
Patrik Winter

# Testiautomaatio Azure DevOps -ympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

12.5.2023

# Tiivistelmä


| | |
|---|---|
| Tekijä: | Patrik Winter |
| Otsikko: | Testiautomaatio Azure DevOps -ympäristössä |
| Sivumäärä: | 32 sivua + 7 liitettä |
| Aika: | 12.5.2023 |
| Tutkinto: | Insinööri (AMK) |
| Tutkinto-ohjelma: | Tieto- ja viestintätekniikan tutkinto-ohjelma |
| Ammatillinen pääaine: | Älykkäät järjestelmät |
| Ohjaajat: | Lehtori Keijo Länsikunnas<br>Lehtori Anne Pajala |


---


Opinnäytetyön tavoitteena oli luoda testiautomaatioympäristö web-sovelluksien ja verkkosivujen automatisoituun testaamiseen.

Testiautomaation avulla pystytään automatisoida ohjelmien testausprosessia ja näin vapauttaa testaajien aikaa muuhun testaamiseen. Testaaminen on tärkeää tuotteen laadun varmistamisessa.

Testit kirjoitettiin käyttäen Robot Framework -automaatiokehystä.
Testeissä käytetään QWeb-kirjasto, joka soveltuu erityisesti Web-sovelluksien testaamiseen. Testien suorittamisesta ja ajastamisesta huolehtii Azure DevOps -ympäristö.



Avainsanat: Testiautomaatio, Robot Framework, Azure DevOps, QWeb, Docker

## Abstract


| | |
|---|---|
| Author: | Patrik Winter |
| Title: | Test automation in Azure DevOps environment |
| Number of Pages: | 32 pages + 7 appendices |
| Date: | 12 May 2023 |
| Degree: | Bachelor of Engineering |
| Degree Programme: | Information and Communication Technology |
| Professional Major: | Smart Systems |
| Supervisors: | Keijo Länsikunnas, Principal Lecturer<br>Anne Pajala, Principal Lecturer |


---


The purpose of the thesis was to create a test automation environment for automated testing of web applications and websites.

With the help of test automation, it is possible to automate the program testing process and thus free up testers' time for other testing. Testing is important in ensuring product quality.

The tests were written using the Robot Framework automation framework.
The tests use the QWeb library, which is especially suitable for Web Application testing. Azure DevOps environment takes care of test executions and scheduling.



Keywords: Test automation, Robot Framework, Azure DevOps, QWeb, Docker

# Sisällys

## Lyhenteet

| | |
|---|---|
| CI | Contunuous Integration. Jatkuva integraatio on ketterän ohjelmistokehityksen toimintatapa. |
| CD | Contunuous Delivery. Jatkuva toimittaminen integraatio on ketterän ohjelmistokehityksen toimintatapa ja jatkoa jatkuvalle integraatiolle. |
| WSL | Windows Subsystem for Linux. Windows-ominaisuus, jonka avulla kehittäjät voivat käyttää Linux-ympäristöä. |
| XPath | XML Path Language. Kieli XML-dokumenttien osien paikantamiseen. |
| YAML | YAML Ain't Markup Language. Tiedostoformaatti, jota käytetään usein konfiguraatioissa. |
| PAT | Personal Access Token. Poletti, jota käytetään tunnistautumiseen. |

# 1 Johdanto

Tämän insinöörityön aiheet ovat testiautomaatio sekä testiautomaation kehittämiseen ja suorittamiseen soveltuvan ympäristön määrittäminen. Työssä käydään läpi tähän tarvittavia työkaluja ja palveluita. Aloitin työni Gofore Oyj -nimisessä yrityksessä työharjoittelijana toukokuussa 2021. Gofore on suomalainen laajat IT-konsultointipalveluita tarjoava julkinen osakeyhtiö. Roolini harjoittelussa ja insinöörityötä kirjoittaessa on testiautomaatioasiantuntija. Harjoittelujakso kesti kuusi kuukautta, jonka jälkeen olen jatkanut täysipäiväisenä työntekijänä.

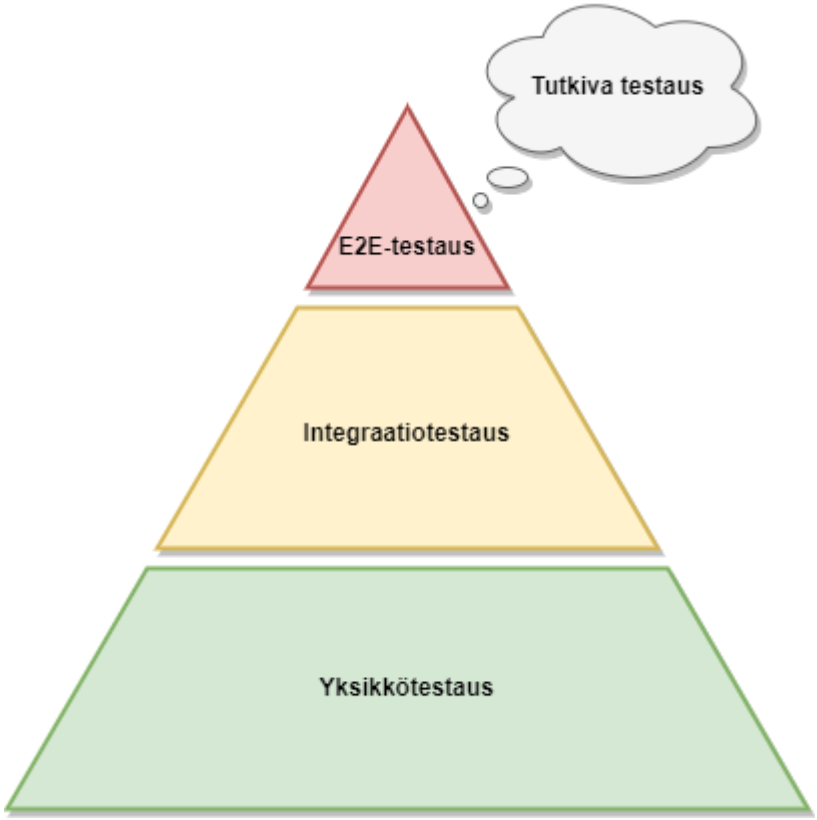
Tämän insinöörityön tavoite on määrittää ympäristö, joka soveltuu testiautomaatio testien kehittämiseen ja suorittamiseen sekä projektin hallinnoimiseen. Ympäristön tarkoitus on olla madollisimman helppokäyttöinen ja helposti ylläpidettävä. Suurimpana tiedon lähteenä insinöörityössä tulee olemaan testiautomaatioasiantuntijana työskentely. Olen päässyt työskentelemään konsulttina useassa projektissa ja usealla eri asiakkaalla. Näissä projekteissa olen päässyt kehittämään testiautomaatiotestejä ja määrittämään testiautomaation kehitys- sekä suoritusympäristöjä.

Opinnäytetyö sisältää seitsemän osaa. Ensimmäisessä osassa käsitellään testiautomaatiota sekä sen mahdollisia hyötyä ja riskejä. Toisessa osassa käsitellään DevOps-toimintamallin keskeisimpiä periaatteita. Kolmannessa osassa tutustutaan Robot Framework -kehykseen. Neljässä osassa perehdytään Docker-virtualisointityökaluun. Viidennessä osassa käydään läpi Azure DevOps -ympäristön tarjoamia ominaisuuksia. Kuudes osa sisältää ympäristön suunnittelun ja toteutuksen. Seitsemännessä ja viimeisessä osassa pohditaan työn onnistumista ja mahdollisia jatkokehitysmahdollisuuksia.

# 2 Testiautomaatio

Testiautomaatio tarkoittaa ohjelmistojen, kuten web-sovelluksien automatisoitua testaamista. Ohjelmistojen testaamisella varmistetaan, että ohjelmisto täyttää

sille ennalta annetut kriteerit. Näitä voi olla esimerkiksi koodin tyyli, koodin toiminnallisuus tai käyttöliittymän ulkoasu. Testiautomaation avulla pyritään automatisoimaan mahdollisimman paljon ohjelmistoihin liittyvää testaamista. Näin manuaalitestaajien ei tarvitse toistaa samoja testejä ja voivat he keskittyä hyödyllisempään työhön, kuten uusien osa-alueiden testaamiseen tai tutkivaan testaamiseen. Tutkivalla testaamisella tarkoitetaan vapaamuotoista ohjelmistojen testaamista, jossa manuaalitestaaja käyttää erilaisia tekniikoita, kokemusta ja intuitiota mahdollisimman usean virheen löytämiseen. Hyödyntämällä testiautomaatiota sekä manuaalitestausta saavutetaan parempi ohjelmiston testauskattavuus kuin ilman testiautomaatiota. Testauksen automatisoimisen vaikeutta kuvataan useasti pyramidilla, jonka alapäässä on helposti ja yläpäässä vaikeasti automatisoitavat testit. Kuvassa 1 näkyy tyypillinen testiautomaatiopyramidi. [1; 2.]




Kuva 1. Testiautomaatiopyramidi. [1]

## 2.1 Yksikkötestaus

Yksikkötestauksen tarkoitetaan lähdekoodin yksittäisien osien, kuten funktioiden toiminnallisuuden varmistamista. Yksikkötesteillä ei saa olla ulkoisia riippuvuuksia, kuten rajapintoja tai tietokantoja. [1; 2.]

Yksikkötestauksen päätarkoitus on tarkastella, kuinka ohjelmiston komponentit toimivat itsenäisesti ilman ulkoisia vaikutuksia. Yksikkötestaamista tehdään ohjelmistonkehitysvaiheessa ja on ensimmäinen testaamisen vaihe. Melkein kaikki yksikkötestit ovat täysin automatisoitavissa, ja automatisoitu yksikkötestaaminen on osa ohjelmistokehittäjien hyviä toimintatapoja. [1; 2.]

## 2.2 Integraatiotestaus

Integraatiotestauksen tarkoituksena on varmistaa, että ohjelmiston eri komponentit toimivat ja kommunikoivat toistensa kanssa halutulla tavalla. Integraatio testaaminen vaikeutuu integraatioiden määrän ja eri riippuvuuksien lisääntyessä. [2; 3.]

Integraatiotestauksen ylläpitäminen vaikeutuu, kun ohjelmiston sisältämien integraatioiden määrä kasvaa. Integraatiotestauksesta saattaa hankaloittaa myös suuri määrä integraatioita ulkoisiin palveluihin, kuten tietokantoihin, rajapintoihin tai muihin ohjelmiin. Ohjelmiston kasvaessa integraatiotestaus vaikeutuu ja testaus menetelmiin kannattaa ottaa mukaan regressiotestaus, jos se ei ole jo käytössä. [2; 3.]

## 2.3 Regressiotestaus

Regressiotestauksella varmistetaan, ettei muutos lähdekoodiin vaikuta negatiivitesti ohjelmistossa oleviin toiminnallisuuksiin. Regressiotestaus varmistaa, että aiemmin toteutetut toiminnallisuudet toimivat. Tämä on erityisen tärkeää isoissa web-sovelluksissa, joihin lisätään jatkuvasti uusia toiminnallisuuksia. [2.]

Regressiotestit tulisi suorittaa mahdollisimman nopeasti lähdekoodiin tehtyjen muutoksien jälkeen, jotta virheet löydettäisiin mahdollisimman nopeasti. Regressiotestejä on myös hyvä suorittaa ajastetusti, sillä kaikki virheet ohjelmistossa eivät välttämättä ilmene heti koodiin tehtyjen muutosten jälkeen. Testiautomaatiolla toteutetut regressiotestit voidaan suorittaa automatisoidusti testiautomaatioputken avulla heti lähdekoodiin tehtyjen muutoksien jälkeen. Testiautomaatioputken suoritus voidaan käynnistää ajastetusti, jolloin myös myöhemmässä vaiheessa ilmenevät virheet voidaan paikantaa nopeasti. [2.]

### 2.4 E2E-testaus

E2E-testauksen tarkoituksena on varmistaa kokonaisien prosessien toimivuus, jotka saattavat käyttää useita ulkopuolisiasi riippuvuuksia ja integraatioita. E2E-testaus on samankaltaista kuin integraatiotestaus. Hyvällä E2E-testuksella voidaan vähentää integraatiotestauksen tarvetta ja on usein isoissa ohjelmistoissa parempi lähestymistapa ohjelmiston eri komponenttien välisen kommunikaation toimivuuden varmistamisessa. [1.]

E2E-testaus on aikaisemmin ollut pääasiassa manuaalitestaajien työtä, ja tästä syystä testien suorittaminen on vienyt paljon manuaalitestaajien aikaa. Tällä hetkellä uudet testiautomaatio työkalut ja DevOps-toimintamalli yrittävät keskittyä E2E-testien automatisoimisen helpottamiseen. DevOps-toimintamallin mahdollistamat testiautomaatioputket ovat oleellinen osa E2E-testauksen autonomisoinnin helpottamista. [1.]

