jamk.fi

Testausautomaatioympäristön luominen ja sen skaalautuvuus

Joona Hautamäki

Opinnäytetyö
Huhtikuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hautamäki, Joona	Opinnäytetyö, AMK	huhtikuu 2020
	Sivumäärä 30	Julkaisun kieli Suomi
	-	Verkkojulkaisulupa myönnetty: x

Työn nimi
Testausautomaatioympäristön luominen ja sen skaalautuvuus

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Esa Salmikangas, Jouni Huotari

Toimeksiantaja(t)
Pinja Oy, Jukka Järvi

Tiivistelmä

Pinja Oy kehittää ohjelmistoratkaisuja teollisuusalan yrityksille, ja yhtenä Pinjan ratkaisuista on GEMA. GEMAa käytetään teollisuuden toimialoilla tuotannonohjaus- ja koneseurantajärjestelmänä. GEMA on suunnattu tuotannon seuraamiseen, laadun parantamiseen ja toimintamallien kehittämiseen.

Työn aiheena oli kehittää toimiva ratkaisu testausautomaatiosta osaksi GEMAn tuotekehitystä. Testausautomaatiolla pystyttäisiin vähentämään nykyistä manuaalitestauksen määrää ja parantamaan tuotteen laatua. Yhtenä tavoitteena oli toteuttaa työstä geneerinen, jotta sitä voitaisiin hyödyntää muissakin Pinjan ohjelmistoprojekteissa.

Opinnäytetyön nykytilanteen selvityksessä käydään läpi käytössä ollutta kehitysmenetelmää ja kuvataan GEMAn toimintaa tarkemmalla tasolla. Nykytilanteessa selvitetään myös testaamisen tarvetta ja nykyisen testaamisen määrää.

Työn aloitus vähäisten testaussuunnitelmien ja GEMAn vaatimusmäärittelyjen takia oli hankalaa. Robot Frameworkin valitseminen nopeutti työn kehitysprosessia rajoittamalla käytettävien ohjelmistojen määrää. Näiden rajoitteiden avulla Docker-kontin valinta onnistui paljon nopeammin. Kun testausautomaatiosta oli toimiva kontti, pystyttiin aloittamaan Jenkinsille testausprosessin määrittäminen.

Lopputuloksena testausautomaatiosta saatiin toimiva ratkaisu GEMAn tuotekehitykseen. Työ on myös geneerinen ja käytettävissä muissa Pinjan projekteissa. Lisäksi työn aikana saatiin paremmin määriteltyä GEMAn testaussuunnitelmaa ja ominaisuuksia.

Avainsanat (asiasanat)
Docker, Robot Framework, pabot, Kubernetes, python, testausautomaatio

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Hautamäki, Joona	Bachelor's thesis	April 2020
		Language of publication: Finnish
	Number of pages: 30	Permission for web publication: x

Title of publication

Creating and scaling test automation environment

Degree programme

Information and Communications Technology, Software Engineering

Supervisor(s)

Salmikangas, Esa; Huotari, Jouni

Assigned by

Pinja Oy; Järvi, Jukka

Abstract

Pinja Oy develops software solutions for industrial companies and one of Pinja's solutions is GEMA. GEMA is used in various fields of industry as a production control and machine monitoring system. GEMA is directed at monitoring production, improving its quality and developing operating models.

The objective was to develop a functional solution for test automation as part of GEMA's product development. Test automation could reduce the current amount of manual testing and improve product quality. One of the goals was to make the work generic so that it could be utilized in other Pinja software projects as well.

The study of the current situation reviews the development method used and describes GEMA's activities at a more detailed level. In the current situation, the need for testing and the current amount of testing are also clarified. The theory phase describes the tools and software that will be used in the work.

The start of the thesis project was difficult due to minor testing plans and GEMA requirement specifications. The choice of Robot Framework accelerated the work development process limiting the number of the software used. With these constraints, Docker container selection was much faster. Once the test automation had a working container, it was possible to begin determining the testing process for Jenkins.

The result of the test automation was a functional solution for GEMA's product development. The work is also generic and available in other Pinja projects. In addition, the work provided a better definition of the GEMA test plan and features.

Keywords/tags (subjects)

Docker, Robot Framework, Pabot, Kubernetes, Python, test automation

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

CI/CD	Continuous integration and continuous delivery. Jatkuva integraatio ja toimitus. Ohjelmistotuotannon menetelmiä, joita hyödynnetään lähdekoodin yhdistämisessä, testaamisessa ja sen hyväksynnässä.
IP-osoite	Internet Protocol. Käytetään IP-verkkoihin kytkettyjen verkkosovittimien yksilöimiseen.
KPI	Key performance indicator. Laskettava tai mitattava arvo, joka ilmaisee suorituskyvyn tehokkuutta.
MES	Manufacturing execution system. MES-järjestelmä on operatiivisen tuotannon johtamisen ja kehittämisen työkalu.
OPC	Open platform communications. Koostuu joukosta standardeja, joita käytetään teollisuudessa tiedonsiirrossa.
PaaS	Platform as a service. Palvelualustan ulkoistaminen, esimerkiksi pilvipalvelut.
pip	Pip Install Packages. Pythonin pakettienhallinta-työkalu.
PLC	Programmable logic controller. Pieni tietokone, jonka avulla voidaan ohjata tuotannon prosesseja.

1 Lähtökohdat

1.1 Toimeksiantaja

Toimeksianto opinnäytetyölle tuli Pinja Oy:ltä (Pinja), joka aikaisemmin toimi nimellä ARROW Engineering Oy (ARROW). ARROW perustettiin vuonna 1993, ja sen päätoimialat olivat tuottaa, toimittaa ja myydä ohjelmistoja teollisuuden alalle. Vuonna 2018 elokuussa tapahtuneen yrityskaupan jälkeen ARROW toimi osana Protacon-konsernia omana tytäryhtiönään. Protacon Group Oy on vuonna 1990 perustettu digitalisaatioon ja teollisuuden uudistamiseen perehtyvä yritys. Protacon-konserni päätti vuonna 2020 maaliskuussa uudelleen brändätä konsernin, josta tuli Pinja Oy. (Yrityskauppa 2018.)

Pinjan yksi osaamisalueita on luoda järjestelmäratkaisuja teollisuuden tuotannon ja kunnossapidon operatiiviseen johtamiseen. Pinjalla kehitetään ohjelmistoratkaisuja teollisuuden digitalisoimiseksi, johon kuuluvat myös tuotteiden myynti- ja tukipalvelut. Yksi ohjelmistoratkaisuista on GEMA. GEMA voi skaalautua tavallisesta koneseurantajärjestelmästä kokonaiseksi MES-järjestelmäksi. (ARROW GEMA n.d.)

1.2 Työn tavoitteet

Opinnäytetyön tavoitteena oli kehittää toimiva ratkaisu web-ohjelmiston testausautomaatiosta. Samalla oli tarkoitus tutkia kehitetyn automaation skaalautuvuutta, jotta se olisi käytettävissä muissa ohjelmistoprojekteissa. Yhtenä tavoitteena oli laajentaa saatavilla olevaa tietoa testauksesta ja sen kattavuudesta sekä tuoda uusia ja erilaisia ratkaisuja testausautomaatioympäristöille. Työ oli tarkoitus ottaa osaksi GEMAn tuotekehitystä, jolla parannettaisiin tuotteen laatua ja vähennettäisiin regressiota.

Projektilla ei ollut aikaisemmin muuta testausautomaatiota käytössä kuin yksikkötestit. Tuotteelle oli suoritettu vain manuaali- ja yksikkötestejä. Jo tuotettu testausdokumentaatio oli lähinnä testaajan testitapauslistoja ja ohjeita tuotteen testaamiseksi ja testien toistamiseksi. Tuotteesta ei ollut tehty varsinaisia vaatimusmäärittelyjä, vaan

kehitys on tapahtunut edellisen tuotteen pohjalta käyttäen myös MarvelApp-protoilua käyttöliittymien suunnittelussa. Laajimmat dokumentit olivat tuotteen toimitusdokumentaatio sekä myynnin tukimateriaalit.

1.3 Tutkimusmenetelmät

Tutkimusmenetelmänä työssä käytettiin soveltavaa tutkimusta. Työssä tutkittiin erilaisia toteutusvaihtoehtoja ja niiden ominaisuuksia vertailtiin toisiinsa käytännössä. Vertailussa otettiin huomioon ympäristön käyttöönotto, skaalautuvuus ja sen ylläpitäminen. Käyttöönotossa otettiin huomioon, kuinka paljon erilaisia muutoksia piti tehdä, jotta toteutus oli käytettävissä myös muissa palveluissa. Skaalautuvuutta tutkittiin, miten monelle palvelulle valittu ratkaisu oli toimiva ilman suuria lisämuutoksia. Ylläpidettävyyttä tutkittiin järjestelmien ja työkalujen eri versioita vasten ja kuinka stabiileja ne olivat keskenään.

2 Nykytilanne

2.1 Kehitysmenetelmät

Toimeksiantajalla on 26 vuoden kokemus ohjelmistokehityksestä teollisuuden parissa. Ainoa ongelma oli ohjelmiston tuotantomalli. Ketteriä menetelmiä ei vielä ollut otettu käyttöön, mutta niiden käyttöönotosta oli jo päätetty ennen työn aloittamista. Tuotekehitykseen toivottiin parempaa kehitysmallia, dokumentaation parantamista ja kattavampaa testaamista ennen uusien versioiden julkaisua tuotantoon.

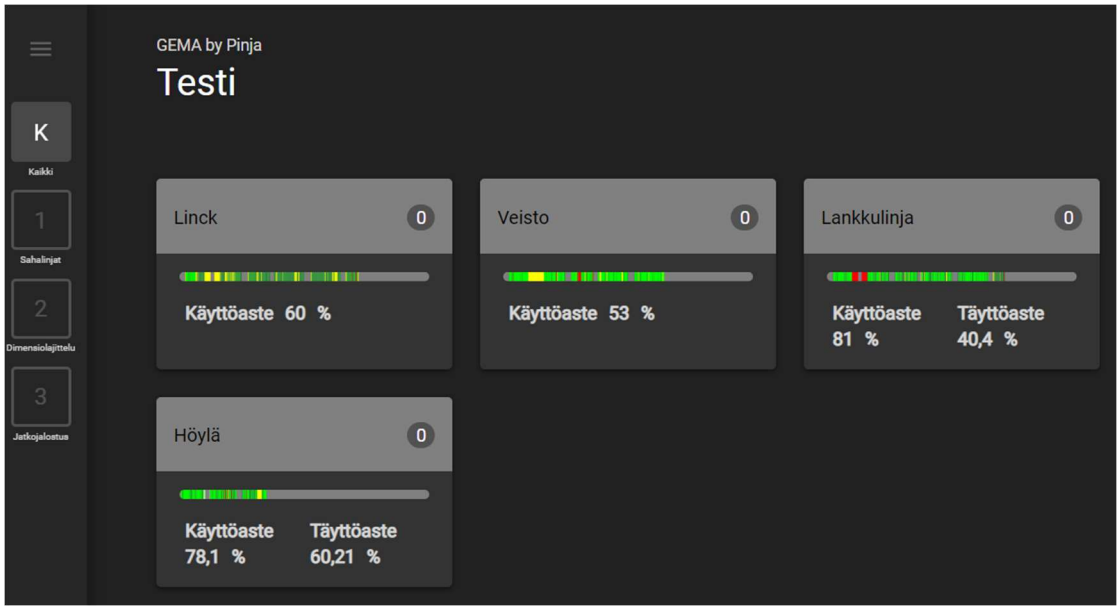
GEMAn tuotekehityksessä käytettiin aluksi kehitysmenetelmänä prototyypittämistä. Prototyypittämisen voi jakaa neljään vaiheeseen. Ensimmäinen vaihe koostuu määrittelystä ja nopeasta suunnittelusta. Määrittelyyn ja suunnitteluun tarvittavat vaatimukset tulevat yleensä asiakkaalta. Toinen vaihe on puhtaasti prototyypin kehitystä, kunnes prototyyppi alkaa vastaamaan asiakkaan vaatimuksia. Vaiheessa kolme ominaisuutta hyväksytetään asiakkaalla ja jatkokehitetään saadun palautteen mukaan. Vaihetta kolme iteroidaan, kunnes ominaisuus vastaa asiakkaan odotuksia.

