jamk.fi

Hyväksyntätestausautomaation hajauttaminen

Tapaus: Contriboard

Arttu Henell

Opinnäytetyö
Toukokuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Arttu Henell	Opinnäytetyö, AMK	23.5.2016
	Sivumäärä 56	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Hyväksyntätestausautomaation hajauttaminen
Tapaus: Contriboard

Tutkinto-ohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Hannu Luostarinen, Raija Hämäläinen

Toimeksiantaja(t)
Jyväskylän ammattikorkeakoulu / Marko Rintamäki

Tiivistelmä

Opinnäytetyön tavoitteena oli tutkia, millä tavoin websovellusten testausautomaatiota voitaisiin tehostaa. Tavoitteena oli jatko-kehittää N4S@JAMK-projektin Contriboard-palvelun hyväksyntätestaamisessa käytettyä järjestelmää siirtämällä testausautomaation suorittaminen pilvipalveluihin. Uusi järjestelmä mahdollistaisi useamman testitapauksen suorittamisen yhtäaikaisesti.

N4S@JAMK on Jyväskylän ammattikorkeakoululla toimiva projekti, joka sai alkunsa vuonna 2014. Projekti on osa Digilen johtamaa Need For Speed -tutkimusohjelmaa ja toimii tämän työn toimeksiantajana.

Opinnäytetyön teoriaosa käsittelee ohjelmistotestauksen eri vaiheita ja työssä käytettyjä työkaluja sekä teknologioita. Toteutusosassa tehdään katsaus järjestelmän rakentamiseen sekä sen aikana kohdattuihin ongelmiin ja haasteisiin.

Järjestelmän tavoitteena oli vähentää hyväksyntätestaukseen kuluvaa aikaa ja nopeuttaa mahdollisten ohjelmistovirheiden tunnistamista, jotta uusien ohjelmistoversioiden julkaiseminen olisi nopeampaa. Parannetun testijärjestelmän mahdollistama pilvilaskennan hyödyntäminen osoittautui toimivaksi ratkaisuksi ja tulokset olivat lupaavia. Työ toteutettiin Amazon EC2 -ympäristössä.

Lopputuloksena saatiin järjestelmä, joka täyttää alussa sille asetetut vaatimukset. Järjestelmän mahdollistama useamman testiskenaarion yhtäaikainen suorittaminen nopeutti testausautomaatiota selkeästi.

Avainsanat (asiasanat)
Hyväksyntätestaus, testausautomaatio, N4S@JAMK, Need For Speed, Contriboard

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Arttu Henell	Bachelor's thesis	23.5.2016
		Language of publication: Finnish
	Number of pages: 56	Permission for web publication: x

Title of publication
Distributing automated acceptance testing
Case: Contriboard

Degree programme
Software Engineering

Supervisor(s)
Hannu Luostarinen, Raija Hämäläinen

Assigned by
JAMK University of Applied Sciences / Marko Rintamäki

Abstract

The goal in this thesis was to research new ways to improve test automation in web applications. The goal was also to further develop the acceptance testing system used in Contriboard service developed by N4S@JAMK project. The new system would distribute test execution to cloud services and allow parallel testing.

N4S@JAMK is a project of JAMK University of Applied Sciences, and it started in 2014. The project is a part of Digile's Need For Speed research program. The thesis was assigned by N4S@JAMK project.

The theoretical section focuses on different levels of software testing and all the tools and technologies used in the thesis. The implementation section reviews the development process, problems and challenges which occurred while building the new system.

The main task of the system was to reduce the acceptance testing time and improve awareness of possible defects in software so that releasing new software versions would be quicker. Using cloud computing was found to be a very practical solution and the results were promising. The system was built using Amazon Elastic Cloud 2.

The result is a system which meets the requirement specifications. Parallel test execution made the test automation process significantly faster.

Keywords/tags (subjects)
Acceptance testing, test automation, N4S@JAMK, Need For Speed, Contriboard

Miscellaneous

Sisältö

Kuviot

Sanasto

Amazon EC2

Skaalautuvia virtuaalipalvelimia pilvilaskentakäyttöön.

Amazon Web Services

Amazonin pilvipalvelualusta

AMI

AMI (Amazon Machine Image) on valmis levykuva, jonka pohjalta voidaan luoda virtuaalikoneita.

Ansible

Esimerkiksi sovellusten ja palveluiden käyttöönottoon tarkoitettu automaatiotyökalu.

CI

CI (engl. continuous integration) tarkoittaa jatkuvaa integraatiota.

Git

Git on yleisesti käytetty ohjelmistokoodin versionhallintaohjelmisto.

Java Platform

Java Platform on laajasti käytetty alusta ohjelmistojen kehittämiseen ja jakamiseen työpöytä-ja palvelinympäristöihin.

Jenkins

Jenkins on alustariippumaton jatkuvan integroinnin palvelu.

Python

Python on helppokäyttöinen tulkattava ohjelmointikieli.

Robot Framework

Robot Framework on hyväksyntätestaukseen tarkoitettu ohjelmistokehys.

Selenium

Selenium on pääasiassa websovellusten testaamiseen tarkoitettu ohjelmistokehys.

Selenium GRID

Selenium-ohjelmistokehyksen Java-palvelin

SSH

SSH on salattuun tietoliikenteeseen tarkoitettu protokolla, jonka yleinen käyttötarkoitus on ottaa yhteys SSH-palvelimeen päästäkseen käyttämään toista tietokonetta konsolin kautta.

SUT

System Under Test eli lyhyemmin SUT on testattava järjestelmä.

Virtuaalipalvelin

Virtuaalipalvelin on samankaltainen kuin fyysinen palvelin mutta ohjelmallisesti toteutettu.

1 Työn lähtökohdat

1.1 Tausta ja tavoitteet

Opinnäytetyön toimeksiantajana toimi Jyväskylän ammattikorkeakoulu. Kesällä 2015 JAMK N4S -projektissa huomattiin ongelmia automatisoidun ohjelmistotestauksen toteutuksessa ja tehokkuuteen. Automatisoitujen testien ja testitapauksien määrän kasvaessa testausaika saattoi kasvaa huomattavan suureksi: yksi toisensa jälkeen ajettavat testiskenaariot veivät huomattavasti aikaa viivästyttäen CI -prosessia. Ongelmaan löytyy eri ratkaisuja, jotka mahdollistavat testaamisen kuorman jakamisen useammalle järjestelmälle. Toimiessaan parempi järjestelmä nopeuttaisi virheiden raportointia ja testikokoelma saataisiin suoritetuksi nopeammin. Tässä työssä perehdytään valittuun tapaan, jolla websovelluksen testausautomaatiota voidaan hajauttaa.

Opinnäytetyötä ei tarvinnut lähteä toteuttamaan täysin tyhjältä pöydältä. Kesältä 2015 jäi talteen paljon käyttökelpoista testimateriaalia, jota pystyi nyt hyödyntämään. Testimateriaalin kirjo on laaja: yli sadan testitapauksen kokoelmassa käydään Contriboard-palvelun toimintoja läpi käyttäjän rekisteröinnistä alkaen.

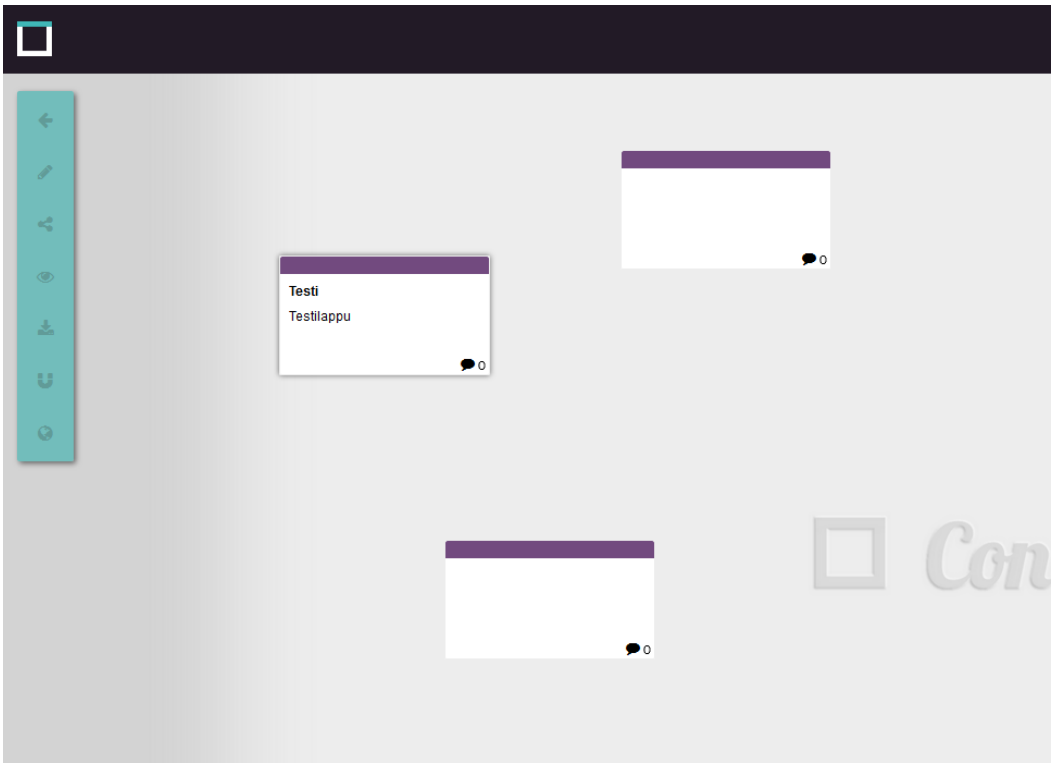
Työn teoriaosuudessa käydään lyhyesti läpi testaamisen eri muotoja, testausautomaatiossa yleisesti käytettyjä työvälineitä ja sitä, mitkä ovat eri testausvaiheiden tehtävät. Lisäksi käydään läpi muita työssä käytettyjä teknologioita ja tekniikoita. Toteutusosassa käydään läpi järjestelmän kehitysprosessia, toteutusta sekä sen aikana kohdattuja ongelmia.

1.2 N4S@JAMK-projekti

N4S@JAMK on Jyväskylän ammattikorkeakoululla toimiva projekti, joka käynnistyi vuonna 2014. N4S@JAMK on osa Digilen johtamaa N4S -tutkimusohjelmaa. Projektin työntekijät koostuvat lähinnä Jyväskylän ammattikorkeakoulun opiskelijoista tai juuri tutkintonsa päätökseen saaneista henkilöistä. Tämän työn laatija otti itsekin osaa projektiin kesällä 2015, jolloin Jyväskylän ammattikorkeakoulun Dynamon toimipisteellä työskenneltiin Contriboard-nimisen tuotteen kehityksen parissa.

1.3 Contriboard-palvelu

Contriboard on N4S@JAMK-ohjelman aikana kehitetty yhteistyösovellus (engl. collaboration tool), jota voidaan käyttää muun muassa apuna projektinhallinnassa sekä ideoinnin tehostamisessa. Käyttäjät voivat koota palvelun avulla erilaisia ideoita ja ajatuksia sähköiseen muotoon. Palvelun käyttäjä luo virtuaaliselle seinätaululle sähköisiä lappuja (ks. kuvio 1). Taululle tehtävät muutokset näkyvät kaikille käyttäjille heti. (Contriboard Fact Sheet 2015.)



Kuvio 1. Contriboard

2 Pilvipalvelut

2.1 Yleistä

Pilvipalvelut vähentävät tarvetta rakentaa ja ylläpitää omia datakeskuksia - samalla henkilöstön määrä pysyy pienempänä johtaen kustannustehokkuuteen. Yritykset voivat nostaa uusia järjestelmiä ja järjestelmäkomponentteja nopeasti ja helposti. Näin muutoksiin pystytään reagoimaan ketterästi. Pilvipalveluiden vahvuus on niiden joustavuus ja taloudellinen hyöty. (Rosenberg & Mateos 2011, 1-3.)

Muutama asia, jotka määrittävät pilvipalvelut (mts. 1-3):

- Keskitetty laskentakapasiteetti

- Skaalautuvuus

- Laskutus käytön mukaan

- Käyttöjärjestelmäriippumattomia

Pilvipalvelujen tietoturva

Huoli tietoturvasta herää, kun sovellus tai käytettävä kapasiteetti ulkoistetaan, ja joskus maantieteellisesti hyvinkin kauas. Kotikäyttäjän kohdalla pilvitoimintamallin pitäisi lisätä tietoturvaa, koska silloin vastuu tietoturvasta siirtyy palveluntarjoajalle. Palveluntarjoajat voivat kuitenkin luvata maat ja taivaat, mitä loppukäyttäjä ei pysty millään varmistamaan. On hyvin vaikea tietää, kuinka turvallinen palvelu oikeasti on. (Heino 2010, 92-94.)

Tietoturvan lisäksi pilvipalvelumalliin liittyy myös tietosuojariskejä. Esimerkiksi asiakkaan tietojen vuotaminen tietomurron myötä julkisuuteen tai epäselvyys siitä, minkä maan ja/tai oikeuskäytännön mukaisesti käyttäjän tietoja palvelussa käsitellään. (Mts. 102-103).

Pilvipalvelut eivät tietenkään ole vaihtoehto kaikissa tapauksissa. Pelkästään tietoturvariskit vaikuttavat siihen päätökseen, voidaanko pilvipalveluita käyttää vai ei. Käyttäjän laittaessa tietojaan pilveen, yksi osa omien tietojen hallinnasta on menetetty. Jos yritys tai muu käyttäjä haluaa tarkemman hallita palvelintaan tai jos palvelimen laitteiston täytyy olla tarkempien määritysten mukaista, voi fyysinen palvelinratkaisu olla parempi vaihtoehto. (Velte, Velte & Elsenpeter 2010, 25-27.)

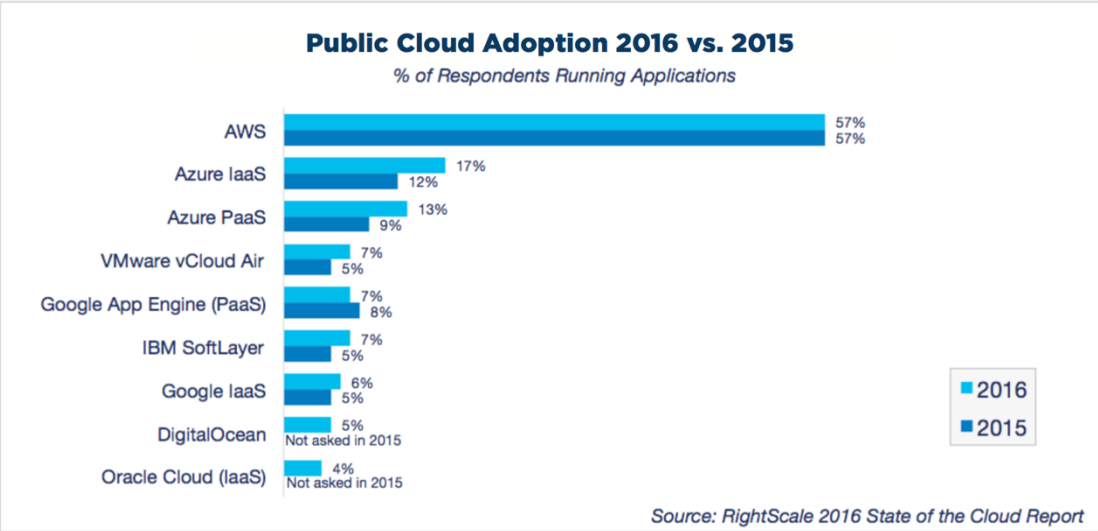
Pilvipalvelun valinta

Markkinoiden suurimmat palveluntarjoajat pilvipalveluissa ovat Amazon AWS-palvelullaan ja Microsoft Azure-palvelullaan. Molemmat tarjoavat hyvin samankaltaisia palveluita: pilvilaskentaa, tallennustilaa, tietokantoja ja niin edelleen.