## 3 DevOps-toimintamalli

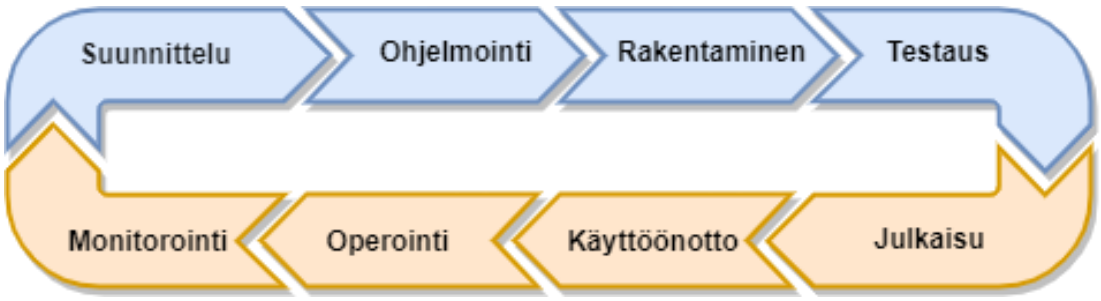
DevOps-toimintamalli on yhdistelmä erilaisia toimintatapoja, työkaluja ja kulttuuria. Käsitteellä ei kuitenkaan ole tarkkaa määritelmää. Sillä tarkoitetaan karkeasti kehityksen, testauksen ja ylläpidon sulavaa yhteistyötä. Toimintatavat, työkalut ja kulttuuri saattavat vaihdella paljon riippuen projektista tai yrityksestä. DevOps-toimintamallin tavoite on mahdollistaa moderni, nopea ja vakaa ohjelmistokehitys.

DevOps-toimintamallia kuvataan useasti syklinä, joka sisältää kahdeksan vaihetta. Sykli alkaa ohjelmiston tai siihen tehtävien muutoksien suunnittelusta. Tämän jälkeen siirrytään kehitysvaiheeseen, jossa ohjelma tai siihen suunnitellut muutokset toteutetaan. Kehitysvaiheessa testataan myös koodin toimivuutta ja varmistetaan laatua. [4.]

Seuraavaksi siirrytään rakentamis- eli integraatiovaiheeseen, jossa eri kehitystiimit tai kehittäjät kokoavat koodit yhdeksi kokonaisuudeksi. Tämä mahdollistaa seuraavan eli testausvaiheen, jossa varmistetaan, että koodin ja sen eri komponentit toimivat yhdessä virheettömästi. [4.]

Tämän jälkeen siirrytään julkaisu vaiheeseen, jossa ohjelmisto tai uusi ohjelmistoversio siirretään jakeluun. Seuraavaksi on vuorossa käyttöönotto-, operointi- ja monitorointivaiheet. Nämä vaiheet ovat tärkeitä, sillä niiden aikana kerätään tietoa ohjelmiston käytöstä ja mahdollisista virheistä. Tämä tieto mahdollistaa ohjelmiston käytön aikana paikannettujen virheiden nopean korjaamisen sekä syklin jatkuvuuden. [4.]

Kerätty tieto on erityisen tärkeää siirryttäessä uudestaan suunnitteluvaiheeseen, jossa tietoa voidaan hyödyntää uusien ominaisuuksien, tai esimerkiksi käyttöliittymämuutoksien suunnittelussa. DevOps-toimintamalli kokonaisuudessaan tuo ohjelmistokehitykseen joustavuutta sekä mahdollisuuden reagoida nopeasti muuttuviin vaatimuksiin ja tarpeisiin. Kuvassa 2 näkyy tavanomainen kahdeksanvaiheinen DevOps-sykli. [4.]




Kuva 2. DevOps-syklin kuvaus. [4]

## 3.1 DevOps-toimintamallin alku

Ohjelmistokehityksen vanhat toimintatavat perustuivat prosesseihin, joita käytetään fyysisien laitteiden kehitykseen, testaamiseen ja julkaisuun. Fyysisien laitteiden kehityksessä pienetkin virheet ovat usein kalliita, mutta virheellisen laitteen julkaisu markkinoille on lähes aina erittäin kallista. Tästä syystä fyysisien laitteiden kehitysprosessit ovat joustamattomia ja hitaita. Virheiden hinta ei kuitenkaan ole sama ohjelmistokehityksessä, jossa pienet virheet voidaan usein korjata, myös tuotteen julkaisun jälkeen usein nopeasti ja pienillä resursseilla. Tästä huolimatta myös ohjelmistokehityksessä vakavat virheet saattavat olla hyvin kalliita ja niiden välttäminen, aikainen paikantaminen sekä nopea korjaaminen on hyvin tärkeää. Testiautomaatio on yksi tärkeä osa DevOps-toimintamallia, jonka avulla osa virheistä voidaan paikantaa nopeasti ja automatisoidusti. [5.]

DevOps-liike alkoi vuosien 2007 ja 2008 aikoina, kun ohjelmistoylläpito- ja ohjelmistokehitysyhteisöt esittivät huolensa vanhanaikaisesta ohjelmistonkehitys mallista. Tänä aikana ohjelmistojen operoinnista vastaavat ammattilaiset ja ohjelmistojen kehityksestä vastaavat ammattilaiset työskentelivät usein erillään toisistaan, ja ryhmien välinen kommunikaatio oli usein huonoa. DevOps-toimintamallilla yritetään ratkaista tätä ongelmaa ja tehdä osastojen välisestä yhteistyöstä mahdollisimman sulavaa. Hyvällä yhteistyöllä saavutetaan parempi tuottavuus ohjelmistonkehityksessä sekä nopeampi reagointi virheisiin. [6.]

## 3.2 DevOps-toimintamalli nykypäivänä

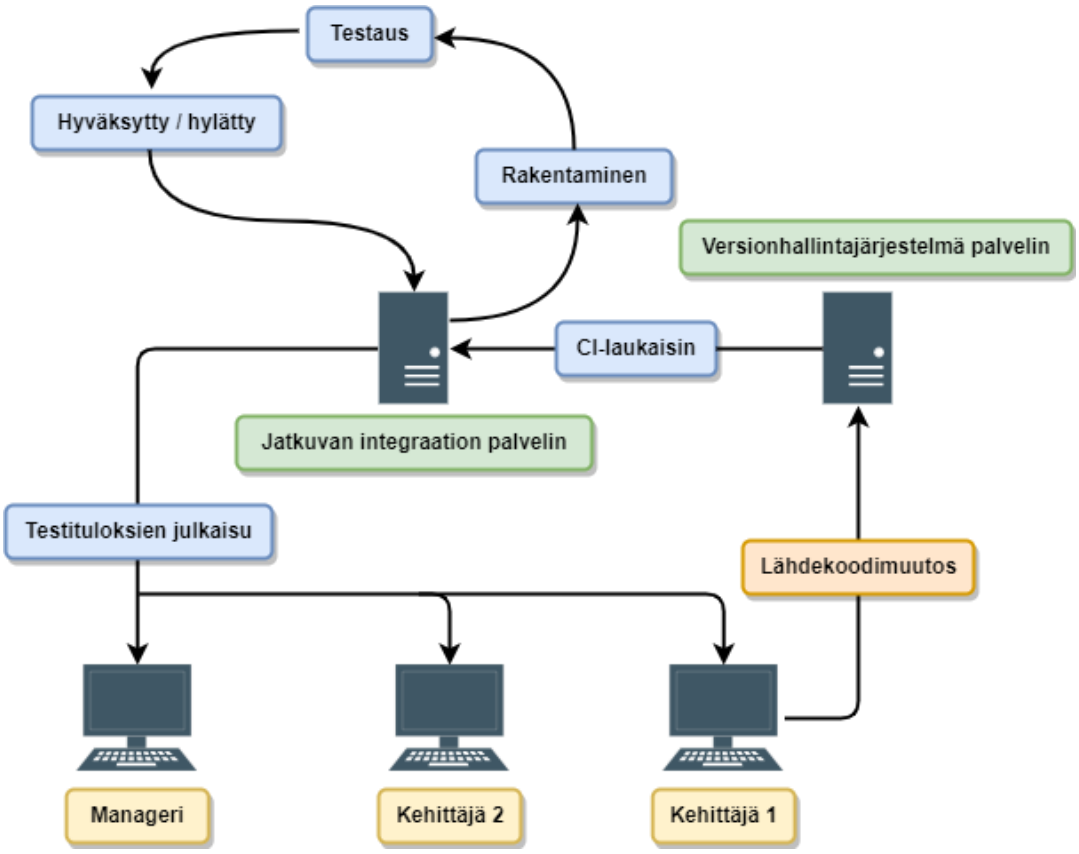
Nykypäivänä DevOps-toimintamalli pitää sisällään paljon enemmän kuin kahden eri osaston yhteistyön helpottamista. Toimintamalli pitää sisällään koko ohjelmistokehityksen kaaren ohjelmiston suunnittelusta käytön monitorointiin. Tavoitteena on helpottaa ja automatisoida mahdollisimman monta manuaalista toimenpidettä liittyen ohjelmistokehitykseen. Nykyään esimerkiksi ohjelmiston rakentaminen, testaus ja julkaisu voidaan toteuttaa yhdellä automatisoidulla CI/CD-putkella heti lähdekoodiin tehtyjen muutoksien jälkeen. Avainkäsitteitä

DevOps-toimintamallissa nykyään ovat jatkuva integraatio, jatkuva toimittaminen, jatkuva julkaisu, testiautomaatio ja erilaisten ympäristöjen automatisoitu pystytys. [6.]

## 3.3 Jatkuva integraatio

Jatkuva integraatio on ohjelmistokehityksen toimintatapa, jossa kehittäjät tekevät mahdollisimman useasti ja mahdollisimman pieniä muutoksia koodiin. Muutoksen jälkeen ohjelmisto rakennetaan ja testataan automatisoidusti, ennen kuin muutoksia viedään ohjelmiston pääversioon. Tämän ansiosta kehittäjät pystyvät julkaisemaan muutoksia nopeammin kuin aikaisemmin, ja toimintatapa myös vähentää riskiä, jossa kehittäjien muutokset ovat konfliktissa keskenään integraatiovaiheessa. [7.]

Jatkuvan integraatio vähentää riskiä, joka muodostuu, kun versionhallinnan päähaaraan viedään samanaikaisesti useita muutoksia. Samanaikaisesti tehdyt muutokset lähdekoodiin lisäävät riskiä, että nämä muutokset vaikuttavat toisiinsa negatiivisesti. Automatisoidulla ohjelmakoodin rakentamisella ja testaamisella säästetään kehittäjien sekä testaajien resursseja. Parhaassa tapauksessa jatkuva integraatio parantaa ohjelmistokehittäjien tehokkuutta ja tuotetun koodin laatua. Jatkuvan integraation käytöllä saavutetaan parempi testauskattavuus, kun osa ohjelman testauksesta on automatisoitua ja manuaalitestaajien ei tarvitse kuluttaa aikaa näiden testien suorittamiseen. Kuvassa 3 on kuvaus jatkuvan integraation vaiheista, jotka alkavat lähdekoodiin ehdotetusta muutoksesta. [7; 8.]




Kuva 3. Jatkuvan integraation työnkulku. [20]

## 3.4 Jatkuva toimittaminen ja jatkuva julkaisu

Jatkuva toimittaminen eli *Continuous Delivery* on ohjelmistokehityksen toimintatapa, joka on jatkoa jatkuvalle integraatiolle. Jatkuvassa toimittamisessa koodin rakentaminen ja testauksen lisäksi automatisoidaan ohjelmiston valmistelu julkaisua varten. Julkaisu voi tapahtua esimerkiksi nappia painamalla, mutta on silti manuaalinen toimenpide. [9; 10.]

Jatkuva julkaisu eli *Continuous Deployment* ja jatkuva toimittaminen sekoitetaan useasti keskenään. Jatkuvassa julkaisussa myös ohjelmiston julkaisu automatisoidaan. Tämä tarkoittaa, että jos automatisoitu rakentaminen ja automaattisesti suoritettavat testit menevät läpi onnistuneesti, niin myös julkaisu tapahtuu automaattisesti. [10.]

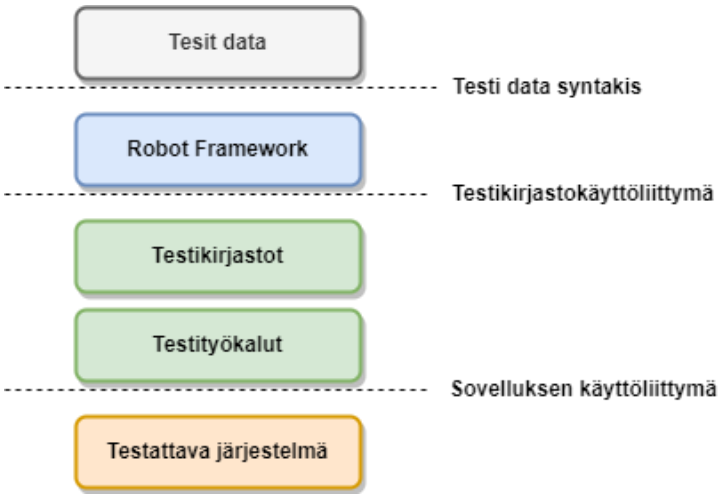
Jatkuvan toimittamisen ja jatkuvan julkaisun suorat hyödyt ovat riskien vähentäminen liittyen muutoksien julkaisussa tuotantoon. Näillä toimintatavoilla onnistutaan pienentämään yksittäisien julkaisujen kokoa, ja tämä helpottaa virheiden korjaamista. Näin myös asiakaspalautteen saa aikaisemmassa vaiheessa käyttöön, jonka avulla esimerkiksi uusien ominaisuuksien hyöty voidaan todeta aikaisemmassa vaiheessa. [10.]

