Testausautomaation kehittäminen ohjelmistokehitysyrityksessä

Sami Ropponen

2019 Laurea

Laurea-ammattikorkeakoulu

Testausautomaation kehittäminen ohjelmistokehitysyrityksessä

Sami Ropponen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2019

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tietojenkäsittely (AMK)

Sami Ropponen

Testausautomaation kehittäminen ohjelmistokehitysyrityksessä

Vuosi 2019 Sivumäärä 28

Tämän opinnäytetyön tavoitteena on tutkia ja testata toimeksiantajan käyttämiä automaatiotestausmenetelmiä. Opinnäytetyö suoritettiin toimeksiantona suomalaiselle taloushallinnon ohjelmistotalolle, Accountor Finago Oy:lle. Opinnäytetyöprosessin aikana toimeksiantaja laajentaa ohjelmistokehitysympäristönsä toimimaan Kubernetes-ohjelmistolla luodussa monipalvelinympäristössä.

Opinnäytetyön toiminnallisessa osuudessa päivitetään testiympäristö toimimaan Zalenium-, ja Kubernetes-työkaluilla toteutetussa infrastruktuurissa ja tutkitaan, kuinka sitä voidaan hyödyntää automaatiotestauksessa. Työn aikana perehdyttiin käytettävien työkalujen sähköisiin materiaaleihin sekä testaukseen ja automaatiotestaukseen liittyvään kirjallisuuteen.

Työn kehittämistehtävänä oli pystyttää toimeksiantajan testiympäristöön lokaali Kubernetes-klusteri, jolla voidaan ajaa automatisoituja käyttöliittymätestejä. Kehittämistehtävän testitulokset olivat onnistuneita, ja yritys sai opinnäytetyöstä hyödyllisiä tutkimustuloksia.

Asiasanat: Automaatiotestaus, testaus, Selenium, Zalenium, Kubernetes

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Sami Ropponen

Test automation development in software development company

Year 2019 Pages 28

The purpose of this thesis was to study test automation methods used by the company. Thesis was done as commission for Finnish accounting software house Accountor Finago Oy. During the thesis commissioner expands its software development environment to Kubernetes clustered environment.

The functional part of this thesis was to expand local test environment to environment with capability of executing Zalenium based automated user interface tests. In the research part of this thesis literature of testing and automation testing was studied and materials of used testing tools were made familiar.

The development task of this thesis was to expand test environment to Kubernetes cluster that can run Zalenium based automation tests. Development task produced successful testing results and good researching results.

Keywords: Automation testing, Testing, Selenium, Zalenium, Kubernetes

Sisällys

1 Johdanto

Tämän opinnäytetyön tavoitteena on tutkia toimeksiantajayrityksen automaatiotestausmenetelmiä ja miten niitä hyödynnetään, kun toimeksiantajana toimiva Procountor ottaa käyttöönsä Kubernetes-ohjelmiston. Kubernetes laajentaa toimeksiantajan tietokantavirtualisointia.

Kehitystehtävän tavoitteena on testata Selenium-ohjelmiston integroimista Kubernetes-monipalvelinympäristöön. Selenium on sovellus web-selaimen automatisointia varten. Toimeksiantaja käyttää Seleniumia automatisoitujen käyttöliittymätestien ajamiseen.

Zalenium on Zalando Techin ja Seleniumin yhteistyönä valmistama avoimen lähdekoodin laajennus Selenium-ohjelmistoon. Zaleniumin avulla yritys muokkaa Seleniumin yhteyksiä, niin kutsuttua Selenium-verkkoa, toimimaan Kubernetes-ohjelmistolla muodostetussa Kubernetes klusterissa. Opinnäytetyöprosessin aikana luodaan lokaali Kubernetes klusteri, jolla voidaan ajaa käyttöliittymätestejä Zaleniumia hyödyntäen.

Jotta tutkimus automaatiotestausmenetelmistä olisi mahdollisimman hedelmällinen, perehdyttiin opinnäytetyön aikana ohjelmistotestaukseen ja sen hyötyihin. Lisäksi opinnäytetyössä tutustutaan, mitä on käyttäytymislähtöinen ohjelmistokehittäminen. Opinnäytetyön tietoperustana käytettiin testauksen painettua kirjallisuutta sekä sähköisiä lähteitä. Painettuina lähteinä käytettiin muun muassa Jussi Pekka Kasurisen (2013) *Ohjelmistotestauksen käsikirjaa* sekä Dorothy Grahamin ja Mark Fewsterin (2012) automaatiotestaukseen perustuvista kirjoista kerättyä opasta *Experiences of Test Automation: Case Studies of Software Test Automation*. Lisäksi tutustuttiin Dan Northin kirjoittamiin materiaaleihin JBehavesta ja käyttäytymislähtöisestä kehityksestä.

Tämän opinnäytetyön tarkoituksena ei ollut tuottaa absoluuttista ratkaisua, joka olisi täydellinen tapa hyödyntää automaatiotestausta. Kuitenkin havaintoja oikeista ratkaisuista voidaan tehdä. Opinnäytetyön tavoitteena on tutkia Procountorin käyttämiä automaatiotestausmenetelmiä ja identifioida ratkaisuja, jotka ovat Procountorin tapauksessa olleet hyviä ratkaisuja. Vertailukohteina toimivat aikaisemmalla toteutuksella toimiva Selenium-verkko sekä opinnäytetyössä esiintyvällä Zalenium-toteutuksella toimiva Selenium-verkko Kubernetes klusterissa.

1.1 Keskeiset käsitteet

Bugi	Ohjelmiston koodista löytyvä ohjelmointivirhe.
Chocolatey	Pakettienhallintasovellus Windowsille.

Docker	Sovellus, jonka avulla ohjelmisto voidaan pakata virtuaaliseen pakettiin eli konttiin.
Hyper-V	Windows 10 –käyttöjärjestelmän sisäänrakennettu virtualisointityökalu.
Integraatiotesti	Testi, jolla testataan ohjelmiston useamman toiminnon toimivuutta yhdessä.
JBehave	Java-pohjainen viitekehys automaatiotesteille.
Jenkins	Java-pohjainen automaatiopalvelin automatisoitujen ajojen (job) hallintaan.
Klusteri	Useista tietokoneista koostuva verkosto, jossa yksi kone toimii palvelimena, joka jakaa tehtäviä muille koneille. Klusterin tehtäviä suorittavia koneita kutsutaan Nodeiksi.
Kontti	Virtualisoitu paketti ohjelmistosta.
Kubernetes	Konttipohjainen monipalvelinympäristö.
Käyttäytymislähtöinen kehitys	Ohjelmistokehitysmenetelmä, jossa ohjelmiston käyttäytyminen dokumentoidaan käyttäjän näkökulmasta.
Minikube	Työkalu Kuberneteksen testaamista varten. Minikuben avulla Kubernetes voidaan asentaa lokaalille ympäristölle.
Node	Tietokone, joka suorittaa sille annettuja tehtäviä. Node voi olla fyysinen tai virtuaalinen tietokone. Tässä opinnäytetyössä Nodeilla tarkoitetaan virtuaalisia koneita.
Regressiotestaus	Regressiotestaus tarkoittaa ohjelmiston jo aiemmin testattujen toimintojen uudelleen testaamista.
Selenium	Sovellus web-selaimen automatisointia varten.

Selenium-hub	Selenium-verkon osa, joka ohjaa automaatiotestejä Nodeille.
Selenium-verkko	Seleniumin muodostama yhteys suoritettavan testin ja web-selaimen välillä.
Testivetoinen kehitys	Ohjelmistokehitysmenetelmä, jossa ensin luodaan yksikkötesti, jonka jälkeen luodaan koodi testin läpäisemiseksi.
Yksikkötesti	Testaa ohjelmiston yhden toiminnon toimivuutta.
Zalenium	Zalando Techin kehittämä uusi Selenium-verkko.

2 Työn lähtökohdat

Tämän opinnäytetyön aihe nousee toimeksiantajayrityksen, Procountorin, käytännöntarpeesta. Procountor ilmoitti syksyllä 2018 uusista käyttöönotettavista työkaluista, joita testattaisiin tulevien kuukausien aikana. Osoitin Procountorin testauspäällikölle kiinnostusta osallistua uusien työkalujen testaamiseen opinnäytetyön muodossa. Aloitin suunnittelemaan opinnäytetyön toteutusta ja perehtymään käyttöönotettaviin työkaluihin opinnäytetyön aiheen hyväksynnän jälkeen marraskuussa 2018. Aihe oli kiinnostava, sillä uudet työkalut eivät olleet minulle ennestään tuttuja.