Iteroinnista siirrytään vaiheeseen neljä, jossa ominaisuus implementoidaan osaksi nykyistä ratkaisua ja siirrytään kehityksestä ominaisuuden ylläpitämiseen.

Opinnäytetyön aikana siirryttiin prototyyppittämisestä scrum-pohjaiseen kehitykseen, joka mahdollistaa testaamisen työmäärän arviointia paremmin sunnittelun aikana. Scrum on ohjatumpi ja paremmin suunniteltu ohjelmiston tuontamalli. Scrumissa kehitystyö jakosoitetaan sprinttien avulla. Sprintit ovat yleensä 1-4 viikon mittaisia jaksoja. Päivän työt käydään läpi scrum masterin vetämänä dailyissä. Dailyt ovat lyhyitä kokouksia, jotka pyritään pitämään noin 30 minuutin mittaisina. Scrumissa tuotteen omistaja määrittelee tehtävät, jotka menevät tuotekehityksen backlogille eli niin sanottuun varastoon. Kehitystiimit aina ennen seuraavan sprintin alkua suunnittelevat seuraavan sprintin valitsemalla tehtävät backlogilta ja arvioivat niiden työmäärän. Sprintin vaihdon yhteydessä yleensä arvioidaan edellisen sprintin suoritustaso. Samaan aikaan pidetään myös retrospektiivinen kokous eli retrot. Retroissa kerätään kehitysideoita parantamaan seuraavaa sprinttiä katsomalla takautuvasti edelliseen sprinttiin.

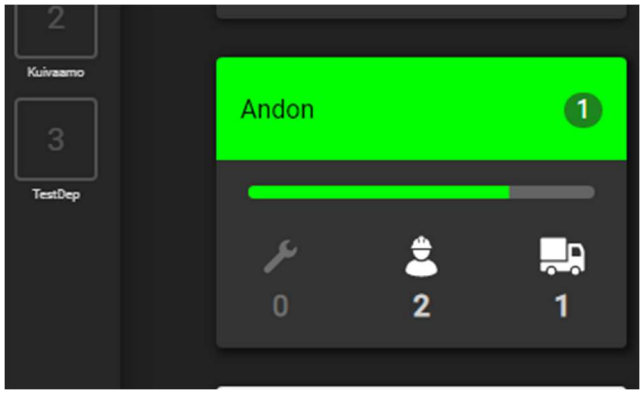
2.2 GEMA

GEMA on yksi Pinjan kehittämistä tuotteista, joka on tarkoitettu teollisuuden digitalisoimiseksi ja tuotannon tehostamiseksi. GEMA on suunnattu tuotannon seuraamiseen, laadun parantamiseen ja toimintamallien kehittämiseen. GEMA-lyhenne muodostuu sanoista ”Genius Manufacturing”. GEMA on siis MES-järjestelmä, joka kerää tietoa tuotannon koneilta ja kokoaa ne yhteen tiivistäen tiedot prosessista, tuotteista ja materiaaleista helposti luettaviksi raporteiksi. Kuviossa 1 on esimerkki testiympäristön tehdasnäkymästä. GEMA on käytössä monilla suurilla suomalaisilla yrityksillä. Suomessa sitä käytetään eniten metsä-, elintarvike- ja koneteollisuudessa. (ARROW GEMA n.d.)



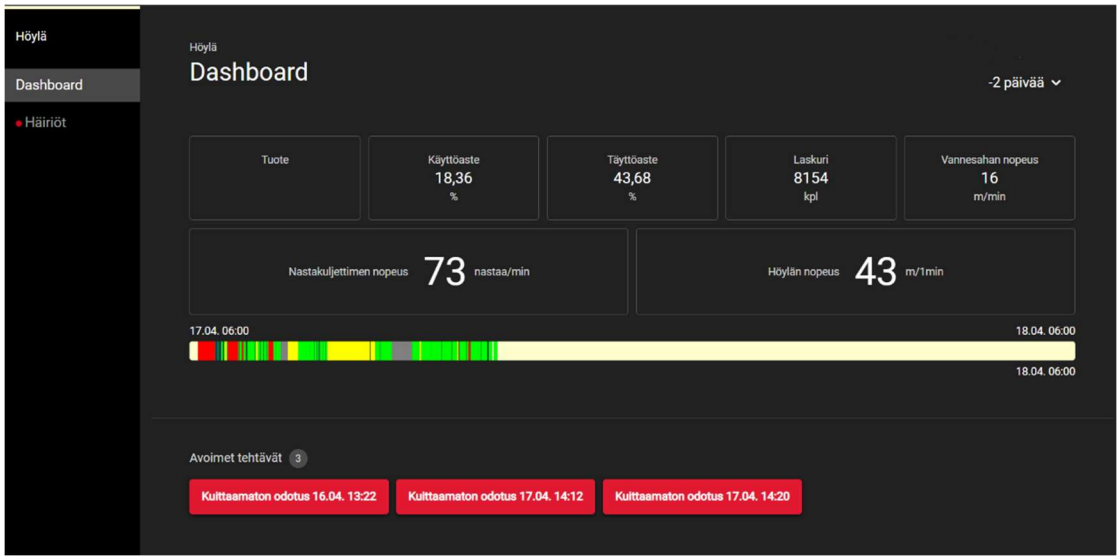
Kuvio 1. GEMAn tehdasnäkymä

Monipuolisten ominaisuuksien ja vähäisen dokumentaation takia tuotteen käyttötarkoituksiin oli todella hankalaa päästä käsiksi. Paras lähestymiskeino oli perehtyä tuotteeseen, tutkia jo olemassa olevia testitapauksia ja lähteä pilkkomaan niitä pienempiin osiin järjestelmän kokonaiskuvan hahmottamiseksi. GEMA ei toimi pelkästään tuotannonseurantajärjestelmänä, vaan siihen myös mahdollista lisätä Andon-koneita (ks. kuvio 2) ja käsityöpisteitä. Andonin avulla tuotannossa voidaan tehdä avunpyyntöjä häiriötilanteissa ja niistä kerättyjen datojen avulla voidaan luoda raportteja (ARROW Andon n.d).



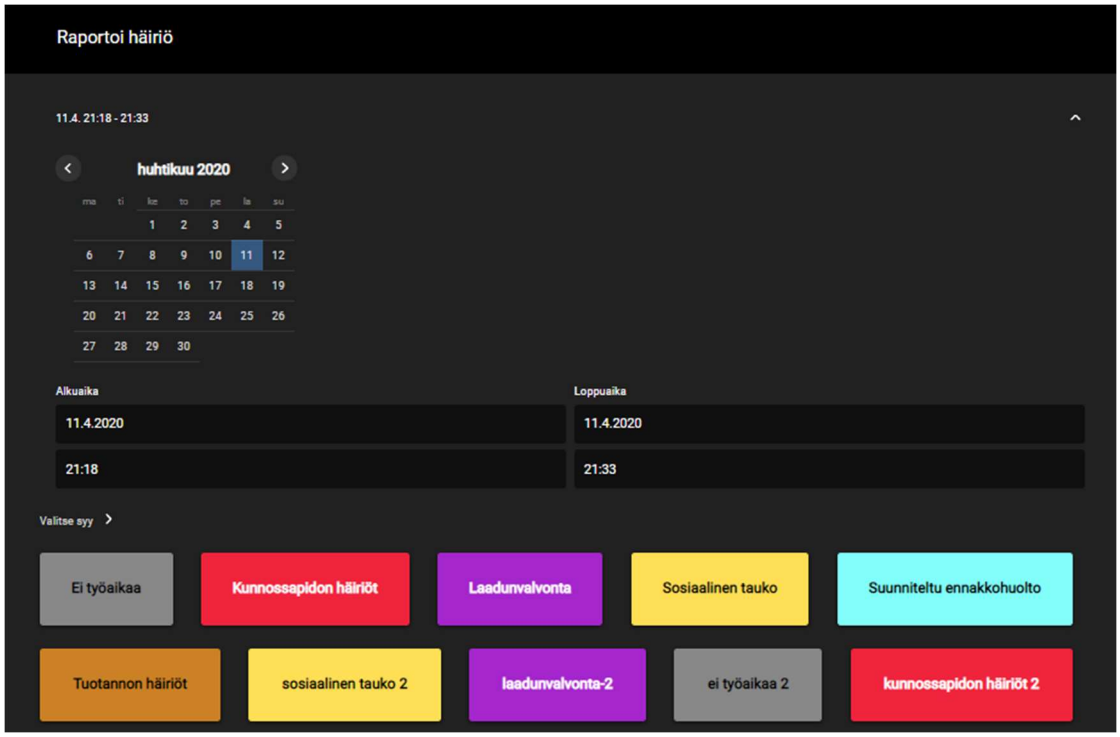
Kuvio 2. Andon-kone tehdasnäkymässä

GEMAn tärkeimpiin ominaisuuksiin kuuluu tuotannon tapahtumien seuranta, jossa visualisoidaan tilapalkkiin, kuinka paljon kone on ollut tuotanto-, odotus- tai häiriötilassa. KPI-arvot eli suoritusindikaattorit kertovat koneen nopeuden esimerkiksi kappalelaskureista tai prosentuaalisen määrän koneen tuotannosta suhteessa suunniteltuun. Häiriöiden kuittaaminen tuotannon aikana on tärkeää. Esimerkiksi jos kone on ollut odotuksella häiriön seurauksena, on tärkeää pystyä raportoimaan häiriön tarkempi syy välittömästi. Tällaisessa tilassa järjestelmä lisää automaattisesti avoimen tehtävän koneelle, jotta käyttäjän on helpompi raportoida tiettyihin aikoihin tapahtuneita pysähdyksiä. Kuviossa 3 näkyy alareunassa yksi avoin tehtävä 16.04. klo 13.22 alkaneesta pysähtymisestä, josta klikkaamalla pääsee kuittaamaan kyseisen hetken syytä.



Kuvio 3. Yksittäisen koneen oma sivu

Raportoinnilla saadaan visualisoitua eri pysähdyssyyt, niiden määrät ja trendit graafeina. Koneiden ongelmia ja pysähdyssyitä on mahdollista selvittää raporteista, kuten johtuvatko pysähtymiset konerikoista, materiaalin loppumisesta tai jostain muusta. Käyttäjä pystyy GEMAn asetuksista konfiguroimaan syitä yleiseltä tasolta konekohtaisiksi. Syille pystyy määrittämään alisyitä ja tarvitseeko syy lisäkommenttia raportoidessa. Kuviossa 4 näkyy esimerkkinäkymä häiriön raportoinnista, jossa alareunassa näkyy koneelle määritetyt pysähdyssyyt.