# 4 Robot Framework

Robot Framework on laajennettava avainsana ohjattu automaatiokehys, joka on toteutettu Python-ohjelmointikielellä. Kehystä voidaan käyttää esimerkiksi testiautomaation tai ohjelmistorobotiikan kehittämiseen. [11.]

Robot Framework soveltuu ympäristöihin, joissa automaatio edellyttää monien erilaisten teknologioiden käyttöä. Robot Framework on avoimen lähdekoodin automaatiokehys, joka on julkaistu Apache License 2.0:n alaisuudessa ja jonka kehityksestä vastaa Robot Framework Foundation. Robot Framework käyttää yksinkertaista syntaksia ja ominaisuuksia voidaan laajentaa käyttämällä ulkoisia tai itse toteutettuja kirjastoja. [11.]

Robot Framework on geneerinen, sovelluksesta ja teknologiasta riippumaton kehys. Ydinkehys ei tiedä testattavasta kohteesta mitään, ja vuorovaikutus kohteeseen tapahtuu kirjastojen avulla. Todelliset testiautomaatio-ominaisuudet saavutetaan käyttämällä erilaisia kirjastoja. Kuvassa 4 näkyy yksinkertainen kuvaus Robot Frameworkin arkkitehtuurista, missä testikirjastot ja työkalut hoitava vuorovaikutuksen testattavan järjestelmän kanssa, ja ydinkehys välittää testidataa testikirjastoille ja työkaluille. [11.]

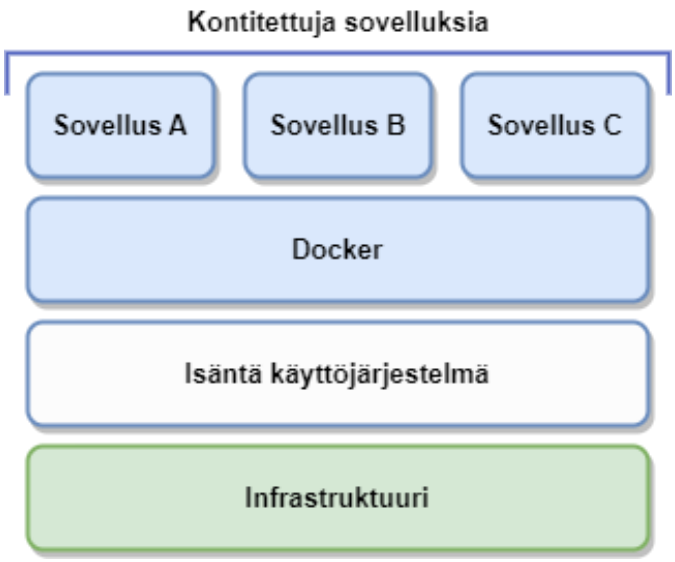



Kuva 4. Robot Frameworkin arkkitehtuuri. [11]

QWeb on avoimen lähdekoodin Robot Framework -kirjasto, joka mahdollistaa web-käyttöliittymien ja web-sovelluksien testauksen. QWeb pohjautuu Selenium WebDriver -ajuriin. QWeb-avainsanat eivät usein tarvitse XPath-paikantimia päästäkseen käsiksi web-elementteihin. Tämä helpottaa testiautomaatiotestien kehittämistä ja ylläpitoa. XPath-paikantimet muuttuvat useammin kuin käyttöliittymässä näkyvät tekstit ja niiden etsiminen on työläämpää. Tekstin käyttäminen paikantimena ei kuitenkaan ole aina mahdollista, ja tästä syystä QWeb tukee myös XPath-paikantimia. [12.]

# 5 Docker

Docker on avoimen lähdekoodin käyttöjärjestelmävirtualisointityökalu, joka avulla kehittäjät voivat luoda, käyttöönottaa ja ajaa sovelluksia Docker-konttien sisällä. Konteissa on kaikki sovelluksen tarvittavat riippuvuudet, kuten kehykset ja kirjastot. Konttien etu on, että ne ovat kevyempiä kuin virtuaalikoneet ja konteja on helpompi siirtää ympäristöstä toiseen. Tästä on etenkin hyötyä uusien ratkaisujen käyttöönottovaiheessa, koska testiympäristössä testatturatkaisu halutaan toimivan samalla tavalla myös tuotantoympäristössä. Kuvassa 5 näkyy yksinkertainen kuvaus siitä, kuinka Docker-moottori välittää isäntä käyttöjärjestelmän resursseja Docker-konteille. [13; 14.]




Kuva 5. Docker-arkkitehtuuri [14.]

Docker toimii Docker-moottorin avulla, joka perustuu asiakas-palvelin arkkitehtuuriin. Asiakkaan ja palvelimen välinen kommunikaatio tapahtuu REST-rajapinnan kautta. [13; 14.]

## 5.1 Docker-kuva

Docker-kuva on vain lukumuotoinen malli, jossa on ohjeet Docker-kontin luomiseen. Kuvamääritys ja luominen tapahtuu Dockerfile-tiedoston avulla. Dockerfile-tiedostossa määritellään vaiheet Docker-kuvan muodostamiseen. Kuvan määritys alkaa määrittämällä Docker-kuva tai käyttöjärjestelmä, johon uusi kuva tulee pohjautumaan. Tämän jälkeen tehdään tarvittavat asennukset ja määritykset. Viimeisenä määritellään, mitä tapahtuu Docker-kontin käynnistyessä. [14.]

## 5.2 Docker-kontti

Docker-kontti on Docker-kuvan ajettava instanssi. Kontteja voi luoda, käynnistää, pysäyttää, siirtää tai poistaa käyttämällä Docker-rajapintaa tai Docker-komentorivityökalua. Docker-kontit ovat kevyitä ja nopeita virtuaalikoneisiin verrattuna, sillä ne hyödyntävät isäntäkoneen resursseja suoraan ilman varsinaista

virtualisoimista, mutta silti eristettynä isäntäkoneesta. Virtuaalikoneet käyttävät myös isäntäkoneen resursseja, mutta virtualisoituina. Tämä tuo isäntäkoneen resurssien ja niitä hyödyntävän järjestelmän välille lisää kommunikaation kerroksia, jonka läpi resurssien hyödyntäminen tapahtuu. Lisäkerrokset lisäävät viivettä, joka tekee resurssien hyödyntämisestä tehottomampaa kuin niiden suora hyödyntäminen. [14.]

Kontit mahdollistavat myös niin sanotun mikropalveluarkkitehtuurin, jossa ohjelmiston osat voidaan ottaa käyttöön ja skaalata asteittain. Tämä on iso etu verrattuna siihen, että jouduttaisiin skaalaamaan koko isoa ohjelmaa yhden komponentin takia. [14; 15.]

# 6 Azure DevOps

Azure DevOps on Microsoftin tarjoama ja ylläpitämä kehittäjille suunnattu ympäristö, jonka sisältää projektin hallintaan ja ohjelmistokehitykseen soveltuvia palveluita. Ympäristön tarjoamia palveluita ovat muun muassa projektin toiminnanohjaus, automaatioputkien rakentaminen, versionhallinnanjärjestelmä ja testauksen suunnittelu. [16.]

## 6.1 Azure Artifacts

Azure Artifacts on Azure DevOps -ympäristön sisäinen palvelu, joka tarjoaa pakettienhallintajärjestelmän. Järjestelmällä kehittäjät pystyvät hallinnoimaan paketteja ja jakamaan koodia. Järjestelmä tukee paketteja, kuten NuGet-, npm-, Maven- ja Python-paketteja. Azure Pipeline -palvelun tarjoamat jatkuvan integraation ja jatkuvan kehityksen putket voivat käyttää palvelun tarjoamia ominaisuuksia. [17; 18.]

## 6.2 Azure Boards

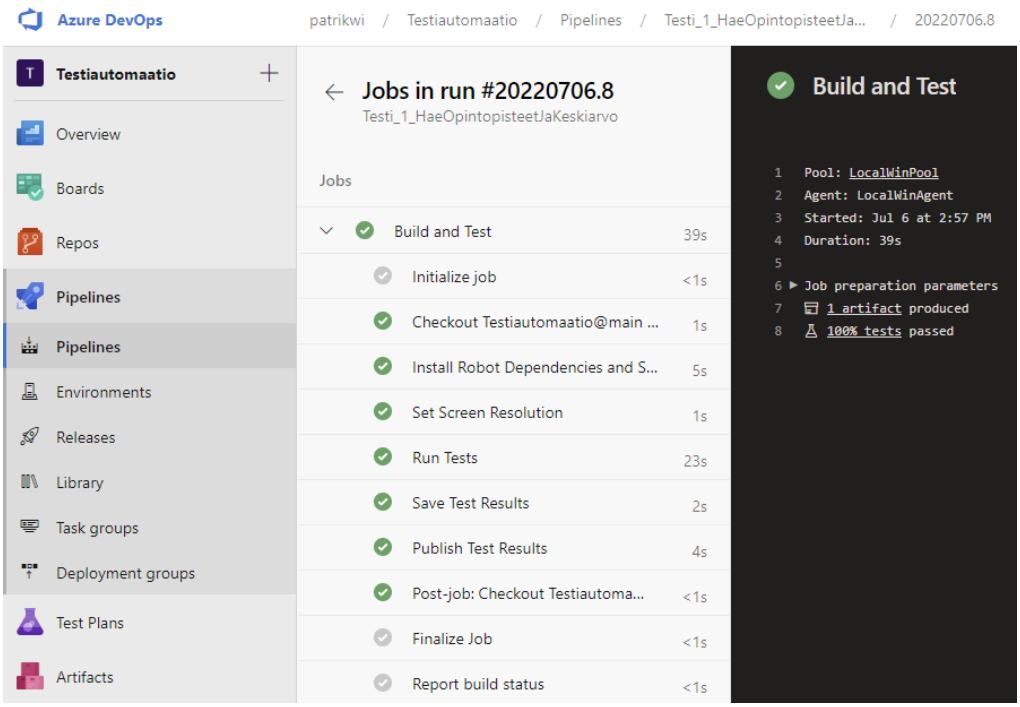
Azure Boards -palvelu tarjoaa ohjelmistokehitystiimeille interaktiivisen ja mukautettavan toiminnanohjausjärjestelmän, jota tarvitaan ohjelmistoprojektien

hallintaan. Järjestelmä tarjoaa natiivin tuen Agile-, Scrum- ja Kanban-ketterän kehityksen menetelmille. Järjestelmä tarjoaa suuren määrän ominaisuuksia ja työkaluja projektien hallinnoimiseen. [19.]

## 6.3 Azure Pipelines

Azure Pipelines -palvelu tarjoaa mahdollisuuden toteuttaa jatkuvan integraation ja jatkuvan toimittamisen putkia. Palvelu tukee useita eri versionhallinnanjärjestelmiä, mutta parhaiden tuettuja ovat Azure Repos, Bitbucket ja GitHub. YAML-kielellä toteutetut putket on tuettu vain edellä mainituissa versionhallintajärjestelmissä. Palvelu toimii monella eri ohjelmointikielellä ja alustalla, sekä toimii useiden avoimeen lähdekoodiin perustuvien projektien kanssa. [19.]

Putkien luominen onnistuu palvelun Pipeline-näkymän kautta. Putki on joukko suoritettavia tehtäviä. Tehtävä voi olla mikä tahansa sallittu prosessi isäntäkoneella, kuten testiautomaatiotestien suoritus. Tehtävillä voi olla useita ja erilaisia esiehtoja suoritukselle. Ehto voi olla esimerkiksi tietty muuttujan arvo, jonka putki saa suorituksen aikana tai aikaisemmin suoritettavan tehtävän onnistuminen. Kuvassa 6 näkyy testiautomaatioputken onnistunut suoritus. Putki hakee testit versionhallintajärjestelmästä, asentaa tarvittavat riippuvuudet ja suorittaa testit. Tämän jälkeen putki tallentaa ja julkaisee testien tuottamat tulokset Azure DevOps -palveluun. [20.]




Kuva 6. Esimerkki onnistuneesti suoritetusta testiautomaatioputkesta.

Putken voi määrittää kokonaan käyttämällä palvelun verkkokäyttöliittymää, jolloin määritys tallentuu ainoastaan luotuun putkeen, mutta putken pohjana voidaan käyttää myös YAML-tiedostoa. YAML-tiedoston avulla putken määritys tai osa määrityksestä voidaan tallentaa versionhallintajärjestelmään. Näin uusien samankaltaisien putkien luominen on nopeampaa ja vanhojen ylläpitäminen helpompaa. [20.]

Usea testiautomaatio putki voi käyttää samaa YAML-tiedostoa pohjana ja ajettavat testit voidaan määrittää palvelun tarjoamien putki kohtaisien muuttujien avulla. Näin muuttamalla yhtä YAML-tiedostoa pystymme vaikuttamaan kaikkiin sitä pohjana käyttäviin testiautomaatioputkiin. Tästä on etenkin hyötyä, kun tehdään muutoksia putken suorittamiin tehtäviin. Putket tallentuvat Azure Pipelines -palveluun, jossa niille on oma kansiorakenne ja niiden suorituksien seuraaminen onnistuu palvelun verkkokäyttöliittymästä. [20.]