2.1 Toimeksiantaja

Accountor Finago Oy on suomalainen taloushallinnon ohjelmistotalo, joka tarjoaa sähköisen taloushallinnon ratkaisuja niin yrityksille kuin tilitoimistoille. Accountor Finago Oy tarjoaa asiakkailleen Procountor ja Tikon -ohjelmistoja, joita käyttävät tällä hetkellä 1400 tilitoimistoa ja niillä hoidetaan 120 000 yrityksen kirjanpitoa Suomessa, Ruotsissa, Norjassa ja Tanskassa. Finago on osa suomalaista Accountor-konsernia, joka toimii seitsemässä maassa ympäri Eurooppaa. (Accountor Finago Oy 2017.)

Työn toimeksiantajana toimii Procountor-ohjelmiston kehitystiimi, joka muodostuu yli kymmenestä Scrum-tiimistä. Scrum on ketterän ohjelmistokehityksen viitekehys, joka koostuu Scrum-tiimeistä. Jokaisella tiimillä on omat roolit, säännöt, tapahtumat ja tuotokset (Schwaber & Sutherland 2017, 4). Procountorin Scrum-tiimit pitävät sisällään kehittäjiä, Scrum-mastereita, tuoteomistajia, suunnittelijoita sekä testaajia.

Kuvio 1. *Scrum-tiimi.* (Mendix 2017.)

Kuvio yksi sisältää havainnollistavan kuvan standardista Scrum-tiimistä, joka koostuu Scrum-masterista, tuoteomistajasta ja kehitystiimistä. Suurin osa testaajista kuuluu johonkin Scrum-tiimiin ja pitää huolta tiimin alueen tuotekehityksen laadunvarmistuksesta. Tämän lisäksi Procountorilla on yksi Scrum-tiimi, joka koostuu pelkästään testaajista. Tämä tiimi on vastuussa Procountorin regressiotestauksesta, testauksen suunnittelusta sekä automaatiotestauksesta.

Procountor on alati kehittyvä taloushallinnon ohjelmistotalo, joka on kasvanut vuosi vuodelta enemmän. Mitä suuremmaksi ohjelmisto ja sen ekosysteemi kasvavat, sitä enemmän resursseja tarvitaan. Viimeisen kolmen vuoden aikana Procountor on nostanut panostaan automaatiotestauksessa, minkä seurauksena infrastruktuurin ylläpitoon tarvitaan todennäköisesti lisää resursseja tulevaisuudessa.

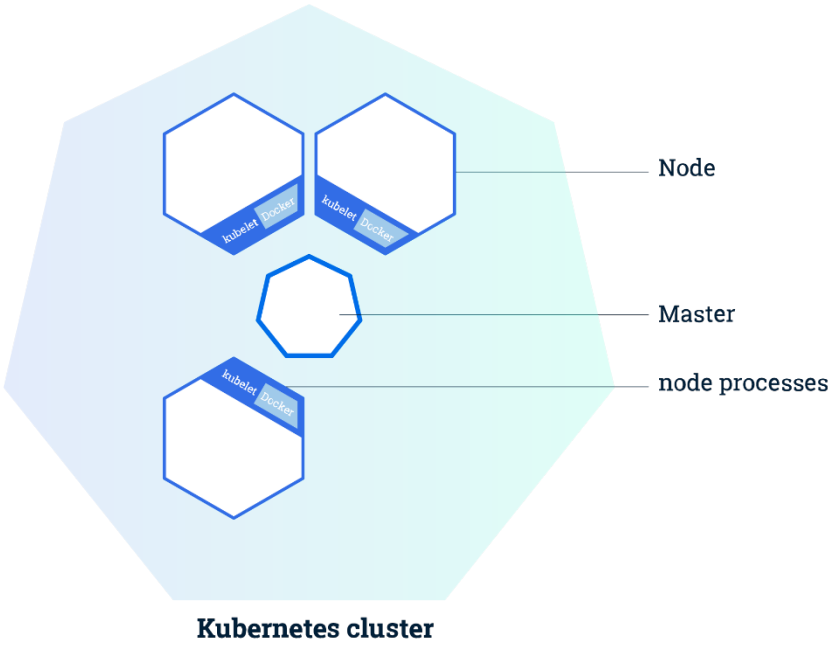
2.2 Procountorin automaatiotestauksen tila

Procountor alkoi panostaa testauksen automatisointiin vuosina 2017 ja 2018, jolloin testien määrä moninkertaistui. Tämän opinnäytetyöprosessin aikana Procountorilla oli yli tuhat skenaariota, yli kolmessasadassa käyttöliittymätestissä. Päivittäin ajettavissa testiajoissa ajettavien käyttöliittymätestien määrät kasvoivat niin suuriksi, että käyttöliittymätestien ajamiseen käytetty web-selainten automatisointityökalu, Selenium, alkoi ylikuormittua. Seleniumin epävakaa Selenium-verkko eli tietoverkko, jota Selenium käyttää testien suorittamiseen, alkoi tuottaa ongelmia. Jos yhteys ajettavan käyttöliittymätestin ja virtualisoidun verkkoselaimen välillä katkeaa, testi epäonnistuu. Epäonnistunut testi tuottaa ongelmia koko testiajolle. Testiajossa, jossa ajettiin kaikki Procountorin käyttöliittymätestit, oli todennäköisempää saada

epäonnistunut ajo kuin onnistunut. Procountor alkoi kyseenalaistaa automaatiotestauksessa käyttämiään menetelmiä.

2.3 Työn toimeksianto

Opinnäytetyön aikana Procountor mukauttaa ohjelmistonsa ekosysteemiä toimimaan Kubernetes-klusterissa. Kubernetes-klusteri on monipalvelinympäristö, jossa on määriteltynä master-kone, joka ohjaa tehtäviä niin sanotuille worker-koneille. Worker-koneita kutsutaan tässä opinnäytetyössä Nodeiksi. Kuberneteksen lisäksi Procountor ottaa käyttöönsä Zaleniumin, joka on Selenium-automaatiotestaustyökalun laajennus. Zaleniumin avulla muokataan Selenium-verkko toimimaan ohjelmiston oman ekosysteemin sisällä. Zalenium- sekä Kubernetes-ympäristöjen ohjelmointivaiheet on rajattu pois opinnäytetyöstä.



Kuvio 2. *Klusteri.* (Kubernetes 2018.)

Kuviossa kaksi on esiteltynä Kubernetes-klusteri, jonka keskipisteenä on Master-kone. Se antaa tehtäviä ympärillä toimiville Nodeille.

Opinnäytetyö on rajattu testauksen näkökulmaan, ja miten tämän kaltainen ekosysteemin muutos vaikuttaa automaatiotestaukseen. Opinnäytetyö ei ole ohjelmointiprojekti, joten työssä ei tulla näkemään ohjelmointiratkaisuja. Procountorin kehitysympäristöön on toteutettu ohjelmistohaara, johon Zalenium on implementoitu valmiiksi.

3 Testaus

Testauksen tarkoituksena on varmistaa, että tuotetaan laadukasta tuotetta. Tietojenkäsittelyn ja tietotekniikan koulutuksia tarjoavan yrityksen Tieturin (2017) mukaan ”Testauksessa yhdistyy ohjelmistokehityksen osaaminen, liiketoiminnan tuntemus, systeemiajattelu ja kommunikointitaito.” Jokainen edellä mainituista tuo lisäarvoa tuotteelle. Testausta löytyy monenlaista, ja testauksesta löytyy monenlaisia eri osa-alueita, kuten esimerkiksi käytettävyystestaus, turvallisuustestaus tai suorituskykytestaus. Kullakin testausmetodilla on tietty tarkoituksensa, ja jokainen niistä tuo varmuutta testauksen laadulle, joka taas tuo varmuutta kehitettävälle tuotteelle. (ISTQB 2011.)

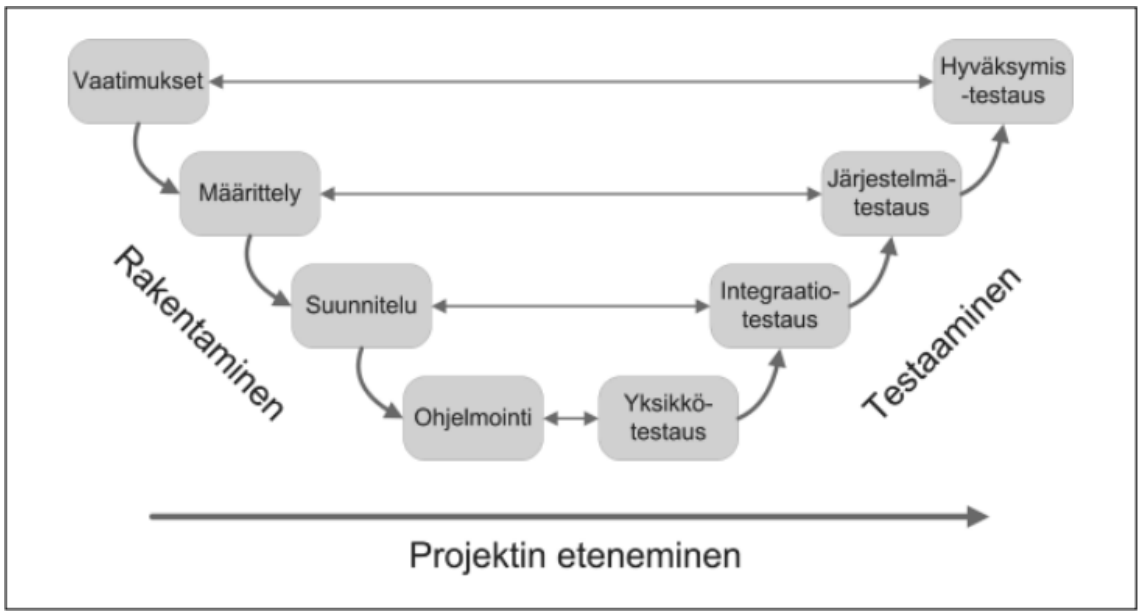
Testaus on ohjelmistokehityksen osa-alue, jonka tarkoituksena on varmistaa, että kehitettävä ohjelmisto toimii halutulla tavalla. Testauksen avulla voidaan verifioida, että ohjelmiston kaikki osa-alueet toimivat yhdessä. Mitä laajemmaksi ohjelmistoprojekti muuttuu, sitä tärkeämmäksi osaksi testaustyö kasvaa. (Kasurinen 2013, 10.)