Kuvio 4. Häiriön raportointi -näkymä

GEMA hyödyntää tiedon keräämisessä suoraan logiikoita tai esimerkiksi Moxan tuottamia PLC-tiedonkeruulaitteita, joista tiedot kerätään OPC-palvelimelle. GEMAn ympärille on luotu eri palveluita hoitamaan tiedonkeruun ja arvojen laskeminen, jotta palvelu toimisi vakaammin ja mahdollisimman reaaliajassa. HarvesterService kerää OPC:lta tiedot ja lähettää ne tietokantaan. CalculationService laskee kaikki KPI-laskennat, tilapiirrot ja raporteissa tarvittavat arvot. Yhtenä palveluna toimii myös NotificationCenter, jonka avulla lähetetään halutuille henkilöille tai työryhmille sähköposti tai tekstiviesti, jos jollakin Andon-koneella tulee häiriö.

2.3 Testiympäristö

Tuotekehitystä varten oli pystytetty kolme ympäristöä, joissa oli tuotteesta kolme eri versiota: DEV, FREEZE ja STABLE. DEV-ympäristöön tulee kaikista viimeisimmät muutokset, FREEZE-ympäristöön taas käännetään release-kandidaatit, joita testataan ja korjataan, ennen kuin tuotteesta tehdään julkaisu ja se käännetään STABLE-ympäristöön. Ympäristöjen lisäksi Pinjalla oli käytössä pilvipalveluina Jenkins ja Google Cloud Platform ja siellä Kubernetes Engine. Yhtenä toisena kokonaisuutena

heillä oli myös käytössä Microsoftin Azure DevOps. Työssä päädyttiin käyttämään Jenkinsiä Kubernetes-integraatiolla, koska se on Pinjalla yleisimmässä käytössä ja siellä voi laajemmin toteuttaa erilaisia ratkaisuja.

3 Työkalut ja ohjelmistot

3.1 Robot Framework

Robot Framework on Pythonilla kehitetty automaatiotestaamiseen tarkoitettu ohjelmisto. Sitä kehitetään avoimena lähdekoodina, joka tarjoaa sen käyttäjille enemmän mahdollisuuksia tutustua kehittäjien ratkaisuihin ja niiden perusteella muokata ohjelmistoa omaan käyttöön. Avoimen lähdekoodin hyviä puoli on sen maksuton käyttö, eikä tarvita lisenssiä. Testien kirjoittamiseen ja kuvaamiseen Robot Frameworkissa käytetään avainsanapohjaista rakennetta. Avainsanapohjaisessa testien kirjoittamisessa etuina ovat, että testit ovat paremmin ihmiselle luettavassa muodossa ja siinä käytetään helppoa syntaksia. (Klarck 2019.) Robot Frameworkiä käytettiin työssä testausautomaation päätyökaluna.

3.2 SeleniumLibrary

SeleniumLibrary on Robot Frameworkissa web-pohjaiseen testaamiseen tehty kirjasto. Kirjastossa sisäisesti hyödynnetään Seleniumia, joka on Javalla kirjoitettu avoimen lähdekoodin ohjelmistokehys. Selenium tarjoaa toistotyökalun funktionaalisten testien kirjoittamiseen ilman testauksen skriptikielen opettelua. SeleniumLibrary vaatii toimiakseen jonkin selaimen ohjaimen, Pythonin 2.7, 3.4 tai sitä uudemman version ja Robot Frameworkin. Viimeisimpien selaimien ohjaimien asentamiseen voi hyödyntää työkalua WebdriverManager. (SeleniumLibrary 2020.) Suurin osa Robot Frameworkille tehdyistä testeistä käytti SeleniumLibraryn avainsanoja, koska työssä painotettiin GEMAn käyttöliittymän testaamista.

3.3 Pabot

Pabot on työkalu Robot Frameworkille, jonka avulla pystyy ajamaan testejä rinnakkain. Pabot on Mikko Korpelan Pythonilla kehittämä projekti, se on lisensoitu Apache-2.0:lle. Rinnakkain ajamisella nopeutetaan testausprosessia ja säästetään aikaa pitkissä testeissä. Pabotille voi erikseen määrittää, kuinka monia testejä se pystyy ajamaan rinnakkain ja myös sen, kuinka se jakaa testit rinnakkain. (Korpela 2020.) Pabotin avulla työssä testausprosessin kokonaissuoritusaikaa saatiin pienemmäksi testien rinnakkaisajolla.

3.4 Cypress

Cypress on web-pohjaisten ohjelmistojen automaatiotestaamiseen tarkoitettu työkalu, joka ei sisäisesti hyödynnä Seleniumia. Cypress ei suoranaisesti ole pelkkä automaatio- tai yksikkötestaus-ohjelmistokehys, vaan se keskittyy end-to-end testaamiseen. End-to-end testaamisella otetaan huomioon kaikki alusta loppuun, funktionaalisesta testaamisesta aina rajapintoihin saakka niin, että se kattaa koko systeemin. Cypressin testit kirjoitetaan JavaScriptillä, koska testit itsessään ajetaan suoraan selaimessa. (How it works n.d.) Työssä Cypressiä käytettiin vertailukohteena Robot Frameworkille.

3.5 Python

Python on yksinkertainen, korkean tason tietorakenteiden tulkattava ohjelmointikieli, ja se on myös avointa lähdekoodia. Se on vuonna 1990 julkaistu, ja kehittäjä on Guido van Rossum. Tulkattavan kielen ohjelmia ei käännetä erikseen, vaan ne ovat valmiita ajettavaksi. Sen takia asioiden testaaminen on nopeampaa, kun erikseen ei tarvitse käyttää aikaa ohjelmien kääntämiseen. (General Python FAQ n.d.) Python toimi työssä suuressa osassa, koska suurin osa työkaluista ja ohjelmistoista käytti sitä. Pythonilla työssä myös toteutettiin omien avainsanojen luontia Robot Frameworkille.

3.6 Docker

Docker on käyttöjärjestelmätason virtualisointiin tarkoitettu PaaS-tuote. Kyseisellä virtualisoinnilla haetaan kevyempää ratkaisua pilvipalveluiden toteuttamiseen tai esimerkiksi lyhytaikaisia suorituksia varten tarkoitettuja ympäristöjä. Dockerilla pystytetyistä ympäristöistä puhutaan kontteina. Kontti toimii yhdellä käyttöjärjestelmän ytimellä, jolloin se käyttää vähemmän resursseja kuin virtuaalikone. Docker Engine on ohjelman nimi, joka pitää kontteja ylhäällä. (What container n.d.) Dockerin avulla saavutetaan ympäristöjen standardointi, kun samasta kuvakkeesta pystytetyt ympäristöt vastaavat toisiaan. Työssä käytettiin Pythonin ylläpitämää konttia testausympäristön luomiseen.

3.7 Google Cloud Platform ja Kubernete Engine

Google Cloud Platform on Googlen tarjoama palvelupaketti pilvilaskennasta, missä on samankaltainen rakenne sen loppukäyttäjille kuin muissa sen tarjoamissa palveluissa. Palvelupaketti tarjoaa erinäköisiä moduuleja laskennasta, tietovarastoihin, tiedon analysointiin ja koneoppimiseen asti. (Overview 2020.)

Kubernetes Engine on yksi Googlen Cloud Platformin tarjoamista moduuleista ja sitä käytettiin työssä. Kubernetes on konttien orkestrointiin tarkoitettu ohjelma. Sen avulla automatisoidaan konttien pystyttäminen, skaalautuvuus ja hallinta. Kubernetesin avulla voidaan pystyttää useampia kontteja samaan aikaan. Kontit voivat toimia täysin omina ympäristöinään tai ne voivat toimia samassa ”kotelossa” eli podissa, jolloin kontit jakavat saman IP-osoitteen ja voivat jakaa saman tietovaraston. (What is Kubernetes 2020.)

3.8 Jenkins

Jenkins on ilmainen avoimen lähdekoodin automatisointipalvelu. Se on Javalla kehitetty ja tarjoaa yli 1600 plug-in-moduulia. Jenkins on CI/CD:hen suunnattu palvelu, jonka avulla pystyy helposti määrittelemään uusia projekteja, joissa voi suorittaa skriptien ajamisesta konttiklusterien pystyttämiseen. Projekteissa voidaan ajaa halu-

tulla tavalla koodien yhdistäminen, kääntäminen, testaaminen tai julkaiseminen halutulle alustalle. Jenkinsfile on tiedosto, joka perustuu Groovy-pohjaiseen ohjelmointikieleen. Groovy on oliokeskeinen ohjelmointikieli, mitä käytetään Java-alustoilla. Jenkinsfileä käytetään Jenkins CI/CD-ketjun vaiheistamiseen kuten, koodin kääntämiseen, testaamiseen ja julkaisemiseen. (Heller 2020.)

3.9 Github

Github tarjoaa verkkosivuillaan ohjelmistokehitykseen palvelun, jonka avulla koodin jakaminen ja julkaiseminen on helppoa. Github käyttää Gitin hajautettua versionhallintajärjestelmää. Git on avoimen lähdekoodin projekti, jonka aloitti Linus Torvalds eli Linux-käyttöjärjestelmän alkuperäinen kehittäjä. Githubiin kuuluu myös muita erilaisia ominaisuuksia kuten wikisivut, kehitystoiveet, tehtävien ylläpitäminen ja bugiseuranta. Github tarjoaa ilmaiseksi julkisten ohjelmavarastojen ylläpitämisen. Nykyään voi myös yhtä tunnusta kohti luoda yhden yksityisen ohjelmavaraston. (Finley 2012.)

3.10 Microsoft Azure DevOps

Microsoft Azure DevOps on Microsoftin ylläpitämä palvelu, joka tarjoaa työkalut versiohallintaan Gitillä tai TFVC:llä, raportointiin, vaatimusten ja projektin hallintaa, automaattisiin käännöksiin, testaamiseen ja julkaisujen hallintaan. TFVC eli Team Foundation Version Control on Microsoftin kehittämä ratkaisu versionhallinnasta. Microsoft tarjoaa palvelua heidän ylläpitämänään pilvipalveluna tai lisenssinä, jos haluaa itse pyörittää palvelua. (Azure DevOps n.d.) Työssä DevOpsia käytettiin ratkaisun tutkimiseen eri palveluiden välillä.

4 Työn toteutus

4.1 Tavoitteet toteutukselle

GEMAn tuotekehitykseen haluttiin automaatiotestausta, jotta se saataisiin lisättyä osaksi nykyistä ohjelmistotuotantoa. Vaatimuksissa haettiin automaatiotestaamista tuotteen front-endille. Toimeksiantaja halusi hyväksyntä- ja toimintotestien automatisointia, joka vähentäisi kuormaa nykyiseltä manuaalitestaajalta ja samalla parantaisi

tuotteen laatua. Aluksi täytyi valita, mitä työkalua aiottiin hyödyntää testien kirjoittamiseen ja suorittamiseen. Ensimmäiseksi vaihtoehdoksi nousi Robot Framework, koska siitä oli jo aikaisemmin hieman kokemusta ja näkemys siitä, että se toimisi tässä projektissa mahdollisesti hyvänä työkaluna. Jotta työkalujen valitseminen ei tapahtuisi täysin oman kokemuksen perusteella, vaihtoehtoiseksi työkaluksi valittiin Cypress.