Agentti on ohjelmisto, joka suorittaa putkien sisältämät tehtävät. Agentti pystyy suorittamaan yhtä putkea kerrallaan ja putken suorituksen aikana tulleet uudet

suorituspyynnöt menevät jonoon odottamaan, että agentti vapautuu. Agentti voidaan asentaa erilaisiin ympäristöihin, kuten virtuaalikoneelle tai omalle henkilökohtaiselle koneelle. Microsoft tarjoaa myös heidän ylläpitämiä agentteja. Agentit ovat osa agenttipoolia, joka voi sisältää useamman agentin. [21.]

Microsoftin ylläpitämät agentit päivitetään ja ylläpidetään. Microsoftin ylläpitämiä agentteja käyttäessä jokaiselle putken suoritukselle tulee uusi uniikki virtuaalikone ja agentti. Virtuaalikone ja agentti tuhotaan, kun putki on suoritettu. Näistä syystä Microsoftin ylläpitämät agentit eivät välttämättä sovellu kaikkiin käyttötarkoituksiin. [21.]

Itse ylläpidetty agentti mahdollistaa paremman hallinnan agentin ympäristöstä, kuten ohjelmistojen asennuksesta ja ohjelmistopakettien versioista. Lisäksi konetason välimuisti ja määritykset säilyvät, mikä saattaa lisätä nopeutta. Agentteja voidaan asentaa tietokoneille, jotka käyttävät on Windows-, Linux- tai macOS-käyttöjärjestelmää. Agentin voi myös asentaa Docker-kontin sisälle. [21.]

## 6.4 Azure Repos

Azure Repos -palvelu on versionhallintajärjestelmän, jonka avulla voidaan hallita projektin lähdekoodia. Versionhallintajärjestelmät ovat ohjelmistoja, joiden avulla voidaan seurata koodiin tehtyjä muutoksia ajan mittaan. Versionhallintajärjestelmät tallentavat tilannekuvan pysyvästi jokaisen muutoksen jälkeen. Tilannekuva voidaan ottaa myöhemmin käyttöön, jos tälle on tarvetta. [22.]

Azure Repos tukee kahdentyyppistä versionhallintaa: GIT hajautettua versionhallintaa ja TFVC keskitettyä versionhallintaa. GIT on nykyään yleisimmin käytetty versionhallintajärjestelmä ja on nopeasti muodostumassa standardiksi. [22.]

### 6.5 Azure Test Plans

Azure Test Plans -palvelu tarjoaa verkkoselain pohjaisia testauksenhallinta- ja seurantatyökaluja. Palvelussa voidaan määrittää muun muassa manuaalitestausta, tutkivaa testausta ja hyväksyntätestausta. Palvelu on myös integroitu toimimaan yhdessä Azure Pipelines -palvelun kanssa. Testauksenhallintatyökaluja voidaan käyttää jatkuvan integraatioon ja jatkuvan toimittamiseen liittyvän testauksen tukena. [23.]

## 7 Toteutus

Tavoitteena oli toteuttaa ympäristö, joka soveltuu testiautomaatio testien kehittämiseen ja suorittamiseen sekä projektin hallinnoimiseen. Ympäristön tuli olla mahdollisimman helppokäyttöinen ja yksinkertainen, jotta kehittäjien työ olisi mahdollisimman suoraviivaista.

Ympäristön luominen alkoi Azure DevOps -ympäristön käyttöönotolla. Tämän jälkeen palveluun luotiin projekti, joka sisältää muita Azure DevOps -ympäristön tarjoamia palveluita. Projektin luomisen jälkeen oli vuorossa Azure Repos -palvelun tarjoaman versionhallintajärjestelmän käyttöönotto. Versionhallinnan käyttöönoton jälkeen oli vuorossa itse hallinnoidun agentin asennus ja konfigurointi. Seuraavaksi oli vuorossa Docker Desktop -ohjelman lataus sekä asennus ja Docker-kuvan määritys. Docker-kuvan määrityksen jälkeen kirjoitettiin Shell -skripti, jolla testiautomaatio testit käynnistetään Docker-kontin käynnistyessä. Tämän jälkeen luotiin yksinkertainen web-käyttöliittymä testiautomaatio testi. Seuraavaksi luotiin automaatioputket käyttämällä Azure Pipelines -palvelua. Viimeisenä on vuorossa putkien ajaminen ja tuloksien analysointi.

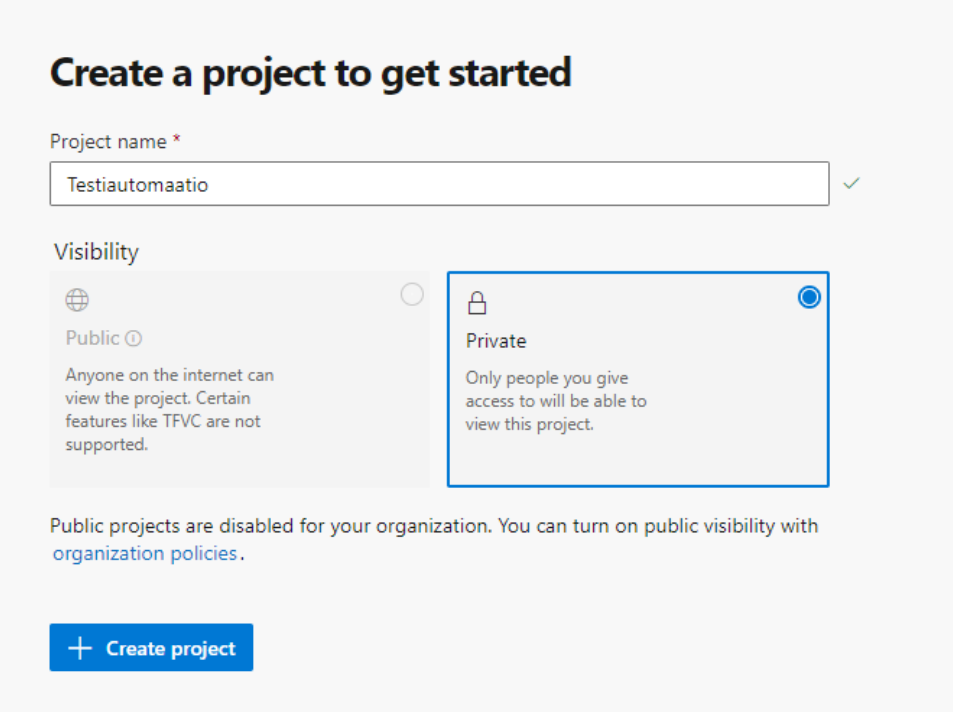
### 7.1 Azure DevOps -ympäristön käyttöönotto

Azure DevOps -ympäristön käyttöönotto alkoi Azure DevOps -palveluun kirjautumisella, joka edellyttää Microsoft-tiliä. Ensimmäisen kirjautumisen jälkeen käyttäjälle luodaan automaattisesti organisaatio, mikäli tiliä ei ole jo liitetty

organisaatioon. Organisaation asetukset ja määritykset koskevat kaikkia sen sisältämiä projekteja.

## 7.2 Projektin luonti ja versionhallintajärjestelmän käyttöönotto

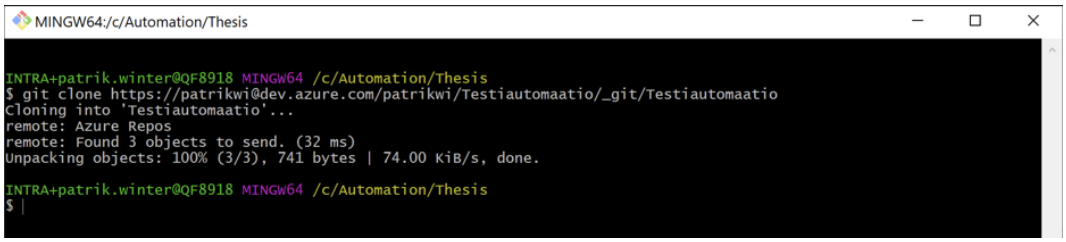
Projekti luotiin suoraan organisaation etusivulta. Etusivulla on nappi ”Create project”, jota painamalla luotiin ensimmäinen projekti. Napin painamisen jälkeen projektille määritellään pakollinen nimi ja näkyvyys. Näkyvyys voi olla julkinen, eli koko internetin nähtävillä tai yksityinen, jolloin projekti on näkyvissä vain kutsutuille. Projektille voidaan myös antaa kuvaus, valita versionhallintajärjestelmä ja määrittää haluttu projektin kehityksen lähestymistapa. Kuvassa 7 näkyy näkymä projektin luomisvaiheesta, jossa määriteltiin Azure DevOps -projektin nimi ja näkyvyys.




Kuva 7. Projektin luominen.

Projektin luomisen jälkeen oli vuorossa versionhallintajärjestelmän käyttöönotto. Käyttöönotto alkoi avaamalla palvelun Repos-näkymä. Näkymästä painettiin nappia ”Initialize” ja valittiin GIT, joka alusti projektin Azure Repos -palvelun

versionhallintajärjestelmään. GIT-repositorion nimeksi tuli automaattisesti Azure DevOps -projektin nimi. Tämän jälkeen luotiin PAT, jota tarvittiin repositorion kloonaamisen yhteydessä. Kuvassa 8 näkyy GIT-repositorion kloonaaminen lokaalille laitteelle.




Kuva 8. GIT-repositorion kloonaaminen.

## 7.3 Robot Framework -testi

Seuraavaksi oli vuorossa toteuttaa yksinkertainen testiautomaatiotesti käyttämällä Robot Framework -kehystä ja QWeb-kirjastoa. Testin tekemistä varten repositorioon luotiin kaksi kansiota, jotka sisälsivät kolme robot-tiedostosta. Kansioiden nimiksi annettiin r*esources* ja *tests*. Resources-kansioon luotiin tiedostot *keywords.robot* ja *resources.robot*. Tests-kansioon luotiin testitiedosto nimeltä *HaeSuoritetutOpintopisteetOpiskelija.robot*. (liite 1).

*Keywords.robot*-tiedosto toteutettiin avainsanat, joiden tarkoitus on suorittaa useassa testissä toistettavia prosesseja. Esimerkkikoodissa 1 näkyvä avainsana on toteutettu käyttämällä QWeb-kirjaston tarjoamia avainsanoja, joilla haetaan opiskelijan keskiarvo ja opintopisteiden määrä. Tämän jälkeen avainsana palauttaa haetut tiedot testitapauksen käyttöön. (liite 2).

```
*** Settings ***
Documentation      Robot Framework keywords

*** Keywords ***
OpTy_HaeOpintopisteetJaKeskiarvo
     [Documentation]    Get student points and GPA.
        ClickElement           //*[@id\= "switch-general"]
        ClickText              Opiskelijan työpöytä
        ClickText              HOPS
```

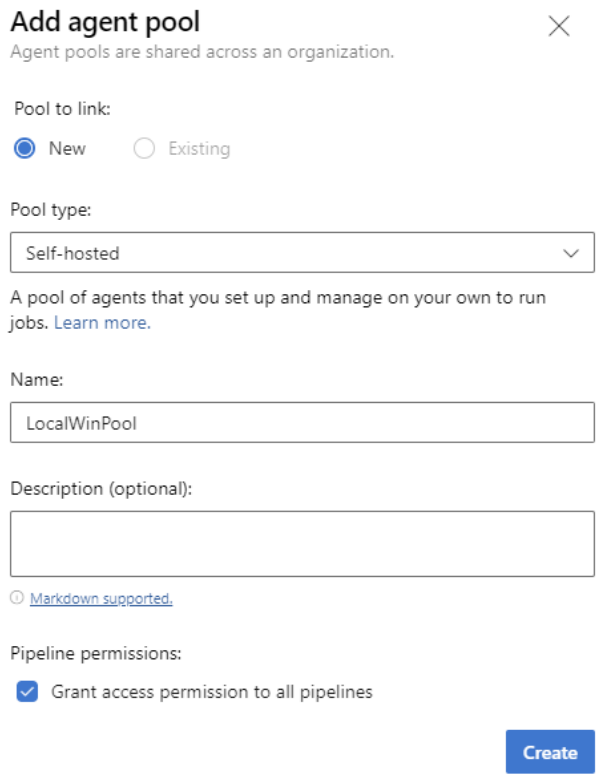
```
${pisteet}=           GetText  Opintopisteitä yhteensä:
...                   between=Opintopisteitä yhteensä: ???
${keskiarvo}=         GetText  Keskiarvo:
...                   between=Keskiarvo: ???
LogScreenshot

[Return]  ${pisteet} ${keskiarvo}
```

Esimerkkikoodi 1. Esimerkki avainsanasta.

*Resources.robot*-tiedosto sisältää avainsanoja liittyen verkkoselaimen alustamiseen ja sulkemiseen sekä OMA-palveluun kirjautumis- ja uloskirjautumisavainsanat. (liite 3).