Testaaja nimikkeenä antaa hyvin suppean kuvan sen roolista. Aivan kuten ohjelmiston koodanneita henkilöitä kutsutaan usein nimellä kehittäjä. Kehittäjien työtehtäviin kuuluu paljon muutakin kuin vain koodin kirjoittaminen. Samankaltaisesti testaajien työhön voi usein sisältyä koodin kirjoittamista, dokumentointia, haastatteluita tai esimerkiksi niin kutsuttujen testaus työpajojen järjestämistä koeryhmille. (Kasurinen 2013, 10.)

Kasurisen (2013, 16) mukaan ”Testaustyö on sinänsä hyvin epäkiitollista, koska testauksen onnistuessa kukaan ei näe hyvin tehtyä työtä, mutta puutteellinen tai huolimattomasti tehty testaustyö loistaa ohjelmistosta kuin majakka pimeydessä.” Tämä merkitsee sitä, ettei onnistunut ohjelmisto indikoi onnistuneen testaustyön tuloksia kehitystiimin ulkopuolisille tahoille. Kehitystiimi voi kuitenkin, riippuen käytetyistä menetelmistä, mitata testaustyötä esimerkiksi vertaamalla ennen julkaisua löytyneiden bugien määrää julkaisun jälkeen löytyneiden bugien määrään. Tämä testauksen mittaamisen menetelmä antaa kuitenkin hyvin mustavalkoisen käsityksen käytetyistä resursseista sekä työmäärästä ja niiden hyödyistä, sillä se ei kerro projektin vaiheiden laajuutta tai monimutkaisuutta.

3.1 Ohjelmistotestaus

Testaus on osa ohjelmistokehitysprosessia prosessin alusta loppuun. Ohjelmistokehitysprosessissa ohjelmiston tekninen testaus on usein jaettu V-mallin mukaiseen kolmeen osaan: yksikkötestaukseen, integraatiotestaukseen sekä järjestelmätestaukseen (kuvio 3). Näitä kutsutaan testaustasoiksi. (Kasurinen 2013, 51.)



Kuvio 3. *Testauksen V-malli.* (Kasurinen 2013, 51.)

Yksikkötestaus on testauksen osa, joka suoritetaan ohjelmoinnin yhteydessä tai heti sen jälkeen. Yleensä yksikkötestauksen suorittaa itse kehittäjä. Yksikkötestauksessa testataan ohjelmiston yhden funktion tai toiminnallisuuden toimivuutta. Sen tarkoituksena on tarkistaa, että ohjelmoitu koodi toteutuu oikein. Yksikkötesti voidaan toteuttaa rakentamalla funktiokutsuja, joihin ohjelmoidun komponentin pitää osata vastata oikein. Funktiokutsu voi esimerkiksi pyytää komponentilta jotain syötettä tai toimintoa. Jos komponentti antaa väärän syötteen tai toimii väärin, tiedetään heti, että ohjelmoidussa komponentissa on virhe. Yksikkötestauksessa suurin ongelma on, ettei sen avulla saada vastausta siihen, kuinka ohjelmoitu komponentti kommunikoi muiden komponenttien kanssa. (Kasurinen 2013, 51—52.)

Integraatiotestaus on testauksen osa, joka suoritetaan yksikkötestauksen jälkeen. Se keskittyy funktion tai komponentin toimintaan ja sen kanssakäymiseen muiden komponenttien kanssa. Integraatiotestauksella on tarkoitus varmistaa, että ohjelmiston komponentit ovat yhteensopivia toistensa kanssa. Integraatiotestit ovat laajempia kuin yksikkötestit, mutta ne eivät vielä kuvaa kokonaisen järjestelmän toimintaa. (Kasurinen 2013, 54.) Integraatiotestauksen tarkoituksena on tarkastaa, että ohjelmisto toimii suunnitellulla tavalla.

Järjestelmätestaus on testauksen osa, joka kattaa koko järjestelmän. Järjestelmätestaukseen voidaan ottaa mukaan sellaiset komponentit, jotka ovat läpäisseet sekä yksikkötestit että integraatiotestit. Komponentit tulee olla yksikkötestattu eli varmistettu, että ne antavat oikeanlaisia syötteitä. Yksikkötestauksen jälkeen komponentit tulee olla integraatiotestattu eli varmistettu, että ne toimivat oikeanlaisesti muiden niiden ympärillä toimivien komponenttien kanssa. Järjestelmätestaus suoritetaan yleensä testiympäristössä, jolloin saadaan kokonaiskuva järjestelmän toimivuudesta, mutta siihen voidaan vielä tehdä muutoksia. (Kasurinen 2013, 56.) Järjestelmätestauksen tarkoitus on tarkastaa, että ohjelmisto täyttää sille annetut määritykset.

Testaustasojen jälkeen suoritetaan vielä hyväksymistestaus. Hyväksymistestaus on ohjelmistokehitysprosessin testaustyön viimeinen osio. Hyväksymistestausta kutsutaan ohjelmiston tarkastamiseksi, sillä siinä tarkastetaan täyttääkö järjestelmä sille annetut vaatimusmäärittelmät, joilla se voidaan todeta valmiiksi. Hyväksymistestaus suoritetaan yleensä kohdeympäristössä eli sillä ei ole testiympäristön rajoitteita. (Kasurinen 2013, 57).

3.2 Automaatiotestaus

Tänä päivänä yhä useampi yritys on alkanut hyödyntää automaatiotestausta. Vaikka automaatiotestaus on ollut olemassa jo vuosikymmenet, on se kasvattanut suosiotaan vasta 2000-luvulla. Automaatiotestauksen käytön levinneisyyttä on hidastanut epätietoisuus sen hyödyistä. Suurimmalle osalle projekteista esteenä ovat yleensä rajalliset resurssit. Epätietoisuus tekniikoista ja hyötyjen suhteesta resursseihin on kriittinen tekijä projektin suunnittelussa. Vuosien saatossa erilaisten automaatiotestaustekniikoiden määrä on kuitenkin hiljalleen kasvanut. Jokainen ohjelmistoprojekti haluaa kehittää omaa tekemistään ja löytää uusia, parempia ratkaisuja menestyä markkinoilla. Yhä useampien ohjelmistoprojektien positiiviset tulokset automaatiotestauksen hyödyntämisestä ovat kasvattaneet automaatiotestauksen suosiota. (Graham & Fewster 2012, 31.)

Automaatiotestaus oikein toteutettuna on oivallinen tapa helpottaa yrityksen ohjelmistokehitystä. Etenkin ketterissä ohjelmistokehitysprojekteissa automaatiotestausta hyödynnetään jatkuvasti. Hyvin suunnitellulla automaatiotestauksella voidaan kattaa laajojakin alueita ohjelmiston testauksesta ja näin ollen vähentää yrityksen kuormaa manuaalitestaukselta. Automaatiotestauksen salaisuus on nopea palaute laajojen ekosysteemien regressiotestauksessa. Automaatiotestauksella voidaan verifioida hyvinkin laajoissa ohjelmistoprojekteissa, jotka koostuvat useista eri tiimeistä, että ohjelmaan tehdyt muutokset toimivat yhdessä.

Täydellinen testaaminen ei yksinkertaisesti ole mahdollista. Täydellisellä testauksella tarkoitetaan testaustyötä, jossa kaikki mahdolliset tapaukset kaikilla mahdollisilla muuttujilla otetaan huomioon. (Kasurinen 2013, 17.) Testauksen automatisointi on kuitenkin merkittävä

harppaus kohti sitä. Automaatiotestauksella testattavien käyttötapausten määrää voidaan kasvattaa merkittävästi.