Automaatiotestauksen lisääminen osaksi tuotteen kehitystä ei ollut ainoa vaatimus. Työhön asetettiin myös yhdeksi vaatimukseksi, että se olisi hyvin skaalautuva muihin Pinjan projekteihin. Skaalautuvuudessa tuli ottaa huomioon eri ohjelmistot, jotka pitää järjestelmään asentaa, kuinka paljon tilaa testitulokset vievät ja kuinka nopeasti sovellus oli käytettävissä uudessa projektissa. Tähän oli olemassa jo hyvä ratkaisu: Docker. Valmiita ratkaisuja Robot Framework -konteista oli jo olemassa, mutta yhdelläkään niistä ei ollut olemassa mitään ylläpitoa, ja ne eivät muutenkaan vastanneet niitä tarpeita, joita tässä työssä vaadittiin.

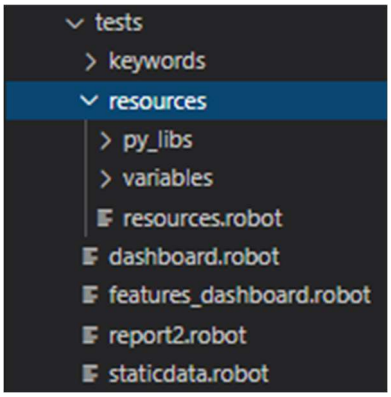
4.2 Testaustyökalut

4.2.1 Työkalun valinta

Robot Frameworkin ja Cypressin väliltä päättäminen oli aluksi vaikeaa, koska pohjatietämys itse GEMAsta oli silloin heikolla tasolla. Paras lähestymistapa oli tutustua olemassa oleviin testitapauksiin ja niiden avulla lähteä suorittamaan tutkivaa testausta testipalvelimella toiminnassa olevalle tuotteelle. Mitä enemmän tuote tuli tutuksi, sitä helpommaksi kävi testien suunnittelu ja kirjoittaminen. Työkalun valintaprosessissa kuitenkin tuli ottaa huomioon, että jatkossa testien kirjoittaminen, ajaminen ja ylläpitäminen kävisi mahdollisimman helposti. Isoa tuotetta testattaessa testitapauksien määrä kasvaa radikaalisti ja niiden suoritusaika tulee ottaa huomioon. Robot Frameworkin ominaisuudet, kuten testien yksinkertaisuus, ylläpidettävyys ja kevyt suorittaa tulevat pidemmällä aikavälillä hyödyllisemmiksi ja niiden vuoksi Robot Framework valittiin työssä testauksen päätyökaluksi.

4.2.2 Robot Framework

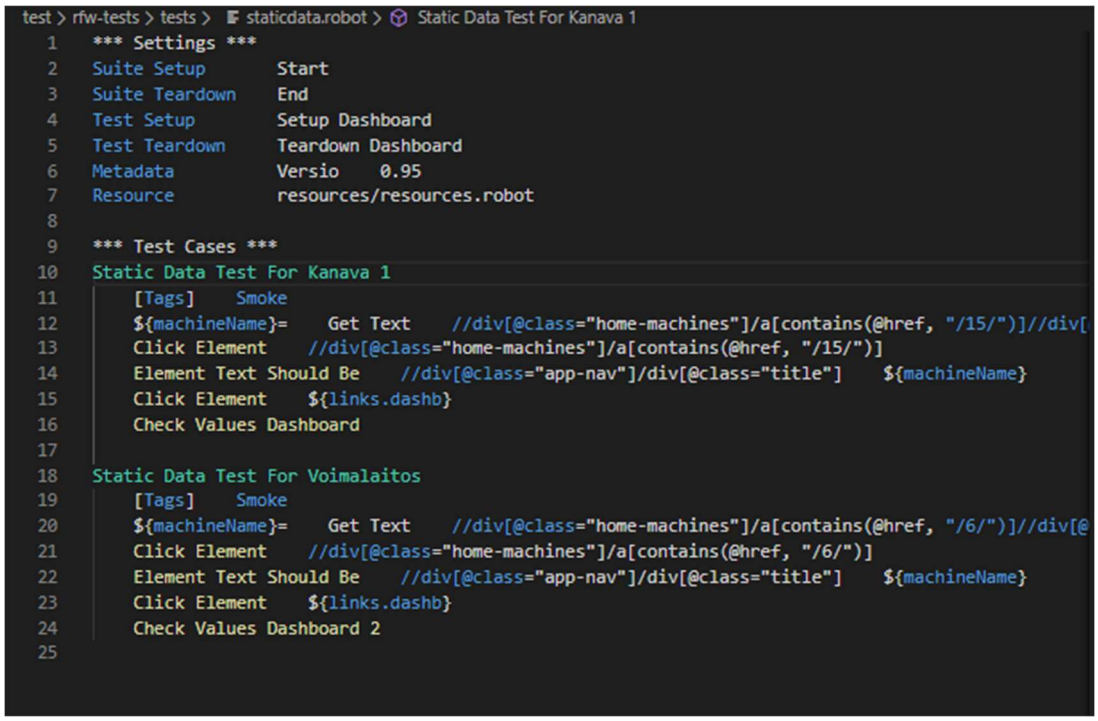
Robot Frameworkillä oli vaikeuksia aloittaa itse testien kirjoittaminen ja suorittaminen, koska siihen vaadittiin paljon alustusta ennen kuin sai pienen joukon testitapauksia ajettua läpi. Ensin täytyi työpisteelle asentaa viimeisin versio Pythonista, jonka avulla pääsee vasta asentamaan Robot Frameworkin ja sille tarpeelliset kirjastot. Selaimien ajurit pystyttiin asentamaan, joko WebdriverManagerin avulla tai manuaalisesti. Python ja selaimien ajurit piti myös erikseen vielä määrittää Windows-koneella ympäristömuuttujiin. Kun työkalut, tarpeelliset ajurit ja kirjastot oli asennettu, pystyi aloittamaan testitapausten kirjoittamista ja rakenteen suunnittelua. Testien kirjoittamisen ja ylläpidon voi millä tahansa työkalulla tehdä hankalaksi, jos niille ei suunnittele rakennetta valmiiksi. Robot Frameworkissa rakenne jaetaan yksinkertaisuudessaan testi- ja resurssitiedostoiksi. Kuviossa 5 näkyy tiedostorakenne, jota työssä käytettiin. Tests-kansion juuressa on 4 eri testitiedostoa, joihin määriteltiin suoritettavat testitapaukset. Resources-kansion juuresta löytyy pääresurssi-tiedosto, johon määriteltiin työssä käytettäviä kirjastoja ja lisäosia. Keywords-kansioon lisättiin käyttäjän määrittämät omat avainsanat ja variables-kansiossa omaan yaml-tiedostoon lisättiin tarvittavat muuttujat.



Kuvio 5. Tiedostorakenne Robotin testeille

Resurssitiedostossa (ks. liite 1) määritellään, mitä kirjastoja, käyttäjän luomia avainsanoja ja mahdollisia muuttujia testeissä hyödynnetään. Testitiedostoissa määritellään testitapaukset (ks. kuvio 6), joita itse Robot Framework ajaa läpi suoritusvaiheessa. Testitiedostoissa täytyy määritellä resurssitiedosto, mahdolliset metatiedot

ja muut asetukset. Ilman resurssitiedostoa Robot Framework ei ymmärrä avainsanoja, joita testitapauksissa käytetään.



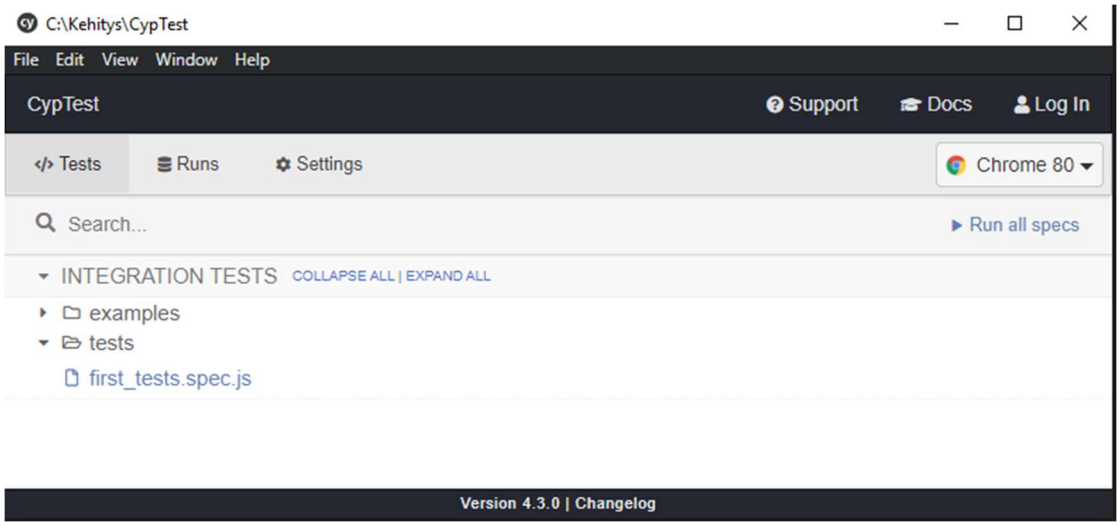
Kuvio 6. Esimerkkitestitiedosto

Robot Frameworkillä testien suorittamiseen tarvitaan vain komento ”robot” ja mistä itse testit löytyvät. Jos komennossa ei erikseen kerrota, mihin Robotti luo raportit testiajosta, tiedostot luodaan suoraan kansiorakenteen juureen, jossa komento suoritetaan. Robot Framework luo suorituksen lopuksi raportin, joka koostuu kolmesta eri tiedostosta. XML-tiedosto sisältää kaikki vaiheet, mitä Robotti suoritti alusta loppuun testitiedostossa, ja kahteen HTML-tiedostoon se luo helposti luettavat versiot testiajon suorituksista. Raportista on esimerkki liitteessä 2.

4.2.3 Cypress

Cypressin käyttöönottaminen verrattuna Robot Frameworkkiin oli paljon yksinkertaisempi. Cypressin asentamiseen ja käyttämiseen vaaditaan Node.js, joka on tarkoitettu JavaScript-koodin suorittamiseen. Cypressin asentamisen jälkeen se on valmis

käytettäväksi. Työkalua käynnistäessä se käy ensin läpi käyttöjärjestelmän kaikki selaimet, jotka ovat käytettävissä testaamiseen. (Cypress Systen Requirements 2020.) Sen mukana tulee liuta testiesimerkkejä, joilla pääsee hyvin mukaan testien kirjoittamiseen. Cypressin tulee käynnistää samassa sijainnissa, mihin se asennettiin. Cypress lisää kaikki testitapaukset, jotka löytyvät suoritetusta tiedostosijainnista. Kuviossa 7 ohjelmaikkunan otsikossa näkyy missä sijainnissa Cypress on käynnistetty ja mistä sijainnista se hakee testitapaukset.