## 7.4 Agenttipoolin lisääminen

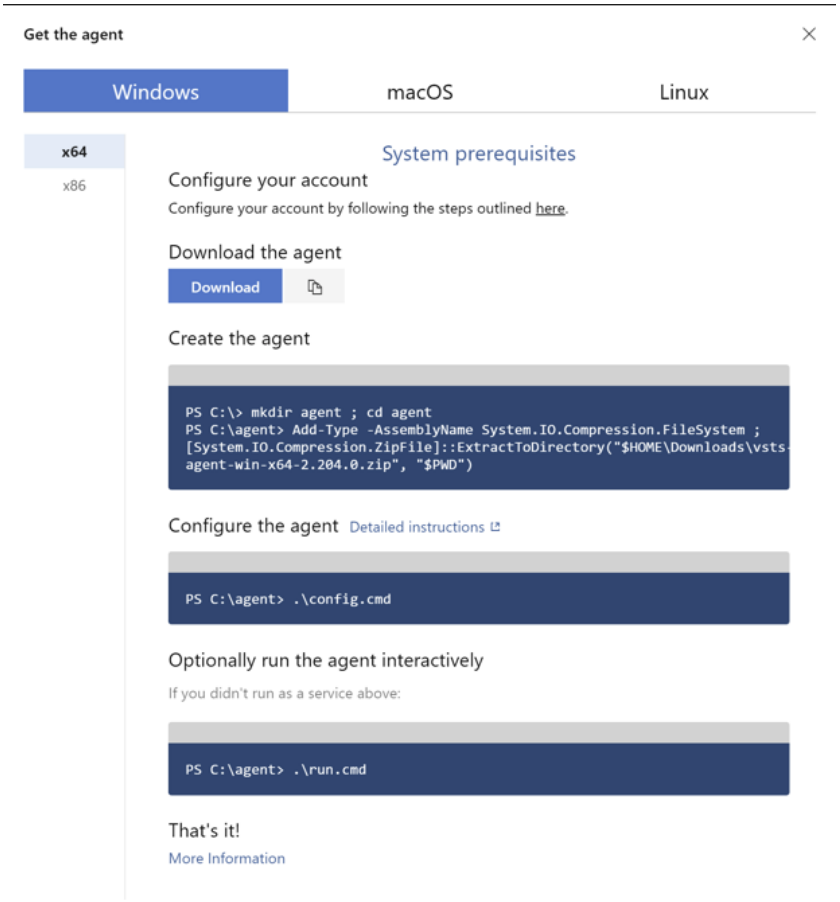
Pooli lisättiin Azure DevOps -projekti asetuksista löytyvän osion alta. Näkymästä löytyy nappi ”Add pool”. Poolin luonnin yhteydessä valittiin poolin tyypiksi ”Self-hosted”. Vaihtoehto olisi ollut Azure Scale Set, joka on pooli automaattisesti työmäärän mukaan skaalautuville virtuaalikoneille, mutta tämän käyttöön saaminen on maksullista. Poolin nimeksi annettiin LocalWinPool ja valittiin määritys, että kaikki Azure Pipelines -putket voivat käyttää poolia. Kuvassa 9 näkyy agentti poolin määritysvaihe.




Kuva 9. Agenttipoolin luonti.

## 7.5 Agentin asentaminen

Seuraavaksi oli vuorossa agentin asennus tietokoneelle. Agentin asentaminen alkoi avaamalla agenttipoolin asetukset -sivu ja painamalla nappia ”New agent”. Tämä tuo esille ohjeet agentin asentamiseen ja linkin agentin asennustiedoston lataamiseen. Agentti asennettiin tietokoneelle, jonka haluttiin suorittavan automaatioputket, joka oli tässä tapauksessa henkilökohtainen tietokone, jonka käyttöjärjestelmä oli Windows 10. Kuvassa 10 näkyy näkymä, josta Windows-agentin asennustiedoston voi lataa. Kuvassa näkyy myös tarvittavat komennot, joilla agentti asennetaan, määritetään ja käynnistetään.

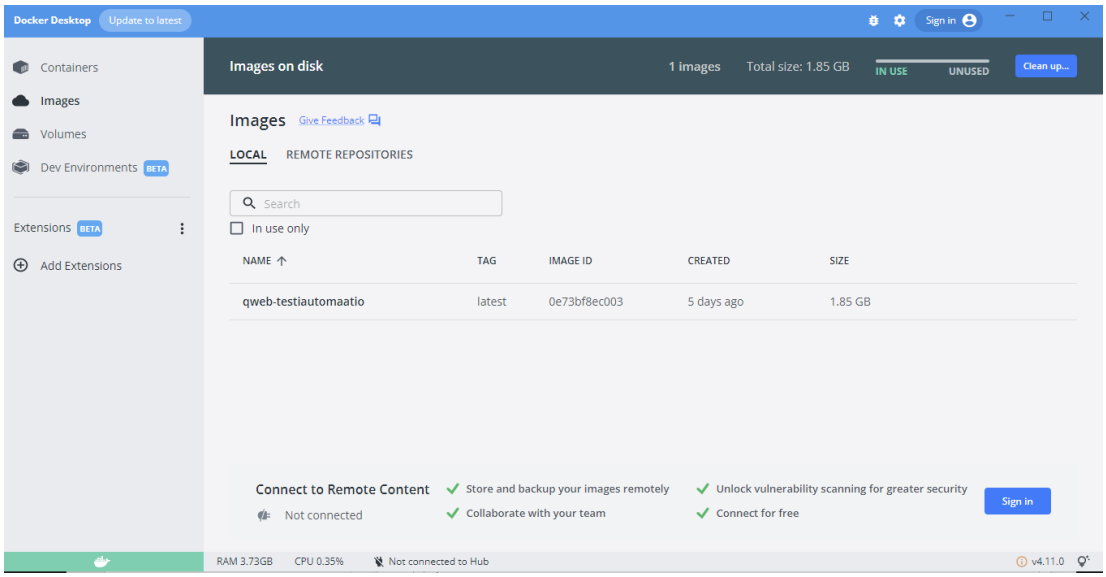



Kuva 10. Agentin asennusohjeet

Agentti asennettiin suorittamalla kuvan 10 ohjeissa olevat komennot henkilökohtaisen Windows-koneen PowerShell-komentorivillä. Agentti käynnistettiin interaktiivisena, jotta web-käyttöliittymän testiautomaatio testien suoritus olisi mahdollista.

## 7.6 Docker Desktop -ohjelman asennus

Tämän jälkeen oli vuorossa Docker Desktop -ohjelman asennus. Tämä onnistui käyttämällä Docker-yhtiön virallisilta nettisivuilta ladattavaa asennusohjelmaa. Docker Desktop -ohjelma mahdollistaa Docker-konttien ajamisen Windows-ympäristössä. Kuvassa 11 näkyy Docker Desktop -ohjelman käyttöliittymä.




Kuva 11. Docker Desktop -ohjelma.

Windows-käyttöjärjestelmän sisältämän WSL-ominaisuuden ansiosta Windows-ympäristössä voidaan ajaa Linux-terminaalia ja tämä mahdollistaa Linux-käyttöjärjestelmää käyttävien konttien ajamisen. Linux-terminaalin pystyi lataamaan ja asentamaan Microsoft Store -kauppan kautta.

## 7.7 Docker-kuvan määrittäminen

Seuraavaksi oli vuorossa Docker-kuvan määrittäminen. Tämä tapahtui luomalla dockerfile-tiedosto ja määrittämällä tiedostoon testiautomaatio kontin vaatimat asennukset ja muut määritykset. Dockerfile-tiedosto tallennettiin projektin repositoriossa sijaitsevaan docker-kansioon. (liite 4).

Dockerfile-tiedostoon määriteltiin kontin vaatimat asennus- ja määritysvaiheet sekä skriptin nimi, joka suorittaa robot-tiedostojen testit. Asennukset ja määritykset tehtiin dockerfile-tiedoston syntaksin käyttämillä käskyillä. Tiedosto sisälsi yhteensä viisitoista käskyä. Käskyt tekevät seuraavat asiat:

- Ottaa käyttöön Docker -pohjakuvan Ubuntu 20.04 ”FROM” käskyllä.

- Luo ”build-time”-muuttja, joka sisältää halutun Chrome-selain versionumeron ”ARG”-käskyllä.

- Luo ympäristömuuttujat käytetyille tiedostopoluille ja määritä käyttöliittymä epäinteraktiiviseksi ”ENV”-käskyillä.

- Lataa ja asentaa testiautomaation vaatimat paketit käyttämällä apt-komentorivityökalua. Komennot suoritetaan ”RUN”-käskyllä.

- Lataa ja asentaa Chrome-selain käyttämällä hyväksi aikaisemmin määriteltyä muuttujaa, joka sisältää halutun Chrome-selaimen versionumeron. Komennot suoritetaan ”RUN”-käskyllä.

- Lataa ja asentaa Robot Framework -kehys ja QWeb-kirjastoa käyttämällä Pythonin pakettimanageria. Komennot suoritetaan ”RUN”-käskyllä.

- Asettaa oikean aikavyöhykkeen käyttämällä juuri asennettua tzdata-pakettia. Komennot suoritetaan ”RUN”-käskyllä.

- Muuta testiautomaatio -skriptin rivinvaihto Unix-muotoon juuri asennetun dos2unix-paketin avulla. Komennot suoritetaan ”RUN”-käskyllä.

- Luo testiautomaation vaatimat kansiot. Anna avoimet käyttöoikeudet kansioihin, sekä avoin suoritusoikeus testiautomaatio -skriptiin. Komennot suoritetaan ”RUN”-käskyllä.

- Ottaa käyttöön työkansion ”WORKDIR”-käskyllä.

- Viimeisenä määrittää testiautomaation suoritusskriptin kutsu, joka suoritetaan kontin käynnistyessä ”CMD”-käskyllä

## 7.8 Testiautomaation käynnistys skriptin kirjoittaminen

Seuraavaksi luotiin Shell-skripti, jonka testiautomaatiokontti suorittaa käynnistyessään. Skripti tekee kolme tärkeää asiaa kontin sisällä. Määrittää ja tarkistaa tarvittavat muuttujat testille. Käynnistää virtuaalinäytön ja ikkunamanagerin. Lopuksi skripti käynnistää testiautomaation suorittamisen. (liite 5).

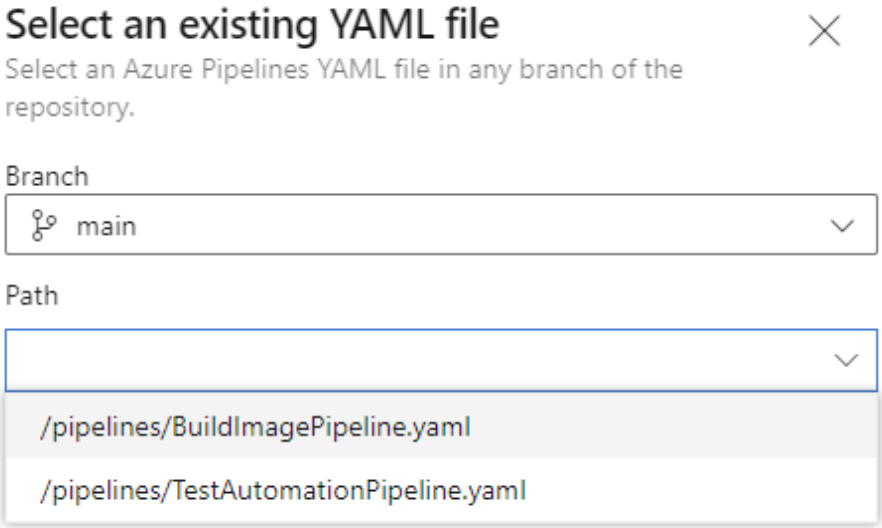
## 7.9 Docker-kuvan luominen putken avulla

Seuraavaksi oli vuorossa Docker-kuvan luominen. Kuva luontia varten tehtiin YAML-tiedosto, jossa määriteltiin agentin suorittamat tehtävät, joilla Docker-kuva luodaan agentin isäntäkoneelle. Tiedosto nimettiin BuildImagePipeline.yaml ja tallennettiin projektin repositoriossa sijaitsevaan pipelines-kansioon, josta sitä käytettiin itse putken luomisvaiheessa. (liite 6).

Putken luominen alkoi Azure Pipelines -palvelun kuvaketta painamalla ja seuraavaksi painamalla nappia ”New pipeline”. Tämän jälkeen valittiin versionhallinta järjestelmäksi Azure Repos ja projektin repositorio, jossa putken määrityksiä sisältävä YAML-tiedosto oli. Tiedostoksi valittiin aikaisemmin luotu BuildImagePipeline.yaml-tiedosto Pipeline-kansiosta. Putki nimettiin samoin kuin YAML-tiedosto, mutta nimi olisi voinut olla mikä vain. Kuvassa 12 näkyy BuildImagePipeline.yaml-tiedoston valinta putken luomisvaiheessa. (liite 6).

Putkien YAML-tiedostossa määritettiin kolme tehtävää, jotka agentti suorittaa järjestyksessä ylhäältä alkaen putken käynnistetään. Suoritettavat työtehtävät ovat:

- Projektikoodin kopiointi versionhallintajärjestelmästä.
- Docker-kuvan luominen, käyttämällä versionhallinnasta kopioidun dockerfile-tiedostoa ja suorittamalla Docker-komentorivi komento, jolla tiedostosta luodaan valmis kuva.
- Vanhojen käyttämättömien Docker-kuvien poisto.




Kuva 12. YAML-tiedoston valitseminen.

Putkeen lisättiin vielä jatkuvan integraation laukaisin, joka käynnistää automaattisesti putken suorituksen, jos projektin repositorion päähaarassa sijaitsevaan dockerfile-tiedostoon tehdään muutos.