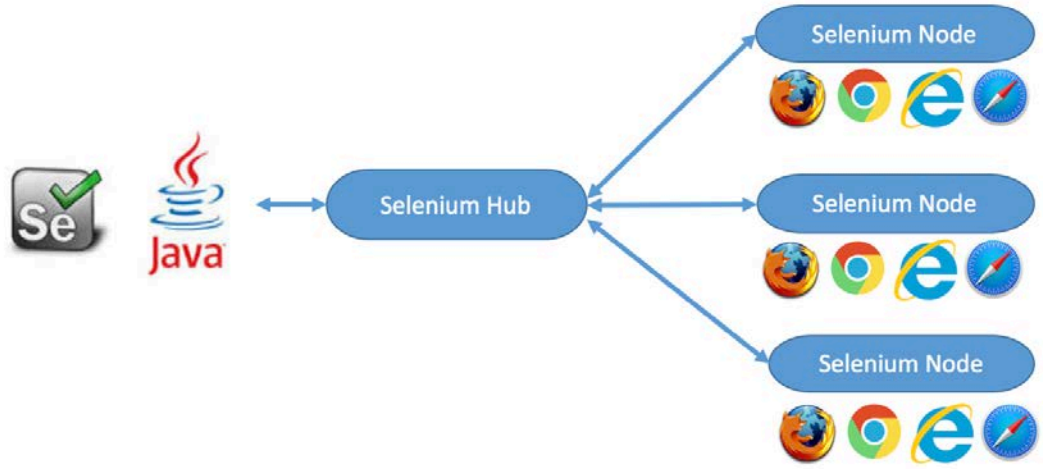
4 Toimintaympäristö

Toimintaympäristö-kappaleessa käsitellään opinnäytetyöprosessin aikana käytettyjä sovelluksia ja työkaluja. Sovelluksia käytetään testauksen ja testausautomaation edistämiseksi.

4.1 Selenium

Selenium on työkalu web-selainten automatisointia varten. Sen yleisin käyttötarkoitus on web-ohjelmistojen automaatiotestaus. Procountor käyttää Seleniumia muun muassa automatisoiduissa käyttöliittymätesteissä. Selenium lukee JBehave-viitekehyksellä kirjoitettuja skeenaarioita askel kerrallaan ja osoittaa askeleet virtualisoituun web-selaimeen. (Seleniumhq 2008.)

Kun käyttäjä käynnistää testin ajon, Selenium hakee testin Selenium-hubiin, jossa se lukee testille annetut parametrit, kuten virtualisoitavan käyttöjärjestelmän ja web-selaimen. Tämän jälkeen Selenium-hub ohjaa testit käynnistetylle Nodelle. Node on virtuaalikone, joka käynnistää web-selaimen annetuilla parametreillä. Tämän jälkeen Selenium suorittaa testin määritelmän mukaiset tehtävät. (Seleniumhq 2008.) Kuviossa viisi toimii havainnollistava kuva Selenium-verkosta, jossa Selenium-Hub ohjaa Java-ympäristöstä käynnistetyt automaatiotestit Nodeille.



Kuvio 4. *Selenium-verkko.* (Chubarov 2017.)

4.2 JBehave

Procountor käyttää automaatiotestauksessaan JBehave-viitekehystä, joka on Java-pohjainen viitekehys käyttäytymislähtöisille automaatiotesteille. JBehave hyödyntää käyttäytymislähtöisen kehityksen menetelmää, joka on kehittyneempi muoto testivetoisestä kehityksestä. Testivetoisessa kehityksessä kirjoitetaan ensin yksikkötesti, jonka jälkeen implementoidaan tarvittava koodi testin läpäisemiseksi (Koutifaris 2018). Testivetoisen kehityksen ongelmana on testien vaikea luettavuus sekä ymmärrys siitä, mitä testataan ja mitä ei testata (North 2006).

Käyttäytymislähtöisessä kehityksessä käyttöliittymätesti on dokumentoitu käyttötapaus ohjelmiston käytöstä. Testi sisältää käyttötapauksen eli skenaarion, joka taas sisältää askeleina dokumentoituja vaiheita ohjelmiston käytöstä. Askeleet ovat kokonaisia lauseita, jolloin testit ovat helppolukuisia ja niitä voidaan käyttää käyttötapausten dokumentoimiseen. (North 2006.) JBehave-automaatiotesti muodostuu neljästä elementistä, Story-, Step-, Page object-, sekä Story config -tiedostosta.

Story-tiedosto on tiedosto, joka sisältää automaatiotestin käyttötapauksia. Käyttötapaukset ovat määritelty skenaarioiksi. Yksi automaatiotesti voi sisältää monta skenaariota. Skenaariot ovat ennalta määritettyjä tapahtumaketjuja, joiden mukaan käyttäjä käyttää sovellusta. Skenaarion määrittämä tapahtumaketju on puolestaan pilkottu askeleiksi. Skenaarion askeleet ovat dokumentoitu kokonaisina lauseina tiedostoon.

```
Narrative:
As a Procountor user
I want to open the software manual
So that I can learn how to use Procountor

Scenario: Opening the manual using side bar navigation
Given a user with mainuser role is logged in
When the user clicks manual link in side bar help menu
Then the manual should open in a new tab
And Procountor should stay opened in another tab
```

Kuvio 5. *Openmanual.story*

Kuvion kuusi mukaisessa käyttäjätarinassa käyttäjä avaa ohjelmiston käyttöohjeet ohjelmistosta. Käyttäjätarina alkaa tarinan kuvauksella. Käyttäjätarinan kuvauksessa määritellään käyttäjä, mitä käyttäjä haluaa tehdä ja tarinan lopputulos. Kuvauksen jälkeen alkaa tarinan skenaario, jolla kuvataan käyttötapausta askel kerrallaan. Käyttäjätarinan skenaario sisältää Given-, When- ja Then-arvot. Given-arvolla määritellään alkuarvot skenaariolle, kuten millaisilla käyttöoikeuksilla käyttäjä kirjautuu sisään. When-arvolla määritellään käyttäjän tekemä toimenpide, kuten esimerkiksi napin painallus tai kentän täyttäminen. Then-arvolla määritellään toimenpiteestä seuraava ohjelmiston käyttäytyminen.

Step-tiedosto on tiedosto, jossa Story-tiedoston kokonaisina lauseina määritellyt askeleet ovat määritelty Java-metodeiksi. Java-metodi on se osa testiä, jossa testin itse tekeminen tapahtuu. Esimerkiksi painikkeen klikkaus voidaan määritellä Java-metodissa.

Page object -tiedosto on tiedosto, jossa määritellään web-sivu, johon testi osoitetaan. Web-sivu saattaa usein sisältää kymmeniä eri näppäimiä, tekstikenttiä, linkkejä sekä muita web-sivuille ominaisia elementtejä. Jokaiselle web-sivulle luodulle elementille on sitä luodessa annettu nimi tai ID. Page object -tiedostossa määritellään, millä Java-metodissa käytetyllä objektilla halutaan osoittaa minkäkin ID:n elementtiin.

Story config -tiedosto on tiedosto, johon on kerätty kaikki Step-tiedostot, joita testin ajamiseen tarvitaan. Story config -tiedostoa käytetään testin ajamiseen.

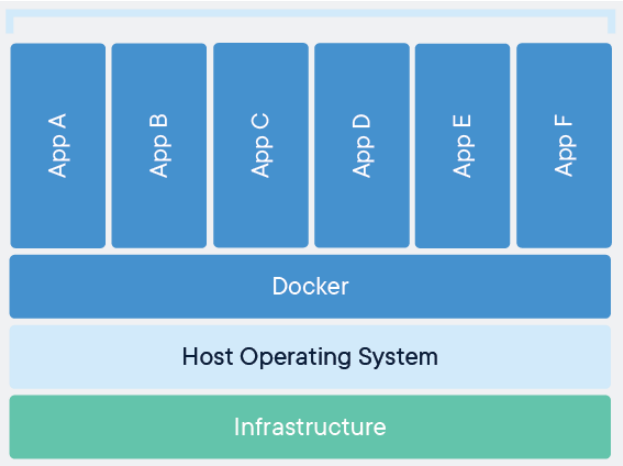
4.3 Jenkins

Jenkins on Java-pohjainen automaatiopalvelin, jota käytetään ajastettujen ajojen hallintaan. Jenkins-ajoja käytetään muun muassa koodimuutosten koontiversioiden sisäänajoissa sekä automaatiotestiajojen ajamisessa. Se tarjoaa käyttäjälleen palvelinten lisäksi hallintapaneelin sekä kaavioita ajamiesi ajojen tuloksista.

Procountor käyttää Jenkinsiä yli 200 000 ajon hallintaan päivittäin. Nämä ajot sisältävät muun muassa automatisoituja testiajoja, kuten yksikkötestejä, integraatiotestejä sekä käyttöliittymätestejä. Automatisoituja käyttöliittymätestejä varten Jenkins tarjoaa visuaaliset HTML-raportit ajojen tuloksista.