Kuvio 7. Cypressin käyttöliittymä

Cypressillä testien kirjoittaminen ei kuitenkaan ollut helpoimmasta päästä. Jos JavaScript on käyttäjälle uusi juttu, on hänellä todennäköisesti hankaluuksia aloittaa testien kirjottamista. Huono puoli Cypressiä on myös testien rinnakkaissuorittaminen. Kun testien määrä kasvaa, kasvaa myös niiden kokonaissuoritusaika. Tuota kyseistä aikaa saadaan pudotettua, jos testejä voidaan ajaa rinnakkain. Teknisesti testejä voisi suorittaa rinnakkain samassa ympäristössä, mutta Cypressin tekijät ei sitä suosittele. Cypressin toimiessa todella raskaasti tekijät suosittelevat pystyttämään Cypress-instansseille omat ympäristöt kuormituksen pienentämiseksi (Cypress Parallezation 2020).

4.3 Virtuaaliympäristö

4.3.1 Ympäristön valinta

Ympäristön valinnassa tuli ottaa huomioon, että se olisi mahdollisimman kevyt, helppo pystyttää ja ylläpitää. Oman palvelimen varaaminen pelkästään testaamiselle varaisi vain turhaan resursseja silloin, kun ei ajeta testejä. Lisäksi palvelimen ylläpitäminen versiopäivityksineen tuli huomioida. Oman palvelimen pystyttämisessä oli paljon huonoja puolia, joten piti lähteä selvittämään muita vaihtoehtoja. Pinjalla monissa projekteissa käytettiin Google Cloud Platformia, mihin aina tarvittaessa Kubernetes Enginelle luotiin virtuaaliympäristöjä Docker-konteilla. Docker-kontit ovat kevyitä, nopeita pystyttää, helppoja ylläpitää ja niistä on olemassa paljon valmiita kuvakkeita eri käyttötarkoituksiin.

Robot Frameworkin käyttämiseen oli olemassa monia erilaisia kontteja, joista yksikään ei vastannut halutun testaamisen tarpeita. Testiympäristöstä halusin mahdollisimman pienen ja kevyen, joten hyväksi vaihtoehdoksi osoittautui linux-pohjaisista käyttöjärjestelmistä Alpine. Alpine sivujen mukaan Alpine-kontti ei vie tilaa kuin 8 megabittiä, joten kontin uudelleen lataaminen ei veisi aikaa eikä kaistaa (Alpine n.d). Lataamisen jälkeen kontin käyttöönottamisessa kuluisi vielä aikaa, koska sinne pitäisi erikseen asentaa Pythonin paketit ja Pythonilla pitäisi vielä asentaa Robot Framework ja sille tarvittavat kirjastot.

Docker Hub on paikka missä pystyy luomaan, hallitsemaan ja toimittamaan kontteja kenen tahansa käyttöön (Rouse 2017). Sieltä voi etsiä virallisten tahojen tekemiä kontteja ja ottaa niitä käyttöön omien käyttötarpeiden mukaan. Python ylläpitää sivustolla kontteja eri käyttöjärjestelmillä, joihin on valmiiksi asennettu jokin tietty versio Pythonista. Niiden joukosta löysin työhön juuri sopivan kuvakkeen kontista, jossa käyttöjärjestelmänä toimi viimeisin versio Alpinesta ja Pythonista. Kyseisen kontin löytäminen pienentää aikaa käyttöönotossa ja vähentää työtä alun konfiguraatiossa.

4.3.2 Ohjelmistojen asentaminen

Selaimelle tarkoitettujen automaatiotestien suorittamiseksi, täytyy konttiin asentaa virtuaalinen ruutupuskuri. Kyseinen ohjelma mahdollistaa graafisten ohjelmien käyttämisen virtualisoidussa ympäristössä, jossa ei ole fyysisiä näyttöjä käytössä. Siksi konttiin asennetaan X Virtual Framebuffer. Ennen testaamisen aloittamista se täytyy alustaa halutuilla parametreilla, kuten haluttu resoluutio ja mihin porttiin se halutaan toimintaan (Wiggins n.d).

Konttiin tarvitsi asentaa kevyt selain, jotta automaatiotestaus tapahtuisi mahdollisimman kevyesti ja rinnakkaisten testien ajaminen ei kuormittaisi ympäristöä liikaa. Yksi GEMAlle asetetuista rajoitteista oli, että sen tulisi toimia Chromella. Chrome on Googlen ylläpitämä versio Chromiumista. Chromium on Googlen avoimen lähdekoodin projekti. Chromen ja Chromium eivät eroa toisistaan paljoa, mutta loppukäyttäjälle erot ovat merkittävät. Chrome esimerkiksi sisältää codecs-lisenssin, mikä tarjoaa tuen usealle erille videoformaatille eli videoiden katsominen onnistuu paremmin eri sivuilla. Chrome myös sisältää Googlen automaattiset päivitykset, virheiden ja poikkeusten raportoinnin ja Adobe Flash -lisäosan. (Hoffman 2018) Kuitenkaan näitä ominaisuuksia GEMA ei tarvitse selaimelta, joten testaamiseen riittää Chromium. Kun Chromiumia suoritetaan kontissa, joka on pieni ja kevyt, pitää käynnistysasetuksiin lisätä muutama argumentti selaimen toiminnan varmistamiseksi. Laitteistokiihdytys tulee ottaa pois käytöstä argumentilla "--disable-gpu", koska kontissa ei ole erillistä näytönohjainta. Myös jaetun muistin käyttäminen otetaan selaimelta pois käytöstä argumentilla "--disable-dev-shm-usage", koska konteissa on oletuksena vain 64 megabittiä tilaa ja se ei riitä isojen sivujen renderöimiseen Chromiumissa. Muistin loppuminen johtaisi selaimen kaatumiseen, mitä testien suorituksen aikana ei haluta.

Robot Frameworkin asentaminen onnistuu pip-työkalulla, joka on tarkoitettu Python-pakettien asentamiseen. Robot Frameworkin lisäksi pip:llä asennetaan Selenium-Library ja pabot. Näiden jälkeen ympäristö on valmis testien ajamiseen, mutta parannuksia voi vielä tehdä asennusvaiheisiin. Luomalla valmiiksi muutaman shell script -tiedoston pystyy kontin käyttöä yksinkertaistamaan. Asennusvaiheelle luodaan oma skriptitiedosto, jossa tiivistetään kaikki aikaisemmat vaiheet ja lisätään asennusvai-

heet kirjaamaan lokia. Asennuslokeista ympäristön pystyttämisen jälkeen voi tarkastella eri ohjelmistojen asennusversioita ja onnistuiko kaikkien ohjelmistojen asentaminen halutusti. Pip:llä halutut paketit voidaan asentaa niin, että paketit on kerrottu yhdessä tiedostossa. Kuviossa 8 yläosassa näkyy työssä asennetut paketit ja kuvion alaosassa asennusskriptin vaiheet.

requirements.txt

test > rfw-tests > hlp > requirements.txt

```
1  robotframework
2  robotframework-seleniumlibrary>=4.2.0
3  robotframework-pabot
4  pyyaml
```

install.sh

test > rfw-tests > hlp > install.sh

```
1  #!/bin/bash
2  echo "Installing chromium chromium-chromedriver udev xvfb nmap and requirements.txt"
3  mkdir test/rfw-tests/logs/
4  mkdir test/rfw-tests/reports/
5  apk add --no-cache tzdata chromium chromium-chromedriver udev xvfb nmap > test/rfw-tests/logs/apk_ins.log
6  pip install --no-cache-dir -r test/rfw-tests/hlp/requirements.txt > test/rfw-tests/logs/pip_ins.log
7  apk update -q
8  apk upgrade -q
9  rm -rf /var/cache/apk/* test/rfw-tests/hlp/requirements.txt
10 chmod +x test/rfw-tests/hlp/entry_point.sh
11 dos2unix test/rfw-tests/hlp/entry_point.sh
12 apk info -vv | sort > test/rfw-tests/logs/apk.log
13 pip freeze > test/rfw-tests/logs/pip.log
```

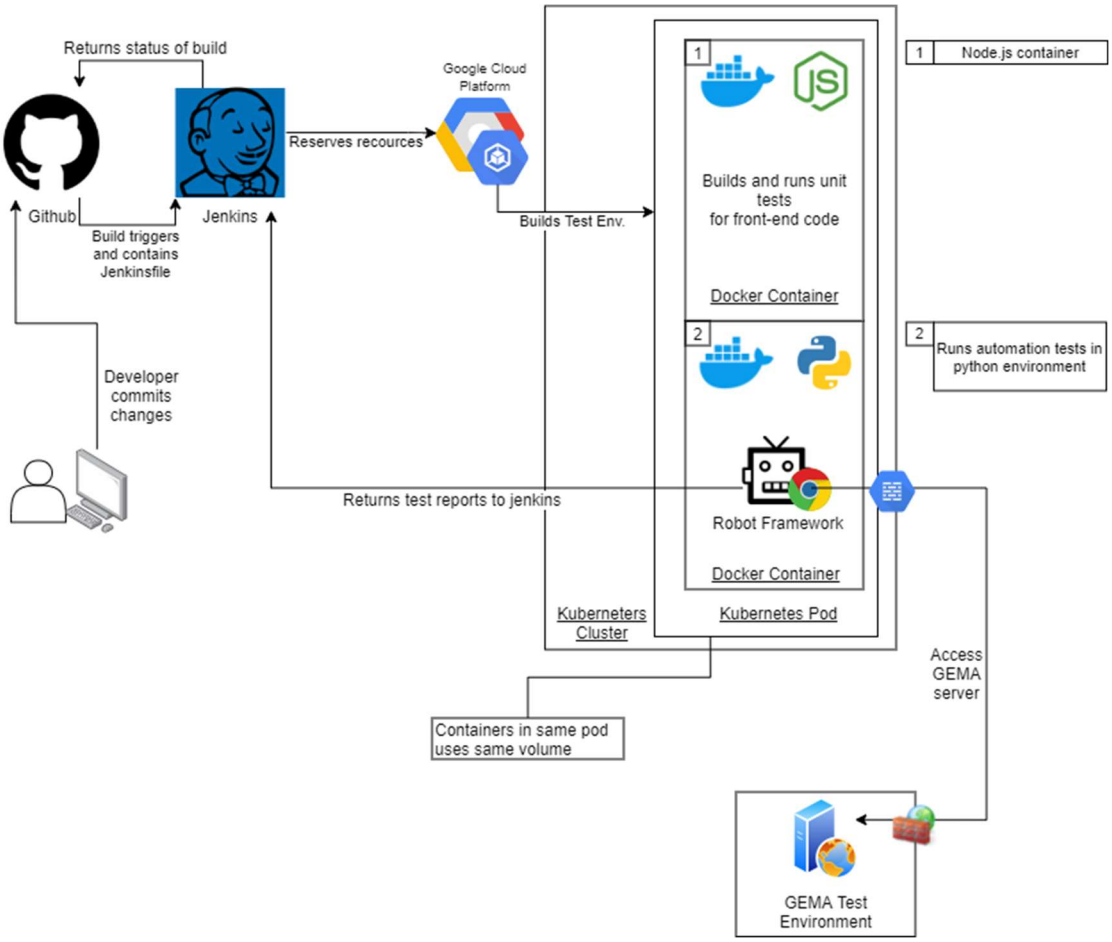
Kuvio 8. Asennusskripti ja pip:n asennuspaketit

4.4 Implementointi tuotantoon

Kontti oli tässä vaiheessa valmis testattavaksi tuotannossa ja sen kääntämisympäristöissä. Automaatiotestien suorittaminen haluttiin osaksi GEMAn front-end koodin julkaisuprosessiin. Front-end-koodin kääntäminen ja yksikkötestien suorittaminen tapahtui Pinjan ylläpitämissä ympäristöissä. Front-end-projektin mukana oli olemassa Jenkinsfile, millä määritetään koko koodin kääntäminen ja yksikkötestaaminen eli Jenkins pipeline. Pinjan Jenkins aloittaa aina määritetyn prosessin, kun se löytää projektista Jenkinsfilen. Jenkinsissä on käytössä kubernetes-lisäosa (Kubernetes plugin, 2020), mikä mahdollistaa orkestroidun konttien pystyttämisen Jenkinsfileen määritettynä. Kubernetes-lisäosa luo dynaamisesti agentin kubernetes klusteriin eli tässä tapauksessa Google Cloud Platformissa toimivaan kubernetesiin. GEMAn front-end käyttää kääntämiseen ja yksikkötesteihin Node.js-konttia, mikä taas ei sovellu Robot Frameworkin pyörittämiseen.