Viimein oli aika suorittaa putki, joka luo docker-kuvan agentin isäntäkoneelle. Suorituksen jälkeen docker-kuva oli luotu onnistuneesti ja valmis seuraavaksi toteutettavan testiautomaatioputken käyttöön.

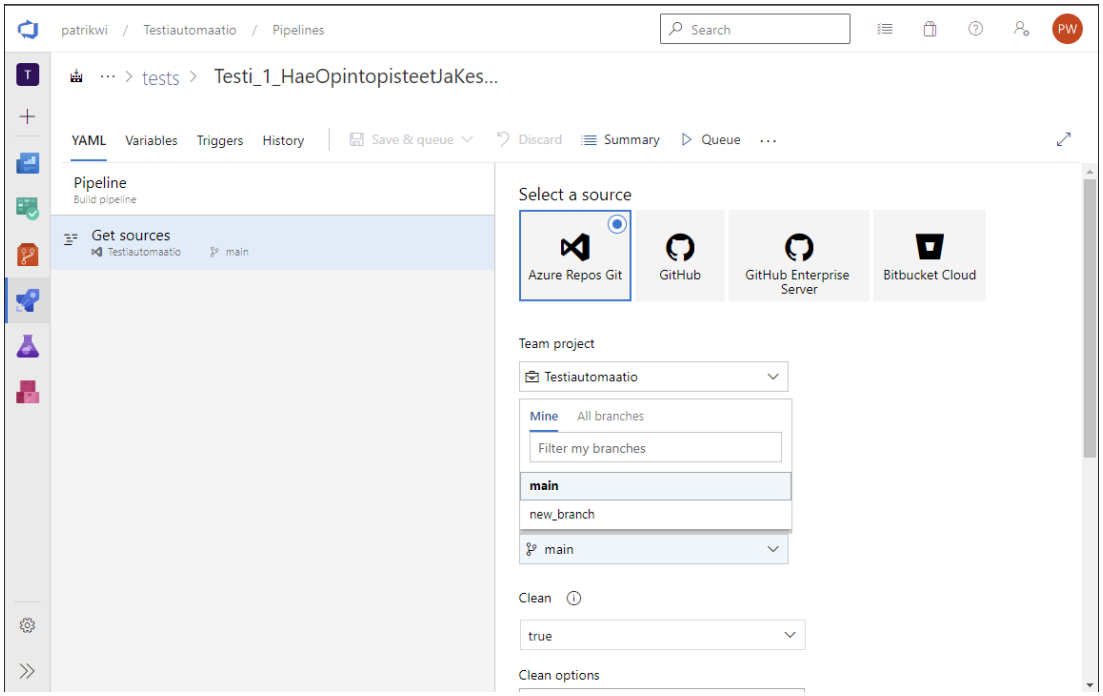
## 7.10 Testiautomaatioputken toteutus

Testiautomaatioputken toteuttamista varten luotiin YAML-tiedosto nimeltä TestAutomationPipeline.yaml. Tämän tiedoston tarkoitus oli olla pohja testiautomaatioputkien toteuttamisessa. Tiedostoon määriteltiin testiautomaatio testin suoritukseen ja sen tuottamien tuloksien julkaisuun tarvittavat tehtävät. Tiedosto tallennettiin repositoriossa sijaitsevaan pipelines-kansioon. (liite 7).

Testiautomaatio putken YAML-tiedostossa määritettiin viisi tehtävää, jotka agentti suorittaa järjestyksessä ylhäältä alkaen putken käynnistetään. Tehtävien kuvaukset ovat seuraavat:

- Projektikoodin kopiointi versionhallintajärjestelmästä.
- Docker-kontin käynnistys, mikä myös käynnistää testin suorituksen.
- Julkaisee suorituksen tuottamat tiedostot artefaktina, myöhempää analysointia varten.
- Julkaisee testien suoritustietoja Tests-välilehdelle.
- Poistaa pysähtyneet Docker-kontit.

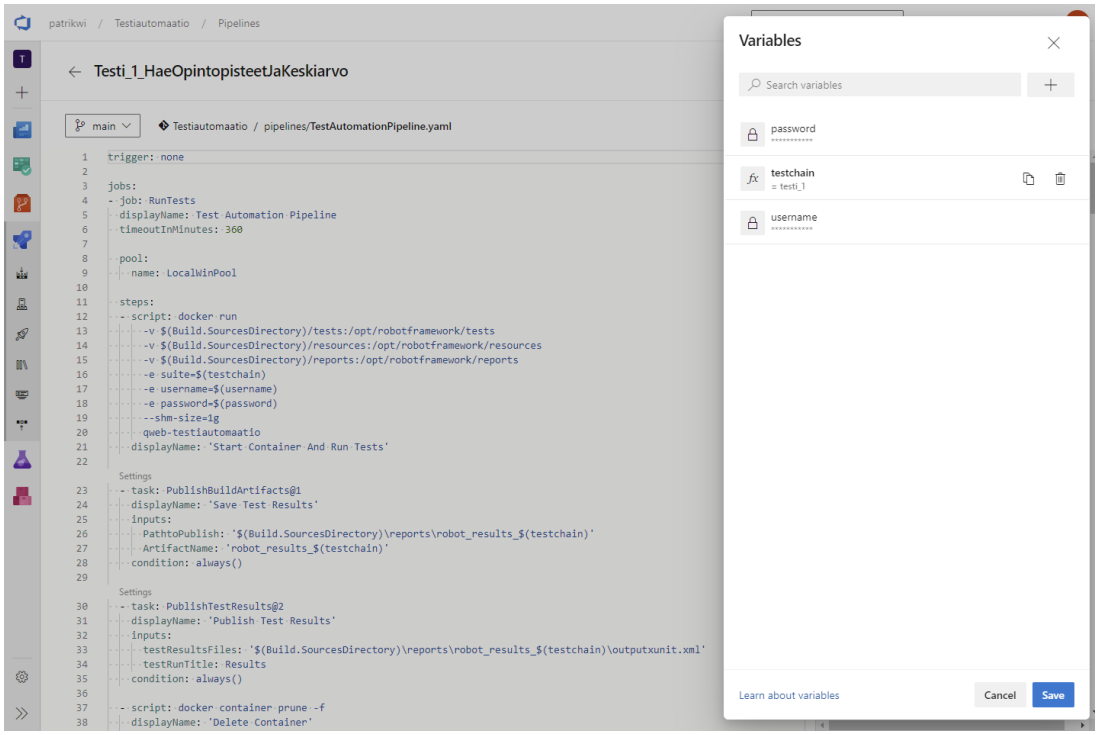
Seuraavaksi oli vuorossa itse testiautomaatio putken luominen TestAutomation-Pipeline.yaml YAML-tiedoston avulla. Testiautomaatioputken nimeksi tuli automaattisesti projektin nimi ja oletus repositorion haara oli päähaara. Nämä muutettiin halutuiksi putken *triggers*-näkymä alta. Kuvassa 13 näkyy *triggers*-näkymä, mistä esimerkiksi putken käyttämä repositorion haara voidaan vaihtaa putken luomisen jälkeen.




Kuva 13. Putken nimen muuttaminen ja repositorion haaran vaihto.

Tämän jälkeen pystyimme määrittää testiautomaatio putken suorittama tietty Robot Framework -testitiedosto projektin repositoriosta antamalla putkelle

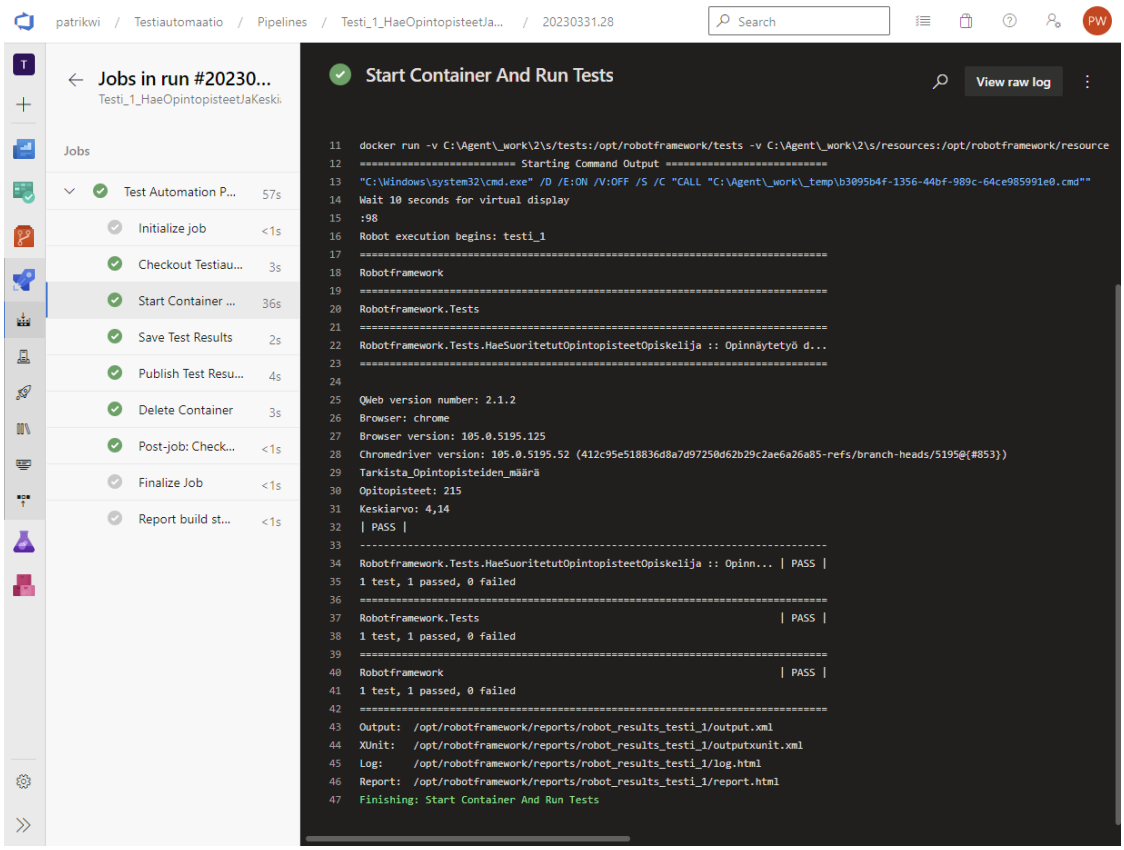
putkikohtainen muuttuja, joka sisälsi testitiedoston *Force Tag* -arvon. Testiautomaatioputki edellytti, että testitiedostoon oli määritetty *Force Tag* -arvon, jolla tiedosto valitaan suoritettavaksi. Testiautomaatioputkelle annettiin myös testin vaatimat käyttäjätunnus ja salasana salattuina putkikohtaisina muuttujia. Viimein kun muuttujat oli määritetty ja tallennettu, oli ensimmäinen testiautomaatio putki valmis suoritettavaksi. Kuvassa 14 näkyy valmis testiautomaatioputki ja sille annetut putkikohtaiset muuttujat.




Kuva 14. Putkikohtaisten muuttujien määritys.

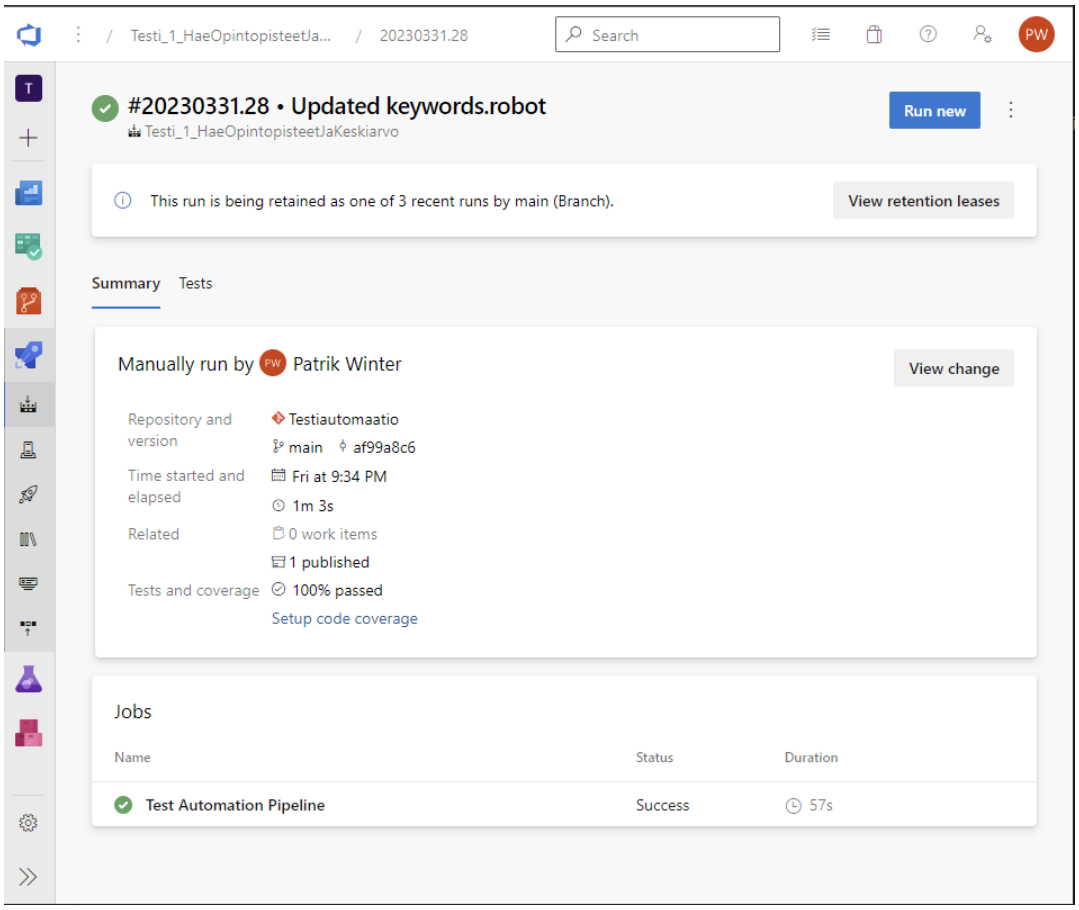
## 7.11 Testiautomaatioputken suorittaminen

Seuraavaksi oli vuorossa testiautomaatioputken suorittaminen ja suorituksen analysointi. Ensimmäisenä tarkastettiin, että kaikki agentin suorittamat työtehtävät suoritettiin halutulla tavalla ja tarkastuksen jälkeen voitiin todeta, että testiautomaatioputki suoritettiin halutulla tavalla. Kuvassa 15 näkyy onnistuneesti suoritettu testiautomaatioputki sekä testin terminaalissa.