4.4 Docker ja Kubernetes

Teknologian kehittyessä virtualisoinnin käyttö on kasvanut viimeisten vuosikymmenten aikana. Erityisesti pilvipalveluiden yleistyminen on nostanut virtualisoinnin tärkeyttä ohjelmistojen ja sovellusten ekosysteemissä. (Vase 2016.) Ennen Dockeria yleisin virtualisointiin käytetty tekniikka oli virtualisointi virtuaalikoneilla. Virtuaalikone on virtuaalinen kuva fyysisestä koneesta, jolla voidaan suorittaa toimintoja aivan kuten fyysisellä koneella. Vuonna 2013 julkaistu avoimen lähdekoodin sovellus Docker mullisti virtualisoinnin tuomalla markkinoille konttiteknologian. Dockerin avulla ohjelmisto pakataan pakettiin eli konttiin, johon sisältyy vain tarvittavat resurssit ohjelmiston nopeaa käynnistämistä varten.

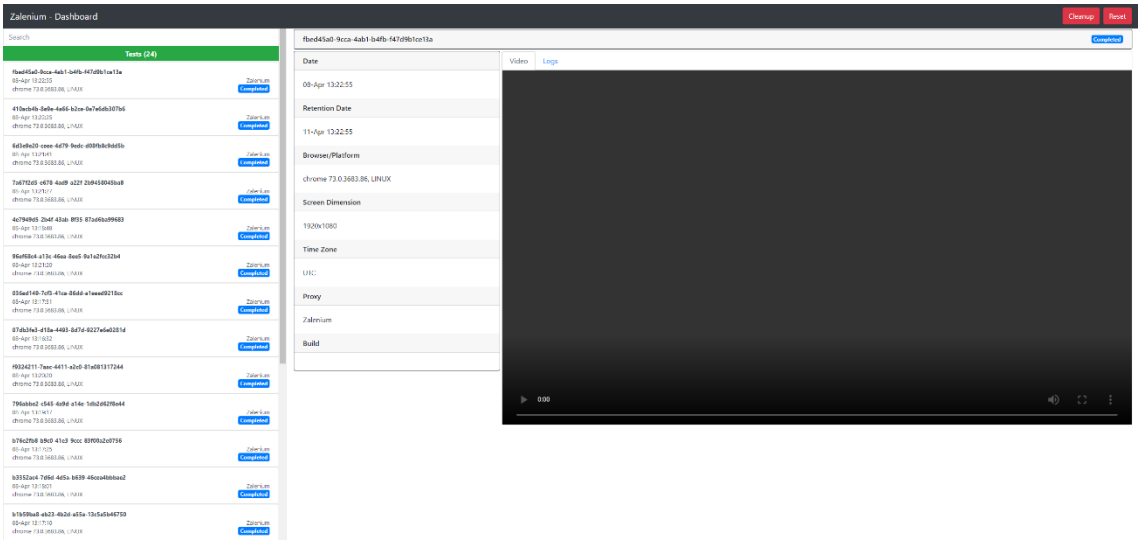


Kuvio 6. *Docker-kontti.* (Docker 2019.)

Kuvio seitsemän on havainnollistava kuva Docker-kontista, jossa kaikki tarvittavat ohjelmiston osat on pakattu yhteen. Kubernetes on ohjelmisto, joka laajentaa ohjelmiston Docker-konttiteknologiaa. Kuberneteksen avulla Docker-kontit sijoitetaan klusteroituun monipalvelinympäristöön.

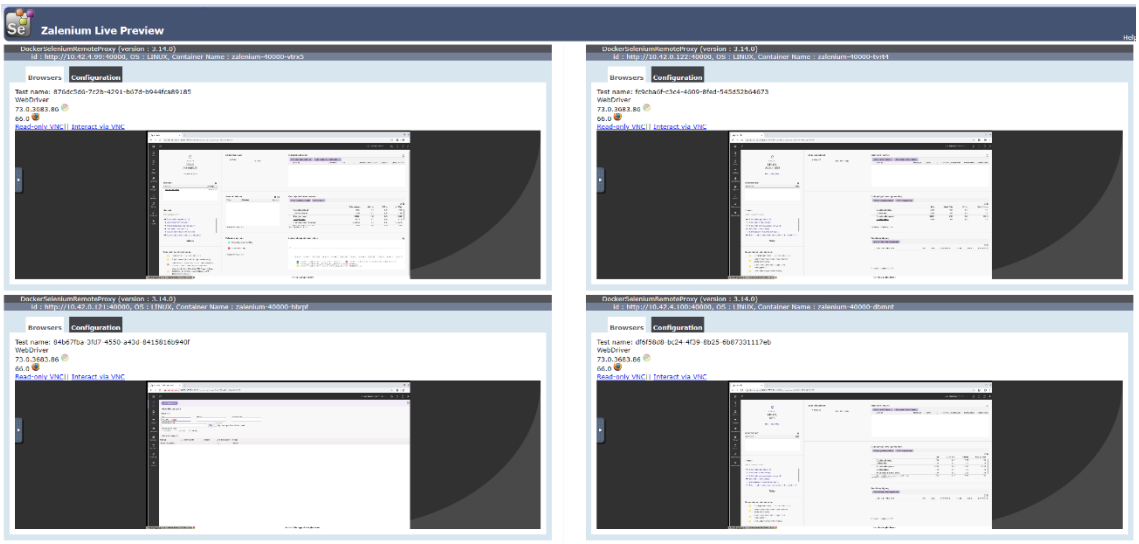
5 Uusi Selenium-verkko

Zalenium on yhdessä Seleniumin ja Zalando Techin kehittämä monipuolinen ja joustava Docker-kontteihin pohjautuva Selenium-verkko -infrastruktuuri. Zalenium tarjoaa käyttäjälleen Selenium-verkon, jonka avulla yhteys ajettavan testin sekä virtualisoidun verkkoselaimen välille voidaan luoda ilman kolmatta osapuolta. Lisäksi Zalenium tuo käyttäjälle työkaluja automaatiotestauksen helpottamiseksi, kuten videotallennuksen, live-seurannan, hallintapaneelin ja paljon muuta. (Zalando Tech 2018.)



Kuvio 7. *Zalenium-hallintapaneeli.*

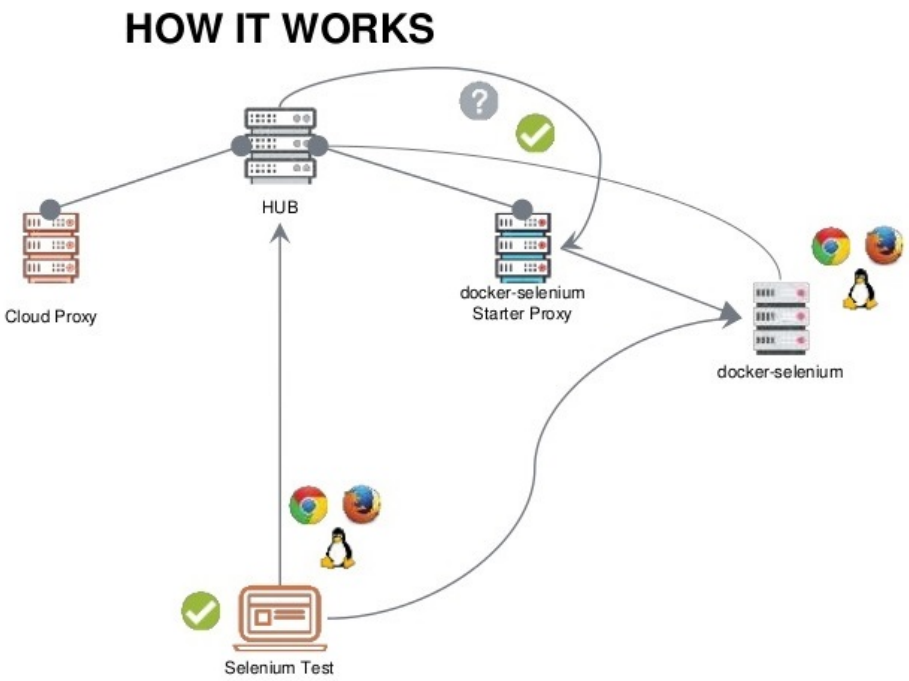
Kuviossa kahdeksan on esimerkkinä näytönkaappaus Zalenium-hallintapaneelista, jossa on käynnissä ajastettu testiajo. Vasemmassa laidassa näkyvät jonossa olevat testit, ja oikeassa laidassa yhden valitun testin videonäkymä. Zalenium hallintapaneelin avulla käyttäjä voi hallinnoida testiajoja. Hallintapaneelissa näkyvät ajettujen testien status, jonossa olevat testit sekä dataa testien ajoista. Lisäksi hallintapaneelissa on live-seurantaa varten oleva ikkuna, josta voi seurata valitun testin etenemistä videona.



