-komennolla. (30.) YAML-tiedoston alussa määritellään haluttu YAML-tiedostomuodon versio. Version valinta vaikuttaa siihen, millaisia argumentteja Docker Compose -työkalulle voi määritellä YAML-tiedostossa. (31.)

Kuvassa 12 näkyy projektin ”docker-compose.yml”-tiedosto. Tiedostossa määritellään projektin palvelut nimeltä test ja app. Test-palvelulle asetetaan automaatiotesteistä aiemmin luotu awstest-niminen näköistiedosto kontin näköistiedostoksi. App-nimiselle palvelulle asetetaan dataa keräävästä ohjelmasta aiemmin rakennettu wsapp-niminen kontin näköistiedosto.

docker-compose.yml

```
version: "2.2"
services:

  test:
    image: awstest

    ports:
      - "2048:2048"
    network_mode: "host"

  app:
    image: wsapp
    ports:
      - 5000:5000

```

KUVA 32. Palvelukokonaisuuden määrittävä YAML-tiedosto

”Docker-compose up”-komennolla Compose-työkalu käynnistää palvelukokonaisuuden, jossa käynnistetään dataa keräävän ohjelman kontti sovelluksena nimeltä app_1 ja automatisoidut yksikkötestit ajava kontti palveluna nimeltä test_1. Kuvasta 13 näkee tilanteen palveluiden ajon suorittamisesta. Kuvassa palvelun

app_1 käynnistäminen on luonut paikallisen web-palvelimen portissa 5000 ja automaatiotestien kontin käynnistäminen on ajanut yhden testitapauksen, jossa onkin kutsuttu testiä suorittaessa dataa keräävän ohjelman rajapinnan con-päätepistettä.

```
Creating network "aws-test-automation_default" with the default driver
Creating aws-test-automation_app_1  ... done
Creating aws-test-automation_test_1 ... done
Attaching to aws-test-automation_test_1, aws-test-automation_app_1
app_1    |  * Serving Flask app "app" (lazy loading)
app_1    |  * Environment: production
app_1    |    WARNING: This is a development server. Do not use it in a production deployment.
app_1    |    Use a production WSGI server instead.
app_1    |  * Debug mode: on
app_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
app_1    |  * Restarting with stat
app_1    |  * Debugger is active!
app_1    |  * Debugger
test_1   | Run options: --seed 33394
test_1   |
test_1   | # Running:
test_1   |
test_1   |
test_1   | 2020-04-27 18:01:26 +0000: ====================
test_1   | 2020-04-27 18:01:26 +0000: Test setup (Test case: test_own_thing)
app_1    | Thread created
app_1    | 172.29.0.1 - - [27/Apr/2020 18:01:30] "GET /con HTTP/1.1" 200 -
test_1   | 2020-04-27 18:01:30 +0000: Test teardown (Test case: test_own_thing)
test_1   | 2020-04-27 18:01:30 +0000: ====================
test_1   | .
test_1   |
test_1   | Finished in 5.777310s, 0.1731 runs/s, 0.0000 assertions/s.
test_1   |
test_1   | 1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
aws-test-automation_test_1 exited with code 0
```

KUVA 43. Palvelukokonaisuuden ajon esimerkki

6.3 Kontitettujen testien ajaminen Jenkins-automaatiopalvelimella

Olemassa olevien taustajärjestelmää testaavien automatisoitujen yksikkötestien kokonaisuus on ennen opinnäytetyön aloittamista suoritettu ajastetusti Jenkins-automaatiopalvelimella. Jenkins-palvelimelle on määritelty, että yksikkötestien lähdekoodi noudetaan Git-versionhallintajärjestelmästä, kun testit halutaan suorittaa. Linus Torvalds perusti avoimen lähdekoodin Git-versionhallintajärjestelmän ohjelmiston vuonna 2005 ja se on työkalu ohjelmistokehityksen lähdekoodin hallintaan (33). Jenkins-ohjelmisto on myös avoimeen lähdekoodiin perustuva ohjelmisto automaatiopalvelimen ylläpitämiseksi. Jenkins-ohjelmiston avulla voidaan automatisoida kaikenlaisien ohjelmistojen rakentamiseen, toimittamiseen, testaamiseen ja käyttöönottoon liittyviä toiminnollisuuksia (34). Kun automatisoitavat testit on paketoitu ja rakennettu näköistiedostona saatavaksi, voi Jenkins-