Kääntöprosessiin otetaan mukaan aikaisemmin Robot Frameworkiä varten luotu Python-kontti. Kontti lisätään samaan koteloon, missä Node.js-kontti toimii. Kotelossa kontit jakavat keskenään saman levyosion, minne ladataan Githubista GEMAn front-end-lähdekoodi. Lähdekoodin mukana on Python-konttiin tarvittavat asennustiedostot ja Robot Frameworkillä ajettavat automaatiotestit. Saman levyosion jakaminen pienentää levytilan käyttöä ja vähentää lataamistarpeita. Kääntämis- ja testaamisprosessin jälkeen asennuslokit ja automaatiotestien tulokset halutaan jakaa käyttäjälle nähtäväksi, siinä käytetään hyväksi Jenkinsin HTML Publisher -lisäosaa. Lisäosalle kerrotaan mitä tiedostoja halutaan ottaa mukaan, mistä ne löytyvät ja millä nimellä ne julkaistaan. Lokit ja raportit julkaistaan Jenkinsissä osaksi käännöshistoriaa. Koko prosessi kuvattuna kuviossa 9.

Jenkinsfilessä voi erikseen rajoittaa tiettyjä vaiheita määrittelemällä ehtoja. Säännöllisen lausekkeen (regex) avulla loin ehdot automaatiotestien suorittamiseen. Testit ajettaisiin vain silloin, kun koodista ollaan tekemässä käännöstä master-, dev- tai release kandidaatti -haarakkeesta. Testien suoritus rajoitetaan tiettyihin haarakkeisiin, ettei testiautomaatio ole käyttämässä turhaan Google Cloud Platformin resursseja, kun koodista lisätään muutoksisia versionhallintaan. Ilman määriteltyjä rajoituksia regexillä, testiautomaatiot suoritettaisiin jokaisessa lähdekoodin muutoksessa. Liitteenä 3 työssä käytetty Jenkinsfile esimerkkinä.



Kuvio 9. Käännös- ja testiprosessi

5 Testaus ja skaalautuvuus

5.1 Automaatiotestit

Kun testausympäristö oli todettu vakaaksi ja toimivaksi, niin oli mahdollista aloittaa automaatiotestien kirjoittaminen. Mitä enemmän testejä oli Robot Frameworkille kirjoitettu, sitä paremmin ympäristöä pystyttiin testaamaan, tarkastelemalla testien suoritusaikoja ja ympäristön kuormitusta. Automaatiotestejä ollessa tarpeeksi, pystyi nykyisessä ympäristössä aloittamaan testaamista ja lähteä etsimään optimaalisia asetuksia testien ajamiseen.

Ensimmäiseksi ongelmaksi ympäristössä tuli keskusmuistin riittävyys. Kun halutaan suorittaa selaimella toimivaan käyttöliittymään tarkoitettuja testejä rinnakkain, tulee huomioida, että X Virtual Framebuffer ja neljä rinnakkain auki olevaa selainta vievät

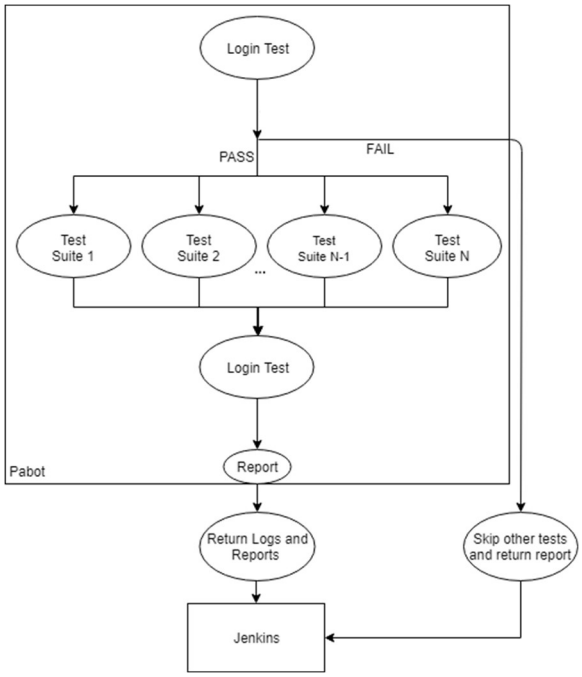
melkein 1 gigabitin verran keskusmuistia. Keskusmuistia siis tulee kyseisessä ympäristössä varata ainakin 1,5 gigabittiä.

Pabotille optimaalisin asetus rinnakkaisuuksien määräksi yhdellä virtuaalisella prosessorilla oli kaksi. Jos rinnakkaisuuksia olisi pabotilla käyttänyt kolmea tai neljää olisi kokonaissuoritusaika testeillä kaksinkertaistunut. Testauksessa käytettiin neljää testitiedostoa, joissa yhteensä oli 27 testitapausta, kuten kuviossa 10 näkyy. Kuviossa Total-sarakkeessa näkyy ensin testitapausten kokonaismäärä ja sen jälkeen montako testitapausta kussakin testitiedostossa on.

Statistics by Suite	Total
Suites	27
Suites.Dashboard	4
Suites.Features Dashboard	15
Suites.Report2	6
Suites.Staticdata	2

Kuvio 10. Testien jakautuminen testitiedostoissa

Automaatiotestien rakenne GEMAlle toteutettiin niin, että jokainen testitiedosto alkaa kirjautumisen testaamisella. Kirjautumistesti on tietoturvallisesti ja tuotteen käytön kannalta kriittisin. Jos kirjautumistesti ei mene läpi on testitiedoston muita testejä turha suorittaa. Testitiedostojen rinnakkaissuoritusmäärä riippuu pabotille määritetystä arvosta. Kuviossa 11 ”Test Suite” eli testitiedostoja suoritetaan rinnakkain N määrä, joka on pabotille erikseen kerrottu rinnakkaisuuksien määrä.



Kuvio 11. Testien suoritus prosessi vuokaaviona

Pabot vakiona suorittaa testit tiedostonimen perusteella aakkosjärjestyksessä. Suorituksen aikana pabot luo väliaikastiedoston ”.pabotsuitenames”, johon se tallentaa testien nimet niiden suoritusajan perusteella ja jos tiedosto on vielä seuraavalla suorituskerralla olemassa, niin pabot aloittaa testien ajamisen tiedostosta, jossa oli pisin suoritusaika.

5.2 Testaamista Azuressa

Toiseksi alustaksi työn testausympäristön pystyttämiseksi valittiin Microsoftin Azure DevOps. Tutkiminen ja testaaminen oli hyvä aloittaa siitä, kuinka aikaisemmin luotu testausympäristö skaalautuu DevOpsille ja kuinka paljon muutoksia DevOpsin ympäristössä tulee ottaa huomioon. Työtä tehdessä ei ollut mahdollisuuksia käyttää DevOpsin Kubernetes klusteria, johon olisi voinut pystyttää nykyisen version testausympäristöstä ilman isoja muutoksia.

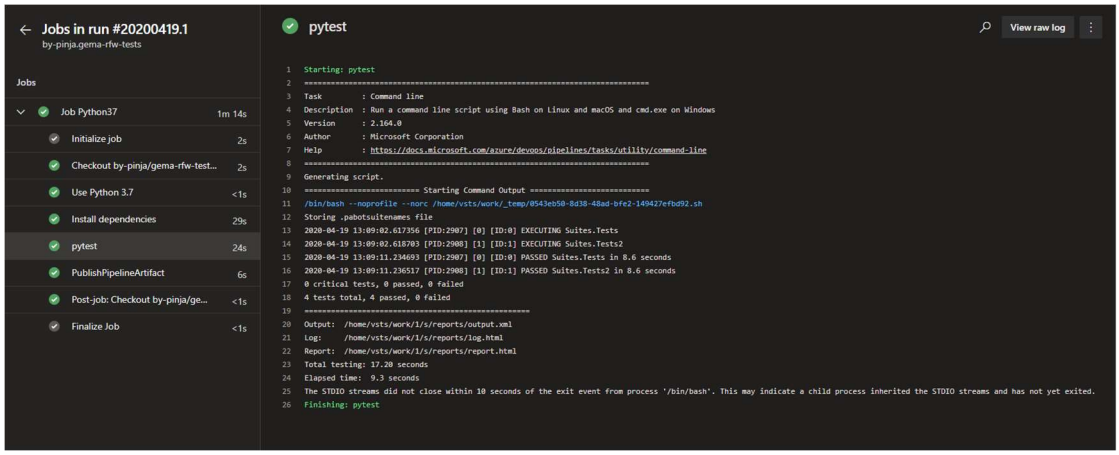
DevOpsiin siirtyessä luotiin ensin uusi projekti, mihin suoritetaan testit ja tallennetaan tulokset. DevOpsissa ympäristön luonnin myös pystyy hyvin yksinkertaisesti määrittämään yhden tiedoston avulla ja tässä projektissa käytettiin azure-pipelines.yml tiedostoa, joka löytyy liitteenä 4. DevOpsin pipelinen määrittelytiedosto ei

kokonaisuudeltaan poikkea paljoa Jenkinsfilestä, suurimmat erot ovat erityyppiset tiedostot ja vaiheiden määrittely.

Ensin tiedostoon määriteltiin mitä virtuaalikuvaketta ympäristössä tullaan käyttämään. Jos projektille haluaa rinnakkaissuorituksia eri ympäristöissä tai ohjelmistoilla, niin ne voidaan määrittää matriiseissa. Kyseisessä projektissa käytettiin Ubuntun viimeisintä versiota ja matriisissa otetaan käyttöön Pythonin 3.7 versio. Vaiheet DevOpsin pipelinessa jaetaan neljään. Ensimmäisessä vaiheessa kerrotaan järjestelmälle, että kyseisessä ympäristössä otetaan matriiseissa määritelty versio Pythonista käyttöön. Toisessa vaiheessa suoritetaan skripti, missä asennetaan tarvittavat työkalut ja ohjelmistot. Skriptin pystyi ilman suuria muokkauksia ottamaan aikaisemman ympäristön luonnista ja muokkaamaan Alpinen asennuskomennot sellaisiksi, että ne toimivat Ubuntulla. Kolmannessa vaiheessa tulivat X Virtual Framebufferin käynnistäminen ja automaatiotestien suorittaminen. Viimeisessä vaiheessa lokit ja testien tulokset julkaistaan artefaktina osana pipelinen työtä, missä käyttäjä voi tarkastella suorituksen tuloksia.