Kuvio 8. *Zalenium Live Preview.*

Kuviossa yhdeksän on esimerkkinä näytönkaappaus Zalenium Live Preview -näkymästä. Zalenium tarjoaa hallintapaneelin lisäksi käyttäjälleen seurantapaneelin. Toisin kuin hallintapaneelista, seurantapaneelista voi seurata live-seurantaa kaikista samanaikaisista testeistä. Samanaikaisten testien määrä riippuu ennalta määritellyistä parametreistä. Parametrit määritellään käytettävissä olevien resurssien mukaan.

Selenium on jo itsessään todella monipuolinen ja hyödyllinen työkalu web-pohjaisille ohjelmistoille, mutta sen ongelmana on ollut Selenium-verkon epävakaus. Seleniumilla on ollut vaikeuksia mukautua hyvin erilaisiin ympäristöihin ja eri tavoin rakennettuihin tietoverkkoihin. Eniten ongelmia on luonut yhteyden muodostaminen ympäristön ulkopuolisiin tekijöihin, kuten pilvipalveluihin. (Zalando Tech 2018.) Yhteysongelmien ylipääsemiseksi Zalenium päätti lähestyä ongelmaa sieltä, missä itse selaimen virtualisointi tapahtuu eli Docker-konteista. Zalenium tarjoaa nopeammin ja vakaammin ajettavia käyttöliittymätestiajoja Chrome- ja Firefox-selaimilla, sillä ajot tapahtuvat omassa tietoverkossa, jossa virtualisoitu Node rakennetaan tyhjästä ajon aikana, ja hävitetään kun ajo on suoritettu. (Zalando Tech 2018.) Kuvio 10 havainnollistaa Selenium-verkon Zalenium toteutusta.



Kuvio 9. *Zalenium.* (Zalando Tech 2018.)

Minikube on työkalu Kuberneteksen testaamista varten. Minikuben avulla lokaalista ympäristöstä luodaan klusteri, jossa fyysinen tietokone toimii palvelintietokoneena. Virtuaalisia Nodeja voidaan luoda riippuen palvelin tietokoneen resursseista.

5.1 Lähtökohdat

Toteutin Minikuben lokaaliasennuksen fyysiselle Windows 10 - käyttöjärjestelmän tietokoneelle, joka toimii klusterin palvelintietokoneena. Klusterin arvoiksi annetaan yksi palvelintietokone sekä yksi virtuaalinen Node.

Minikuben järjestelmävaatimuksena oli Windows 10 Pro -käyttöjärjestelmä (Kubernetes 2019). Vaihtoehtoisesti asennukseen olisi voinut käyttää myös Windows 10 Enterprise tai Education –käyttöjärjestelmiä, sillä kaikki kolme käyttöjärjestelmää ovat Hyper-V tuettuja. Hyper-V on Microsoftin virtualisointityökalu, joka korvaa aikaisemmissa Windows käyttöjärjestelmissä tuetun Microsoft Virtual PC:n. (Microsoft 2018.)

5.2 Työkalut asennukseen

Minikuben Windows-asennukselle löytyy paljon eri tapoja, joita kukin yritys voi hyödyntää omien tarpeidensa mukaan. Osa työkaluista on käyttöjärjestelmäriippuvaisia, joten kaikkia työkaluja ei ole tarjolla jokaiselle käyttöjärjestelmälle. Opinnäytetyön Minikube-asennuksessa käytettyjä työkaluja olivat muun muassa IntelliJ IDEA, Chocolatey, Kubectl ja Helm.

IntelliJ IDEA on JetBrainsin kehittämä Java-integroitu kehitysympäristö (JetBrains 2009). Kehitysympäristöä voidaan käyttää lukuisien eri ohjelmointikielten ohjelmistojen ohjelmointiin. IntelliJ IDEA:a käytetään muun muassa Procountorin ohjelmoimiseen. Se mahdollistaa helpon tavan haarauttaa ohjelmistoa, jolloin ohjelmistoa voidaan koodata samanaikaisesti. IntelliJ IDEA on asennettuna opinnäytetyössä käytetyssä tietokoneessa valmiiksi, joten sen asennusta ei dokumentoida opinnäytetyössä.

Chocolatey on pakettienhallintasovellus Windowsille. Sen avulla voidaan asentaa ohjelmistopaketteja komentoriviltä. Ohjelmisto voidaan pakata pakettiin, jotta sen siirtäminen, lataaminen ja asentaminen olisi mahdollisimman kevyttä. Ohjelmistopaketti sisältää ainoastaan ne resurssit, joita tarvitaan ohjelmiston suorittamiseksi halutussa ympäristössä. Näin esimerkiksi erilaiset asennustyökalut voidaan jättää kokonaan pois viemästä tilaa. (Chocolatey 2011.) Opinnäytetyössä Chocolatey-työkalua käytetään Minikuben ja Kubectl:n resurssien lataamiseen lokaalille tietokoneelle.

Kubectl on komentorivityökalu, jota käytetään Kubernetes-klusterin hallintaan komentoriviltä. Sen avulla voidaan tarkastella klusterin resursseja ja sitä voidaan käyttää muun muassa luomaan, lisäämään tai poistamaan komponentteja klusteriin. (Kubernetes 2019.) Opinnäytetyössä Kubectl:ää käytetään Kubernetes-klusterin rakentamiseen ja hallinnoimiseen.

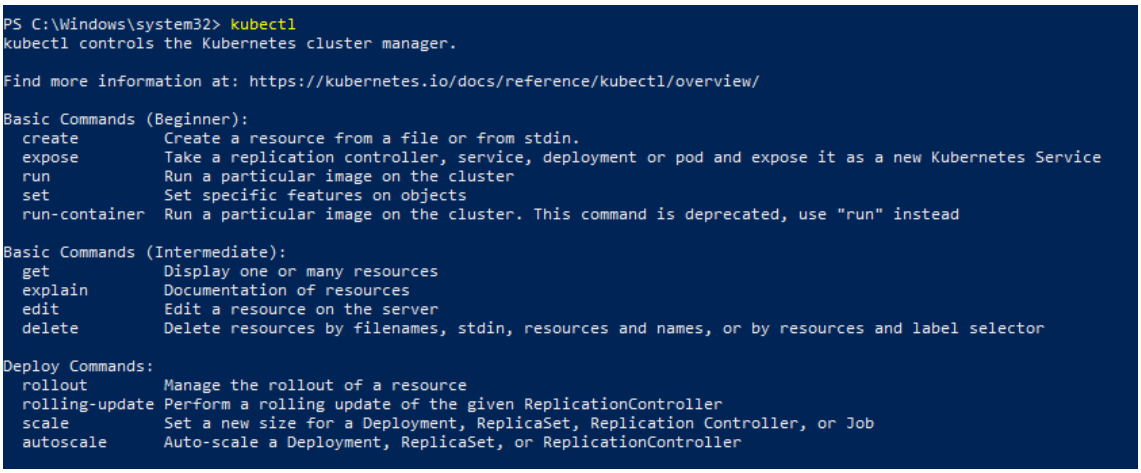
Helm on pakettienhallintasovellus Kubernetekseen. Sen avulla voidaan jakaa ja siirrellä halutun tyyppistä Kubernetes-klusteria. Helm hakee pilvihakemistosta organisaatioiden määrittämiä klustereita, jolloin saman klusterin replikoiminen useammalle lokaalille koneelle onnistuu vaivatta. Asennuksen aikana Helmiä käytetään lataamaan Helm-hakemistosta Procountorin määrittämä klusteri. (Helm 2018.) Opinnäytetyössä Helmiä käytetään Procountorin määrittämien Klusteri-resurssien hakemiseen pilvessä toimivasta Helm-hakemistosta.

5.3 Asennus

Kuberneteksen asennus voidaan aloittaa, kun asennuksen oletusvaatimukset on täytetty. Docker-sovellus tulee olla asennettuna koneelle Docker-kuvien lataamista varten. Procountorin Docker-kuvien lataamiseen Docker tarvitsee kirjautumistunnukset Procountorin yksityiseen Docker-hakemistoon. Kun Docker on käynnissä ja yhteys Procountorin Docker-hakemistoon on luotu, voidaan siirtyä asennuksessa Chocolatey -pakettienhallintasovelluksen asennukseen. Chocolatey tarjoaa verkkosivuillaan asennuskomennon, jolla Chocolatey-asennus suoritetaan komentoriviltä. Kun sovelluksen asennus suoritetaan komentoriviltä, säästytään turhien resurssien, kuten erillisten asennustyökalujen lataamiselta tietokoneelle.