automaatiopalvelimelle määritellä testien ajon suoritettavaksi Docker-kontin näköistiedostosta Jenkins-automaatiopalvelimen Docker-laajennuksen avulla. Tämä ominaisuus mahdollistaa myös usean kontin rinnakkaisen ajamisen Jenkins-automaatiopalvelimella eli usean erilaisen testikokonaisuuden ajamisen samanaikaisesti, koska seuraavan ajettavan testikokonaisuuden ei tarvitse odotella ympäristömuuttujien vapautumista omiin tarpeisiin vaan kontissa ajettuna kaikki tarvittava on paketoitu konttiin ja ympäristö toimii eristettynä muista konteista (32). Tätä mahdollisuutta voisi tulevaisuudessa hyödyntää hajauttamalla valtavan testikokonaisuuden usealle eri kontille ajettavaksi rinnakkain, jolloin testien suorittamiseen menisi vähemmän aikaa.

7 YHTEENVETO

Opinnäytetyön päämääränä oli kehittää ohjelma WebSocket API:lta saapuvan datan keräämiseen sekä selvittää, miten ohjelma olisi mahdollista ajaa Docker-kontissa ja kuinka olemassa olevat automaatiotestit voitaisiin saattaa myös Docker-kontissa suoritettaviksi.

Pivo-mobiilisovelluksen taustajärjestelmässä käytettiin pitkään REST API -rajapintaa datan välittämiseksi palvelimelta asiakkaan sovellukselle. Toiminnallisuuden parantamiseksi ja tietojen päivittämisen nopeuttamiseksi sovelluskehityksessä otettiin REST API:n lisäksi käyttöön myös WebSocket API. WebSocket-protokolla mahdollistaa kahden suuntaisen keskustelun palvelimen ja asiakkaan sovelluksen välillä, jolloin dataa voi toimittaa palvelimelta asiakkaan sovellukselle ilman erillistä pyyntöä. Arvaamatta palvelimelta saapuva data oli mahdotonta vastaanottaa automaatiotesteissä nykyisin puittein, joten ratkaisuksi opinnäytetyössä kehitettiin ohjelma, joka voi kerätä saapuvan datan ja pystyy välittämään sen automaatiotesteille testien sitä pyytäessä.

Konttitekniikka oli minulle ennen opinnäytetyöni aloittamista täysin uusi aihe ja aiheen selvittämiseen meni jokseenkin paljon aikaa, vaikka se ei sinänsä ole kovin monimutkainen. Konttitekniikka on mielestäni mielenkiintoinen ja siinä on paljon potentiaalia. Opinnäytetyön jälkeen minulla on vielä henkilökohtaisesti paljon opittavaa, jotta voisin saada konttitekniikasta parhaan mahdollisen hyödyn irti.

Konttitekniikka nopeuttaa ohjelmistokehittämistä, tekee siitä mutkattomampaa ja sujuvoittaa huomattavasti tuotantoon vientiä, koska konttiin voi pakata sisään kaiken tarvittavan, sitä on helppo siirrellä ja yhden Linux- tai Windows-ytimen päällä voi ajaa monta konttia samanaikaisesti.

Automaatiotestit ja WebSocket API:lta dataa keräävä ohjelma saatiin opinnäytetyössä pakattua kontteihin. Konttien näköistiedostojen ajaminen onnistuu opinnäytetyön lopuksi Docker Compose -työkalun avulla rinnakkain ja automaatiotestit saavat datan ohjelman rajapinnan avulla käsiteltäväksi.