Microsoftin tuotteena Azure tarjoaa integraatioita omille sovelluksilleen, esimerkiksi Visual Studiolle. Microsoft veloittaa asiakkaitaan minuuttitaksalla. (Microsoft Azure n.d.)

Palveluiden välillä on käynnissä jatkuva hintasota ja toisaalta hinnan määrittämisessä on niin paljon muuttujia, ettei vertailu ole helppoa. Päätöstä palveluiden välillä ei välttämättä kannata tehdä hinnan perusteella. Käyttäjän kannattaa miettiä käyttötarkoitusta: onko tarvetta pilvitallennustilalle vai pelkälle laskennalle? Vai molemmille? Paras neuvo on kokeilla palveluja - sekä Amazon että Microsoft tarjoavat kokeilumahdollisuuden.

Alkuvuodesta 2016 tehdyn markkinatutkimuksen mukaan Amazon oli ylivoimaisesti käytetyin pilvipalveluiden tarjoaja (ks. kuvio 2). Microsoft Azure on kuitenkin viimeisen vuoden aikana noussut kohtuullisesti. (State of the Cloud Report 2016.)



Kuvio 2. Pilvipalveluiden markkinaosuudet (State of the Cloud Report 2016.)

2.2 Palvelumallit

Tunnetuimmat palvelumallit pilvilaskennassa ovat seuraavat:

Software as a Service (SaaS):

SaaS-mallissa ohjelmistoja ei enää hankita perinteisen lisenssimallin mukaisesti, vaan tilauspohjaisesti. SaaS-malli siis tuo muutoksen ajatusmalliin, jonka mukaisesti ohjelmistoja on toimitettu asiakkaille. Tuotteita ostetaan joko kuukausi– tai vuosimaksulla, ja ohjelmistot toimivat usein pilvipalveluissa. Tyypillinen mallin mukainen palvelu on esimerkiksi Netflix. SaaS-malli on laajentumassa myös työpöytäsovelluksiin, muun muassa Microsoftin Office -tuotteeseen tai Adoben tuotevalikoimaan. SaaS-

mallin mukaiset ohjelmistot voivat olla myös ilmaisia, jolloin liikevaihtoa tehdään esimerkiksi mainostuloilla. (Cloud computing service models, Part 3: Software as a Service 2011)

SaaS -ohjelmistot toimivat siis usein ainakin jollain tasolla pilvipalveluissa, josta seuraa sekä hyviä että huonoja puolia. Etuja ovat päivitysten helppous: tuote päivittyy automaattisesti ilman käyttäjältä vaadittavia toimia ja käyttäjällä on mahdollisuus käyttää palveluun millä tahansa laitteella. (mt.)

Platform as a Service (PaaS):

PaaS on käsitteenä hieman sekava ja se sekoitetaan usein IaaS- tai SaaS-malleihin. PaaS -malli tuo ohjelmistokehittäjille samoja asioita kuin SaaS toi ohjelmistoille. Kehittäjillä on mahdollisuus rakentaa ja julkaista websovelluksia ulkoistettuun infrastruktuuriin. Infrastruktuuri sisältää kehittämiseen tarvittavia työkaluja, eikä ohjelmistokehittäjän tarvitse huolehtia lisensseistä. Kehittäjän ei tarvitse ajatella ohjelmiston skaalautuvuutta tai käyttäjämäärien noususta johtuvista ongelmia, koska alustaa pystytään joustavasti laajentamaan. (Cloud computing service models, Part 2: Platform as a Service 2011)

Infrastructure as a Service (IaaS):

IaaS-mallissa palvelimet ja palvelinsalit ulkoistetaan tarpeen mukaan. Suurin hyöty tästä yrityksille on kuormituksen siirtäminen pilveen, kun tarve isolle laskentateholle kasvaa. Yritykselle koituu säästöjä, kun ylimääräisiin palvelimiin ei tarvitse käyttää resursseja. Esimerkiksi suorituskyky -tai kuormitustestausta varten ei ole tarvetta kasvattaa palvelimien määrää pysyvästi, vaan tarvittaessa voidaan hankkia resursseja pilvestä. (Cloud computing service models, Part 1: Infrastructure as a Service 2011)

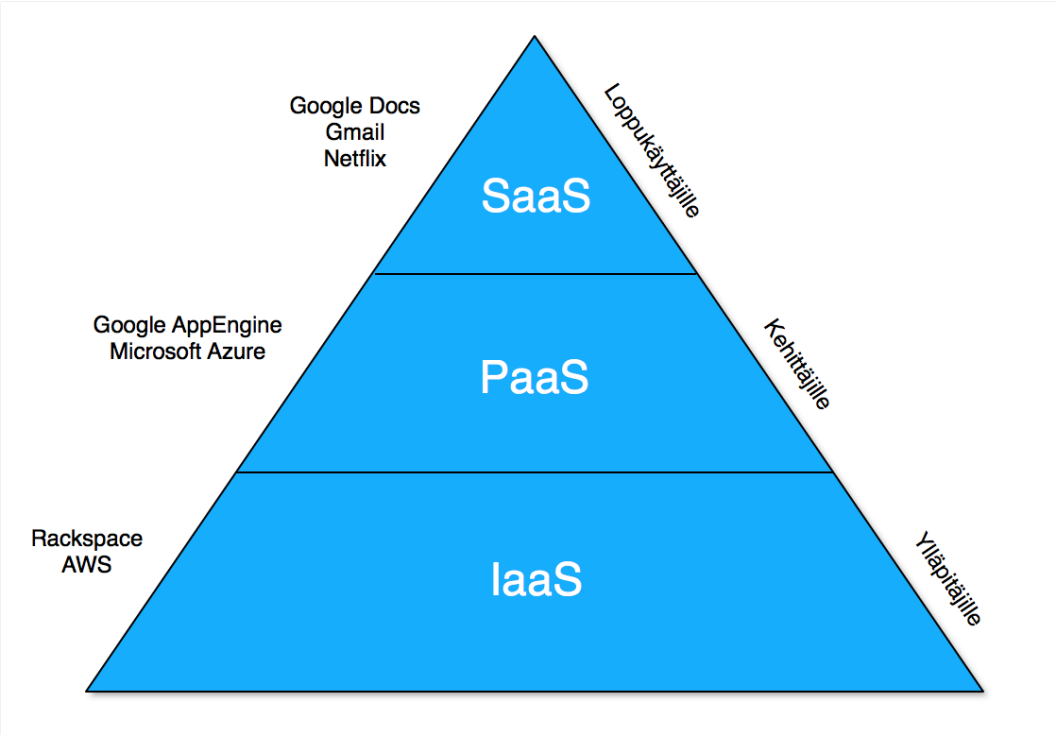
Lyhyesti:

- SaaS ohjelmistot ovat tarkoitettu loppukäyttäjille, verkon yli toimitettuna.

- PaaS: joukko kehittäjille tarkoitettuja työkaluja ja palveluita, tehden kehittämisestä ja julkaisemisesta nopeampaa. Työkalujen lisenssoinnista ja hankinnasta ei tarvitse huolehtia.

- IaaS: rautaa ja ohjelmistoja, jotka pitävät rattaat pyörimässä. (palvelimia, tallennustilaa, tietokantoja, verkkoja jne.)

Kuviossa 3 palvelumallit.



Kuvio 3. Palvelumallit

2.3 Amazon Web Services

Amazon AWS on yksi markkinoiden monista pilvipalveluiden tarjoajista. Sen palveluhin kuuluvat muun muassa pilvilaskenta, erilaiset tallennustilaratkaisut ja tietokantapalvelut. (About AWS n.d.)

Amazon EC2

Amazon Elastic Compute Cloud (lyhyemmin EC2) on keskeinen osa Amazonin AWS-pilvipalvelualustaa. EC2 tarjoaa lukuisiin eri käyttötarkoituksiin virtuaalipalvelimia, joiden suorituskykyä voidaan skaalata sen hetkisen tarpeen mukaan. Virtuaalipalvelimia voidaan luoda valmiiden levykuvien eli AMIen (Amazon Machine Image) pohjalta. Luodut virtuaalipalvelimet toimivat kuin fyysiset vastineensa. Suorituskykyä voidaan kasvattaa automaattisesti lisäämällä virtuaalikoneita. Laskutus muodostuu päällä oloaika per kuukausi -mallin mukaan hinnan kasvaessa mitä tehokkaampaa instanssia käytetään. (Amazon EC2 - Virtual Server Hosting n.d.)

3 Ohjelmistotestaus

Kasurinen kertoo kirjassaan Ohjelmistotestauksen käsikirja (2013) testauksen olevan työtä, jonka aikana varmistetaan, että toteutettavasta ohjelmistotuotteesta tulee toivotun kaltainen ja sen ominaisuudet varmasti toimivat niin kuin oli tarkoitus. (Kasurinen 2013, 10.)

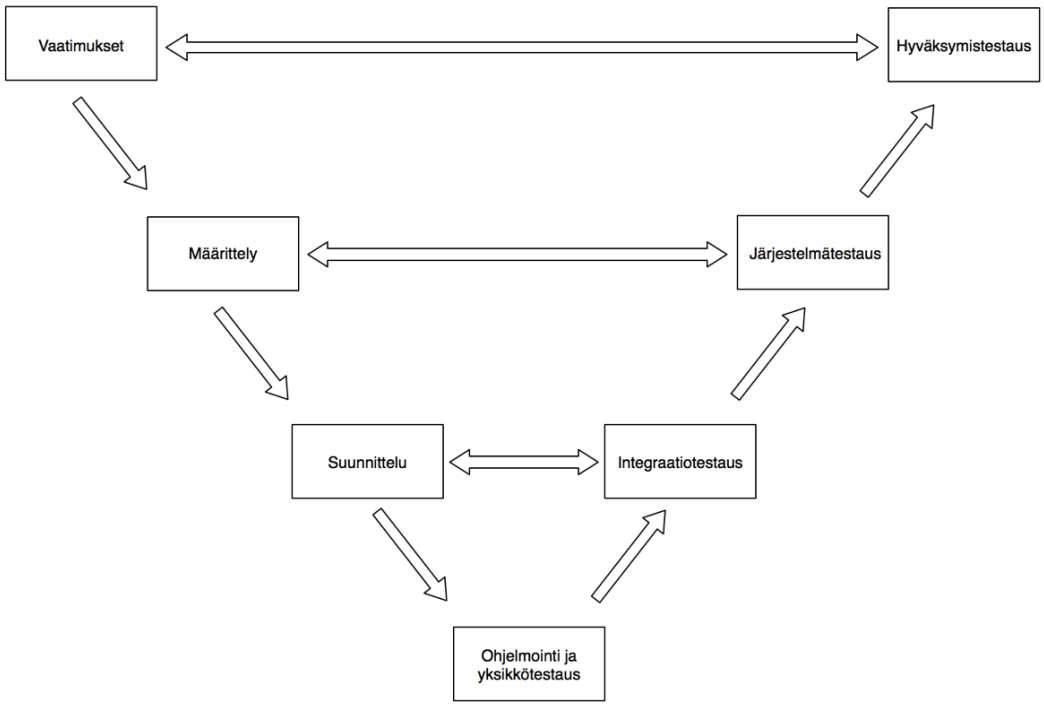
The Art of Software Testing -teoksesta löytyy vasta-argumentti:

Testaaminen terminä käsitetään usein väärin. Testaamisen tarkoituksena ei ole näyttää, että lopullinen tuote toimii määritelmän mukaisesti. Testaajan on oletettava, että virheitä löytyy. (Myers, Badgett & Sandler 2012, 5-6.)

Testaus on siis prosessi, jossa testattavaa kohdetta ajetaan tarkoituksena löytää virheitä. Testaaminen lisää laatua, jonka asiakas huomaa tuotteen käyttöönoton jälkeen. (Mts. 6)

3.1 Ohjelmistotestauksen V-malli

Ohjelmistotestaus jaetaan tasoihin ohjelmistokehityksen eri vaiheissa. Testausvaiheessa katsotaan, vastaako sen hetkinen taso samalla tasolla olevaa suunnitteluvaihetta. V-mallia (ks. kuvio 4) voidaan toki aina muokata paremmin projektiin ja organisaatioon sopivaksi. Contriboard-palvelun tapauksessa erillistä integraatiotestausta ei tehty. Ohjelmoijat tekivät yksikkötestejä, järjestelmätestaus tehtiin manuaalisesti ja hyväksyntätestaus automatisoidusti (Robot Framework).



Kuvio 4. Ohjelmistotestauksen V-malli

3.2 Yksikkötestaus

Yksikkötestaus (engl. unit testing) on testausmuodoista yleisin, ja sitä käytetään käytännössä kaikissa ohjelmisto-organisaatioissa. Yksikkötestien tekeminen aloitetaan jo ohjelmoinnin yhteydessä ja useimmiten itse kehittäjän toimesta. Tästä seuraa se etu, että ongelmat toteutuksessa löydetään jo hyvin varhaisessa vaiheessa. Jos testi epäonnistuu, voi ohjelmoija korjata vian, ennen kuin ohjelmiston osa tai komponentti on osa laajempaa kokonaisuutta. (Kasurinen 2013, 51-52.)

Yksikkötestauksen tavoitteena on eristää ohjelman eri osat ja näyttää, että yksittäiset ohjelmiston palaset toimivat niin kuin on tarkoitus. Tämä on myös yksikkötestauksessa usein ilmenevä ongelma: ohjelmistot koostuvat useista pienistä komponenteista ja niiden välisestä vuorovaikutuksesta. Yksittäinen komponentti pystyy harvoin tekemään mitään itsenäistä. (Mts. 51-52)

xUnit -työkalut

xUnit on yhteinen nimitys usealle Smalltalkin SUnit-työkalusta kehitetylle yksikkötestaamisen ohjelmistokehykselle. Aluksi SUnit käännettiin menestyksekkäästi Java –ohjelmointikielelle, jonka ansiosta vastaava työväline löytyy valtaosasta tällä hetkellä eniten käytetyistä ohjelmointikielistä. (xUnit n.d.)

3.3 Integrointitestaus

Yksikkötestausta seuraava työvaihe on integrointitestaus (engl. integration testing), joka on loogista jatkoa yksikkötestaukselle. Integrointitestauksessa järjestelmä tai sovellus on tarkoituksena saada toimimaan yhtenä kokonaisuutena yhdistämällä pienemmät osat. Kun kokonaisuus toimii, siihen lisätään uusi komponentti ja toimivuus tarkastetaan uudelleen. Esimerkiksi Contriboard-palvelu sisältää useita eri osia, näistä API, IO, IMG ja CLIENT, mutta integrointitestaus-vaihe ohitettiin. Integrointitestaus korvattiin järjestelmätesteillä. (Kasurinen 2013, 54.)

Integrointijärjestys voidaan tehdä monella eri tavalla mutta seuraavat tavat ovat yleisimmät (mts. 55.):

Alhaalta ylöspäin (engl. bottom up testing)**:**

Järjestelmään tuodaan ensimmäisenä matalimman tason moduulit. Näitä ovat esimerkiksi moduulit, jotka kommunikoivat raudan tai järjestelmän kanssa (kuten verkkoyhteydet tai tietokanta).