Kun Chocolatey on asennettu tietokoneelle, sitä käytetään Kubectl- ja Helm-työkalujen lataamiseen ja asentamiseen. Ensiksi haetaan Kubectl-työkalu. Chocolatey-sovelluksella on oma

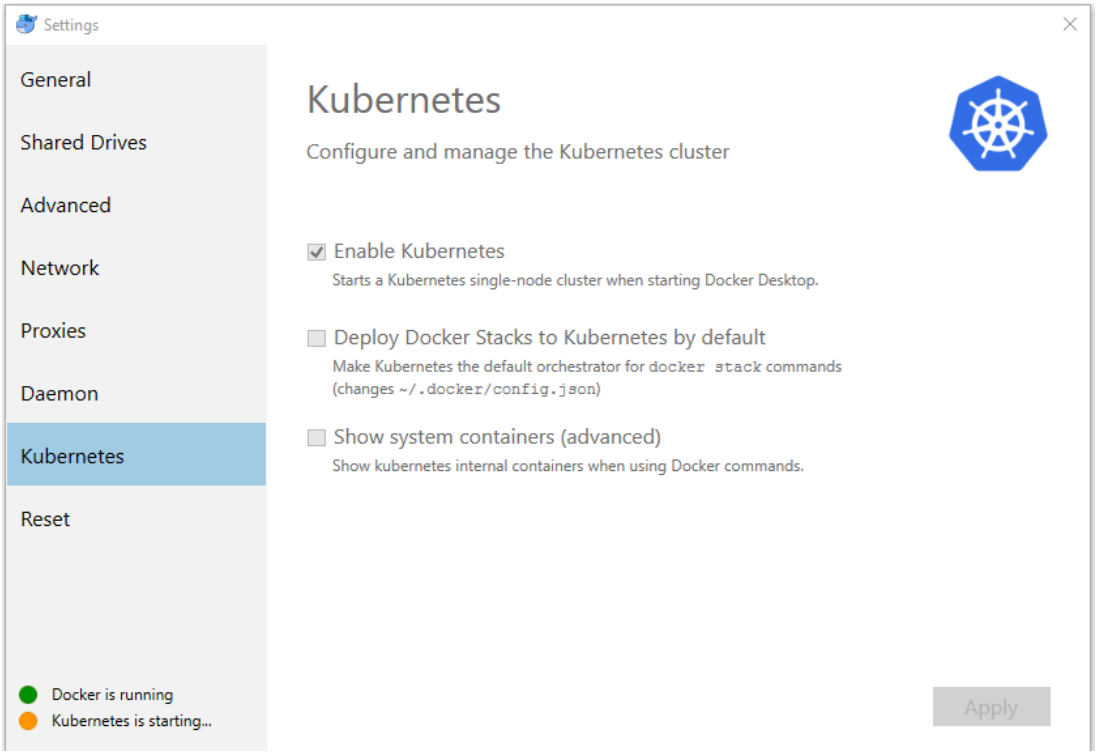
Chocolatey-hakemisto, joka toimii pilvessä. Chocolatey-hakemistoon voidaan upottaa kustomoituja paketteja sovelluksista, jotka sisältävät juuri tiettyyn tarkoitukseen sovitetut resurssit sovelluksesta. Syöttämällä komentoriville Kubectl-työkalun asennuskomennon, Chocolatey hakee Kubectl-paketin hakemistosta ja asentaa sen tietokoneelle. Asennuksen onnistuminen voidaan varmistaa syöttämällä komentoriville komento 'kubectl'. Jos Kubectl-asennus on onnistunut, komentorivi listaa käyttäjälle listan hyödyllisiä komentorivikomentoja, joita käyttäjä voi käyttää klusterin hallinnoimiseen (Kuvio 11).



Kuvio 10. *Kubectl komentorivillä.*

Helm-työkalu asennetaan samalla tavoin kuin Kubectl. Helm-työkalu asennetaan valmiiksi tietokoneelle, mutta sitä ei vielä käytetä mihinkään. Helm-työkalua käytetään Procountorin yksityisen klusterin replikoimiseen sen jälkeen, kun lokaali Kubernetes-klusteri on saatu rakennettua. Syöttämällä komentoriville Helm-työkalun asennuskomennon, Chocolatey hakee tarvittavat resurssit hakemistosta.

Kun Docker on käynnissä, sitä voidaan hallinnoida sen asetuksista. Dockerin asetuksista löytyy Kubernetes-välilehti, joka antaa käyttäjälle mahdollisuuden laajentaa Docker-ympäristön Kubernetes-klusteriksi. Kubernetes antaa eri vaihtoehtoja, kuinka klusterin rakentaminen halutaan toteuttaa. Tässä opinnäytetyössä luodaan tavallinen, yhden Noden klusteri, joten ainoastaan "aktivoi klusteri" -painike otetaan käyttöön (Kuvio 12).



Kuvio 11. *Kubernetes Dockerissa.*

Docker ilmoittaa, kun Kubernetes-klusterin rakentaminen on valmis. Onnistunut klusterin rakentaminen voidaan varmistaa Kubectl-työkalulla. Suorittamalla komentoriviltä komennon 'kubectl get namespaces' voidaan tarkastaa, että klusterin rakentaminen on onnistunut, ja klusterin resurssin ovat käytössä (Kuvio 13).

```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    21m
docker        Active    18m
kube-public   Active    21m
kube-system   Active    21m
```

Kuvio 12. *Aktiivinen klusteri komentorivillä.*

Kun klusterin asennus on onnistunut, se voidaan yhdistää Helm-työkaluun. Käyttämällä komentoriviä Helm-hakemistosta voidaan hakea yksityinen Procountorin klusteri, ja yhdistää se lokaaliin klusteriin. Kun lokaalille ympäristölle on onnistuttu luomaan Procountorin klusteria replikoiva lokaali klusteri, Procountorin ohjelmointihaara, johon Zalenium on implementoitu, voidaan ottaa käyttöön. Käyttämällä IntelliJ IDEA -kehitysympäristöä, voidaan Procountorista hakea sellainen ohjelmistohaaran, johon Zaleniumin implementointi on toteutettu. Käyttäytymislähtöisiä käyttöliittymätestejä voidaan nyt ajaa lokaalissa Kubernetes-ympäristössä käyttäen Zalenium toteutuksella toteutettua Selenium-verkkoa.

6 Tulokset

Varsinaisia benchmarking-tuloksia siitä, onko Zalenium kaikista ideaalisin automaatiotestaustyökalu Procountorin tarpeisiin, ei opinnäytetyöstä saatu. Opinnäytetyössä päästiin kuitenkin kehittämistehtävän tavoitteisiin. Kubernetes-klusteri pystyttäminen lokaalille testiympäristölle onnistui ja käyttöliittymätestejä voitiin ajaa Zalenium-toteutuksella. Positiivisia hyötyjä aikaisemman Selenium-verkon ja uuden Zalenium-toteutuksella toteutetun Selenium-verkon välillä havaittiin.

6.1 Tulokset testiympäristöstä

Testiympäristössä ajetuissa käyttöliittymätesteissä ei ollut suuria muutoksia huomattavissa, sillä testejä ajettiin yksittäisinä testeinä. Selenium on jo itsessään toimiva työkalu yksittäisten testien ajamiseen, niin lokaalissa ympäristössä kuin pilviympäristössä. Kuitenkin suurimmat muutoksen olivat huomattavissa testin käynnistämisen kestossa. Itse testin käynnistäminen oli aikaisempaa nopeampaa. Verkkoselain, jolla testi toteutetaan, käynnistetään lokaalin ympäristön sisällä. Näin ollen yhteyden muodostamiseen ei kulunut yhtä paljon aikaa kuin ennen.

Tulokset testille sopivan virtuaaliselaimen rakentamisen nopeutumisesta ovat iso positiivinen havainto projektin kannattavuudesta. Jo pelkästään yhden osa-alueen suorituskyvyn parantaminen voi luoda merkittäviä muutoksia testiajoihin, jotka sisältävät satoja testejä ja yli tuhat skenaariota.

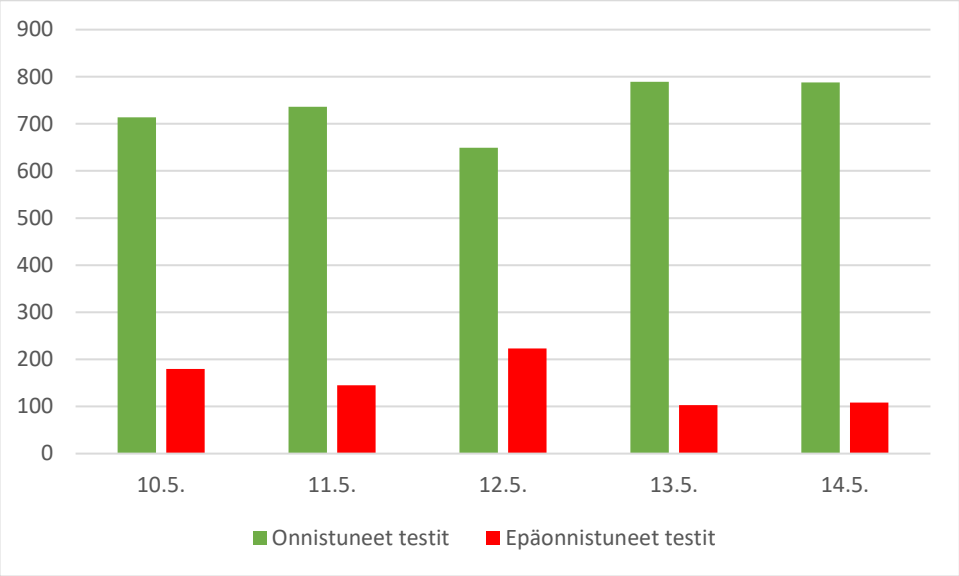
6.2 Tulokset Jenkinsistä

Testiajoja testattiin lokaalilla ympäristöllä sekä palvelimella suurin piirtein samoihin aikoihin. Kun yksittäisistä testeistä saatiin positiivisia tuloksia, varmistuttiin, että käyttöliittymätestit onnistuvat implementoidulla Zalenium-toteutuksella. Suurempien testiajojen testaaminen aloitettiin. Jo yksittäisissä testeissä havaittu testien nopean käynnistämisen vaikutus huomattiin testiajoissa odotettua suuremmin.