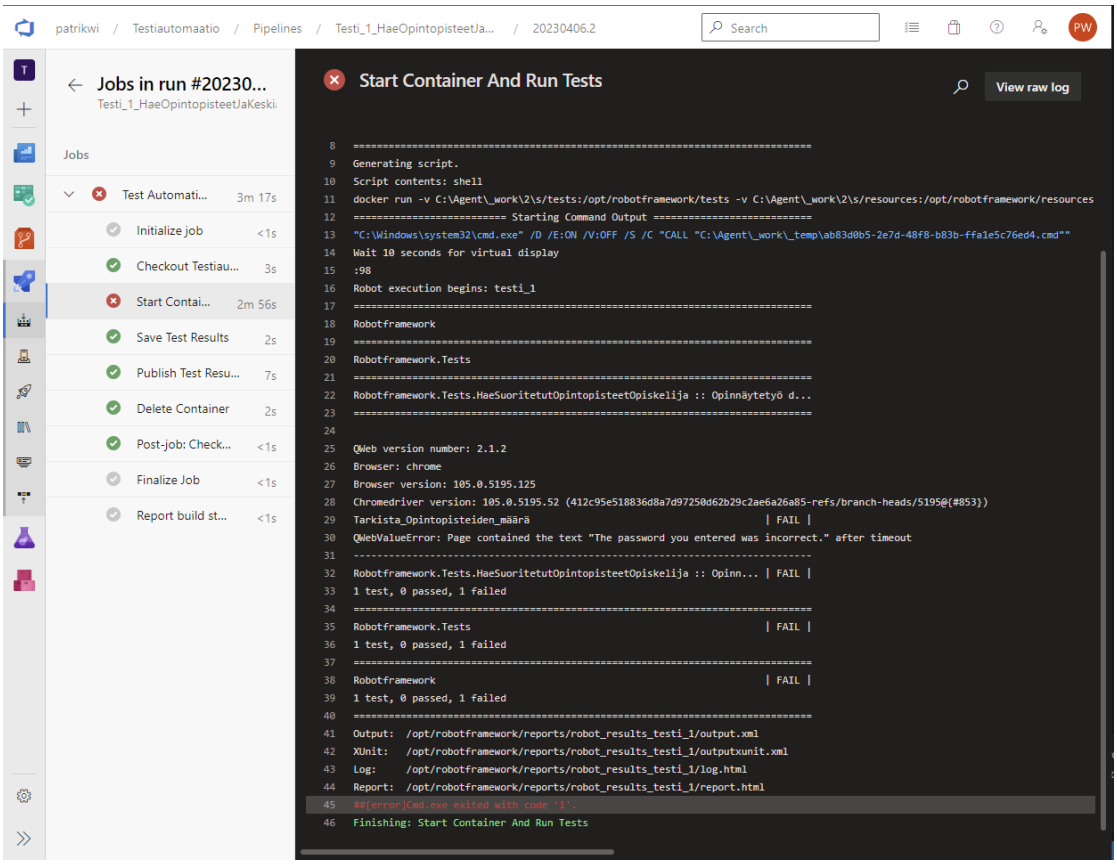

Kuva 15. Onnistuneesti suoritettu testiautomaatioputki.

Tämän jälkeen tarkasteltiin testiautomaatiotestin tuottamia tiedostoja. Tässä tapauksessa Robot Framework -testi tuotti neljä tiedostoa ja yhden kansion, joka sisälsi kuvakaappauksia testi suorittamisesta. Tiedostot olivat *log.html*, *output.xml*, *report.html* ja *xunitoutput.xml.* Tässä tapauksessa testien kannalta tärkeimpiä tiedostoja ovat kuvakaappaukset ja *log.html*. *Log.html* sisältää kaikkien avainsanojen suoritukset ja sen avulla on helpointa paikantaa, missä avainsanassa ja missä kohtaan testiä virhe tapahtui. Testin tuottamat tiedostot löytyivät testin tuottaman osion alta, josta tiedostot pystyttiin lataamaan. Kaikki tiedostot muodostuivat halulla tavalla. Kuvassa 16 näkyy tietoa manuaalisesti käynnistetystä putken suorituksesta ja "1 published"-linkki testin tuottamiin tiedostoihin.




Kuva 16. "1 published"-linkistä pääsee käsiksi suorituksen tuottamiin lokia ja kuva tiedostoihin.

Viimeisenä testattiin, että testiautomaatio putki toimii oikein myös silloin, kun testiautomaatiotesti kohtaa virheen. Tämä testattiin muuttamalla kirjautumiseen vaadittava salasana vääräksi, ja putken suoritus epäonnistui halutulla tavalla. Kuvassa 17 näkyy testin epäonnistumisen testaaminen.




Kuva 17. Testin epäonnistuminen.

# 8 Päätelmä ja jatkokehitysmahdollisuudet

Insinöörityön tavoitteena oli toteuttaa testiautomaatio testien kehittämiseen ja testien ajastettuun suorittamiseen sekä projektin hallinnoimiseen soveltuva ympäristö. Tavoitteena oli, että testien suoritusympäristö olisi mahdollisimman muuttumaton, jotta ympäristön vaikutus testien suorittamiseen olisi mahdollisimman pieni. Viimeinen tavoitteena oli, että ympäristö olisi mahdollisimman helppokäyttöinen kehittäjille ja helppo ylläpitää.

Tavoitteisiin päästäkseen insinöörityössä käytettiin Azure DevOps -palvelua. Azure DevOps -palvelun tarjoama ympäristö oli itselleni entuudestaan tuttu ja tämä oli yksi syy palvelun valinnalle työhön. Azure DevOps -ympäristö tarjoaa myös valmiit työkalut testituloksien seuraamiseen ja analysoimiseen. Esimerkiksi testiautomaatioputken suorittama PublishTestResults@2-tehtävä luo

välilehden testin tuottamista tuloksista. Välilehdelle pystyy myös julkaista testin ottamat kuvakaappaukset. Palvelu yhdistää monta työn vatimaa työkalua ja palvelua. Ympäristössä oli myös hyvät mahdollisuudet jatkokehittämiselle, koska palvelulla on hyvä yhteensopivuus muiden Azure-pilvipalveluiden kanssa.

Toinen mahdollinen palvelu, joilla vastaavanlaisen ympäristön toteuttaminen olisi ollut mahdollista, olisi ollut esimerkiksi GitHub, joka tarjoaa GitHub Actions Runnerin ja GitHub Workflow'n. Nämä toimivat samalla periaatteella kuin Azure DevOps -agentti ja Azure DevOps -putki. GitLab tarjoaa myös samankaltaisia palveluita.

Insinöörityössä käytettiin myös Docker-virtualisointityökalua, jolla testien suoritusympäristö saatiin identtinen jokaiselle testien suoritukselle. Docker-työkalun avulla testien suoritusympäristön ylläpitäminen, kuten riippuvuuksien tai selainversioiden hallinta on yksinkertaista. Testien suoritusympäristöjen ylläpitämiseen kuluva aika on vaikeasti ennustettavaa, jos ympäristöä ei olla muodostettu virtuaalikoneen levykuvasta tai Docker-kuvasta. Pienikin muutos saattaa rikkoa ympäristön toiminnan, kuten esimerkiksi selaimen version automaattinen päivittyminen. Samasta Docker-kuvasta luodut Docker-kontit ovat aina keskenään identtisiä käynnistyessään, ja esimerkiksi selaimen versio on aina sama. Docker-kuvasta luodun ympäristön päivittäminen on myös yksinkertaisempaa kuin ei-virtualisoidun tai virtuaalikoneen levykuvasta luodun ympäristön päivittäminen. Levykuvan päivittäminen esimerkiksi edellyttää, että päivitettävästä levykuvasta luodaan ensin virtuaalikone, jonka jälkeen virtuaalikoneelle asennetaan halutut päivitykset. Tämän jälkeen virtuaalikone pitää vielä alustaa ennen, kuin siitä voidaan luoda päivitetty levykuva. Insinöörityössä toteutetun Docker-kuvan päivittäminen edellyttää ainoastaan dockerfile-tiedostoon uusien päivitysten määrittämisen, jonka jälkeen tiedostosta voidaan luoda uusi päivitetty Docker-kuva. Esimerkiksi kontin käyttämän Chrome-selainversion ja ChromeDriver-ajurin version päivittäminen onnistuu muuttamalla yhtä dockerfile-tiedostossa määritettyä muuttujaa.

Putkien määrittämisessä käytettiin YAML-tiedostoja, joka helpottaa putkien ylläpitämistä. Putken sisältämät tehtävät suorittava agentti oli asennettu henkilökohtaiselle tietokoneelle, joka vaikuttaa testien suoritusympäristön vakauteen, mutta tämä oli tietoinen valinta. Parhaassa tapauksessa agentit olisivat asennettuna virtuaalikoneille tai Docker-konttien sisälle, joita hallinnoitaisiin esimerkiksi Kubernetes-työkalun avulla, mutta tarvittavat palvelut, kuten virtuaalikoneet ovat maksullisia.

Insinöörityössä päästiin mielestäni, niille annettuihin tavoitteisiin. Kehittämismahdollisuuksia tietenkin aina löytyy. Jakokehitysmahdollisuuksia työssä voisi olla vaikka Pabot-työkalun käyttöönotto, jolla voitaisiin saavuttaa testien rinnakkain suoritus ominaisuuksia ilman, että joutuu maksamaan siitä. Muita kehityskohteita voisi olla esimerkiksi testisuorituksien videonauhoitus tai parempi testituloksien visualisointi. Kehityskohteiden suunnittelussa on ainoastaan mielikuvitus rajana.

## Lähteet


1 Hristov, Anton. How automated testing enables DevOps. Verkkoaineisto. Atlassian. <https://www.atlassian.com/devops/devops-tools/test-automation>. Luettu 8.7.2022.

2 Kinsburner, Eran. 2020. What Is Test Automation? Verkkoaineisto. Perfectio. <https://www.perfecto.io/blog/what-is-test-automation>. 16.9.2020. Luettu 8.7.2022.

3 Choudary, Archana. 2020. What is Integration Testing? A Simple Guide on How to Perform Integration Testing. Verkkoaineisto. Edureka. <https://www.edureka.co/blog/what-is-integration-testing-a-simple-guide-on-how-to-perform-integration-testing>. 25.10.2020. Luettu 8.7.2022

4 DevOps Infinite Loop-DevOps processes simplified for beginners? Verkkoaineisto. <https://awkwardgen.com/devops-infinity-loop-for-beginners/>. 29.6.2021. Luettu 15.7.2022.

5 Daw, Graham. 2022. DevOps – what is it and why is it so important? Verkkoaineisto. Futurice. <https://futurice.com/blog/what-is-devops>. 25.3.2022. Luettu 13.7.2022.

6 Buchana, Ian. History of DevOps. Verkkoaineisto. Atlassian. <https://www.atlassian.com/devops/what-is-devops/history-of-devops#:~:text=The%20DevOps%20movement%20started%20to,of%20dysfunction%20in%20the%20industry.>. Luettu 13.7.2022.

7 Fowler, Martin. 2006. Continuous Integration. Verkkoaineisto. <https://martinfowler.com/articles/continuousIntegration.html>. 1.5.2006. Luettu 14.7.2022.

8 Pittet, Sten. How to get started with Continuous Integration. Verkkoaineisto. Atlassian. <https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration>. Luettu 14.7.2022.

9 Pittet, Sten. Continuous integration vs. delivery vs. deployment. Verkkoaineisto. Atlassian. <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>. Luettu 14.7.2022

10 Fowler, Martin. 2013. ContinuousDelivery. Verkkoaineisto. <https://martinfowler.com/bliki/ContinuousDelivery.html>. 30.5.2013. Luettu 14.7.2022.

11 Robot Framework User Guide Version 5.0. Verkkoaineisto. <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>. 23.3.2022. Luettu 14.7.2022.