Ylhäältä alaspäin (engl. top down testing)**:**

Periaate tässä tavassa on sama kuin edellä mainitussa, mutta testaus aloitetaankin järjestelmähierarkian huipulta kohti rautatason toimintoja.

Voileipätestaus (engl. sandwich testing)**:**

Alhaalta ylöspäin ja ylhäältä alaspäin -tapojen yhdistelmä, eli integraatiota lähestytään molemmista suunnista yhtäaikaisesti.

Big Bang

Tässä menetelmässä iso osa kehitetyistä moduuleista yhdistetään kerralla isommaksi kokonaisuudeksi. Big Bang -metodi voi parhaimmillaan lyhentää integrointitestaukseen kuluvaa aikaa hyvinkin paljon. Toisaalta integrointiprosessi on monimutkainen, jos testitapaukset ovat huonosti suunniteltuja. Tällöin lopputuloksena on sekasorto.

3.4 Järjestelmätestaus

Järjestelmätestaus (engl. system testing) nimensä mukaisesti on lähes valmiin, jo rakennusvaiheen ohittaneen kokonaisuuden testaamista. Yleisesti ajatellen tämän testivaiheen tavoitteena on varmistaa, että järjestelmä toteuttaa kaikki sille asetetut tavoitteet ja toimii kokonaisuutena. (Kasurinen 2013, 56-57.)

Järjestelmätestauksessa käytetään todellisen tilanteen eli lopputuotteen kaltaista testiympäristöä (engl. system under test, lyhyemmin SUT). Vasta hyväksymistestausvaiheessa tuote siirretään lopulliseen kohdeympäristöön. Ero hyväksymistestauksen ja järjestelmätestauksen välillä on siinä, että järjestelmätestauksessa virheitä etsitään myös yksittäisistä komponenteista, eivätkä mahdolliset muutokset ole epätavallisia. Hyväksymistestauksessa järjestelmään tehtäviä muutoksia vältetään ja keskitytään toiminnallisuuksien testaamiseen. (Mts. 56-57.)

3.5 Hyväksymistestaus

Ohjelmistotestauksen V-mallin viimeinen työvaihe eli hyväksymistestaus (engl. acceptance testing) osoittaa, onko testattava ohjelma riittävän korkealaatuinen ja kyvykäs täyttämään sille vaatimusmäärittelyssä annetut vaatimukset. Hyväksymistestauksessa järjestelmää ajetaan sen lopullisessa kohdeympäristössä ja lopullinen tarkastus suoritetaan. Jos tuote läpäisee hyväksymistestit, voidaan järjestelmän olettaa olevan halutun kaltainen. Toisaalta asiakas ja kehittäjä voivat tarpeen mukaan muuttaa testejä tai kehittää uusia. Asiakas siis osallistuu tarkastukseen ja hyväksyy tuotteen, jolloin tuote lakiteknisesti siirtyy asiakkaan omaisuudeksi. (Kasurinen 2013, 57.)

3.6 Testausmenetelmät kehitysvaiheessa

Ohjelmistotestauksen V-mallin tasojen sisällä testausta voidaan toteuttaa eri tavoilla, joista yleisimpiä ovat mustalaatikko, lasilaatikko ja harmaalaatikko.

3.6.1 Mustalaatikko

Mustalaatikkotestaamisessa testaaja ei tiedä tarkemmin, kuinka testattava kohde toimii tai mitä sen sisällä tapahtuu. Vaatimusmäärittelyn perusteella tehtyjen testitapauksien syötteet annetaan ohjelmalle tai järjestelmälle ja tarkkaillaan mitä testattava kohde tekee. Kun ohjelman sisälle ei nähdä, on annettavien syötteiden oltava oikein ja lopputuloksen oltava testitapauksessa määritellyn kaltainen. (Lewis 2009, 39-40.)

3.6.2 Lasilaatikko

Lasilaatikkotestaus on mustalaatikkotestauksen täydellisempi versio, jossa ohjelmalle tai järjestelmälle annetaan syötteitä samalla tarkastellen sekä testattavan kohteen reagointia syötteisiin että järjestelmän sisällä tapahtuvia muutoksia. Koska tässä menetelmässä tarkkaillaan ohjelman sisällä tapahtuvia asioita lähdekooditasolle asti, asettaa se tiettyjä vaatimuksia myös testaajalle. Testaajan pitää tuntea järjestelmä lähdekoodia myöten, ja siksi ymmärrys ohjelmoinnista pitää olla riittävällä tasolla. Lasilaatikkotestauksen etu on sen järjestelmällisyys. (Lewis 2009, 40.)

3.6.3 Harmaalaatikko

Harmaalaatikkotestaus on yhdistelmä musta- ja lasilaatikkotestausta. Testaaja tutkii vaatimusmäärittelyä ja kommunikoi kehittäjien kanssa ymmärtääkseen ohjelman rakenteen. Menetelmässä pyritään käymään läpi vaatimusmäärittelyä ja toisaalta tarkastamaan samalla lähdekoodia. Harmaalaatikkomenetelmä sopii tilanteisiin, jossa ei kokonaisvaltaisesti pystytä käyttämään lasilaatikkomenetelmää. (Lewis 2009, 41.)

3.6.4 Regressiotestaus

Regressiotestaus tarkoittaa yksinkertaisesti uudelleentestaamista. Kun ohjelmiston lähdekoodiin tehdään muutoksia, voidaan vanhoja testejä ajamalla tarkistaa, rikkoiko uuden koodin lisääminen jo toimivia ominaisuuksia. Regressiotestaus ei ole erillinen testaamisen muoto kuten järjestelmätestaus tai yksikkötestaus vaan yleisnimi testaustyölle ja regressiotestausta voidaankin tehdä testaamisen jokaisessa vaiheessa. Ideaalissa tilanteessa regressiotestaukseen käytettäisiin mahdollisimman vähän aikaa, mutta ei kuitenkaan niin, että uusien vikojen löytymisen todennäköisyys pienenisi. (Regression Testing n.d.)

3.7 Testausmenetelmät ennen julkaisua

3.7.1 Kuormitustestaus

Kuormitustestaus on yksi suorituskykytestauksen muoto, jolla sovelluksen vakautta ja luotettavuutta voidaan testata. Sovellus altistetaan muun muassa isoille käyttäjämäärille tai websovelluksen tapauksessa verkkoyhteyttä voidaan häiritä ohjelmallisesti. Oikeanlaisella kuormitustestauksella löydetään esimerkiksi käyttäjien tekemien muutoksien synkronisaatioon liittyviä ongelmia. Järjestelmä ajetaan hajoamispisteeseen, jotta löydetään ohjelmistovirheet, jotka tekisivät hajoamisesta vaarallisen. Testattavan järjestelmän ei ääriolosuhteissakaan pitäisi korruptoida tai hävittää dataa. (Stress Testing 2007.)

3.7.2 Käytettävyystestaus

Käytettävyystestauksella voidaan kartoittaa käyttäjien tarpeita ja odotuksia tuotetta kohtaan. Tarkoituksena on siis tutkia testattavaa kohdetta ja saada informaatiota siitä, kuinka käyttäjä onnistuu ohjelmaa tai järjestelmää käyttämään ja pääseekö hän haluamaansa lopputulokseen. Käytettävyyden tutkimisessa voidaan keskittyä muun muassa seuraaviin asioihin (Lewis 2009, 359-360.):

Helppokäyttöisyys: Kuinka nopeasti käyttäjä oppii käyttämään täysin tuntematonta järjestelmää?

Johdonmukainen toiminta: Esimerkiksi hyperlinkin klikkaaminen avaa pop-up -ikkunan, kun taas painikkeen klikkaaminen vie käyttäjän uuteen ikkunaan.

Käyttöliittymä: Onko käyttäjälle näkyvä applikaation osa riittävän selkeä? Voiko käyttäjä selvittää sijaintinsa sivustolla mahdollisimman helposti?

Ilmoitukset: Ovatko ohjelman antamat ilmoitukset tarkoituksenmukaisia ja helposti ymmärrettäviä? (Myers, Badgett & Sandler 2012, 144.)

Vaatimukset: Katsotaan käyttäjän näkökulmasta vastaako toiminta vaatimusmäärittelyssä kirjattuja lupauksia. (Myers, Badgett & Sandler 2012, 145.)

3.7.3 Luotettavuustestaus

Jokaisen testaamisen osa-alueen tavoitteena on parantaa sovelluksen tai järjestelmän luotettavuutta, mutta luotettavuuteen voidaan kiinnittää erityistä huomiota vaatimusmäärittelyssä. Luotettavuuden testaaminen voi olla hankalaa. Yhden määritelmän mukaan luotettavuus tarkoittaa aikaa, jonka testattavan kohteen on toimittava tietyssä ympäristössä. Luotettavuuden todentamisessa käytetään MTBF:ää (engl. mean time between failures) eli virheiden välistä keskimääräistä aikaa. Myös kuormitustestaus on yksi osa luotettavuustestausta. (Myers, Badgett & Sandler 2012, 127-128.)

3.7.4 Tietoturvatestaus

Viime vuosina tietoturva ja yksityisyys ovat nousseet puheenaiheeksi ohjelmistoalalla, jonka myötä tietoturvaongelmiin on alettu kiinnittää enemmän huomiota. Tietoturvaa voidaan testaamisen avulla parantaa suunnittelemalla testitapauksia jo tiedossa olevan, samankaltaisen järjestelmän tai sovelluksen kohtaaman ongelman pohjalta. Tavoitteena on saada aikaan vastaavia tietoturva-aukkoja. Websovellukset tarvitsevat usein korkeamman tason tietoturvatestausta maksutapahtumien ja muiden yksityisten tietojen käsittelystä johtuen. (Myers, Badgett & Sandler 2012, 125.)

3.8 Testausautomaatio

Testausautomaatio (engl. test automation) on testaustoiminnan muoto, jonka ydinajatus on vapauttaa testaajia muihin työtehtäviin. Tämä saavutetaan rakentamalla

erilaisia automaattityövälineitä toistuvasti suoritettavia testitapauksia varten ja siksi testaajalle ohjelmointitaito on välttämätön taito. Testausautomaation hyödyt tulevat esiin esimerkiksi uusien ohjelmistokäännöksien (build) luomisen yhteydessä. Perustestit ovat resurssien käytön kannalta järkevämpää siirtää automatisoiduiksi. (Kasurinen 2013, 76.)

Testausautomaation käyttäminen ei kuitenkaan poista tarvetta manuaaliselle testaamiselle. Joissain tapauksissa niin sanottuja oikeita ongelmia voi löytää vain ihminen - tutkivaa testausta ei voi luonteensa vuoksia automatisoida. Tämän lisäksi automaatioalustan rakentaminen voi ollaa isokin investointi. Regressiotestauksessa, laadunvarmistuksessa ja kuormitustestauksessa testausautomaatio on hyödyllistä ja kätevää. (Mts. 76)

Testausautomaation lähestymistavat (Test automation n.d.):

Graafisen käyttöliittymän testaaminen (engl. GUI testing)

Testausohjelmistokehys toteuttaa käyttöliittymän toimintoja kuten näppäimistön painalluksia ja hiiren klikkauksia, tarkkailee käyttöliittymään tulevia muutoksia ja varmistaa, että lopputulos oli oikeanlainen.

Ohjelmointirajapintatestaus (engl. API driven testing)

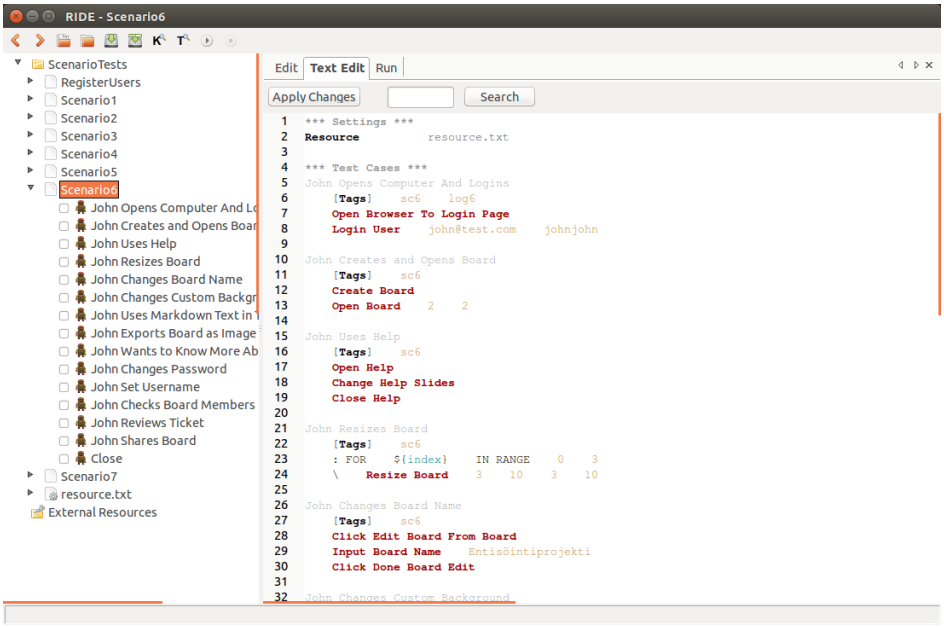
API eli ohjelmointirajapintatestaamisessa testataan APIa (engl. application programming interface) suoraan. Ohjelmointirajapinta ei sisällä käyttöliittymää, joten testaus suoritetaan viestitasolla (engl. message layer). API -testaus on yhä tärkeämpi kohde testausautomaatiota, koska rajapinnoilla on ensisijainen yhteys ohjelman business logiikkaan, ja toisaalta käyttöliittymätestit ovat hankalia ylläpidettäviä käyttöliittymään jatkuvasti tulevien muutoksien takia.

4 Käytetyt tekniikat ja teknologiat

4.1 Robot Framework -työkalu

Robot Framework on Nokia Siemens Networksilla alkujaan kehitetty, hyväksyntätestaamiseen (engl. acceptance testing) tarkoitettu avoimen lähdekoodin yleiskäyttöinen testausautomaatiotyöväline. Työkalu soveltuu sekä web- ja työpöytäsovellusten-testaamiseen. Robot Frameworkin käyttö perustuu avainsanaohjattuun testausmenetelmään. Robot Framework -ohjelmistokehystä voidaan laajentaa ohjelmoimalla uusia testikirjastoja Python -tai Java -ohjelmointikielillä. (Robot Framework n.d.)

Testitapauksien luomiseen voidaan käyttää RIDE -editoria. (Ks. kuvio 5.) RIDE sisältää kehitysympäristölle tyypilliset ominaisuudet, muua muassa koodintäydennyksen, syntaksin erottelun ja virheraportoinnin. Liitteenä 2 testeistä generoitu html -raportti.