Vanhalla toteutuksella eli Selenium-verkolla ilman Zaleniumia, testiajot kestivät huomattavasti kauemmin. Vanhalla toteutuksella ajetut testiajot, joissa ajettiin kaikki Procountorin käyttöliittymätestit, kestivät väistämättä yli 12 tuntia. Keskiarvo vanhan toteutuksen kokonaisen testiajon pituudesta oli noin 16 tuntia. Testiajojen pituudet vaihtelivat usein ja tuottivat usein lukuisia epäonnistuneita käyttöliittymätestejä.

Testiajoja testatessa testejä ajettiin aluksi muutamien testien ajoissa. Ajoihin valittiin testit, joista voitiin olla varmoja, että ne onnistuvat uudessa ympäristössä. Parhaat vertailukohteet saatiin, kun ajettiin testiajo, joka sisältää noin puolet Procountorin käyttöliittymätesteistä. Puolikas testiajo sisälsi tarkalleen 477 skenaariota, ja se kesti kolme tuntia ja 20 minuuttia.

Testituloksien mukaan kokonainen testiajo kestäisi noin seitsemän tuntia. Tulos on yli puolet nopeampi kuin keskiverto kokonainen testiajo ilman Zalenium-toteutusta. Tulokseen vaikutti nopeampien testien määrän lisäksi epäonnistuneiden testien määrä. Testejä epäonnistui uudella toteutuksella huomattavasti vähemmän kuin vanhalla. Kuvio 13 sisältää taulukon päivittäisten testiajojen tuloksista 10.5. – 14.5. Mittausjaksolta saaduissa testiajot olivat vakaita ja satunnaisten epäonnistumisten määrä oli minimaalinen.



Kuvio 13. *Testiajot 10.5. – 14.5.*

7 Yhteenveto

Laajojen muutosten sovittaminen Procountorin kokoiseen ekosysteemiin on iso prosessi. Organisaation on sitouduttava kehitysajan sekä siirtymisajan mahdollisiin väliaikaisiin resurssien menetyksiin mahdollisesti pitkiksikin ajoiksi. Kuberneteksen ja Zaleniumin implementointi Procountorin ekosysteemiin vei useita kuukausia. Lisäksi osa-alueiden siirtäminen uuteen ekosysteemiin oli aikaa vievää.

Opinnäytetyön kehittämistehtävä saatiin päätökseen onnistuneesti. Lokaalin testiympäristön rakentaminen Kubernetes-toteutuksella onnistui sekä testiympäristössä pystyttiin suorittamaan Zalenium-toteutuksella toimivia käyttöliittymätestejä. Opinnäytetyö antoi tutkimustuloksia Zalenium- ja Kubernetes-työkaluista, joita Procountorilla ei aikaisemmin ollut.

8 Pohdinta

Testauksessa on tärkeää, että käytössä olevia resursseja käytetään oikein. Esimerkiksi automaatiotestauksessa on projektin kannalta hyvin tärkeää, että käytetään ohjelmistolle sekä

ympäristölle sopivia työkaluja. Keskittymällä oikeisiin osa-alueisiin ja käyttämällä oikeita työkaluja, saadaan testaustyön hyödyt mahdollisimman suureksi, ja hukkaan menevä työ minimoitua.

Mitä laajemmaksi ohjelmisto kasvaa, sitä tärkeämmäksi käytettävien resurssien hallinnointi tulee. Procountorin ongelmana automaatiotestien ajamisessa oli ajettavien testien epävakaa yhteyden muodostaminen virtualisoituun web-selaimeen. Procountorin kokoisen Web-ohjelmiston testauksessa on tehtävä valintoja, mihin kaikkeen automatisointia halutaan käyttää. Onko esimerkiksi mahdollisimman monen web-selaimen kattaminen tietylle osalle ohjelmistoa yhtä tärkeää kuin mahdollisimman suuren osan ohjelmiston funktionaalisista toiminnoista kattaminen yhdellä tai kahdella web-selaimella.

Procountorin kohdalla automaatiotestauksessa haluttiin keskittyä ohjelmiston sisältä tulevien bugien havaitsemiseen, eikä niinkään ulkopuolisten tekijöiden aiheuttamiin ongelmiin ohjelmiston käytössä. Procountor haluaa testata useampia selaimia, kun ohjelmistoon implementoidaan jokin uusi web-elementti, joka saattaa vaikuttaa ohjelmiston käytettävyyteen. Useampien selainten testaamisessa hyödynnetään workshop-testausta. Kun Procountoriin implementoidaan jokin uusi web-elementti ja halutaan verifioida, ettei se tuota ongelmia ohjelmiston käytössä, järjestetään testaus-workshop. Tämä merkitsee sitä, että useampi henkilö testaa samaa osa-aluetta, mutta eri selaimella. Useimmiten workshop koostuu testaajista, mutta ei ole erikoista, että workshopissa testaamassa olisi muitakin henkilöitä. On yleistä, että workshopissa saattaa olla mukana muun muassa kehittäjiä tai asiakaspalvelijoita.

Projekti näyttää tältä osin onnistuneelta. Uudet työkalut ja uusi ympäristö on integroitu onnistuneesti. Työkalujen toiminta näyttää tällä hetkellä lupaavalta, mutta uusista työkaluista saadaan kunnollista dataa vasta kun projekti on saatu päätökseen ja työkalujen käyttö normalisoituu. Esimerkiksi työkalujen ajan kanssa esiintyvää rasitusta ja muita vikatilanteita voidaan havainnoida vasta useamman kuukauden käytön jälkeen.

Lähteet

Painetut

Graham, D. & Fewster, M. 2012. Experiences of Test Automation: Case Studies of Software Test Automation. 1. painos. Indiana: Pearson Education

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. 1. painos. Jyväskylä: Docendo.

Schwaber, K. & Sutherland, J. 2011. The Scrum Guide™, The Definitive Guide to Scrum: The Rules of the Game.

Sähköiset

Accountor Finago Oy. 2017. Procountor + Tikon = Finago. Viitattu 10.2.2019. https://finago.com/fi/meista/ajankohtaista/finago/

Chocolatey. 2011. Viitattu 11.4.2019. https://chocolatey.org/

Chubarov, Maxim. 2017. Viitattu 7.3.2019. http://www.mchubarov.com/2017/02/running-selenium-grid-with-docker-in.html

Helm. 2018. Viitattu 11.4.2019. https://helm.sh/

ISTQB. 2011. Viitattu 14.4.2019. https://www.istqb.org/certification-path-root/foundation-level/foundation-level-in-a-nutshell.html

JetBrains. 2009. https://www.jetbrains.com/

Koutifaris, Andrea. 2018. Test Driven Development: what it is, and what it is not. Viitattu 7.3.2019. https://medium.freecodecamp.org/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2

Kubernetes. Documentation. Viitattu 11.3.2019 https://kubernetes.io/

Mendix. 2017. The Ideal Scrum Team Composition for Agile Development. Viitattu 16.5.2019. https://www.mendix.com/blog/the-road-to-adopting-scrum-team-composition/

Microsoft. 2018. Hyper-V on Windows 10. Viitattu 15.3.2019. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/

North, Dan. 2004. JBehave. Viitattu 7.3.2019. https://jbehave.org/

North, Dan. 2006. Introducing BDD. Viitattu 7.3.2019. https://dannorth.net/introducing-bdd/

SeleniumHQ. 2008. Introduction to Selenium. Viitattu 11.3.2019. https://www.seleniumhq.org/docs/01_introducing_selenium.jsp#

Tieturi. Viitattu 14.4.2019. https://www.tieturi.fi/koulutukset/testaus

Vase, Tuomas. 2016. Docker ja konttiteknologiat – ratkaisuja ja suorituskykyä. Viitattu 21.2.2019 https://eu.landisgyr.com/better-tech/docker-ja-konttiteknologiat-ratkaisuja-ja-suorituskyky%C3%A4

Zalando Tech. 2018. Zalenium. https://opensource.zalando.com/zalenium/

Kuviot