12 QWeb keyword-based test automation for the web. Verkkoaineisto. Github. <https://github.com/qentinelqi/qweb/>. 9.6.2022. Luettu 14.7.2022.

13 Shivam Arora. Verkkoaineisto. Simplilearn. <https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker>. 21.2.2023. Luettu 27.3.2023

14 Ajeet Singh Raina. Verkkoaineisto. TestProject. <https://blog.testproject.io/2020/05/30/the-ultimate-docker-tutorial-for-automation-testing-using-testproject/> 30.5.2020. Luettu 30.3.2023.

15 What are containers? Verkkoaineisto. IBM. <https://www.ibm.com/topics/containers>. Luettu 30.3.2023.

16 What is Azure DevOps? Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>. 23.5.2022. Luettu 15.7.2022.

17 Azure Artifacts overview. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/artifacts/start-using-azure-artifacts?view=azure-devops >. Luettu 15.7.2022.

18 Artifacts in Azure Pipelines – overview. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/artifacts-overview?view=azure-devops&tabs=nuget >.14.7.2022. Luettu 15.7.2022.

19 What is Azure Boards? Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops>. 8.3.2022. Luettu 15.7.2022.

20 What is Azure Pipelines? Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops>. 2.4.2022. Luettu 15.7.2022.

21 What is Azure Test Plans? Verkkoaineisto. <https://learn.microsoft.com/en-us/azure/devops/test/overview?view=azure-devops>. 29.6.2021. Luettu 15.7.2022.

22 Azure Pipelines agents. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser>. 6.4.2022. Luettu 15.7.2022.

23 What is Azure Repos? Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops>. 27.6.2022. Luettu 15.7.2022.

## Liite 1. HaeSuoritetutOpintopisteetOpiskelija.robot koodi

```
*** Settings ***
Documentation               Opinnäytetyö demo testi
...
Resource                    ${CURDIR}${/}..${/}resources${/}re-
sources.robot
Test Timeout                90 minutes
Suite Setup                 OMA Suite Setup
Suite Teardown              OMA Suite Teardown
Test Setup                  OMA Test Setup
Test Teardown               OMA Test Teardown
Force Tags                  testi_1


*** Variables ***

# Suite output
${KESKIARVO}                ${NONE}
${OPINTOPISTEET}            ${NONE}


*** Test Cases ***
Tarkista_Opintopisteiden_määrä
    [Documentation]
    [Tags]

    Appstate                Login_OMA

    ${OPINTOPISTEET}  ${KESKIARVO}=                   OpTy_HaeOpinto-
pisteetJaKeskiarvo
    Log To Console          \nOpitopisteet: ${OPINTOPISTEET} \nKes-
kiarvo: ${KESKIARVO}
```

## Liite 2. Keywords.robot koodi

```
*** Settings ***
Documentation       Testi avainsanoja.

*** Keywords ***
# Opintosuunnittelu ja Tarkastelu
OpTy_HaeOpintopisteetJaKeskiarvo
    [Documentation]         Hae opitopisteet ja keskiarvo

    ClickElement            ${USERMENU}
    ClickText               Opiskelijan työpöytä
    ClickText               HOPS
    ${pisteet}=             GetText                     Suoritettuja
opintopisteitä:    between=Suoritettuja opintopisteitä: ???
    ${keskiarvo}=           GetText                     Keskiarvo:
between=Keskiarvo: ???
    LogScreenShot

    [Return]  ${pisteet}  ${keskiarvo}
```

## Liite 3. Resources.robot koodi

```
*** Settings ***
Documentation               Avainsanoja liittyen kirjautumiseen, se-
lain asetuksiin, testi asetuksiin ja navigaatioihin.
...                         Myös vakio muuttujat.
Library                     QWeb
Resource                    keywords.robot

*** Variables ***
# Testi muuttujia
${DEFAULT_USERNAME}         ${NONE}
${DEFAULT_PASSWORD}         ${NONE}

# Selain muuttujia
${DEFAULT_BROWSER}          chrome
${DEFAULT_TIMEOUT}          150s
${DRAW_BOX}                 True

# Navigaatio muutujia
${OMA_URL}                  https://oma.metropolia.fi/
${USERMENU}                 xpath\=//*[@class\="username"]

*** Keywords ***
OMA Suite Setup
    [Documentation]         Avaa selain, aseta resoluutio, aseta
avainsanojen aikakatkaisu 150s ja aseta hakutila.
    Open Browser            about:blank                 ${DE-
FAULT_BROWSER}
    SetConfig               WindowSize                  1920x1080
    SetConfig               DefaultTimeout              ${de-
fault_timeout}
    IF  '${DRAW_BOX}' == 'True'
        SetConfig               SearchMode        draw
    END

OMA Suite Teardown
    [Documentation]         kirjaudu ulso ja sulje selain.
    Close All Browsers
    Sleep                   2s

OMA Test Setup
    [Documentation]         Aseta QWeb asetuksia.
    SetConfig               LineBreak             None
    SetConfig               CssSelectors          True
    SetConfig               CheckInputValue       True

OMA Test Teardown
    [Documentation]         Ota kuvakaapaus kun testi epäonnistuu.
    IF    '${TEST STATUS}' == 'FAIL'
        LogScreenshot           screenshot_failed_${TEST NAME}.png
    END

Login_OMA
    [Documentation]         Kirjaudu OMA metropolia palveluun.
    GoTo                    ${OMA_URL}
    TypeText                Username                    ${DE-
FAULT_USERNAME}
    TypeText                Password                    ${DE-
FAULT_PASSWORD}
    ClickText               Login
```

```
    VerifyNoText            The password you entered was incorrect.
    VerifyNoText            The username you entered cannot be identi-
fied.
    VerifyText              Näytä työtilat
    LogScreenshot

Logout_OMA
    [Documentation]         Kirjaudu ulos OMA metropolia palvelusta.
    GoTo                    ${OMA_URL}
    ClickElement            ${USERMENU}
    ClickText               Kirjaudu ulos

Main_Page
    [Documentation]         Palaa etusivulle.
    GoTo                    ${OMA_URL}

Appstate
    [Documentation]         Varmista testin aloitus piste.
    [Arguments]             ${app}
    IF  '${app}' == 'Login_OMA'
        Login_OMA
    ELSE IF  '${app}' == 'Main_Page'
        Main_Page
    END
```

## Liite 4. Dockerfile koodi

```
# Pohjakuvasta Ubuntu 20.04.
FROM ubuntu:20.04

# Luo build-time muuttuja Chromen versiolle.
ARG CHROME_VERSION="105.0.5195.125-1"

# Luo ympäristömuuttujat.
ENV DEBIAN_FRONTEND noninteractive
ENV ROBOT_WORK_DIR /opt/robotframework
ENV ROBOT_DOCKER_DIR /opt/robotframework/docker
ENV ROBOT_REPORTS_DIR /opt/robotframework/reports

# Asenna apt paketit.
RUN apt-get update && apt-get install -yq \
    apt-utils \
    git-core \
    xvfb \
    psmisc \
    xsel \
    unzip \
    libgconf-2-4 \
    libncurses5 \
    libxml2-dev \
    libxslt-dev \
    libnss3 \
    libatk-bridge2.0-0 \
    libgbm-dev \
    libz-dev \
    libgtk-3-0 \
    xclip \
    tzdata \
    wget \
    curl \
    matchbox \
    python3-pip \
    python3-tk \
    python3-dev \
    xauth \
    dos2unix \
    scrot

# Lataa ja asenna Chrome.
RUN wget --no-verbose -O /tmp/chrome.deb
https://dl.google.com/linux/chrome/deb/pool/main/g/google-chrome-sta-
ble/google-chrome-stable_${CHROME_VERSION}_amd64.deb \
    && apt install -y /tmp/chrome.deb \
    && rm /tmp/chrome.deb

# Lataa ja asenna ChromeDriver. Käyttää Chromen versiota sopivan Chro-
meDriver version asentamiseen.
RUN CHROME_VERSION=$(google-chrome --version) \
    && MAIN_VERSION=${CHROME_VERSION#Google Chrome } && MAIN_VER-
SION=${MAIN_VERSION%%.*} \
    && CHROMEDRIVER_VERSION=`curl -sS chromedriver.storage.goog-
leapis.com/LATEST_RELEASE_$MAIN_VERSION` \
    && mkdir -p /opt/chromedriver-$CHROMEDRIVER_VERSION \
    && curl -sS -o /tmp/chromedriver_linux64.zip
http://chromedriver.storage.googleapis.com/$CHROMEDRIVER_VER-
SION/chromedriver_linux64.zip \
```

```
    && unzip -qq /tmp/chromedriver_linux64.zip -d /opt/chromedriver-
$CHROMEDRIVER_VERSION  \
    && rm /tmp/chromedriver_linux64.zip \
    && chmod +x /opt/chromedriver-$CHROMEDRIVER_VERSION/chromedriver \
    && ln -fs /opt/chromedriver-$CHROMEDRIVER_VERSION/chromedriver
/usr/local/bin/chromedriver

# Asenna python paketit.
RUN pip3 install \
    robotframework==6.0.2 \
    QWeb==2.1.2

# Aseta aikavyöhyke ja luo .Xauthority tiedosto.
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
    && touch ~/.Xauthority \
    && export TZ=Europe/Helsinki \
    && ln -snf /usr/share/zoneinfo/$TZ /etc/localtime \
    && echo $TZ > /etc/timezone

# Kopio robot käynnistys skripti.
COPY run_tests_ubuntu_linux.sh /opt/robotframe-
work/docker/run_tests_ubuntu_linux.sh

# Muuta skriptin EOL Unix muotoon.
RUN dos2unix /opt/robotframework/docker/run_tests_ubuntu_linux.sh

# Luo raportti ja työkansiot. Anna käyttöoikeuksia.
RUN mkdir -p ${ROBOT_REPORTS_DIR} \
    && mkdir -p ${ROBOT_WORK_DIR} \
    && chmod 777 -R ${ROBOT_WORK_DIR} \
    && chmod 777 -R ${ROBOT_REPORTS_DIR} \
    && chmod 777 /opt/robotframework/docker/run_tests_ubuntu_linux.sh

# Ota käyttöön työkansio.
WORKDIR /opt/robotframework

# Robot skriptin ajo.
CMD /opt/robotframework/docker/run_tests_ubuntu_linux.sh -s ${suite} -
u ${username} -p ${password}
```

## Liite 5. Run_tests_ubuntu_linux.sh koodi

```
#!/bin/bash
# Aja Robot Framework Framework skripti virtuaalinäytöllä.
Help()
{
  echo "Variables:"
  echo "-s test tag (mandatory)"
  echo "-u username (mandatory)"
  echo "-p password (mandatory)"
}

# Muuttujien  määritys ja tarkistus.
while getopts ":hs:u:p:" opt; do
  case $opt in
    h)  Help
        exit;;
    s)  suite=$OPTARG;;
    u)  username=$OPTARG;;
    p)  password=$OPTARG;;
    *)  echo "Error: Variable error"
        Help
        exit 1;;
  esac
done

# Aseta virtuaalinäyttö and ikkunamanageri.
Xvfb :98 -screen 0 1920x1080x24  &
export DISPLAY=:98
echo "Wait 10 seconds for virtual display"
sleep 10s
echo $DISPLAY
matchbox-window-manager -use_titlebar no &

# Aja skripti.
echo "Robot execution begins: $suite"
python3 -m robot -x outputxunit.xml --outputdir reports/robot_re-
sults_$suite  -i $suite -v default_username:$username -v default_pass-
word:$password .
```

## Liite 6. BuildImagePipeline.yaml koodi

```
trigger: none

jobs:
- job: BuildImage
  displayName: Build Docker Image
  timeoutInMinutes: 360

  pool:
    name: LocalWinPool

  steps:
  - script: cd docker && docker build -t qweb-testiautomaatio .
    displayName: 'Build Image'

  - script: docker image prune -f
    displayName: 'Delete Dangling Images'
```

## Liite 7. TestAutomationPipeline.yaml koodi

```
trigger: none

jobs:
- job: RunTests
  displayName: Test Automation Pipeline
  timeoutInMinutes: 360

  pool:
    name: LocalWinPool

  steps:
  - script: docker run
      -v $(Build.SourcesDirectory)/tests:/opt/robotframework/tests
      -v $(Build.SourcesDirectory)/resources:/opt/robotframework/re-
sources
      -v $(Build.SourcesDirectory)/reports:/opt/robotframework/reports
      -e suite=$(testchain)
      -e username=$(username)
      -e password=$(password)
      --shm-size=1g
      qweb-testiautomaatio
    displayName: 'Start Container And Run Tests'

  - task: PublishBuildArtifacts@1
    displayName: 'Save Test Results'
    inputs:
      PathtoPublish: '$(Build.SourcesDirectory)\reports\robot_re-
sults_$(testchain)'
      ArtifactName: 'robot_results_$(testchain)'
    condition: always()

  - task: PublishTestResults@2
    displayName: 'Publish Test Results'
    inputs:
      testResultsFiles: '$(Build.SourcesDirectory)\reports\robot_re-
sults_$(testchain)\outputxunit.xml'
      testRunTitle: Results
    condition: always()

  - script: docker container prune -f
    displayName: 'Delete Container'
    condition: always()
```