Kuvio 5. RIDE

Robot Frameworkin käyttö ei pääsääntöisesti vaadi testaajalta syvää perehtymistä ohjelmointiin pois lukien tilanteet, joissa joudutaan muokkaamaan tai luomaan uusia testikirjastoja esimerkiksi Python-ohjelmointikielellä. Syntaksiltaan Robot Frameworkin testitapaukset on helppo määrittää. Testit muodostuvat muutamasta tiedostosta. Testitapauksissa tehtävät asiat määritellään erillisessä tiedostossa, nimeltään usein *settings* tai *resource*. Kuvion 6 yläosassa näkyy, kuinka käyttäjän sisäänkirjautumisen

vaiheet määritetään *resource* -tiedostossa, ja alaosassa kuinka *Open Browser To Login Page* -avainsanan käyttö tapahtuu itse testitapauksissa. Liitteenä 1 on esimerkkinä yksi testiskenaario, josta voi tarkastella testien rakennetta ja sisältöä.

```
*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LoginUrl}    ${Browser}    None    ${GRID}
    Set Window Size    1920    1200
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}
    Location Should Be    ${LoginUrl}
    Title Should Be    ${LoginTitle}
    Wait Until Page Contains Element    //form[@class='form']
```

```
*** Settings ***
Resource          resource.txt

*** Test Cases ***
Open Login Page
    [Tags]    reg
    Open Browser To Login Page
```

Kuvio 6. Robot Framework -esimerkki

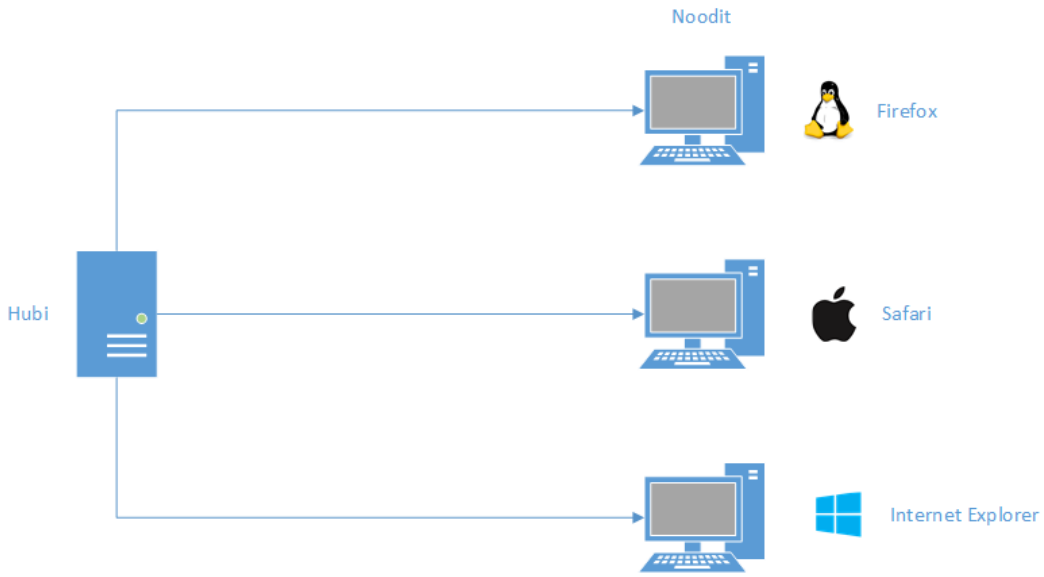
4.2 Selenium-työkalu

Selenium on erityisesti websovelluksien testaamiseen suunniteltu työkalu, jonka toiminta perustuu selaimen automatisointiin. Seleniumin avulla voidaan simuloida käyttäjän toimia. Seleniumia voidaan käyttää ilman, että käyttäjä opettelee joko Seleniumin oman skriptauskielen (Selenium IDE) tai muun tuetun ohjelmointikielen (Python, Java, C#). Vaihtoehtoisesti Seleniumin toiminnallisuus voidaan myös yhdistää Python -kirjastolla Robot Framework -ohjelmistokehykseen. (Selenium n.d.)

Selenium GRID

GRID on Selenium -ohjelmistokehyksen Java -pohjainen palvelin, joka mahdollistaa testien ajamisen etätietokoneilla. GRID mahdollistaa testaamisen hajauttamisen useammalle järjestelmälle vähentäen testaamiseen kuluvaa aikaa. Osaa testitapauksista on mahdoton ajaa rinnakkain. Esimerkki Contriboard-palvelusta: mikäli ensimmäisessä testiskenaariossa käyttäjä loisi uuden ”seinälapun”, mutta toisessa samanaikaisesti ajettavassa testiskenaariossa sama käyttäjä poistaisi äskettäin luodun lapun, konfliktit ovat todennäköisiä.

GRID käyttää hubeja ja noodeja. Testit käynnistetään hubissa, mutta itse suoritus tapahtuu erillisissä koneissa, noodeissa. Hubi on siis keskuskone, ja noodit sen työntekijöitä. Hubi ohjaa testit vapaille noodeille. Noodi-koneet voivat pohjautua eri alustoihin (OS X, Windows, Linux) ja eri selaimiin (Firefox, Google Chrome, Internet Explorer ja Safari), jolloin samalla testauskattavuus laajenee ja pystytään mahdollisesti löytämään myös alustakohtaisia ohjelmistovirheitä. GRIDin koneet voidaan toteuttaa virtuaalikoneilla tai fyysistä laitteistoa käyttäen. Kuviossa 7 esimerkki, kuinka koneet voitaisiin konfiguroida. (Selenium GRID n.d.)



Kuvio 7. Selenium GRID

Selenium WebDriver

Selenium WebDriver on Selenium-ohjelmistokehyksen ajuri, joka antaa selaimelle suoria käskyjä käyttäen selaimen natiivia tukea automatisoinnille. Verrattuna vanhempaan Selenium RC -ajuriin WebDriver sisältää paremman tuen dynaamisille sivuille, joilla sisältö saattaa muuttua ilman sivun uudelleen latausta. Eri selaimia varten on käytettävissä erilaiset ajurit, mutta ainoastaan Firefoxin ajuri tulee paketin mukana. (Selenium WebDriver n.d.)

4.3 Jenkins-työkalu

Jenkins on jatkuvan integroinnin automaatiopalvelin, jonka käyttö perustuu "tehtäviin" (jobs). Tehtävät voi käynnistyä esim. versionhallintaan tulevien muutoksien jälkeen tai manuaalisesti hallintapaneelin kautta. Jenkinsin avulla ohjelmistotuotteiden tuotantoprosessia voidaan ohjata tehokkaammin, jolloin kehittäjät pystyvät integroimaan muutokset nopeammin ja siten käyttäjät saavat useammin uusia ohjelmistoversioita. Jenkinsin käyttöä voidaan laajentaa erilaisilla liitännäisillä (engl. plugin). Näistä mainittakoon esimerkiksi GitHub-liitännäinen, jonka avulla Jenkins luo automaattisesti uuden buildin versionhallintaan tulleiden muutoksien jälkeen. (Meet Jenkins n.d.)

4.4 Ansible-työkalu

Ansible on palveluiden ja sovellusten käyttöönottoon, konfiguraatioiden hallintaan ja tietokoneiden hallintaan, "orkestrointiin" kehitetty työkalu. Ansiblen avulla asioita voidaan automatisoida. (How Ansible Works n.d.)