Pipelinea tehdessä, linkitetään ensin Github-projekti DevOpsiin ja kerrotaan mitä tiedostoa palvelun pipeline tulee käyttämään projektia kääntäessä. Tiedoston määrittämisen aikana pystytään pipelinen asetuksiin määrittämään ympäristömuuttujia, joita tarvitaan testaamisen aikana ja ympäristöä luodessa. Ympäristömuuttujia pystyy DevOpsissa myös määrittämään salaisiksi, mikä mahdollistaa esimerkiksi salasanojen tai muun salattavien tietojen määrittämisen.

Kun projektin määrittely DevOpsissa on valmis, voi pipelinen suorittaa manuaalisesti tai miten azure-pipeline.yml tiedostossa on määritelty. Jotta DevOpsissa voitiin todeta automaatiotestien toimivuus, luotiin sitä varten kaksi testiasua, joissa kummasakin oli kaksi identtistä testitapausta. Testitapauksissa testattiin, että selaimen aukaiseminen, kuvankaappaus selainikkunasta ja selaimen konsolilokien tallentamiseen tarkoitettu avainsana toimii halutusti. Kuviossa 12 näkyy DevOps pipelinen suoritusvaiheet ja pytest-ikkunassa Robot Frameworkin ajamat testit.



Kuvio 12. DevOps Pipelinen suorittama työ

6 Johtopäätökset ja pohdinta

Työn tavoitteena oli toteuttaa GEMAn tuotekehitykseen toimiva ratkaisu automaatiotestausympäristöstä. Tavoitteena oli myös tehdä ratkaisusta geneerinen, jotta sitä voidaan käyttää hyödyksi muissa Pinjan projekteissa. Lisäksi tavoitteena oli laajentaa omaa osaamista testaamiseen, sen työkaluihin, ympäristöihin ja sen automatisointiin liittyen. Työn tuloksena saatiin GEMAn tuotekehitykseen mukaan testausautomaatio, joka pohjautuu kontteihin ja sen ansioista on skaalatuvissa muihinkin projekteihin. Työtä tehdessä opin paljon uutta Dockerista, Pythonin koodaamisesta, Robot Frameworkista ja sen muista työkaluista, CI/CD-työkaluista, Kubernetesista ja testaamisen käytänteistä.

Työssä onnistuttiin luomaan kevyt ja skaalautuva ratkaisu testausautomaatiosta ja sen tuominen osaksi GEMAn tuotekehitystä. Testausautomaation lisäksi työssä GEMAlla saatiin määriteltyä automaatiotestejä. Määriteltyjen testien avulla tuotteelle on helppoa lähteä suunnittelemaan ja toteuttamaan lisää automaatiotestejä vanhoille sekä uusille ominaisuuksille. Automaatiotesteihin saatiin myös kehitettyä omia avainsanoja. Omien avainsanojen avulla pystyy esimerkiksi käymään läpi häiriöraportoinnissa syiden puurakenteen kaikki kohdat ja keräämään kaikki infotasoa korkeammat lokit tiedostoon selaimen konsolista.

Työstä jäi uupumaan konkreettiset tutkimustulokset testaustyökalujen väliltä, joilla olisi saanut parempaa vertailutietoa esimerkiksi testien kuormittavuudesta ja kokonaissuoritusajoista. Skaalautuvuuden tutkiminen ei onnistunut halutusti, koska Azure DevOpsin Kubernetes ei ollut silloin käytettävissä, jolla olisi voinut testata nykyistä ratkaisua kyseisessä palvelussa.

Lopullinen työ rajoittuu käyttämään konttia, johon on valmiiksi asennettu tietty versio Pythonista. Sellaisen kontin käyttö rajoittaa testaustyökalun valintaa ja pitkällä juoksulla voi aiheuttaa versioiden vanhentumista. Esimerkiksi uudempi versio jostain tietystä ohjelmistosta kuten Robot Frameworkista ei toimi, jos käyttäjä ei muista seurata onko Alpinesta tai Pythonista tullut uutta konttia, jossa on uudemmat versiot käytössä. Työn lopullinen tulos vastaa toimeksiantajan asettamia vaatimuksia ja työllä on kaikki valmiudet jatkokehitystä varten.

Työtä tehdessä opin ymmärtämään paremmin testausautomaation tärkeydestä osana tuotekehitystä. Testausautomaation avulla tuotteesta pystyttiin löytämään virheitä, joita aikaisemmin ei ollut todettu tai pystytty toistamaan erilaisten selainasetusten takia. Automaatiotestejä kirjoittaessa tuotteesta myös paljastui paljon sellaisia ominaisuuksia, joita aikaisemmin ei ollut otettu testaussuunnitelmissa huomioon. Tästä opin ominaisuuksien määrittelyn tärkeydestä ja sen dokumentoinnista.

Jatkokehitys

Työ saataisiin paremmin skaalautuvaksi, jos nykyisestä ratkaisusta tekisi oman kontin ja se julkaistaisiin Docker Hubiin. Se vaatisi silloin enemmän ylläpitotyötä, mutta helpottaisi työn jakelua muihin projekteihin. Oman kontin ylläpidossa tulisi ottaa huomioon alkuperäisen kontin eri versiot. Testitulosten raporttien palauttamiseen voisi luoda oman paikan, missä tulokset ryhmiteltäisiin GEMA-version mukaan. Testitulosten ollessa yhdessä paikassa, helpottuisi niiden hyödyntäminen osana tuotekehitystä. Kontista voisi myös tehdä eri versioita, joissa käytettäisiin muuta kuin Alpinea käyttöjärjestelmänä. Käyttöjärjestelmän vaihto esimerkiksi Ubuntuun mahdollistaisi selaimista tuoreimpien versioiden käytön. Konttiin myös voisi luoda mahdollisuudet käyttää Cypressiä testaustyökaluna, mistä voisi lähteä jatkokehittämään rinnakkaisten Cypress-konttien orkestrointia.

Lähteet

ARROW Andon. N.d. Andon-tuotteen esittely ARROW Engineering Oy:n verkkosivuilla. Viitattu 7.2.2020. https://www.arroweng.fi/ratkaisut/andon/ .

ARROW GEMA. N.d. GEMA-tuotteen esittely ARROW Engineering Oy:n verkkosivuilla. Viitattu 7.2.2020. https://www.arroweng.fi/ratkaisut/gema/ .

Alpine About. N.d. Alpine-järjestelmän kotisivut. Viitattu 8.4.2020. https://alpine-linux.org/about/ .

Azure DevOps. N.d. Azure DevOpsin esittely Microsoftin sivuilla. Viitattu 6.4.2020. https://azure.microsoft.com/en-us/services/devops/ .

Cypress Parallezation. 2020. Cypressin kuvaus rinnakkaistamisesta. Muokattu 6.4.2020. Viitattu 7.4.2020. https://docs.cypress.io/guides/guides/parallelization.html .

Cypress System Requirements. 2020. Ohje Cypressin asentamiseen. Muokattu 6.4.2020. Viitattu 7.4.2020. https://docs.cypress.io/guides/getting-started/installing-cypress.html#System-requirements .

Finley, K. 2012. What Exactly Is GitHub Anyway? Artikkeli TechCrunch-sivuilla. Julkaistu 14.7.2012. Viitattu 6.4.2020. https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/ .

General Python FAQ. N.d. Pythonin useimmiten kysytyt kysymykset -sivu. Viitattu 3.4.2020. https://docs.python.org/3/faq/general.html .

Heller, M. 2020. What is Jenkins? The CI server explained. Artikkeli InfoWorld-sivuilla. Julkaistu 9.3.2020. Viitattu 6.4.2020. https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html .

Hoffman, C. 2018. What's the Difference Between Chromium and Chrome?. Artikkeli How-To Geek-sivuilla. Muokattu 29.3.2018. Viitattu 8.4.2020. https://www.howtogeek.com/202825/what%E2%80%99s-the-difference-between-chromium-and-chrome/ .

How it works. N.d. Cypress-työkalun kuvaus. Viitattu 3.4.2020. https://www.cypress.io/how-it-works .

Klarck, P. 2019. Robot Frameworkin esittely Github-sivulla. Viitattu 27.3.2020. https://github.com/robotframework/robotframework .

Korpela, M. 2020. Pabot-työkalun esittely Github-sivulla. Viitattu 3.4.2020. https://github.com/mkorpela/pabot .

Kubernetes plugin. 2020. Kubernetes-lisäosan esittely Github-sivulla. Viitattu 15.4.2020. https://github.com/jenkinsci/kubernetes-plugin .

Overview. 2020. Google Cloud Platformin yleiskatsaus. Viitattu 3.4.2020 https://cloud.google.com/docs/overview .

Rouse, M. 2017. Docker Hub. Artikkeli TechTarget-sivuilla. Muokattu 9.2017. Viitattu 8.4.2020. https://searchitoperations.techtarget.com/definition/Docker-Hub .

SeleniumLibrary. 2020. SeleniumLibrary esittely Github-sivulla. Viitattu 27.3.2020. https://github.com/robotframework/SeleniumLibrary .

What container. N.d. Kuvaus konteista Dockerin sivuilla. Viitattu 3.4.2020. https://www.docker.com/resources/what-container .

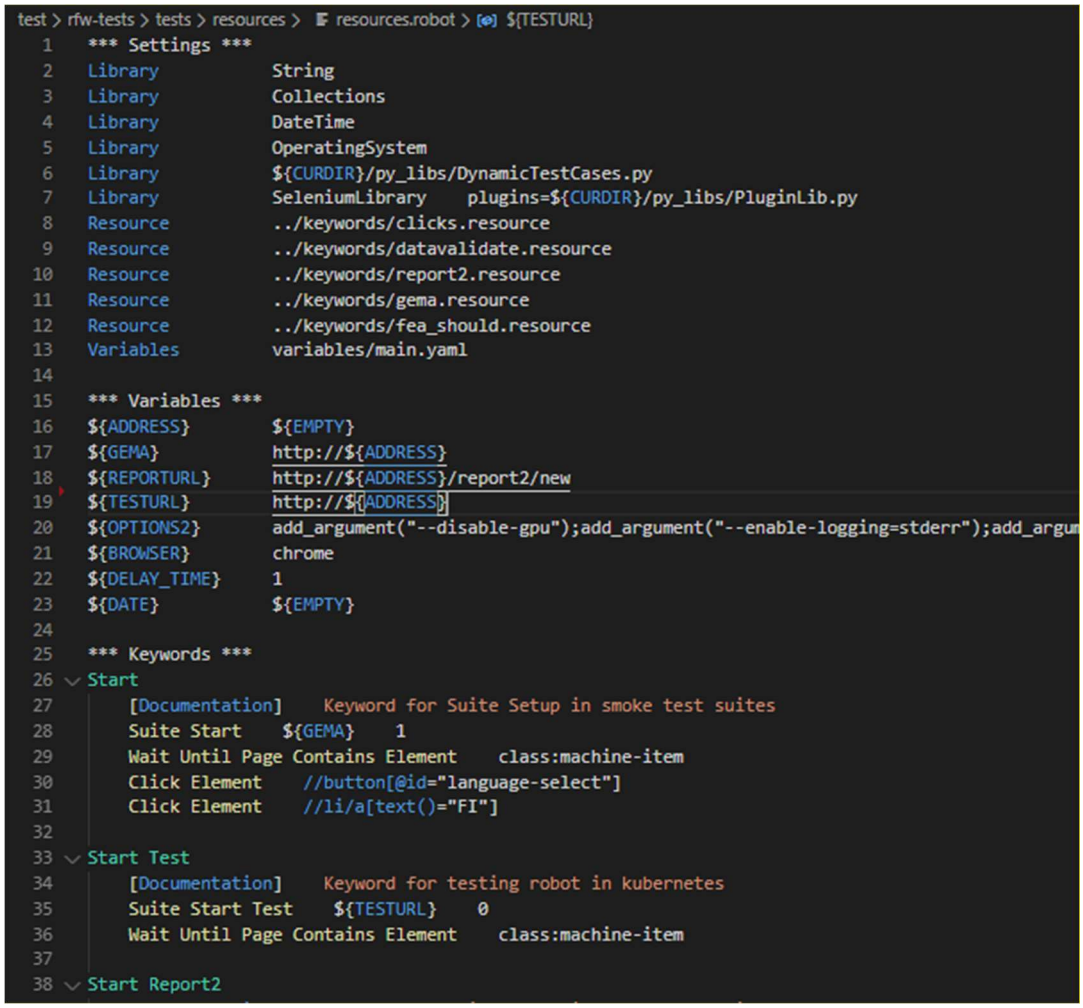
What is Kubernetes. 2020. Kubernetesin esittely. Viitattu 3.4.2020. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ .

Wiggins, D. N.d. X Virtual Framebuffer julkaisudokumentaatio. Viitattu 8.4.2020. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml .

Yrityskauppa. 2018. Uutinen Protaconin sivuilla. Julkaistu 2.8.2018. Viitattu 6.9.2019. https://www.protacon.com/ajankohtaista/yrityskauppa-protacon-osti-teollisuuden-digitalisoijan-arrown/ .

Liitteet

Liite 1. Robot Framework testien pääresurssi-tiedosto



```
test > rfw-tests > tests > resources > resources.robot > ${TESTURL}
*** Settings ***
Library           String
Library           Collections
Library           DateTime
Library           OperatingSystem
Library           ${CURDIR}/py_libs/DynamicTestCases.py
Library           SeleniumLibrary    plugins=${CURDIR}/py_libs/PluginLib.py
Resource          ../keywords/clicks.resource
Resource          ../keywords/datavalidate.resource
Resource          ../keywords/report2.resource
Resource          ../keywords/gema.resource
Resource          ../keywords/fea_should.resource
Variables         variables/main.yaml

*** Variables ***
${ADDRESS}        ${EMPTY}
${GEMA}           http://${ADDRESS}
${REPORTURL}      http://${ADDRESS}/report2/new
${TESTURL}        http://${ADDRESS}
${OPTIONS2}       add_argument("--disable-gpu");add_argument("--enable-logging=stderr");add_argu
${BROWSER}        chrome
${DELAY_TIME}     1
${DATE}           ${EMPTY}

*** Keywords ***
Start
    [Documentation]    Keyword for Suite Setup in smoke test suites
    Suite Start    ${GEMA}    1
    Wait Until Page Contains Element    class:machine-item
    Click Element    //button[@id="language-select"]
    Click Element    //li/a[text()="FI"]

Start Test
    [Documentation]    Keyword for testing robot in kubernetes
    Suite Start Test    ${TESTURL}    0
    Wait Until Page Contains Element    class:machine-item

Start Report2
```

Liite 2. Robot Framework testien rapotti

Basic Tests Log

Generated
20190516 12:48:25 UTC+03:00
186 days 22 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	4	4	0	00:00:19	
All Tests	4	4	0	00:00:19	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Basic Tests	4	4	0	00:00:29	

Test Execution Log

SUITE **Basic Tests** 00:00:28.822
Full Name: Basic Tests
Source: C:\Kehitys\RobotTests\basic_tests.robot
Start / End / Elapsed: 20190516 12:47:57.104 / 20190516 12:48:25.926 / 00:00:28.822
Status: 4 critical test, 4 passed, 0 failed
4 test total, 4 passed, 0 failed

SETUP resources . **Start** ${BROWSER} 00:00:07.089
Start / End / Elapsed: 20190516 12:47:57.234 / 20190516 12:48:04.323 / 00:00:07.089
KEYWORD SeleniumLibrary . **Open Browser** ${URL}, ${BROWSER} 00:00:05.466
KEYWORD SeleniumLibrary . **Set Selenium Speed** ${DELAY_TIME} 00:00:00.000
KEYWORD SeleniumLibrary . **Reload Page** 00:00:01.622

TEARDOWN resources . **End** 00:00:03.038
Start / End / Elapsed: 20190516 12:48:22.888 / 20190516 12:48:25.926 / 00:00:03.038
KEYWORD SeleniumLibrary . **Close Browser** 00:00:03.038

TEST **First test for Robot** 00:00:02.026
Full Name: Basic Tests.First test for Robot
Start / End / Elapsed: 20190516 12:48:04.323 / 20190516 12:48:06.349 / 00:00:02.026
Status: PASS (critical)
KEYWORD SeleniumLibrary . **Element Text Should Be** class:nav-title, Aloitusnäkymä 00:00:02.026

TEST **Change Page** 00:00:06.302
Full Name: Basic Tests.Change Page
Start / End / Elapsed: 20190516 12:48:06.349 / 20190516 12:48:12.651 / 00:00:06.302
Status: PASS (critical)
KEYWORD resources . **Click Action** 00:00:02.056
Start / End / Elapsed: 20190516 12:48:06.349 / 20190516 12:48:08.405 / 00:00:02.056
KEYWORD SeleniumLibrary . **Click Element** //*[@id="root"]/div/div/div/div[1]/div[3]/div/div/div[1]/div 00:00:02.056
KEYWORD SeleniumLibrary . **Click Element** //*[@id="root"]/div/div/div/div[1]/div[1]/div/div[1]/div/div/div/div[2]/ul/li[2]/a 00:00:02.219
KEYWORD SeleniumLibrary . **Element Text Should Be** class:nav-title, Työt 00:00:02.027

TEST **Testing Mouse Over** 00:00:06.117
Full Name: Basic Tests.Testing Mouse Over
Start / End / Elapsed: 20190516 12:48:12.651 / 20190516 12:48:18.768 / 00:00:06.117
Status: PASS (critical)
KEYWORD SeleniumLibrary . **Mouse Over** class:action-button 00:00:02.027
KEYWORD SeleniumLibrary . **Mouse Out** class:action-button 00:00:04.088

TEST **Go back to Main Page** 00:00:04.119
Full Name: Basic Tests.Go back to Main Page
Start / End / Elapsed: 20190516 12:48:18.769 / 20190516 12:48:22.888 / 00:00:04.119
Status: PASS (critical)
KEYWORD resources . **Click Back** 00:00:02.082
Start / End / Elapsed: 20190516 12:48:18.770 / 20190516 12:48:20.852 / 00:00:02.082
KEYWORD SeleniumLibrary . **Click Element** //*[@id="root"]/div/div/div/div[2]/div[3]/div/div/div[1] 00:00:02.081
KEYWORD SeleniumLibrary . **Element Text Should Be** class:nav-title, Aloitusnäkymä 00:00:02.035

Liite 3. Työssä käytetty Jenkinsfile

```
podTemplate(label: pod.label,
  containers: pod.templates + [
    containerTemplate(
      name: 'node',
      image: 'node:10',
      alwaysPullImage: true,
      ttyEnabled: true,
      command: '/bin/sh -c', args: 'cat',
      resourceRequestMemory: '500M',
    ),
    containerTemplate(
      name: 'python',
      image: 'python:3.8.2-alpine3.11',
      alwaysPullImage: true,
      ttyEnabled: true,
      command: '/bin/sh -c', args: 'cat',
      resourceRequestCpu: '1',
      resourceLimitCpu: '2',
      resourceRequestMemory: '1200M',
      envVars: [
        containerEnvVar(key: 'DISPLAY', value: ':99'),
        containerEnvVar(key: 'TZ', value: 'Europe/Helsinki'),
        containerEnvVar(key: 'MONITOR', value: '1920x1080x16'),
        containerEnvVar(key: 'ROBOT_THREADS', value: '2'),
        containerEnvVar(key: 'ROBOT_OPTIONS', value: '--variable ADDRESS:194.136.236.130:100 --nostatusrc -n Noncritical')
      ]
    )
  ]
) {
  def project = 'gema-frontend'
  def branch = (env.BRANCH_NAME)

  node(pod.label) {
    try {
      stage('Checkout') {
        checkout scm
      }
      stage('Build') {
        container('node') {
          sh """
            npm install
          """
        }
      }
      stage('Test') {
        container('node') {
          withEnv(["CI=true"]) {
            sh "npm test"
          }
        }
      }
```

```
      //if (branch == 'dev' || branch == 'master') {
      // /.*rfw-tests\w*/
      // /dev$|master|.*(rfw-tests|v[0-9.]*-rc[0-9]*)/
      if (branch ==~ /dev$|master|.*(rfw-tests|v[0-9.]*-rc[0-9]*)/) {
        stage('RFW Build') {
          container('python') {
            sh """
              chmod +x test/rfw-tests/hlp/install.sh
              dos2unix test/rfw-tests/hlp/install.sh
              sh test/rfw-tests/hlp/install.sh
            """
          }
        }
        stage('RFW Test') {
          container('python') {
            sh """
              sh test/rfw-tests/hlp/entry_point.sh
            """
          }
        }
        stage('RFW Report') {
          publishHTML(target: [
            allowMissing: false,
            alwaysLinkToLastBuild: true,
            keepAll: true,
            reportDir: 'test/rfw-tests/reports/',
            reportFiles: '*.html',
            reportName: 'RobotResults',
            reportTitles: ''
          ])
          publishHTML(target: [
            allowMissing: true,
            alwaysLinkToLastBuild: true,
            keepAll: true,
            reportDir: 'test/rfw-tests/logs/',
            reportFiles: '*.log',
            reportName: 'ChromeLogs',
            reportTitles: ''
          ])
        }
      //}
      }
    }
    catch (e) {
      currentBuild.result = "FAILED"
      throw e
    }
  }
}
```

Liite 4. azure-pipelines.yml tiedosto

```yaml
trigger:
- master

pool:
  vmImage: 'ubuntu-latest'
strategy:
  matrix:
    Python37:
      python.version: '3.7'

steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '$(python.version)'
  displayName: 'Use Python $(python.version)'

- script: |
    python -m pip install --upgrade pip
    echo "Installing chromedriver udev xvfb nmap and requirements.txt"
    mkdir logs/
    apt-get update
    apt-get install -y tzdata udev xvfb nmap > logs/apk_ins.log
    npm install chromedriver
    sudo ln -s /home/vsts/work/node_modules/chromedriver/lib/chromedriver/chromedriver /usr/bin
    pip install --no-cache-dir -r hlp/requirements.txt > logs/pip_ins.log
    apt-get list --installed | sort > logs/apk.log
    pip freeze > logs/pip.log
  displayName: 'Install dependencies'

- script: |
    sudo Xvfb :99 -screen 0 $(MONITOR) -nolisten tcp &
    pabot --processes $(ROBOT_THREADS) --outputDir ./reports/ $(ROBOT_OPTIONS) ./tests/tests*.robot
  displayName: 'pytest'

- task: PublishPipelineArtifact@1
  inputs:
    path: $(System.DefaultWorkingDirectory)
    artifact: report-$(Build.BuildID)
```