Konttien ajaminen onnistuu paikallisella tietokoneella Docker-ohjelman avulla ja kokonaisuus oli tarkoitus vielä viedä Amazon Web Services palveluiden avulla pilvilaskentaympäristöön, jotta siitä saisi paremman hyödyn irti. Aika ei ollut valitettavasti työn ja opinnäytetyön tiukan aikataulun puolesta suotuisa, joten tämä on tarkoitus toteuttaa loppuun myöhemmin, kuitenkin niin pian kuin aikataulu töiden puolesta antaa myöten.

OP:n käytössä oleviin Amazonin pilvilaskentapalveluiden resursseihin on määritelty tarkasti erilaisia käyttöoikeuksia, joten konttipalveluiden kehittämisessä tarvitsisin kollegoiden apua ja oikeuksia. Tämä on erittäin hyvä asia, sillä se vähentää riskiä, että kokemattomuudesta saattaisi aiheutua pilvipalveluiden tietovarastoille jotain vahinkoa.

Lopputuloksesta on kuitenkin jo tässä vaiheessa valtavasti hyötyä automaatiotestien kattavuuden parantamiseksi. Tulos avaa mahdollisuuden kehittää paljon uusia automatisoitavia testitapauksia, joissa merkityksellistä on tutkita ja vertailla WebSocket API:lta saatavaa dataa.

Dataa keräävää ohjelmaa täytyy jatkossa kehittää niin, että se pystyisi keräämään vaikkapa taulukkoon muistiin aikaleiman kera saapuneen datan, koska tällä hetkellä se ylikirjoittaa uuden datan edellisen päälle. Lisäksi jatkossa voisi selvittää, kuinka dataa kuuntelevan ja keräävän säikeen voisi keskeyttää hallitusti, koska tällä hetkellä se odottaa WebSocket-yhteyden aikakatkaisua. Nyt automaatiotestien hyödyntäessä konttitekniikkaa, voisi automaatiotestejä tulevaisuudessa hajauttaa useammalle eri kontille, jolloin niiden suorittaminen Jenkins-automaatiopalvelimella veisi vähemmän aikaa.

LÄHTEET

1. Pivon tarina. Pivo. Saatavissa: https://pivo.fi/pivon-tarina/. Hakupäivä 15.4.2020.

2. Maksa puhelimella kaikissa arjen tilanteissa. Pivo. Saatavissa: https://pivo.fi/maksut/. Hakupäivä 15.4.2020.

3. Module MiniTest. Ruby-lang docs. Saatavissa: https://docs.ruby-lang.org/en/2.1.0/MiniTest.html. Hakupäivä 15.4.2020.

4. Mobiilimaksut kaikkien pankkien asiakkaille. Pivo. Saatavissa: https://pivo.fi. Hakupäivä 24.4.2020.

5. Minitest/{test,spec,mock,benchmark}. Seattlerb. Saatavissa: http://docs.seattlerb.org/minitest/index.html. Hakupäivä 23.4.2020.

6. What is DevOps? 2020. Atlassian. Saatavissa: https://www.atlas-sian.com/devops. Hakupäivä 24.4.2020.

7. Ketterän yrityskulttuurimuutoksen teknisenä ajurina toimii DevOps ja muu-tokseen on sitoutunut koko organisaatio. OP. Saatavissa: https://www.op.fi/op-ryhma/ura-oplla/toissa-meilla/devops. Hakupäivä 15.4.2020.

8. Ketterä toimintatapa. OP. Saatavissa: https://www.op.fi/op-ryhma/tietoa-ryh-masta/op-lyhyesti/kettera-toimintatapa. Hakupäivä 15.4.2020.

9. Devops. 2020. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Devops. Hakupäivä 15.4.2020.

10. DevOps. 2020. Wakaru. Saatavissa: https://www.wakaru.fi/valmennus/par-haat-kaytannot/ketterat-menetelmat/devops/. Hakupäivä 15.4.2020.

11. DevOps toi ohjelmistokehittäjä Oskari Johanssonin arkeen sujuvuutta, osaamisen skaalautumista ja vaikuttamismahdollisuuksia. OP. Saatavissa:

https://www.op.fi/op-ryhma/ura-oplla/toissa-meilla/op-olemme-me-uratari-noita/uratarina-oskari-johansson. Hakupäivä 15.4.2020.

12. Wallenius, Niklas 2020. Konttiteknologia – mitä kontit ovat ja mitä hyötyä niistä on? Saatavissa: https://niklaswallenius.fi/teknologiat/konttiteknologia-mita-hyotya/. Hakupäivä 15.4.2020.

13. Kotilainen, Samuli 2017. Koodi sujahtaa konttiin – sovellusten kehittäminen mullistuu. Tivi. Saatavissa: https://www.tivi.fi/uutiset/koodi-sujahtaa-konttiin-sovellusten-kehittaminen-mullistuu/7931ccac-1cd7-3c40-b338-be25469cf1dd. Hakupäivä 15.4.2020.

14. Wallenius, Niklas 2020. Mikä on Docker ja mitä hyötyä siitä on? Saatavissa: https://niklaswallenius.fi/digitalisaatio/mika-on-docker/. Hakupäivä 15.4.2020.

15. What is Docker? 2020. AWS. Saatavissa: https://aws.amazon.com/docker/. Hakupäivä 15.4.2020.

16. Dockerfile reference. Docker Docs. Saatavissa: https://docs.docker.com/en-gine/reference/builder/. Hakupäivä 15.4.2020.

17. Python. Docker Official Images. 2020. Docker hub. Saatavissa: https://hub.docker.com/_/python. Hakupäivä 21.4.2020.

18. Naik, Yathi 2017. Hardening Docker containers, images, and host - security toolkit. Stackrox. Saatavissa: https://www.stackrox.com/post/2017/08/har-dening-docker-containers-and-hosts-against-vulnerabilities-a-security-tool-kit/. Hakupäivä 23.4.2020.

19. Niemistö, Tero 2017. Kovenna konttisi - 10+1 keinoa konttien tietoturvan parantamiseen. Digia. Saatavissa: https://blog.digia.com/kovenna-konttisi-10-keinoa-konttien-tietoturvan-parantamiseen. Hakupäivä 23.4.2020.

20. About AWS. 2020. AWS. Saatavissa: https://aws.amazon.com/about-aws/. Hakupäivä 16.4.2020.

21. Regions, Availability Zones, and Local Zones. 2020. AWS. Saatavissa: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-regions. Hakupäivä 23.4.2020.

22. What Is IAM? 2020. AWS. Saatavissa: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html. Hakupäivä 23.4.2020.

23. Listing IAM Groups. 2020. AWS. Saatavissa: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_groups_manage_list.html. Hakupäivä 23.4.2020.

24. Modifying a Role (Console). 2020. AWS. Saatavissa: https://docs.aws.amazon.com/IAM/latest/UserGuide/roles-managingrole-editing-console.html. Hakupäivä 23.4.2020.

25. Amazon Elastic Container Service. 2020. AWS. Saatavissa: https://aws.amazon.com/ecs/. Hakupäivä 23.4.2020.

26. What Is Amazon Elastic Container Registry? 2020. AWS. Saatavissa: https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html. Hakupäivä 15.4.2020.

27. Working with WebSocket APIs. 2020. AWS. Saatavissa: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html. Hakupäivä 15.4.2020.

28. What is Amazon API Gateway? 2020. AWS. Saatavissa: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html. Hakupäivä 15.4.2020.

29. Websocket_client 0.57.0. Pypi. Saatavissa: https://pypi.org/project/websocket_client/. Hakupäivä 21.4.2020.

30. Overview of Docker Compose. Docker Docs. Saatavissa: https://docs.docker.com/compose/. Hakupäivä 27.4.2020

31. Compose file versions and upgrading. Docker Docs. Saatavissa: https://docs.docker.com/compose/compose-file/compose-versioning/. Hakupäivä 27.4.2020.

32. Aernouts, Sander 2017. Containerized testing. Xpirit. Saatavissa: https://xpirit.com/containerized-testing/. Hakupäivä 29.4.2020.

33. Git. 2020. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Git. Hakupäivä 14.5.2020.

34. Jenkins User Documentation. Jenkins. Saatavissa: https://www.jenkins.io/doc/. Hakupäivä 14.5.2020.