Playbook

Playbookit ovat Ansiblen konfiguraation ja orkestroinnin kieli. Playbookit kirjoitetaan YAML-syntaksia noudattaen. Syntaksi on minimaalinen ja muistuttaa hieman JSON-tiedostomuotoa. Tarkoituksena ei ole muistuttaa mitään ohjelmointikieltä. Päinvastoin, playbookkien luomisen on haluttu olevan mahdollisimman yksinkertaista. (YAML 2016.) Kuviossa 8 syntaksiesimerkki.

```yaml
# Employee records
-  martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
-  tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

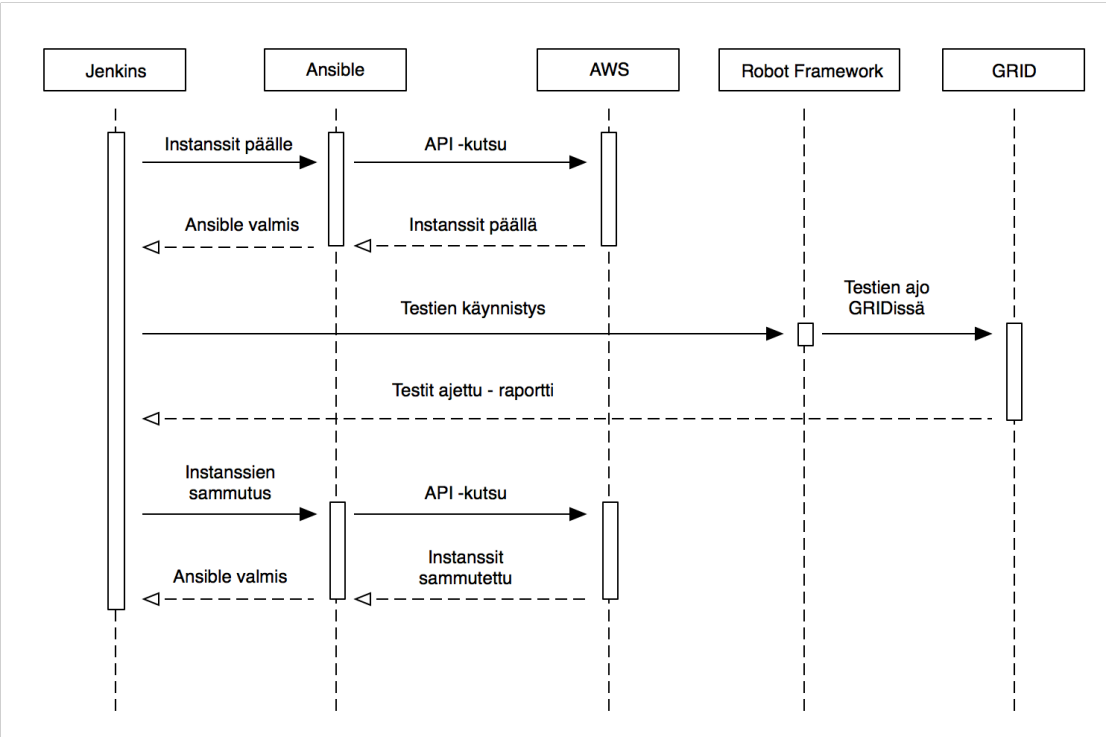
Kuvio 8. YAML-syntaksi (YAML 2016)

5 Ongelma nykytilanteessa ja työn tavoite

Contriboard-palvelun testaamisessa automatisoitujen testien hidas suorittaminen viivästytti jatkuvan integroinnin ketjua. Virheiden tunnistaminen ja tallentaminen oli hidasta, joka taas hidasti muutoksien tekemistä lähdekoodiin. Kun sama prosessi joudutaan tekemään monesti, aikaa kuluu hitaan testausautomaation takia liian paljon. Testit ajettiin Digital Oceanin pilvessä, mutta minkäänlaista hajauttamista toteutuksessa ei ollut. Testiskenaariot ajettiin yksitellen.

Tavoite

Uudessa toteutuksessa testien ajaminen hajautettaisiin useammalle pilvipalvelussa sijaitsevalle virtuaalikoneelle, jotta testiskenaarioita voitaisiin ajaa samanaikaisesti useampi kappale. Virtuaalikoneiden hallinta (sammutus, käynnistys jne.) tapahtuu Ansiblella ja testien käynnistäminen sekä niiden raportointi Jenkinsillä. Kuviossa 9 on sekvenssikaavio järjestelmän toiminnasta. Ansiblen konfigurointiin ja toimintaan sekä Jenkinsin tehtäviin palataan myöhemmin toteutusosassa luvussa 6.



Kuvio 9. Uuden järjestelmän suunnittelua

6 Toteutus

Tässä osiossa käydään läpi järjestelmän toteutusta ja toteutuksen aikana kohdattuja ongelmia. Uusi toteutus aloitettiin pilvipalvelun konfiguroinnilla, josta edettiin muiden järjestelmän komponenttien asentamiseen. Lopuksi tehtiin vertailua vanhan ja uuden järjestelmän välillä sekä tutkitaan nopeuseroa testien suorittamisessa.

6.1 Amazon-pilvipalveluiden käyttöönotto

Oli alusta asti sovittu, että Selenium GRID rakennettaisiin Amazonin pilvilaskenta-alustalle. AWS oli jo aiemmin huomattu toimivaksi ja luotettavaksi ratkaisuksi ammattikorkeakoulun N4S@JAMK-projekteissa. Fyysisille palvelimille ei ollut tarvetta, ei taloudellisesti edes mahdollisia, ja käytännön hyöty olisi jäänyt vähäiseksi.

AWS:n valintaan vaikutti muutama asia:

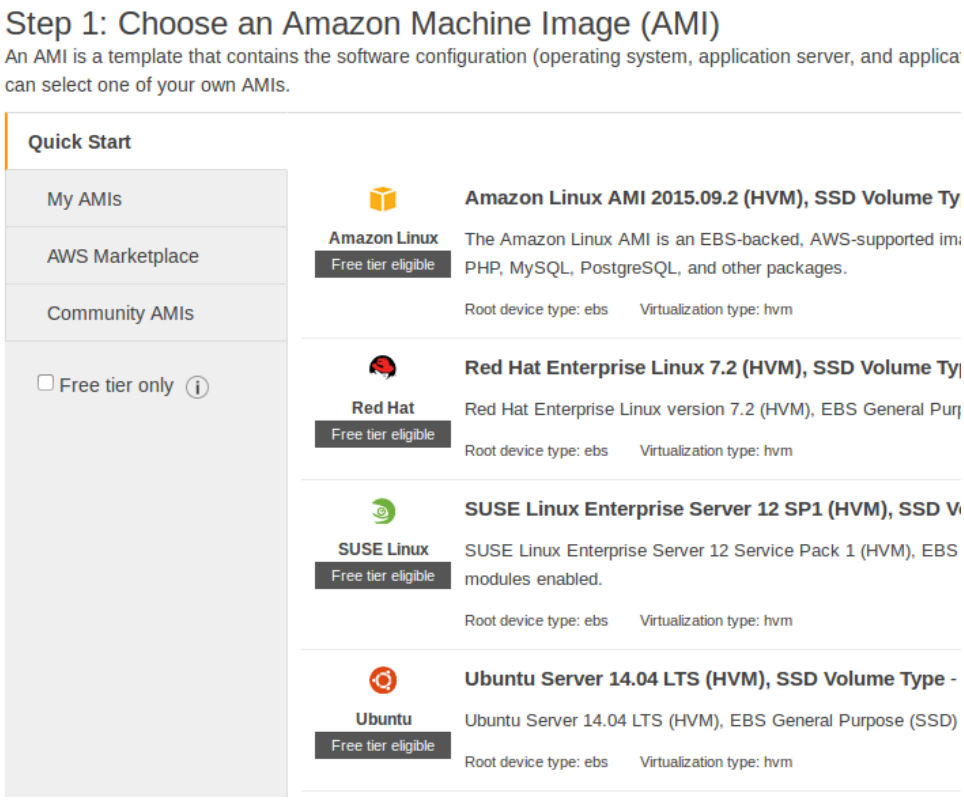
- Toimeksiantaja todennut palvelun luotettavaksi muutaman vuoden käytön perusteella

- Opinnäytetyön tekijän aiempi perehtyminen ko. palveluun.

- Markkinaosuus lupaa laatua.

Instanssien konfigurointi

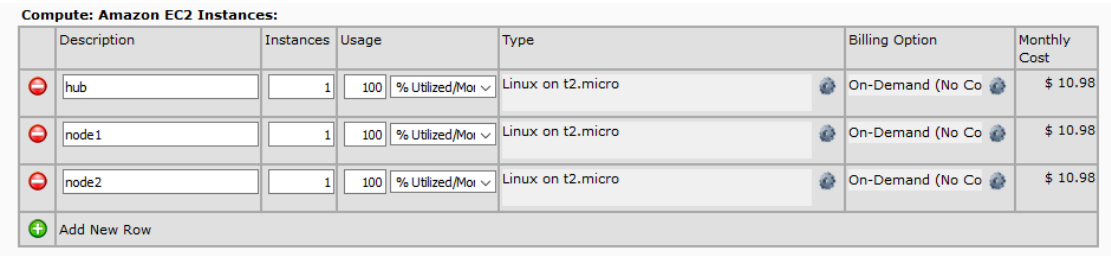
Pilvipalveluiden käytössä on kyse joustavuudesta ja skaalautuvuudesta. Asiakkaalle täytyy olla tarjolla riittävän monipuolinen valikoima eri tilanteisiin sopivia vaihtoehtoja. Ensimmäiseksi valitaan sopiva AMI (Amazon Machine Image). AMI on valmis levykuva, joka sisältää tarvittavat tiedot instanssin eli virtuaalikoneen luomiseen. Yhdestä AMIsta voidaan luoda useampia instansseja. Amazonin EC2-ympäristöön koneita luotiin yhteensä kolme: 1 kpl Selenium Gridin hubi- eli keskuskoneeksi ja 2 kpl noodikonetta hubin käyttöön testejä ajamaan. Instanssi luodaan klikkaamalla *Launch instance* -painiketta. Kuviossa 10 näkyvä Ubuntu Server 14.04 valittiin järjestelmän instanssien pohjaksi.



Kuvio 10. Amazon Machine Image

Niin Amazonin kuin muidenkin palveluntarjoajien hinnasto on melko monimutkainen, ja loppusumma muodostuu lukuisista eri asioista. Esimerkiksi yhdestä kiinteästä IP-osoitteesta joutuu maksamaan noin 4 dollaria lisää.

Kuviossa 11 näkyy hinnat kolmelle EC2-instanssille, joita työssä aluksi käytettiin. Hinnat on laskettu sadan prosentin kuukausittaisen käyttöasteen perusteella.

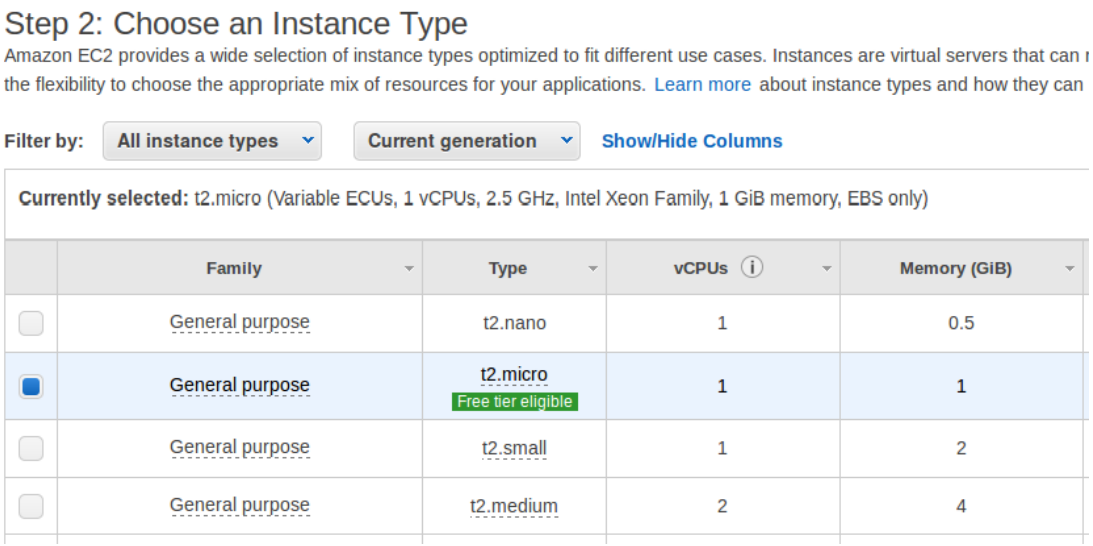


Compute: Amazon EC2 Instances:

	Description	Instances	Usage	Type	Billing Option	Monthly Cost
⊖	hub	1	100 % Utilized/Mo	Linux on t2.micro	On-Demand (No Co	$ 10.98
⊖	node1	1	100 % Utilized/Mo	Linux on t2.micro	On-Demand (No Co	$ 10.98
⊖	node2	1	100 % Utilized/Mo	Linux on t2.micro	On-Demand (No Co	$ 10.98
⊕	Add New Row					

Kuvio 11. Amazon EC2 -kustannukset

Kuviossa 12 Selenium-koneita varten valitaan hinnaston alemmasta osasta *t2.micro*-luokan Ubuntu Server -instanssi. Vaikka kyseisen instanssityypin verkon suorituskyky, muistikapasiteetti ja laskentateho ovat keskinkertaisia, ajateltiin valinnan sopivan tämän työn kaltaiseen toimintaan riittävän hyvin. Kustannukset eivät nousisi korkeaksi pykälää kalliimmallakaan vaihtoehdolla, koska testausautomaatiossa testejä suorittavat koneet harvoin ovat käynnissä ympäri vuorokauden. Usein koneet ylösajetaan uuden koontiversion luonnin jälkeen esimerkiksi yöaikaan, ja kun testaaminen on suoritettu, koneet voidaan ajaa alas.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can r
the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can

Filter by: All instance types | Current generation | Show/Hide Columns

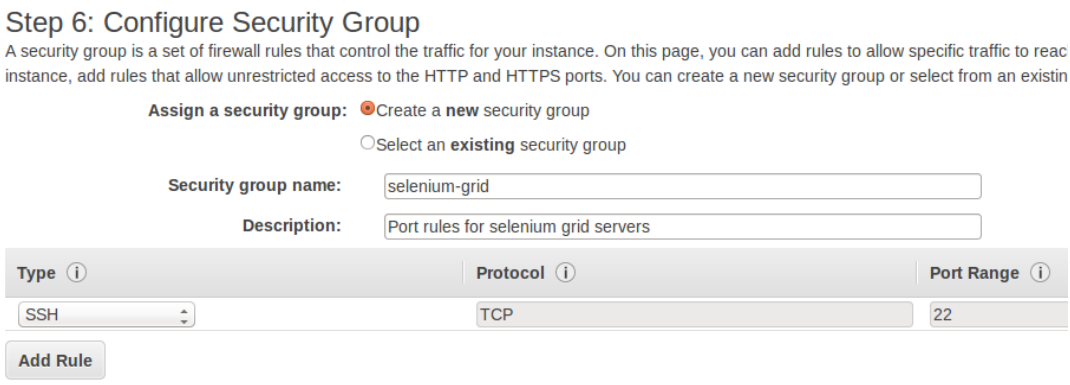
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)
☐	General purpose	t2.nano	1	0.5
☑	General purpose	t2.micro Free tier eligible	1	1
☐	General purpose	t2.small	1	2
☐	General purpose	t2.medium	2	4

Kuvio 12. Amazon EC2 -instanssit

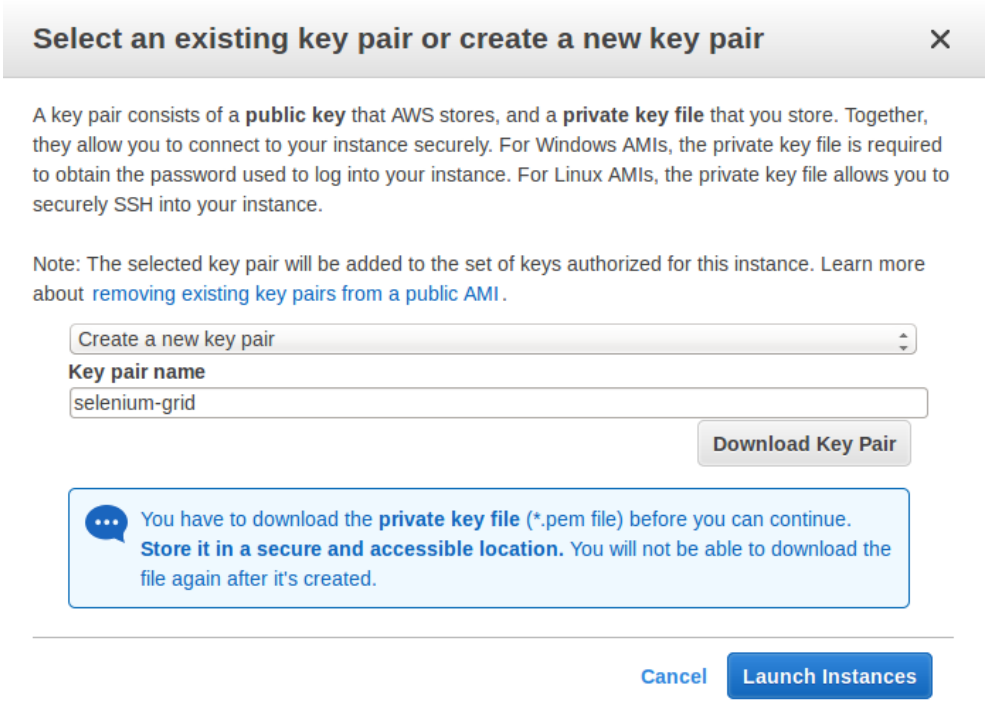
Kuten aiemmin on mainittu, yksi pilvilaskennan eduista on järjestelmän helppo skaalautuvuus. Instanssit voidaan luonnin yhteydessä asettaa Auto Scaling Groupiin. Ominaisuudesta on hyötyä isojen käyttäjämäärien palveluissa. Auto Scaling voi lisätä instanssien määrää, jos palvelun käyttäjämäärät nousisivat yllättäen. Nyt ominaisuus jätettiin huomiotta. Luodaan alustavasti vain yksi instanssi.

Yhteys Amazonin virtuaalikoneeseen saadaan käyttämällä SSH-protokollaa. Instanssin konfiguroinnin viimeisessä vaiheessa asetetaan tietoturvaan liittyviä asetuksia eli ns. Security Group. Security Group on kokoelma erilaisia palomuurisääntöjä, joilla säädetään se verkkoliikenne, jonka täytyy saada yhteys instanssiin. Alustavasti annetaan pääsyoikeus SSH -yhteydelle vain yhdelle tietokoneelle. (Ks. kuvio 13.) Myöhemmin Security Groupeja lisätään ja sallitaan useampia portteja verkkoliikenteelle.



Kuvio 13. Security Group

Konfiguroinnin lopuksi luodaan *pem*–tiedosto. (Ks. kuvio 14.) *Pem*-tiedosto voidaan luoda vain kerran, ja siksi se on tärkeää laittaa turvalliseen paikkaan talteen. Yleisen tavan mukaan tiedosto voidaan tallentaa esimerkiksi käyttäjän kotihakemistosta löytyvään piilotettuun SSH-kansioon, johon SSH-yhteyttä luodessa myöhemmin viitataan.



Kuvio 14. PEM-tiedoston määrittäminen

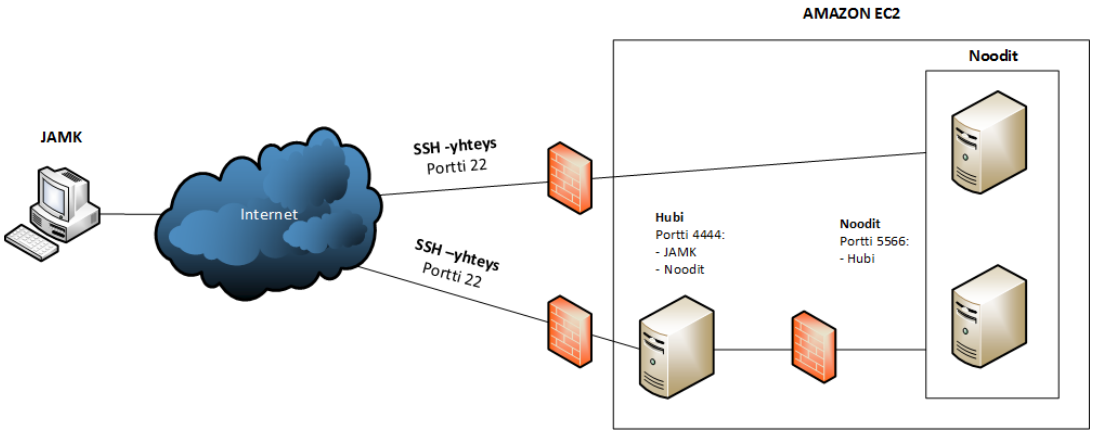
SSH -yhteys Amazon-koneeseen saadaan terminaalikomennolla:

```
ssh -i ~/.ssh/selenium-grid.pem ubuntu@<ip-address>
```

Hubi-konetta varten asetetaan Elastic IP, joka tarkoittaa kiinteää IP -osoitetta. (Ks. kuvio 15.) Tällä tavalla noodien rekisteröinti hubiin on helpompaa, koska virtuaalipalvelimien uudelleenkäynnistykset eivät aiheuta IP -osoitteiden vaihtumista.

Kuvio 15. Elastic IP

Aluksi kolmea konetta varten oli vain yksi yhteinen Security Group -konfigurointi, jolla saatiin aikaiseksi ainoastaan tietoturvaltaan huono toteutus. Palomuuri jäi melkoisen avoimeksi verkkoliikenteelle. Siksi palomuuriasetuksia varten suunniteltiin kaksi erilaista konfigurointia: hubi-koneen verkkoliikenteelle ja noodi-koneen verkkoliikenteelle. Lopullinen verkkoliikenne Amazonin ja kehittäjän tietokoneen välillä näyttää kuvion 16 mukaiselta.



Kuvio 16. Verkkoliikenne.

Instanssien hallinta Ansiblella

Asennetaan Ansible. Ensimmäiseksi asennetaan tarvittavat riippuvuudet:

```
sudo apt-get install software-properties-common
```

Lisätään ohjelmistolähde paketinhallintaan:

```
sudo apt-add-repository ppa:ansible/ansible
```

Päivitetään paketinhallinta:

```
sudo apt-get update
```

Asennetaan ansible-paketti:

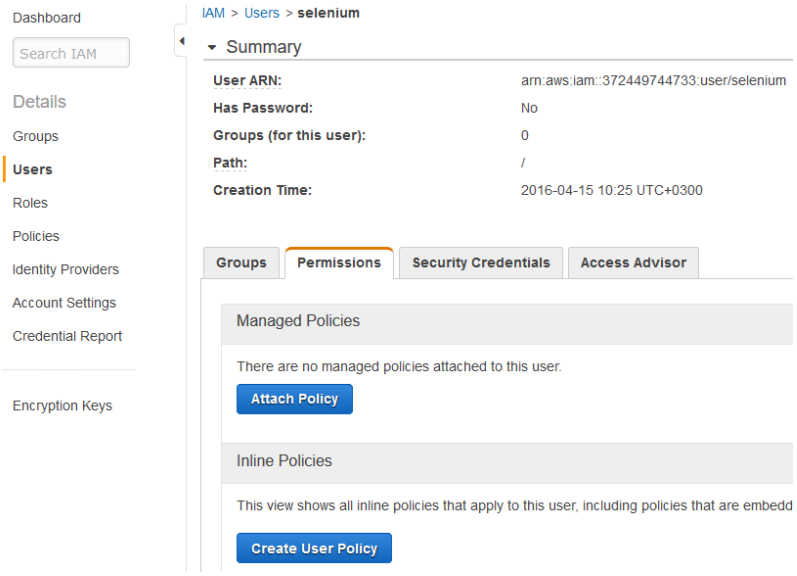
```
sudo apt-get install ansible
```

Asennuksessa tulee mukana *ec2*-moduuli, jota käytetään AWS:n kanssa. Lisäksi tarvitaan *boto*-kirjasto. *Boto* on Pythonin kirjasto, joka tarjoaa rajapinnan Amazonin pilvipalveluihin. Asennetaan *boto*:

```
sudo pip install boto
```

APIn käyttöön tarvittavat avaimet luodaan Amazon hallintapaneelin IAM-valikosta (Identity & Access Management). *Users*-sivupaneelista luodaan uusi käyttäjä ja käyttäjälle generoidaan samalla avaimet APIa varten.

Valitaan luotu käyttäjä listasta ja luodaan oikeudet *Permissions*-välilehdestä. (Ks. kuvio 17.)



Kuvio 17. Amazon IAM

Klikataan *Create User Policy* ja valitaan *Custom Policy*. Laaditaan sopivat oikeudet käyttäjälle. Syntaksin oikeellisuus voidaan tarkistaa *Validate Policy* -painikkeella. Oikeuksiin sisältyvät instanssin käynnistys, uudelleenkäynnistys, pysäytys ja lopetus. Kuviossa 18 AWS API Policy.

Kuvio 18. AWS API Policy

Lopuksi lisätään *~/.boto*-tiedostoon AWS APIn avaimet seuraavassa muodossa (ks. kuvio 19.):

```
[Credentials]
aws_access_key_id = xxxxxxxxxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Kuvio 19. AWS API -avaimet

Ansible Playbookit

Kuviossa 20 voidaan tarkastella instanssien käynnistämiseen ja sammuttamiseen luotuja *playbook*-tiedostoja. Tiedostoissa määritellään EC2-instanssit, jotka joko käynnistetään tai sammutetaan. Lisäksi määritetään Amazonin datakeskus, missä virtuaalikoneet sijaitsevat.

ec2 on Ansiblen moduuli, jolla pilvipalvelun rajapintaa voidaan käyttää.

startInstances.yml

```
- name: Start instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-7a2c12f7'
      - 'i-7b2c12f6'
      - 'i-7c2c12f1'
    region: eu-west-1
  tasks:
    - name: Start instances
      ec2:
        instance_ids: '{{ instance_ids }}'
        region: '{{ region }}'
        state: running
        wait: True
```

stopInstances.yml

```
- name: Stop instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-7a2c12f7'
      - 'i-7b2c12f6'
      - 'i-7c2c12f1'
    region: eu-west-1
  tasks:
    - name: Stop instances
      ec2:
        instance_ids: '{{ instance_ids }}'
        region: '{{ region }}'
        state: stopped
        wait: True
```

Kuvio 20. EC2-instanssien hallintaan luodut Ansible-playbookit

APIn käyttö Ansiblen kanssa aiheutti merkillisiä ongelmia. Rajapinnan *Policy* tarkistettiin toimivaksi sekä Amazonin API-simulaattorilla että *awscli*-komentorivityökalulla. Ansiblen kanssa aiheutui kuvion 21 mukainen virheilmoitus:

```
arttu@tietokone: ~/ansible
arttu@tietokone:~/ansible$ ansible-playbook startInstances.yml
 [WARNING]: provided hosts list is empty, only localhost is available


PLAY [Start sandbox instances] *************************************************

TASK [Start the sandbox instances] *********************************************
An exception occurred during task execution. To see the full traceback, use -vvv
. The error was: <Response><Errors><Error><Code>UnauthorizedOperation</Code><Mes
sage>You are not authorized to perform this operation.</Message></Error></Errors
><RequestID>86410396-0c1b-4747-8692-74ac28976d78</RequestID></Response>
fatal: [localhost]: FAILED! => {"changed": false, "failed": true, "parsed": fals
e}

NO MORE HOSTS LEFT *************************************************************
        to retry, use: --limit @startInstances.retry

PLAY RECAP *********************************************************************
localhost                  : ok=0    changed=0    unreachable=0    failed=1

arttu@tietokone:~/ansible$
```

Kuvio 21. Ansiblen virheilmoitus

UnauthorizedOperation viittaisi äkkiseltään toimimattomiin rajapinnan avaimiin. Asia kuitenkin oli tarkistettu jo aiemmin muilla työkaluilla. Myöskin *Policy* toimi moit-

teetta. Lopulta ongelman aiheuttajaksi muodostui sittenkin Ansiblen kanssa yhteensopimaton *API Policy*. Mitä ilmeisemmin Ansible tarvitsee toimiakseen enemmän valtuutuksia kuin Amazonin oma *awscli*-työkalu. *Policyyn* tehtiin tarvittavat muutokset. Ongelmia oikeuksissa aiheutti *ec2:Describe*-määritys. Jotta *Policy* toimisi myös Ansiblen kanssa, lisättiin oikeuksiin *ec2:Describe*.*

6.2 Testausautomaatiotyökalujen asennus

Robot Framework on kehitetty Python-ohjelmointikielellä, ja asennus onnistuu yksinkertaisesti käyttämällä Pythonin moduulien asentamiseen tarkoitettua *Pip*-työkalua.

Ensimmäiseksi asennetaan Pip käytettävään Python-jakeluun, joka tässä tapauksessa on 2.7. Uusin vakaa versio Robot Frameworkista tukee jo Pythonin versiota 3.x, mutta yhteensopivuusongelmia välttäen asennetaan Robot Framework vanhempaan versioon. Tällä hetkellä vain Robot Frameworkin mukana tulevat standardit kirjastot ovat yhteensopivia Python 3.x -version kanssa. Työssä käytettävässä testimateriaalissa on omia, Pythonilla ohjelmoituja ja Selenium2Library-kirjastoa laajentavia kirjastoja, ja siksi on parempi käyttää aiemmin Contriboard -palvelun testaamisessa toimivaksi todettua kokoonpanoa. Pip:llä asennetut Python -paketit löytyvät Pythonin asennuskansiosta.

Pip:n asennus tapahtuu käyttämällä Ubuntun *apt-get*-paketinhallintatyökalua:

```
sudo apt-get install python-pip
```

Tämän jälkeen asennetaan varsinainen Robot Framework:

```
pip install robotframework
```

Websovellusten testausautomaatioon tarkoitetusta Selenium-ohjelmistokehyksestä on oma kirjasto Robot Frameworkille, jolla Seleniumin toiminnallisuudet saadaan Robot Frameworkin käyttöön. Selenium2Library on Python moduuli, eli asennus tapahtuu samalla tavalla kuin edellisessä esimerkissä:

```
pip install robotframework-selenium2library
```

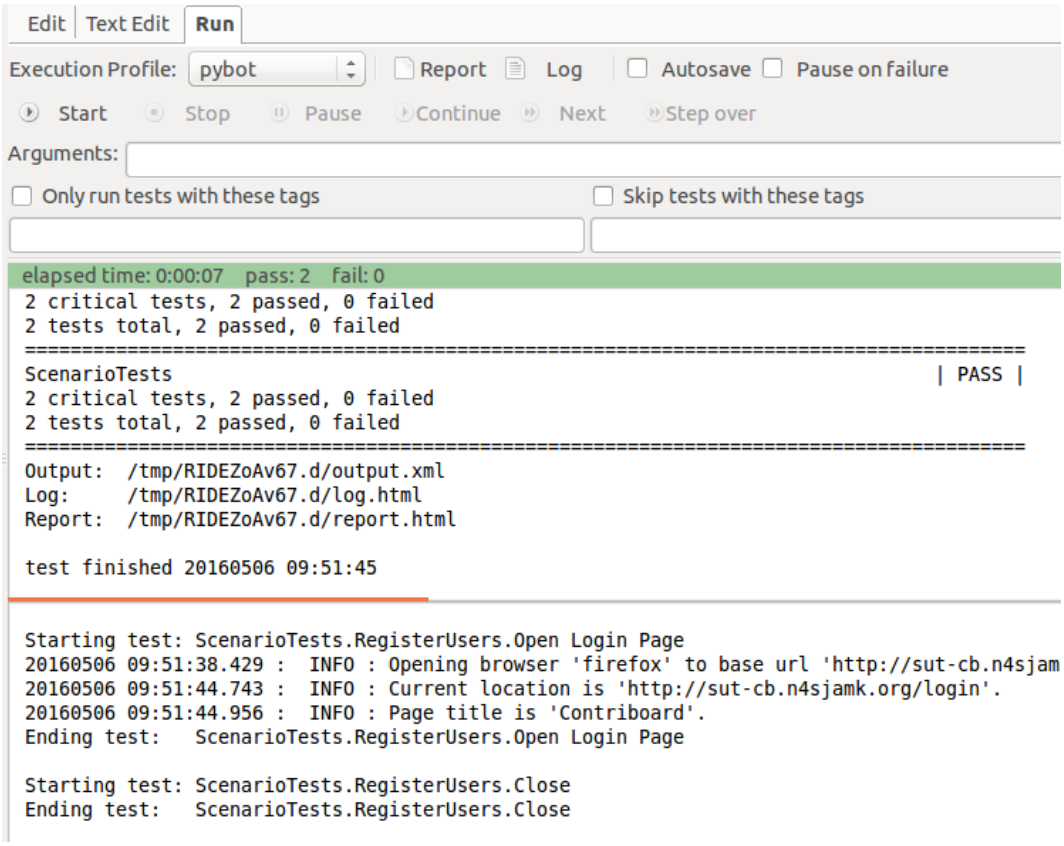
Viimeiseksi testitapausten helpompaa hallintaa varten tarvitaan RIDE-kehitysympäristö:

```
pip install robotframework-ride
```

Testimateriaalina käytetään kesällä 2015 N4S@JAMK-projektissa luotuja testitapauksia. Materiaali löytyy N4S@JAMK-projektin GitHubista, joten repo kloonataan käyttäen *git*-versionhallinnan komentoa:

```
git clone https://github.com/N4SJAMK/teamboard-test.git
```

Työkalujen toimivuus voidaan lopuksi testata ajamalla pari yksinkertaista testitapausta. (Ks. kuvio 22.)



Kuvio 22. Robot Framework -työkalun testaus

Robot Framework ja Selenium toimivat ja kaksi testiä ajettiin onnistuneesti. Testi käynnistää selaimen, menee annettuun verkko-osoitteeseen ja tarkistaa onko sivuston otsikko ”Contriboard”.

6.3 Selenium GRID

Selenium GRID tulee Selenium-palvelimen mukana. Jokainen GRIDin hubi sekä noodi tarvitsevat oman kopionsa tiedostosta. Otetaan SSH-yhteys Amazonin koneisiin:

```
ssh –i /home/arttu/.ssh/selenium-grid.pem ubuntu@<ip-osoite>.
```

Seleniumin palvelin ladattiin osoitteesta http://selenium-release.storage.googleapis.com/index.html. Ladataan tiedosto käyttämällä Linuxin *wget*-työkalua:

```
wget http://selenium-release.storage.googleapis.com/2.47/selenium-server-standalone-2.47.1.jar
```

Nykyisin GRID-komponentti sisältyy Selenium Standalone -palvelimeen eikä erillistä latausta enää tarvita. Jar-tiedoston suorittaminen vaatii toimiakseen Javan, joten ensin asennetaan *OpenJDK*. *OpenJDK*:n asennus tapahtuu jo aiemmin mainittuun tapaan Ubuntun *apt-get*-paketinhallintatyökalulla:

```
sudo apt-get install openjdk-7-jdk
```

Selenium GRID Hub

Hubi käynnistetään komennolla

```
java -jar selenium-server-standalone-2.47.1.jar role –hub
```

Selenium-palvelin ei tarvitse muita ohjelmistoja toimiakseen, joten GRID on valmis testattavaksi. Palvelin käyttää oletuksena porttia 4444 eli verkko-osoite on muotoa `http://<ip-osoite>:4444/grid/console`. Kuviossa 23 näkyy käynnistetty hubi.

Kuvio 23. Selenium Hub

Selenium GRID Node

Testien suorittamiseen vaadittava selain joudutaan käynnistämään ns. headless-tilassa. Headless tarkoittaa esimerkiksi ohjelmistoa, joka toimii ilman graafista käyttöliittymää. Siksi asennetaan ohjelmisto nimeltä *Xvfb*. Xvfb:llä pystytään ajamaan ohjelmistoja laitteistolla, jossa ei ole mahdollisuutta käyttöliittymän näyttämiseen.

Komennolla `sudo-apt-get install xvfb` asennetaan X Virtual FrameBuffer. Asennuksen jälkeen laitetaan palvelu päälle:

```
Xvfb :10 -ac
```

Komento käynnistää palvelun ”näyttöön” kymmenen. Lopuksi asetetaan käyttöjärjestelmän ympäristömuuttuja:

```
export DISPLAY=:10
```

Noodit rekisteröidään hubiin komennolla

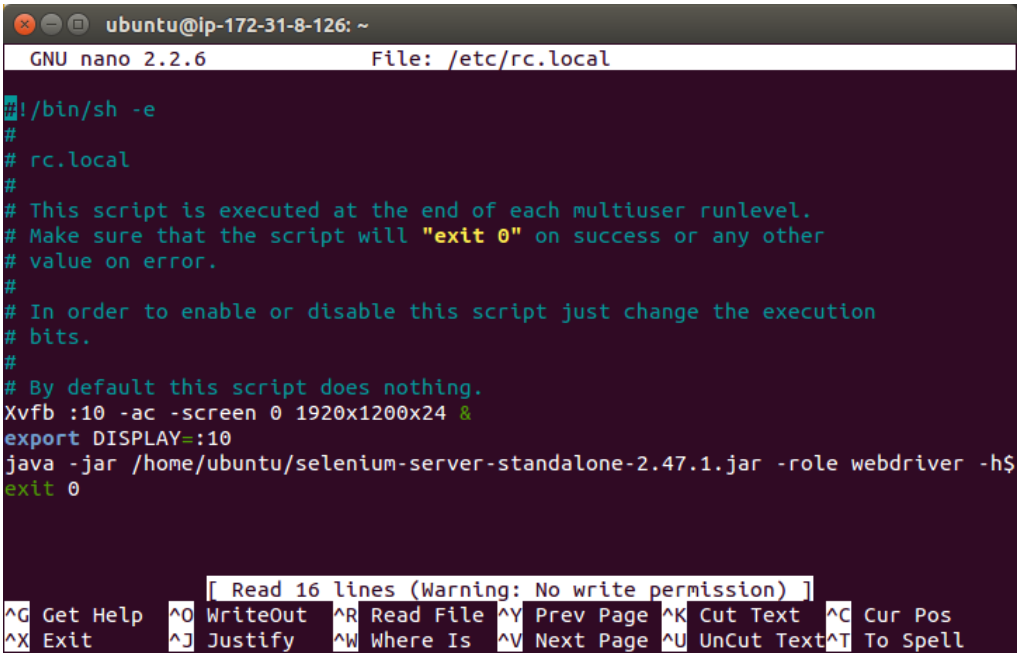
```
java -jar selenium-server-standalone-2.47.1.jar role -webdriver -hub
172.31.8.127:4444/grid/register -port 5566 -browser browserName=fire-
fox,maxInstances=2
```

Komennolla määritetään samalla kaksi kappaletta Firefox-instanssia. Jotta järjestelmän toiminnasta saataisiin järkevämpi, asetetaan tietyt palvelut käynnistymään automaattisesti virtuaalikoneiden käynnistyksen yhteydessä. Linuxissa käynnistysohjelmat voidaan hieman jakelusta riippuen laittaa eri paikkoihin. Nyt käytetään

/etc/rc.local--tiedostoa. Avataan tiedosto komentoriviltä käyttäen *nano*-tekstieditoria:

```
sudo nano /etc/rc.local
```

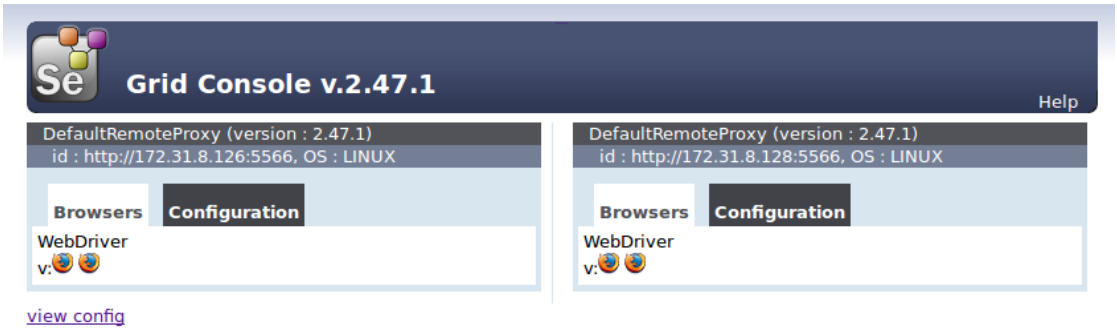
Kirjataan tiedostoon käynnistyksen yhteydessä suoritettavat kuvion 24 mukaiset komennot.



Kuvio 24. rc.local-tiedosto

Tiedostossa siis käynnistetään virtuaalinäyttö, asetetaan ympäristömuuttuja ja rekisteröidään noodi-kone hubiin.

Käynnistetään hubi ja noodit sekä tarkistetaan palvelimen toimivuus. Kuviossa 25 näkyy hubiin rekisteröidyt noodit. Käytettävissä on yhteensä 4 kpl FirefoxWebDrivereita eli parhaimmillaan neljän eri testiskenaarion yhtäaikainen ajaminen on mahdollista.



Kuvio 25. Selenium GRID käyttövalmiina

Selenium GRIDin käyttö ei aiheuta testeihin suuria muutoksia. Testien *resource*-tiedostoon vain kirjataan hubin verkko-osoite muuttujaan *$GRID*:

```
$GRID           http://<ip-osoite>:4444/wd/hub
```

Testeissä osoite annetaan aina selaimen avaamisen yhteydessä. (Ks. kuvio 26.)

```
*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LoginUrl}    ${Browser}    None    ${GRID}
```

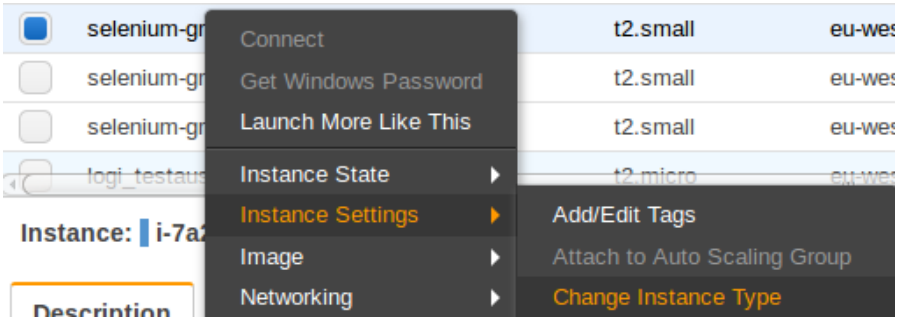
Kuvio 26. Selenium GRID määritys testitapauksissa

Selenium aiheutti runsaasti ongelmia, eikä virheilmoituksista ollut liiemmin apua. Ongelma Selenium-työkalussa ja webteknologioissa ylipäätään on niiden nopea kehittyminen. Esimerkiksi aivan uusimmalle Firefox -selaimelle tuskin löytynee tukea sillä hetkellä uusimmasta saatavilla olevasta FirefoxWebDriver-ajurista ja tuen saaminen kestää aina jonkin aikaa. Tämän lisäksi ajurinkin ominaisuudet saattavat muuttua ilman varoituksia. Tämä huomattiin käytännössä: testitapaukset, jotka kesän 2015 projektin aikana toimivat moitteetta, epäonnistuivat nyt lähes poikkeuksetta. Lopulta syyksi paljastui Seleniumin uuden version mukana tullut muuttunut hiiren klikkaukseen liittyvä algoritmi, joka esti tiettyjen testien onnistumisen. Sama ongelma todettiin myös Chrome-selaimen ja ChromeWebDriver-ajurin kanssa.

Yhteensopivuusongelmat ovat toki arkipäivää ohjelmistokehityksessä. Se, miten uusi algoritmi eroaa vanhasta, jäi epäselväksi eikä testien päivittäminen onnistunut. Kun yhteensopivuusongelmat muodostuivat lopulta ylitsepääsemättömäksi ongelmaksi, ainoaksi keinoksi jäi palaaminen muutamaa versiota vanhempaan selaimeen sekä Selenium-palvelimeen. Lopuksi myös Robot Frameworkin kanssa käytettävä *selenium2library*-kirjasto piti alentaa vanhempaan versioon.

Suunnitteluvaiheessa noodien ei ajateltu olevan kovinkaan raskaita ajettavia. Totuus kuitenkin tuli ilmi hyvinkin nopeasti. Noodi-koneen resurssit riittivät käytännössä vain yhdelle selaimen instanssille: useampi yhtäaikaisesti testejä ajava selain teki toimin-

nasta niin epävakaata, ettei testien luotettavasta suorittamisesta tullut yhtään mitään. Se, muodostuiko laitteiston ns. pullonkaula keskusmuistin puutteeseen vai heikkoon laskentatehoon jäi mysteeriksi. Seleniumin virheilmoitus kaatumistilanteessa jätti tilaa lähinnä arvailulle. Java-sovellukset ovat muistinkäytöltään usein tavallista raskaampia ja näin taisi asian laita olla tälläkin kertaa. Kaikki työssä käytettävät instanssit päätettiin korottaa seuraavan luokkaan, jonka myötä instanssien RAM -muisti kaksinkertaistui ja laskentateho lisääntyi. Päivitys tapahtui AWS:n instanssinäkymästä kuvion 27 mukaisesti.



Kuvio 27. Instanssin tyypin vaihto

6.4 Jenkins-työkalu

Jenkin asennettiin lopulta suoraan Ubuntun paketinhallinnasta. Alun perin asennus suoritettiin Dockerilla, mistä kuitenkin luovuttiin merkillisten ongelmien jälkeen. Dockerin kontit sisältävät omanlaisensa tiedostorakenteen, joka aiheutti ongelmia Robot Framework -skriptien ajamisessa. Konttiteknologiasta ei välttämättä olisi ollut kovinkaan suurta hyötyä tässä tapauksessa.

Asennus

Haetaan ohjelmistolähteen avain:

```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo
apt-key add –
```

Lisätään ohjelmistolähde:

```
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

Päivitetään paketinhallinta ja asennetaan paketti:

```
sudo apt-get update

sudo apt-get install jenkins
```

Kun Jenkinsiä käytetään isommissa projekteissa, on seuraavanlainen kokoonpano melko yleinen: yhtä tietokonetta/virtuaalikonetta käytetään pelkkää Jenkins -palvelinta varten ja sen rinnalle luodaan muita koneita suorittamaan ainoastaan tehtäviä: Tällaista erillistä, tehtäviä suorittavaa konetta kutsutaan englanniksi termillä slave, ns. orjakone. Nyt kun tarvetta on vain yhdelle jobille eli tehtävälle, voidaan käyttää pelkästään yhtä tietokonetta palvelinta sekä tehtävien suorittamista varten.

Komento `whereis jenkins.war` palauttaa seuraavaa:

```
/usr/share/jenkins/
```

Käynnistetään Jenkinsin Jetty -palvelin:

```
java -jar /usr/share/jenkins/jenkins.war
```

Käynnissä olevan Jenkins löytyy oletusasetuksien mukaisesti osoitteesta *localhost* ja portista 8080: `http://localhost:8080`.

Liitännäisten asennus

Liitännäiset on helpoin asentaa käyttöliittymän kautta menemällä Jenkinsin hallintapaneeliin, klikkaamalla *Manage plugins* ja valitsemalla *Available.* Valitaan haluttu liitännäinen ja klikataan joko *Install without restart* tai *Download now and install after restart.*

Testien raportointia varten tarvitaan Robot Framework -liitännäinen Jenkins-asennukseen. Liitännäisellä saadaan esimerkiksi testituloksien raportointi liitettyä Jenkinsin käyttöliittymään, mistä testien tuloksia voidaan tarkastella. Tehtävien ketjuttamiseen asennetaan vielä Parameterized Trigger -liitännäinen.

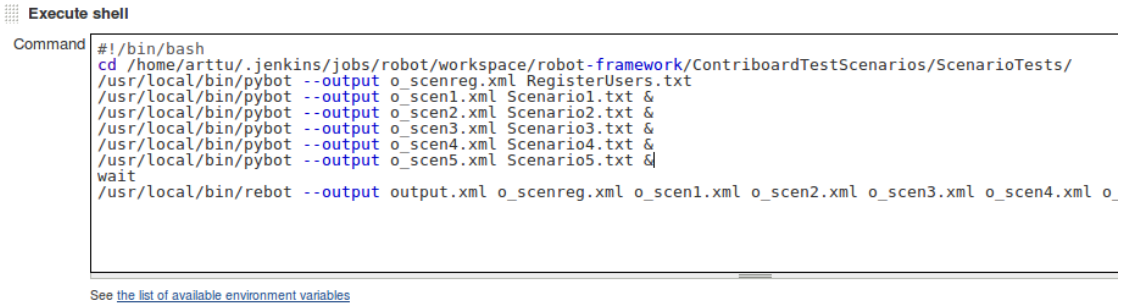
Tarvittavat tehtävät

Tehtäviä luodaan kolme kappaletta: testien ajamiseen ja EC2-instanssien käynnistämiseen sekä sammuttamiseen.

Tehtävä testien ajoa varten

Luodaan uusi tehtävä klikkaamalla etusivun *New item* -painiketta. Annetaan nimeksi *robot* ja valitaan *Build a free-style software project.*

Komennot laitetaan Execute Shell -tekstikenttään. (Ks. kuvio 28.) Ottamalla & -merkit pois komentojen lopusta testiskenaariot ajetaan yksitellen ja voidaan vertailla suorituskykyeroa. Wait –komennolla odotetaan testejä ajavan pybotin loppuun suorittamista, jotta Robot Frameworkin *rebot*–raportointityökalulla voidaan koostaa tulokset yhteen tiedostoon ja tulokset näytetään Jenkinsin käyttöliittymässä.



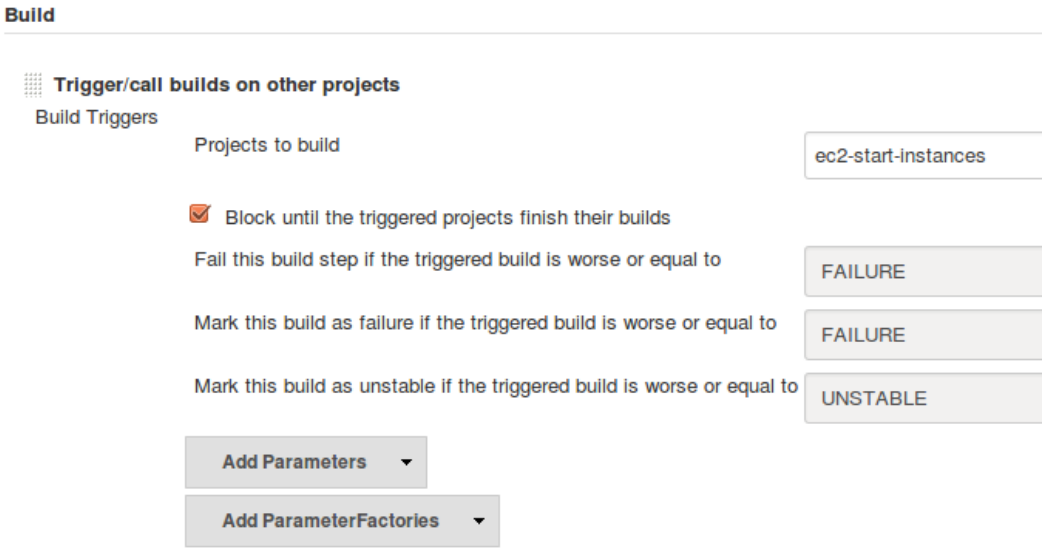
Kuvio 28. Jenkinsin robot-tehtävän komennot

Tehtävä instanssien käynnistämiseen

Ansiblella nostetaan Amazonin virtuaalikoneet ylös ennen *robot*-tehtävän ajamista. Luodaan sitä varten *ec2-stop-instances*-tehtävä. Kuviossa 29 ajetaan *startInstances.yml*-playbook.

Kuvio 29. EC2 -instanssien käynnistys

Kuviossa 30 on *robot*-tehtävä, jossa käytetään Parameterized Trigger -liitännäistä. Kuviossa estetään *robot*-tehtävän ajaminen ennen kuin *ec2-start-instances*-tehtävä on valmis.



Kuvio 30. Jenkins Parameterized Trigger –liitännäinen

Tehtävä instanssien sammuttamiseen

Jenkins Build Graph View -liitännäisellä toisiinsa liittyvät tehtävät saadaan kaavio-muotoon. Työt ketjutetaan seuraavasti:

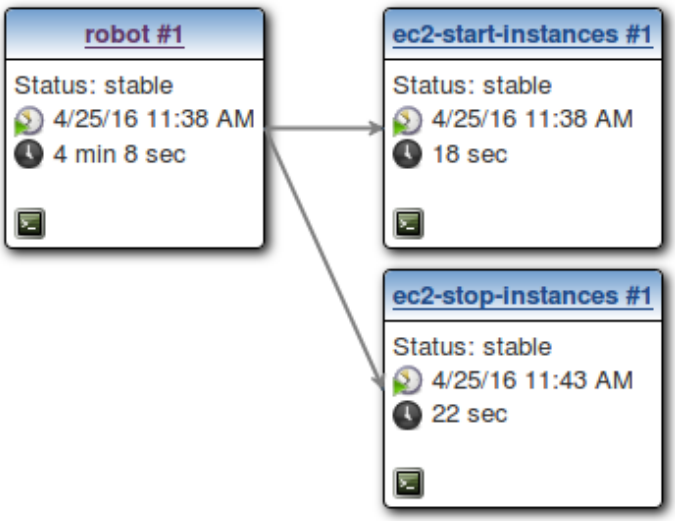
- *Robot*-tehtävä käynnistää ensin *ec2-start-instances*-tehtävän, joka laittaa EC2 -instanssit käyntiin.

- Odotetaan muutama minuutti, jotta hubi- ja noodi-koneet ovat varmasti käynnissä.

- *Robot*-tehtävässä automatisoidut testit käynnistyvät.

- Testien suorittamisen jälkeen käynnistyy *ec2-stop-instances*-tehtävä, joka pysäyttää Amazon EC2 -instanssit.

Kuviossa 31 tehtävät kaaviomuodossa.



Kuvio 31. Jenkins Build Graph

6.5 Järjestelmän testaaminen ja suorituskyky

Käynnistetään Jenkinsin ns. päätehtävä klikkaamalla *robot*-tehtävää ja valitsemalla *Build now*. Kuviossa 32 näkyy konsoli, josta tehtävien ajoa voidaan seurata. Instanssit on liipaistu käyntiin, jonka jälkeen odotetaan hetki virtuaalikoneiden käynnistymistä ja automatisoidut Robot Framework -testit käynnistyvät.

Console Output

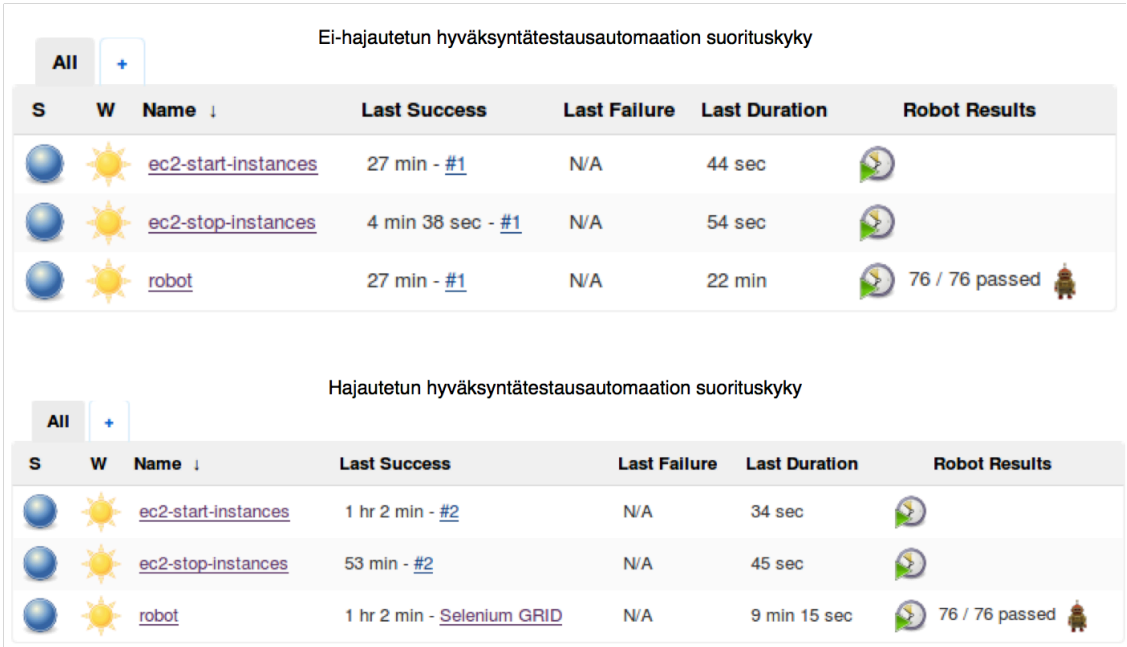
```
Started by user admin
Building in workspace /home/arttu/.jenkins/jobs/robot/workspace
Waiting for the completion of ec2-start-instances
ec2-start-instances #1 completed. Result was SUCCESS
Build step 'Trigger/call builds on other projects' changed build result to SUCCESS
[workspace] $ /bin/sh -xe /tmp/hudson1577537774816298066.sh
+ sleep 2m
[workspace] $ /bin/bash /tmp/hudson8103676191046307933.sh
==============================================================================
RegisterUsers
==============================================================================
Open Login Page                                                       | PASS |
------------------------------------------------------------------------------
Register Jenny                                                        | PASS |
------------------------------------------------------------------------------
Register Evelyn                                                       | PASS |
------------------------------------------------------------------------------
Register James                                                        | PASS |
------------------------------------------------------------------------------
Register Robot                                                        | PASS |
------------------------------------------------------------------------------
Register John                                                         | PASS |
------------------------------------------------------------------------------
Close                                                                 | PASS |
------------------------------------------------------------------------------
RegisterUsers                                                         | PASS |
7 critical tests, 7 passed, 0 failed
7 tests total, 7 passed, 0 failed
```

Kuvio 32. Jenkins-tehtävät toiminnassa

Kuviossa 33 testejä ajetaan Selenium GRIDissä ja kaikki noodit sekä niiden selaimet ovat käytössä. Kuvion alaosassa näkyy ilmoitus, että osa testiskenaarioista on jonossa odottamassa vapaata paikkaa.

Kuvio 33. Selenium GRID toiminnassa

Testaamisen jälkeen päivä paistaa Jenkinsin tehtävänäkymässä. Kuviossa 34 tulokset kun testejä ei hajauteta useammalle järjestelmälle sekä hajautetusti ajettuna.



Kuvio 34. Suorituskykyvertailua

Näinkin pienillä testimäärillä ero on huomattava. Jenkinsin Robot Framework -liitännäinen ei saanut pelkkään testien ajamiseen kulunutta aikaa selville ja siksi kaaviotkin jäivät piirtämättä. Kuviossa 34 näkyvä kokonaisaika on koko Jenkinsin tehtäväketjun suorittamiseen kulunut aika. Ero testien ajamisessa saatiin silti selville. Kuitenkin ajetut testit ja niiden läpimenon liitännäinen näyttää oikein.

7 Tulokset

Lopputulos oli toimiva järjestelmä, joka toteutti alussa sille asetetut tavoitteet. Testausautomaatio ja virheiden raportointi tehostuivat. Uutta ja ei-hajautettua järjestelmää testattiin samoilla testitapauksilla ja tulokset olivat selkeitä. Uusi järjestelmä ajoi testitapaukset läpi yli 50% nopeammin. Ei-hajautettua järjestelmää "simuloitiin" ajamalla testiskenaariot yksi toisensa jälkeen yhtä Amazonin virtuaalikonetta ja Firefox-selainta vasten.

Jos Amazonista otettaisiin vielä muutama instanssi lisää käyttöön, olisi testaaminen entistäkin tehokkaampaa. Nyt osa testiskenaarioista jäi jonoon, mikä aavistuksen hidasti prosessia, mutta ominaisuus ainakin todettiin toimivaksi. On myös huomattava, että testitapauksien määrä on alhainen: normaalisti hyväksyntätestauksessa niitä on satoja. Hyöty kuitenkin näkyy näinkin pienillä testimäärillä.

Kaiken kaikkiaan työ oli erittäin opettavainen. Järjestelmän suunnittelu vei aikaa ja myös tietämys joistakin aihealueista oli alkuun melko olematonta, näistä mainittakoon esimerkiksi jatkuva integraatio. Uusien aiheiden opiskelu vei aikaa ja siksi työn alussa eteneminen oli melko hidasta. Toteutuksen suhteet kädet olivat vapaat ja eri ratkaisuvaihtoehtoja sai miettiä, joka oli positiivista.

8 Pohdinta

Suurinta päänvaivaa projektin aikana aiheuttivat tilanteet, joissa järjestelmän eri komponentteja piti yhdistää yhdeksi toimivaksi kokonaisuudeksi. Tietotaito ei välttämättä heti riittänyt ongelmien ratkaisuun ja taas sai opiskella. Toisaalta nämä haasteet olivat myös erittäin mielenkiintoisia ja opettavaisia. Työn aikana vastaan tuli onnistumia ja vastoinkäymisiä, mutta lopputulokseen voi olla tyytyväinen.

Toteutusvaiheen aikana tuli uusia ideoita, joista osa saatiin mukaan. Jatkokehitykseksi jäi uuden järjestelmän siirtäminen vanhan tilalle ja Jenkinsin Robot Framework -liitännäisen raportoinnin parantaminen, jotka eivät aikarajan puitteissa olleet enää mahdollisia toteuttaa. Selenium GRID ei aiheuttanut testeihin isoja muutoksia ja siksi uuden testausjärjestelmän siirtäminen vanhan tilalle ei pitäisi olla kovinkaan aikaa vievä työ. Toki Jenkinsin tehtäviin tulisi muutoksia, koska esimerkiksi

ennen testien ajamista testattavan järjestelmän koostaminen liipaistaisiin käyntiin versionhallintaan tulevista muutoksista. Tässä mielessä toteutus jäi vielä jokseenkin prototyypin asteelle, mutta toimivaksi sellaiseksi. Tulosten perusteella GRID huomattiin päteväksi vaihtoehdoksi, kun testien ajamista halutaan hajauttaa. Tosin täysin ongelmaton ei myöskään GRID ole. Työkalua moititaan epävakaaksi, joka tuli itsekin huomattua toteutusta tehdessä. Järjestelmä vaati välillä uudelleen käynnistelyä, jotta testien ajo saattoi jatkua kunnolla.

Testausautomaatiotyökalujen -ja järjestelmien rakentaminen vaatii yrityksiltä aina runsaasti resursseja. Lisäksi testitapauksiin joudutaan kiinnittämään lisähuomiota, jos testien rinnakkain ajaminen on tavoitteena. Testitapaukset on suunniteltava siten, ettei testidatan kanssa tule ongelmia. Contriboard-palvelun testiskenaariot pystyttiin ajamaan rinnakkain ilman muutoksia, vaikkei niitä alun perin oltukaan suunniteltu rinnakkaisajo tavoitteena.

Mikäli testitapauksia olisi ollut esimerkiksi puolisen tuhatta kappaletta, oltaisiin saatu tarkempaa tietoa järjestelmän rakentamisen kokonaiskustannuksista - esimerkiksi parempi arvio pilvipalvelujen käytön kuukausittaisista kustannuksista. Virtuaalikoneiden määrää jouduttaisiin nostamaan huomattavasti, joka taas nostaisi kustannuksia nopeasti. Lopullisessa tuotantoon menevässä järjestelmässä kustannukset ja järjestelmän tehokkuus haettaisiin sopivalle tasolle kokeilun kautta.

Lähteet

About AWS. N.d. Palvelun sivusto. Viitattu 1.5.2016. https://aws.amazon.com/about-aws/

Amazon EC2 - Virtual Server Hosting. N.d. Palvelun sivusto. Viitattu 1.5.2016. https://aws.amazon.com/ec2/

Contriboard Fact Sheet. 2015. Contriboard-palvelun wiki. Viitattu 17.3.2016. https://github.com/N4SJAMK/challenge_factory_2015/wiki/Contriboard-Fact-Sheet

Cloud computing service models, Part 1: Infrastructure as a Service. 2011. IBM developerWorks. Viitattu 23.3.2016. http://www.ibm.com/developerworks/cloud/library/cl-cloudservices1iaas/

Cloud computing service models, Part 2: Platform as a Service. 2011. IBM developerWorks. Viitattu 23.3.2016. http://www.ibm.com/developerworks/cloud/library/cl-cloudservices2paas/index.html

Cloud computing service models, Part 3: Software as a Service. 2011. IBM developerWorks. Viitattu 23.3.2016. http://www.ibm.com/developerworks/cloud/library/cl-cloudservices3saas/

Heino, P. 2010. Pilvipalvelut. Hämeenlinna: Talentum.

How Ansible Works. N.d. Työkalun verkkosivusto. Viitattu 20.5.2016. https://www.ansible.com/how-ansible-works

Intro to Playbooks. 2016. Ansible Documentation. Viitattu 3.5.2016. http://docs.ansible.com/ansible/playbooks_intro.html

Kasurinen, J-P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Lewis, William E. 2009. Software Testing and Continuos Quality Improvement. Boca Raton, FL: Taylor & Francis Group.

Meet Jenkins. N.d. Jenkins-työkalun wiki. Viitattu 25.1.2016. https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Microsoft Azure. N.d. Palvelun sivusto. Viitattu 28.4.2016. https://azure.microsoft.com/en-us/overview/what-is-azure/

Myers, G. J., Badgett, T. & Sandler, C. 2012. The Art of Software Testing. Hoboken, New Jersey: John Wiley & Sons, Inc.

Regression Testing. N.d. Microsoft Developer Network. Viitattu 14.2.2016 https://msdn.microsoft.com/en-us/library/aa292167%28v=vs.71%29.aspx

Robot Framework. N.d. Robot Framework -työkalun verkkosivusto. Viitattu 26.1.2016. http://robotframework.org/#introduction

Rosenberg, J. & Mateos, A. 2011. The Cloud at Your Service. Greenwich, CT: Manning Publications Co.

Selenium. N.d. Wikipedia - vapaa tietosanakirja. Viitattu 11.2.2016
https://en.wikipedia.org/wiki/Selenium_%28software%29

Selenium GRID. N.d. Selenium GRID documentation. Viitattu 26.1.2016.
http://www.seleniumhq.org/docs/07_selenium_grid.jsp

Selenium WebDriver. N.d. Selenium WebDriver documentation. Viitattu 30.3.2016.
http://docs.seleniumhq.org/docs/03_webdriver.jsp

State of the Cloud Report. 2016. Viitattu 2.5.2016.
http://assets.rightscale.com/uploads/pdfs/RightScale-2016-State-of-the-Cloud-Report.pdf

Stress Testing. 2007. Microsoft Developer Network. Viitattu 15.2.2016
https://msdn.microsoft.com/en-us/library/bb924374.aspx

Test automation. n.d. Wikipedia – vapaa tietosanakirja. Viitattu 4.5.2016.
https://en.wikipedia.org/wiki/Test_automation

Velte A.T., Velte, T. J. & Elsenpeter, R. 2010. Cloud Computing: A Practical Approach. USA: The McGraw-Hill Companies.

What Is Cloud Computing. N.d. AWS-palvelun sivusto. Viitattu 3.3.2016.
https://aws.amazon.com/what-is-cloud-computing/

xUnit. N.d. Wikipedia - vapaa tietosanakirja. Viitattu 5.5.2016.
https://en.wikipedia.org/wiki/XUnit

YAML. 2016. Ansible Documentation. Viitattu 3.5.2016.
http://docs.ansible.com/ansible/YAMLSyntax.html

Liitteet

Liite 1. Robot Framework -testiskenaario

```
*** Settings ***
Resource          resource.txt

*** Test Cases ***
John Opens Computer And Logins
    [Tags]    sc6    log6
    Open Browser To Login Page
    Login User    john@test.com    johnjohn

John Creates and Opens Board
    [Tags]    sc6
    Create Board
    Open Board    2    2

John Uses Help
    [Tags]    sc6
    Open Help
    Change Help Slides
    Close Help
    [Tags]    sc6


John Changes Board Name
    [Tags]    sc6
    Click Edit Board From Board
    Input Board Name    Example Project
    Click Done Board Edit
    [Tags]    sc6

John Uses Markdown Text in Ticket and Adds Comment
    [Tags]    sc6
    Create Ticket with Markdown Text and Comment    -450    -250    TO-DO

John Exports Board as Image
    [Tags]    sc6
    Export Board Image

John Wants to Know More About the Product
    [Tags]    sc6
    Click About Button

John Changes Password
    [Tags]    sc6
    Change Password    johnjohn    johnjohn2
    Change Password    johnjohn2    johnjohn

John Set Username
    [Tags]    sc6
    Set Username    John

John Checks Board Members
    [Tags]    sc6
    Check Board Members

John Reviews Ticket
    [Tags]    sc6
    Create Ticket    1    1
    Review Tickets

John Shares Board
    [Tags]    sc6
    Share Board From Board

Close
```

```
[Tags]    sc6
Log Out
Close Browser
[Teardown]
```

Liite 2. Robot Framework -raporttiesimerkki

RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 Test Report

Generated
20160509 15:06:57 GMT +03:00
20 hours 55 minutes ago

Summary Information

Status:	All tests passed
Elapsed Time:	00:19:52.298
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	76	76	0	00:19:51	
All Tests	76	76	0	00:19:51	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
log2	2	2	0	00:00:09	
log3	1	1	0	00:00:08	
log4	2	2	0	00:00:08	
log5	1	1	0	00:00:08	
reg	7	7	0	00:01:11	
sc1	19	19	0	00:03:16	
sc2	9	9	0	00:03:33	
sc3	15	15	0	00:04:50	
sc4	12	12	0	00:04:56	
sc5	13	13	0	00:01:41	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5	76	76	0	00:19:52	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . RegisterUsers	7	7	0	00:01:11	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . Scenario1	20	20	0	00:03:40	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . Scenario2	9	9	0	00:03:34	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . Scenario3	15	15	0	00:04:50	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . Scenario4	12	12	0	00:04:56	
RegisterUsers & Scenario1 & Scenario2 & Scenario3 & Scenario4 & Scenario5 . Scenario5	13	13	0	00:01:42	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests