HAAGA-HELIA
ammattikorkeakoulu

Jatkuva integraatio pilvipalveluissa

Panu Salmi

Opinnäytetyö

Tietotekniikan koulutusohjelma

2013

HAAGA-HELIA
University of Applied Sciences

Tiivistelmä

2013

Tietotekniikan koulutusohjelma

Tekijä tai tekijät Panu Salmi	**Ryhmätunnus tai aloitusvuosi** 2009
Raportin nimi Jatkuva integraatio pilvipalveluissa	**Sivu- ja liitesivumäärä** 38 + 11
Opettajat tai ohjaajat Sauli Isonikkilä	

Opinnäytetyössä tavoitteena oli selvittää mahdollisuudet jatkuvan integraation käyttöönottoon pilvipalveluissa. Selvitys kohdistui pilvipalveluiden ominaisuuksien selvittämiseen ja palveluiden käyttöönoton ongelmien ratkontaan.

Selvityksen tutkimusmenetelmäksi valittiin laadullinen tutkimus ja testaukseen valittiin kolme pilvipalvelua. Valinnoissa pyrittiin mahdollisimman erilaisiin ratkaisuihin, jotta tulokset olisivat kattavia. Testaukseen valittiin CircleCi, Cloudbees ja Travis-ci jatkuva integraatio palvelut.

Testausta varten rakennettiin testiohjelmisto ja laadittiin testaussuunnitelma. Testiohjelmisto oli Python-ohjelmointikieleen perustuva verkkosovellus, joka toimii Googlen App Engine pilvipalvelussa ja ohjelmakoodin versionhallintaan valittiin Github internetpalvelu. Testaus toteutettiin samoilla käyttötapauksilla kaikille testauksessa olleille pilvipalveluille, jotta tulokset olisivat vertailukelpoisia.

Tutkimuksen tuloksena kaikkien testattavana olleiden pilvipalveluiden käyttöönotot onnistuivat. Tuloksissa erityisesti CircleCi ja Travis-ci palveluiden ratkaisut olivat käyttäjän kannalta selkeitä ja lähes ongelmittomia ratkaisuja. Cloudbees palvelussa esiintyvät ongelmakohdat olivat selkeästi monimutkaisempia, mutta lopputulos vastasi odotuksia ja näkemys palvelusta jäi positiiviseksi. Tuloksista ei noussut esille selkeästi parasta vaihtoehtoa ja jokaisella palvelulla oli omat selkeät etunsa ja haittansa.

Jatkuvan integraation käyttöönotto kaikilla testauksessa olevilla palveluilla oli suhteellisen selkeää. Jatkuvan integraation tuomaa ohjelmiston laadunvalvontaa voi suositella minkä tahansa yrityksen ohjelmistoprojektin käyttöön. Pilvipalveluiden tuomien etujen myötä palveluiden käyttöönoton kustannukset ovat minimaaliset ja sopimuksen voi irtisanoa helposti.

Asiasanat
jatkuva integraatio, pilvipalvelut, testaus

HAAGA-HELIA
University of Applied Sciences

Abstract

24.4.2013

Tietotekniikan koulutusohjelma

Authors Panu Salmi	**Group or year of entry** 2009
The title of thesis Continuous Integration in cloud computing	**Number of pages and appendices** 38 + 11
Supervisor(s) Sauli Isonikkilä	

The purpose for this thesis was to investigate the possibilities of using continuous integration through cloud services. The report focused on determining the service properties, usability and solving any problems encountered during deployment.

The research method was qualitative research and three cloud services were selected for testing. The aim was to choose distinctively different solutions in order to receive comprehensive results. CircleCi, Cloudbees and Travis-Ci continuous integration services were selected.

For testing purposes, test software and a test plan were created. The test software was made using Python programming language and it functioned as an internet service through Google App Engine cloud platform. For revision control, Github internet service was chosen. Testing was carried out using the exact same use cases for each tested cloud service in order to receive comparable results.

The results of the thesis showed that all the tests were successful. In particular CircleCi and Travis-ci offered their users easily deployable solutions. Cloudbees service had much more complex problems, but the end result corresponded to expectations and thus it still held a positive view in the results. In the results, there was no indication towards a clearly defined best option and each service had its own clear advantages and disadvantages.

The thesis concludes that the deployment of continuous integration systems was relatively clear regarding all the tested services. The usage of automated quality control for software projects is highly recommended. The initial costs of deployment are minimal and the agreement may be terminated at the convenience of the user.

Key words
Continuous integration, cloud computing, testing

Sisällys

1 Johdanto

Opinnäytetyö perustuu yritystaholta esille tulleeseen pyyntöön tehdä tutkimus mahdollisuuksista ottaa jatkuva integraatio palvelu yrityksen ohjelmistoprojektien käyttöön. Erityisesti toiveena oli selvittää millaisia erilaisia palveluita on saatavilla ja mikä näistä olisi sopiva yrityksen omiin tarpeisiin. Yritys ei ole nykyisellään toiminnassa, mutta aiheen voi olettaa olevan ajankohtainen varmasti muillekin.

Tutkimus tullaan toteuttamaan laadullisen tutkimuksen periaattein tutkimalla eri pilvipalveluiden ominaisuuksia, käytettävyyttä sekä palveluista syntyviä kustannuksia. Testaamisen ohella pyritään palveluista löytämään samanlaisia tekijöitä ja vertailemaan näitä. Taustalla on pyrkimys selvittää ovatko nykypäiväiset pilvipalvelut tarpeeksi helppokäyttöisiä ja edullisia, jotta jatkuvan integraation käyttöönottamiselle yrityksen ohjelmistoprojektissa ei voi sanoa olevan hankaluudesta tai kuluista koituvia esteitä ja selvittämään millaisia palvelut käyttäjilleen ovat.

Tutkimus toteutetaan testaussuunnitelman mukaisesti ja sen valmisteluiksi toteutetaan myös testiohjelmisto. Suunnitelmassa palveluiden käyttäminen jaetaan omiin käyttötapauksiinsa ja tehdään oletus palvelun toiminnasta kunkin käyttötapauksen kohdalla. Käyttötapaukseen sisältyy aina yksi järjestelmän kokonaistoimintaan oleellinen ominaisuus, joka on tarpeellista saada järjestelmässä toimintaan ennen kuin seuraavaan tapaukseen voi siirtyä. Tutkimuksen lopuksi pyritään näitä tuloksia vertailemalla tehdä johtopäätöksiä palveluiden oleellisista eroista ja ratkaisuista.

Opinnäytetyön osiossa kaksi selvitetään tutkimuksen taustoja ja tehdään hypoteesi tutkimuksen tuloksesta. Osiossa valotetaan aiheeseen tehtyjä aiempia tutkimuksia ja valitaan aihealueeseen liittyviä tietolähteitä sekä perustellaan lähteiden valinnan syitä. Osio antaa tutkimuksen tekemiselle selkeän selkärangan ja pohjaa tutkimuksen tulosten pohdinnalle.

Osiossa kolme käydään läpi tutkimuksen toteutuksen kannalta oleelliset menetelmät ja viitekehys. Osioissa määritellään jatkuvan integraation sekä pilvipalveluiden yleisiä ominaisuuksia, toimintamalleja ja termistöä. Aihealueiden käsittely ei vielä ota kantaa

testauksen tuloksiin ja keskittyy aihealueeseen kuuluviin oleellisiin tietoihin ja käytäntöihin. Tarkoituksena on selkeyttää myöhemmin tutkimuksessa esiin tulevia termejä sekä vahvistaa käsityksiä palveluiden mahdollisuuksista ja teknisestä pohjasta. Osiossa keskitytään selvittämään tarkemmin mistä jatkuvassa integraatiossa ja pilvipalveluissa on kyse ja mitä ominaisuuksia näihin oleellisesti kuuluu. Järjestelmien toimintoja sekä ominaisuuksia esitetään pääasiallisesti hyötyjen kannalta. Tietoja esitetään myös yksikkötestaamisesta sekä versionhallinnan käytännöistä.

Osiossa neljä käydään läpi itse tutkimus. Testaukseen käytetty ohjelmisto käydään tarkemmin läpi ja kuvaillaan tärkeimpiä testaukseen liittyä tietoja. Osiossa jokainen testattava palvelu käydään seikkaperäisesti läpi ja varmistetaan, että palveluiden käyttötapaukset ovat tarvittaessa toistettavissa. Tutkimuksesa käydään läpi kuinka palvelu asennetaan ja konfiguroidaan, ohjelmiston testit ajetaan sekä suoritetaan ohjelmakoodin käyttöönotto. Osio sisältää myös tietoa valmisteluista käytännön työn toteutukselle. Varsinainen testaussuunnitelma, joihin tehdyt testit perustuvat on opinnäytetyön mukana liitteenä yksi.

Tutkimuksen tulokset palveluiden käyttöönotosta koostetaan osiossa viisi. Palvelut käydään tuloksissa läpi analysoiden eri vaiheiden mutkikkuutta ja esille tulleita ongelmia sekä niiden ratkaisuja. Tuloksissa tehdään myös vertailua palveluiden välillä ja käydään läpi esiin tulleita oleellisia eroavaisuuksia. Arviointia suoritetaan vielä myös aiemmin tehtyä hypoteesia vasten ja selvitetään miltä osin alkuperäinen olettamus on toteutunut.

Viimeisessä osiossa käydään läpi loppupäätelmät ja tehdään suosituksia jatkotoimien kannalta. Osiossa käydään läpi myös muita tutkimuksen aikana esiin tulleita ajatuksia ja havaintoja.

2 Tutkimuksen lähtökohdat

2.1 Jatkuvan integraation tutkimisen taustaa

Opinnäytetyö lähtee selvittämään millaisia erilaisia pilvipalveluita on nykyisin tarjolla jatkuvan integraation käyttöönottoa varten. Aihe on ajankohtainen jatkuvasti monimutkaistuvien tietokonejärjestelmien maailmassa, joissa valmistuneiksi luokiteltujen ohjelmistojen käyttöönotot tuntuvat helposti venyvän kuukausien mittaisiksi projekteiksi ja laadullisen varmistuksen automaatiosta tuskin voi ajatella olevan oleellisesti haittaakaan.

Jatkuvasta integraatiosta löytyy useita tutkimuksia ja kirjoitelmia. Tehdyt teokset useimmiten liittyvät oman jatkuva integraatio palvelimen käyttöönottoon tai jatkuvan integraation ominaisuuksien määrittelyyn. Samoin pilvipalveluiden puolelta löytyy paljon yleistäviä teoksia, jotka keskittyvät määrittelemään pilvipalveluiden ominaisuuksia ja teknistä rakennetta. Opinnäytetyön kannalta on oleellista yhdentää nämä kaksi toteutusta ja tarkastella niiden yhdessä tarjoamia ominaisuuksia ja mahdollisuuksia erityisesti palveluiden käyttäjän näkökannalta.

Jatkuvaa integraatiota varten on pääasialliseksi lähteeksi valittu kirja Continuous Integration Improving Software Quality and Reducing Risk, jonka on kirjoittanut Paul M. Duvall yhdessä Steve Matyaksen ja Andrew Gloverin kanssa. Kirja on vuodelta 2007 ja on niin muodoin jo hiukan liian vanha nykyään nopeasti kehittyvässä tietotekniikan maailmassa. Kirja kuitenkin käsittelee peruskäsitteet riittävällä syvyydellä, jotta siitä saa näkemyksen millaisiin ongelmiin jatkuva integraatio järjestelmillä pyritään löytämään ratkaisu ennen kuin ongelmaa edes on. Kirjan lisäksi lähteenä toimii verkkodokumentti Martin Fowlerilta.

Pilvipalvelut ovat tutkimuksen ytimessä monella tapaa ja on hyvä käydä läpi niiden rakennetta ja ominaisuuksia. Amerikkalainen NIST (National Institute of Standards and Technology) on julkaissut vuonna 2011 yleisen määritelmän pilvipalveluille ja tämä julkaisu toimii hyvänä pohjana pilvipalveluiden yleisimpien piirteiden kuvailuun. Julkaisun lisäksi lähteinä pilvipalveluille on kaksi tutkimusta vuosilta 2008 ja 2009.

Tutkimuksissa käydään läpi pilvipalveluiden palvelurakennetta ja eroja aiempiin ratkaisuihin nähden.

2.2 Jatkuva integraatio pilvipalveluissa tutkimuskohteena

Pilvipalveluita on tarjolla useita erilaisia ja niiden toteutukset vaihtelevat oleellisesti toisistaan. Palveluntarjoajan kannalta on monia erilaisia vaihtoehtoja miten järjestelmänsä voi asiakkaalleen toteuttaa. Järjestelmän ominaisuuksien toteuttamisen kautta luodaan pohja hinnoittelulle lisäämällä palvelun arvoa kuluttajalle. Kuluttajalle ongelmaksi muodostuu epävarmuus palveluiden soveltuvuudesta omiin tarpeisiinsa ja miten hankalaksi käyttöönotto muodostuu. Kuluttajalle ei ole mielekästä, että järjestelmän käyttöönotto kestää useita päiviä, jos se ei lopuksi vastaa odotuksia.

Tutkimuksen aihe on ajankohtainen hyvin moneltakin kannalta. Pilvipalveluja on nykyään saatavilla valtavissa määrin ja pieniä yrityksiä ilmestyy koko ajan enemmän hyödyntämään mahdollisuuksia uudella kentällä. Nämä yritykset hyötyvät valtavasti pilvipalveluiden tuomista rakenteellisista eduista, jotka eliminoivat tarvetta isoille investoinneille ennen kuin mitään varteenotettavaa voisi edes toteuttaa. Yritykset voivat keskittyä kehittämään asiakkailleen mahdollisimman hyvin toteutetun ja helppokäyttöisen palvelun ilman tarvetta keskittyä palvelun rakenteellisiin ongelmiin. Tämä kehitys lupailee hyvää myös kuluttajan kannalta ja tulevaisuudessa vastaavat palvelut tulevat todennäköisesti vain entistä kilpailukykyisemmiksi ja helpommiksi ottaa käyttöön.

2.3 Tutkimuksen tavoitteet

Tutkimuksen aikana tehdään testiohjelmisto, joka tuottaa Python-ohjelmistokoodilla tuotetun web-sovelluksen Googlen App Engine alustalle. Ohjelmistokoodia säilytetään niin ikään internetissä toimivalla versiohallintaohjelmistolla, joka yhdistetään testattavaksi valittuihin jatkuva integraatio järjestelmiin testituloksien hankkimisen ajaksi. Palveluiden käyttöönotoista halutaan todenmukainen kuva ja tämän saavuttamisessa hyödynnetään normaalia monimutkaisempaa ohjelmistoratkaisua testisovelluksessa. Testisovelluksen rakennetta kuvaillaan tarkemmin osiossa 4.1.

Opinnäytetyön tulokseksi oletetaan saavutettavan kevyt ja nopeasti rakennettu jatkuva integraatio järjestelmä kaikilla valituilla pilvipalveluilla. Järjestelmän tulisi tukea ohjelmistoprojektin laadun varmistamisessa ilman suuria kustannuksia, raskaita rakenteita tai vaikeaa käyttöönottoa. On todennäköistä, että pienikin ohjelmistoprojekti hyötyisi pilvipalvelun kautta saavutettavista eduista jatkuvan integraation. käyttöönoton oletetaan olevan helppoa ja nopeaa. Opinnäytetyön lopputuloksena syntyy opastus vertailun tuloksiin, jatkuvan integraation käyttöönottoon ja pilvipalveluiden hyödyntämiseen nykyaikaisessa ohjelmistoprojektissa. Tuloksissa selvennetään myös voisiko lähes mikä tahansa ohjelmistoprojekti hyötyä mahdollisuudesta käyttää jatkuvan integraation periaatteita.

Tutkimuksessa rajauksena toimii testaussuunnitelman rakenne ja testaukseen valitut pilvipalvelut. Testaukseen käytetään ainoastaan yleisesti tarjolla olevia pilvipalveluita. Palveluiksi valittiin CircleCi, Cloudbees ja Travis-ci pilvipalvelut. Jatkuvaa integraatiota ei toteuteta omalle palvelimelleen, koska tämä on jo useasti tutkittu. Oman palvelimen ylläpitäminen useille erilaisille ohjelmistoprojekteille vaatisi myös rakenteellista panostusta järjestelmiin palvelimen hankinnan takia ja suurempaa ylläpitotyötä projektille jatkossa. Pilvipalvelut mahdollistavat palveluiden kääntämisen pois päältä tarvittaessa, jos sitä ei koeta tarpeelliseksi jatkaa.

3 Tutkimusmenetelmät ja viitekehys

3.1 Tutkimusmenetelmät

Tutkimus toteutetaan laadullisena tutkimuksena. Laadullisen tutkimuksen menetelmät antavat kohdennuksen yksittäisiin valittuihin palveluihin. Laadullisella tutkimuksella pyritään vastaamaan kuluttajan kannalta tärkeisiin kysymyksiin palveluiden sisällöstä. Palveluiden takana toimii erittäin monimutkaisia prosesseja ja tutkimusmenetelmän pohjalta lähdetään testauksen ohella myös selventämään mikä yleisesti vastaavissa palveluissa on mahdollista.

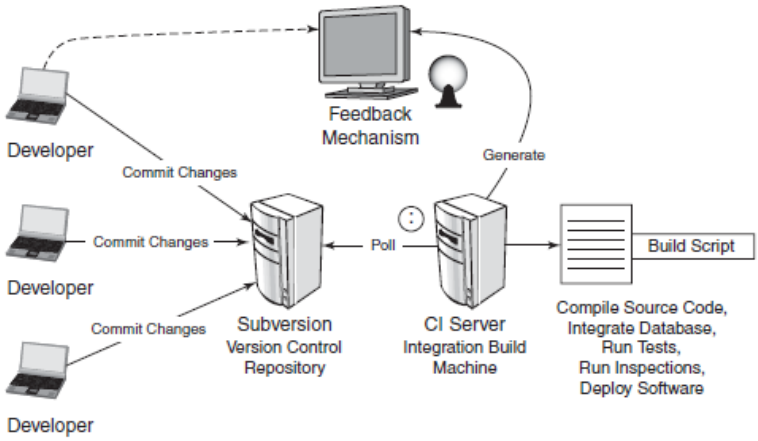
Yksittäiset testaukset suoritetaan mahdollisimman irrallaan toisistaan, jotta testien tulokset eivät vaikuttaisi toisiinsa. Jokaiselle erilliselle tutkittavalle palvelulle suoritetaan testit saman testaussuunnitelman mukaisesti ja tulokset kirjataan ylös. Tämän testauksen avulla toivotaan saavutettavan vertailukelpoista tietoa, jota voidaan yhdistellä ja analysoida tuloksissa.

3.2 Jatkuva integraatio

Ohjelmiston kehittäminen vaatii muutosten suunnittelua, jatkuvaa tulosten seurantaa ja tulosten ohjaamista oikeaan suuntaan. Jatkuva integraatio tuo tähän prosessiin automaatiota siten, että jos ohjelmisto hajoaa muutoksia tehtäessä siitä saadaan välitöntä palautetta ja tarvittavat muutokset ongelmien korjaamiseksi voidaan tehdä nopeammin. (Duvall, ym. 2007, 24.) Fowler käy myös läpi ohjelmakoodin ongelmien kertymistä pidemmällä aikavälillä ja huomauttaa automaattisella ohjelmakoodin seurannalla ongelman olevan lähes eliminoitavissa. Ongelmien kertymisen välttäminen on erittäin tärkeää, koska suuri määrä koodivirheitä voi olla äärimmäisen hankala ja kallis projekti korjata. (Fowler. 2006)

Jatkuva integraatio järjestelmän perusvaatimuksina on yhteys ohjelmakoodin versionhallintaan, ohjelmakoodin kokoaminen, palautteen järjestäminen tarvittaessa sekä muutosten käyttöönotto. Järjestelmää ei voi luonnehtia jatkuvaksi integraatioksi ilman ohjelmistolle suoritettavia testejä. Ilman testausta ei voida olla varmoja ohjelmiston toiminnasta käyttöönoton yhteydessä ja muut ohjelmiston rakentamiseen

osallistuvat eivät voi olla varmoja tekevänsä muutoksia toimivalle pohjalle. (Duvall, ym. 2007, 12, 15.)



Kuva 1. Jatkuva Integraatio järjestelmän kuvaus (Duvall, ym. 2007, 5.)

Jatkuvan Integraation etuihin luetellaan riskien väheneminen, manuaalisen toiston vähentäminen, Ohjelmistokoodin julkaisukelpoisuus, projektin tilanteen näkyvyys ja ohjelmakoodin pitäminen luotettavana. (Duvall, ym. 2007, 29.)

Riskien vähentämisen Fowler kuvailee tärkeimmäksi saatavaksi hyödyksi. Automatisoidun järjestelmän avulla voidaan poistaa projektista epävarmuus sen valmistumisesta. Se auttaa myös näkemään mikä osa koodista ei projektissa toimi. (Fowler. 2006) Duvall huomauttaa, että mitä aiemmin virheet löydetään sitä vähemmän aikaa todennäköisesti niiden korjaamiseen tarvitsee käyttää, koska voidaan nopeasti arvioida minkä muutoksen yhteydessä ongelma on ohjelmistoon sisällytetty. (Duvall, ym. 2007, 29-30.)

Manuaalisten toistuvien tehtävien vähentäminen säästää aikaa, rahaa ja vaivaa koko projektin kannalta. Automaatiolla varmistetaan, että kaikki vaiheet tapahtuvat samalla tavalla jokainen kerta ja oikeassa järjestyksessä. Ajatus tylsien testien ajamisen välttämiseltä on helppoa saada läpi mille tahansa kehitysryhmälle. (Duvall, ym. 2007, 30-31.)

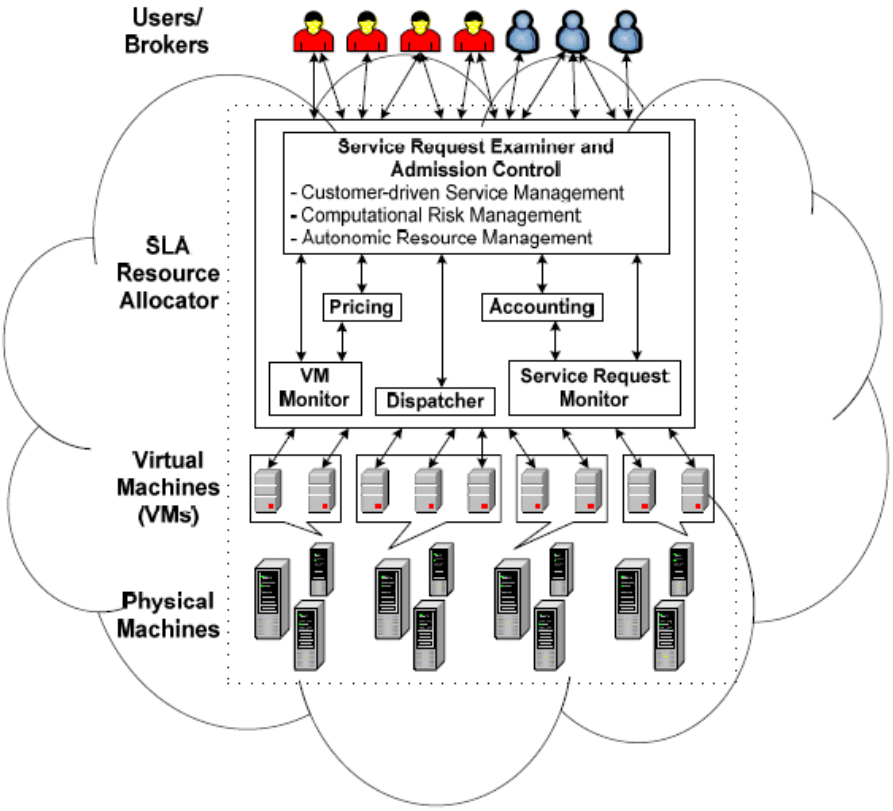
Fowler kuvailee jatkuvan integraation poistavan yhden isoimmista esteistä ohjelmistomuutosten nopeaan käyttöönottoon. Ominaisuus on tärkeää sen kannalta,

että ohjelmiston käyttäjät pääsevät käsiksi uusiin ominaisuuksiin nopeasti ja voivat antaa palautetta tehdyistä muutoksista. (Fowler. 2006) Myös Duvall selvittää samoja piirteitä ohjelmakoodin käyttöönotosta ja huomauttaa, että ilman ohjelmakoodin pitämistä jatkuvasti julkaisukelpoisena voi projektille aiheutua huomattava myöhästyminen, jos ohjelmistoa joudutaan korjaamaan ennen julkaisemista. (Duvall, ym. 2007, 31.)

Jatkuva integraatio antaa mahdollisuuden tehdä tehokkaita ratkaisuja ja luoda parempia ratkaisuja ongelmiin. Projektilla on päivitettyä tietoa ohjelmiston tilasta ja laadullisesta mittauksesta, jota voi käyttää päätösten tekemiseen. Jatkuva integraatio mahdollistaa ohjelmiston kehittäjille varmuuden siitä, että viimeisin saatavilla oleva versio ohjelmakoodista on testattu ja toimiva. Ilman tätä varmuutta kehittäjä ei voi olla varma siitä, että tehdyt muutokset tulevat toimimaan oletetulla tavalla. (Duvall, ym. 2007, 31-32.)

3.3 Pilvipalveluiden rakenteesta

NIST-julkaisussaan Mell ja Grance kuvailevat pilvipalveluiden keskeiseksi ideaksi tarjota kuluttajille mahdollisuutta hankkia tarpeidensa mukaisia, missä tahansa saatavilla olevia ja helposti käyttöönotettavia tietokoneresursseja. Näitä resursseja voi kuluttaja nopeasti ja vähäisellä vaivalla ottaa omaan käyttöönsä. (Peter Mell, Timothy Grance. 2011, 2.) Armbrust ja muut kuvailevat tutkimuksessaan samanlaisesti ja korostavat pilvipalveluiden ratkaisevan monia internet palveluiden tarjoamiseen liittyviä ongelmia. Pilvipalvelut poistavat palveluiden kehittäjiltä tarpeen tehdä suuria investointeja omiin järjestelmiinsä, jotka eivät välttämättä tule saavuttamaan oletettua suosiota ja vastaavasti myöskin tarjoavat joustavasti resursseja mikäli palvelun kasvu on odotettua nopeampaa. (Armbrust, ym. 2009, 1.)



Kuva 2. Korkean tason pilvipalveluiden arkkitehtuurin kuvaus. (Buyya, Rajkumar, ym. 2008, 8.)

Nykypäivän pilvipalvelut lupaavat luotettavia palveluita, jotka tarjotaan moderneista palvelukeskuksista. Palvelut yhdistävät yritykset ja asiakkaat missä tahansa ja milloin tahansa. Pilvipalveluiden kautta tarjotut ohjelmistot ja palvelut ovat usein kriittisessä osassa yritysten toimintaa minkä johdosta palveluntarjoaja on velvollinen toimittamaan tarjotut palvelut palvelusopimusten mukaisesti. (Buyya, Rajkumar, ym. 2008, 2, 4.)

3.3.1 Pilvipalveluiden keskeisimmät ominaisuudet

Tarpeiden mukainen palvelu, jossa palvelun kuluttaja voi tarpeidensa mukaisesti ja omatoimisesti ottaa palveluntarjoajan järjestelmästä resursseja käyttöönsä. Resursseja voi olla esimerkiksi verkkotallennustila tai palvelinaika. (Peter Mell, Timothy Grance. 2011, 2.) Myös Armbrust ja muut (2009, 1) listaavat mahdollisuuden ottaa resursseja vapaasti käyttöönsä yhdeksi kolmesta laitteistojen tarjoamasta uudesta ominaisuudesta, joita pilvipalvelut mahdollistavat.

Laaja saatavuus verkossa, jossa järjestelmään pääsyä tarjotaan verkon ylitse yleisiä tekniikoita hyödyntäen useille eri päätelaitteille, kuten tietokoneelle tai kännykälle. (Peter Mell, Timothy Grance. 2011, 2.) Laaja saatavuus verkossa yhdistää kuluttajia mahdollistaen kuluttajien jakavan tietoa ja tekevän yhteistyötä helpommin. Palvelun pitäminen yllä jatkuvasti on erittäin vaativa haaste järjestelmien tarjoajille. Asiakkaiden kannalta ei ole järkevää tukeutua vain yhteen palveluun ja varsinkin isomissa yrityksissä ei ole kannattavaa ottaa kriittisiä pilvipalveluita käyttöönsä ilman mahdollisuutta nopeasti vaihtaa palvelun tarjoajaa ongelmien ilmaantuessa. (Armbrust, ym. 2009, 4, 14.)

Palveluntarjoajan resurssit yhdistetään palvelemaan useita kuluttajia yhtä aikaa asettaen käyttöön fyysisiä ja virtuaalisia resursseja käyttäjän tarpeiden mukaan. Kuluttajalla ei usein ole tietoa siitä mistä palvelua tarjotaan ellei sitä voi palvelun asetuksissa erillisesti määritellä. (Peter Mell, Timothy Grance. 2011, 2.)

Pilvipalveluiden joustavuudella tarkoitetaan, että ominaisuuksia ja resursseja voidaan joustavasti lisätä tai vähentää, joskus automaattisesti. Kuluttajalle yleisesti näytetään rajattomia resursseja, joita voi vapaasti ottaa käyttöön minkä tahansa määrän. (Peter Mell, Timothy Grance. 2011, 2.) Palveluiden käytön joustavuus tarjoaa erityisesti kuluttajalle suuria etuja palveluun kohdistuvien kulujen kannalta. Pilvipalveluja kulutetaan vain tarpeen mukaisesti ja tämä luo erittäin edullisen kulurakenteen. (Armbrust, ym. 2009, 10.)

Järjestelmät automaattisesti kontrolloivat ja optimoivat resurssiensa käyttöä ja mittaavat samalla kuluttajan käyttämää resurssimäärää riippuen palvelun mallista. Resurssien käyttöä seurataan, kontrolloidaan ja raportoidaan sekä palvelun tarjoajalle, että kuluttajalle mahdollistaen tarkan kuvan palvelun käytöstä. (Peter Mell, Timothy Grance. 2011, 2.)

3.3.2 Palvelumallit

Ohjelmisto palveluna (Software as a Service – SaaS) järjestelmässä kuluttajalle tarjotaan käyttöön ohjelmisto, joka on toiminnassa palveluntarjoajan pilvipalvelussa.

Ohjelmistoon tarjotaan pääsyä useilta eri päätelaitteilta esimerkiksi verkkoselaimen tai erillisen ohjelmiston kautta. Kuluttajalla ei ole pääsyä ohjelmiston pohjalla toimivaan pilvipalveluun tai sen ominaisuuksiin. Poikkeuksena voi olla ohjelmistoon rakennetut ominaisuudet joihin kuluttaja voi vaikuttaa. (Peter Mell, Timothy Grance. 2011, 2.) Tutkimukseen kuuluvat kaksi SaaS palvelua ovat CircleCi ja Travis-Ci.

Alusta palveluna (Platform as a Service – PaaS) keskeisenä mallina on tarjota kuluttajalle käyttöön alusta, johon kuluttaja voi asentaa tekemänsä tai hankkimansa ohjelmiston. Ohjelmiston tulee olla luotu käyttäen ohjelmointikieliä, kirjastoja, palveluita ja työkaluja, joita palveluntarjoajan järjestelmä tukee. Kuluttaja kontrolloi ohjelmistonsa asennusta ja mahdollisia ohjelmiston tarjoamiseen liittyviä asetuksia, mutta ei alla toimivaa pilvipalvelurakennetta. (Peter Mell, Timothy Grance. 2011, 2-3.) Tutkimuksen sisältämä PaaS palvelu on Cloudbees.

Rakenne palveluna (Infrastructure as a Service – IaaS) järjestelmässä kuluttajalle tarjotaan käyttöön laskentatehoa, levytilaa ja muita tietokoneresursseja, jonne kuluttaja voi asentaa haluamansa ohjelmiston, joka voi olla käyttäjärjestelmä ohjelmistoineen. Kuluttajalla ei ole pääsyä alla toimivaan pilvirakenteeseen, mutta kontrolloi pilvessä olevaa käyttöjärjestelmää, levytilaa ja asennettuja ohjelmistoja. (Peter Mell, Timothy Grance. 2011, 3.) Tutkimus ei sisällä IaaS palvelua.

3.3.3 Toimintamallit

Yksityinen pilvipalvelu on tarkoitettu yhden organisaation omaan käyttöön. Sen voi omistaa, hoitaa ja toimittaa organisaatio, kolmas osapuoli tai näiden jokin yhdistelmä. Järjestelmä voi olla toimitilojen yhteydessä tai ulkopuolella. (Peter Mell, Timothy Grance. 2011, 3.)

Yhteisön pilvipalvelu on tarkoitettu tietyn yhteisön omaan käyttöön. Yhteisö koostuu organisaatioista, joilla on samat tarpeet palvelun järjestämisessä. Järjestelmän voi toimittaa yhteisön jäsen, kolmas osapuoli tai näiden jokin yhdistelmä. (Peter Mell, Timothy Grance. 2011, 3.)

Julkinen pilvipalvelu on tarkoitettu avoimeen käyttöön. Sen voi omistaa yritys, koulutuslaitos, hallinnollinen organisaatio tai näiden jokin yhdistelmä. Palvelu tuotetaan palveluntarjoajan toimitiloista. (Peter Mell, Timothy Grance. 2011, 3.)

Yhdistelmä pilvipalvelussa rakenne on yhdistelmä kahden tai useamman erilaisen pilvipalvelun rakenteesta (yksityinen, yhteisö tai julkinen), jotka toimivat erillisinä toimijoina, mutta joita sitoo yhteen standardisoidut tai omistetut teknologiat, jotka mahdollistavat tiedon ja ohjelmistojen siirrettävyyden tai resurssien jakamisen. (Peter Mell, Timothy Grance. 2011, 3.)

3.4 Yksikkötestaus

Yksikkötesti on koodia, jonka ohjelmiston kehittäjä kirjoittaa kokeilemaan yhtä pientä rajattua aluetta ohjelmiston koodissa. Yleisesti yksikkötesti koskee yhtä metodia tietyssä tilanteessa. Yksikkötestien tarkoitus on varmistaa, että tämä metodi toimii juuri niin kuin ohjelmoija olettaa sen toimivan. (Hunt, Andy. Thomas, Dave. 2003, 3.)

Yksikkötestien tarkoitus ei ole ottaa kantaa siihen mitä ohjelmiston välttämättä halutaan toteuttavan tai miten tehokkaasti ohjelmakoodi toimii. Pyrkimyksenä on ennen kaikkea varmistaa, että yksittäiset osiot, jotka muodostavat ohjelmakoodin toimivat odotetulla tavalla ja siten luovat varmuuden järjestelmän toimivuudesta. Mikäli ohjelmiston osat eivät toimi odotetulla tavalla ei millään muulla testauksella ole varsinaisesti merkitystä. (Hunt, Andy. Thomas, Dave. 2003, 3-4.)

Yksikkötestin ajaminen onnistuneesti ei koskaan saisi kestää sekunnin murto-osaa kauemmin. Jos testin ajaminen kestää kauemmin se on joko rikki tai sen kuuluisi olla isompien testikokonaisuuksien kautta toteutettuna. Mikäli testien ajaminen kestää kauan muodostuu niiden ajamisesta pian taakka kehittäjille, jota ruvetaan välttelemään mahdollisimman paljon. Jatkuva Integraatio järjestelmässä testit tulisi ajaa jokainen kerta ohjelmistokoodin päivityksen yhteydessä mikä poistaa testausaikojen aiheuttamaa päänvaivaa kehittäjille. (Duvall, ym. 2007, 141.)

3.5 Versionhallinta

Versiohallintaa voi kuvailla tiedon useiden versioiden hallinnaksi. Yksinkertaistettuna se on jotain mitä useimmat ihmiset tekevät käsin muokatessaan tiedostoja ja tallentaessaan sen uudella nimellä lisäten perään aina aiempaa suuremman luvun. Kuitenkin tämän prosessin hoitaminen on erittäin virhealtista toimintaa ja sitä varten on kehitetty ohjelmistoja jo pitkän aikaa. Nykypäivän versionhallintatyökalut automaattisesti järjestelevät useita tiedostoja ja palvelevat useita käyttäjiä mahdollistaen suuria ohjelmistoprojekteja, joissa tuhannet ihmiset voivat työskennellä yhdessä rakentaen satoja tuhansia tiedostoja. (O'Sullivan, Bryan. 2009, 1.)

Versionhallinta auttaa ylläpitämään muutoksien historiaa projektissasi. Muutoksille jää tieto siitä kuka muutoksen on tehnyt, miksi muutos on tehty, milloin muutos on tehty ja mitä siinä muutettiin. Versionhallinta auttaa selvittämään tilanteita, joissa useampi muutos on ristiriidassa toistensa kanssa ja mahdollistaa aikaisemman tilanteen palauttamisen virheiden sattuessa. Versionhallinta mahdollistaa myös useampien samanaikaisten versioiden ylläpitämisen samasta projektista. (O'Sullivan, Bryan. 2009, 1-2.)

4 Tutkimuksen toteutus

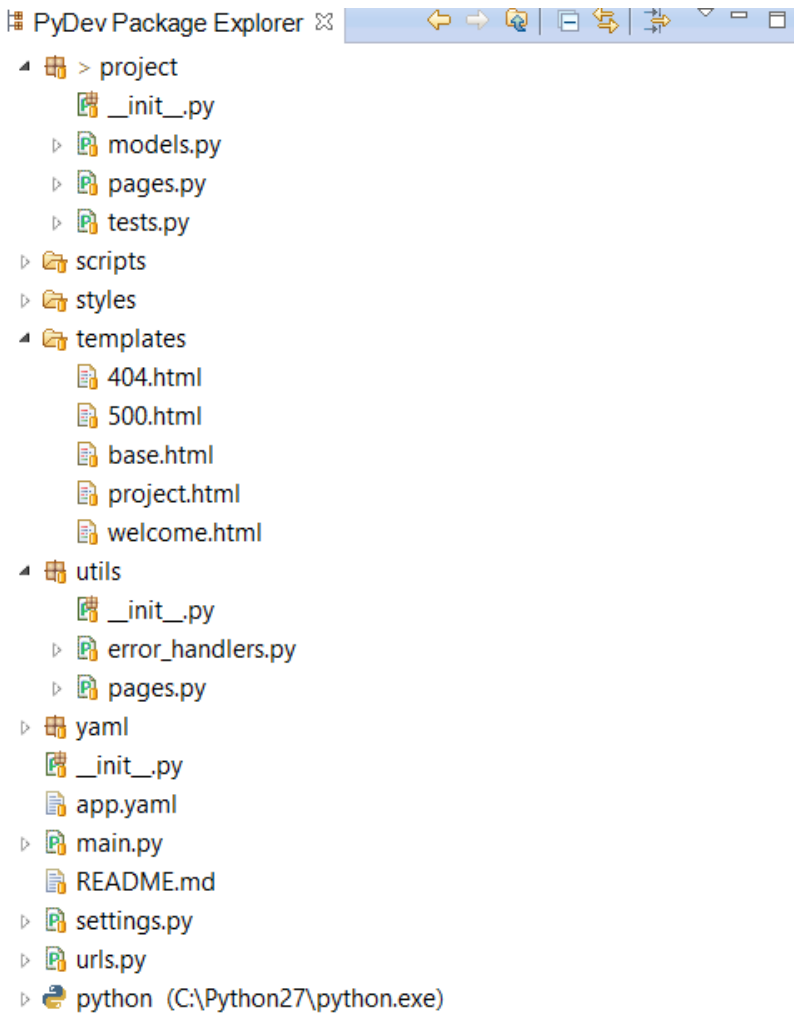
4.1 Testiohjelmisto

Tutkimusta varten rakennettuun testiohjelmistoon tehtiin useita eri työkalujen valintoja. Valinnoissa painotettiin aiemmin koulutusohjelman kanssa tutuiksi tulleita ohjelmistoja sekä palveluita.

Ohjelmistokoodin versionhallintaa hoitaa suosittu GitHub palvelu, jonka voi olettaa olevan hyvä vaihtoehto jatkuvan integraation asentamisen kannalta. Palvelu itsessään on varsin vaivatonta ottaa käyttöön. Ohjelmiston rakentamiseen käytössä on Eclipse-ohjelmointityökalu, johon on ladattu PyDev lisäosa sekä GitHubin tarjoama EGit lisäosa. Googlen App Engine Python kehityskirjastot ovat kehitysympäristössä liitetty ohjelmistoon projektin asetuksien kautta.

Testiohjelmistolla on oleellinen vaikutus testitulosten saamisessa. Yksinkertainen hello-world sovellus ei tarvitse toimiakseen erillisiä Python moduuleja tai kirjastoja ja siksi liian yksinkertaista ohjelmistoa testatessa voisi oleellisia ongelmia jäädä havaitsematta. Testiohjelmiston sisällön tai toiminnallisuuden ei tarvitse olla monimutkainen testaamisen toteutusta varten, mutta on tärkeää varmistua tulosten vastaavuudesta oikean kehitysympäristön kannalta. Mitä vaikeampaa itse testiohjelmiston ympäristön asentaminen on sitä paremmin se paljastaa mahdollisia ongelmia asennuksen kohteena olevan palvelun ympäristön käyttöönotossa. Toimiakseen oikein on jatkuva integraatio palveluiden kyettävä käyttämään testiympäristön kirjastoja ajaessaan testejä tai suorittaessaan käyttöönottoa.

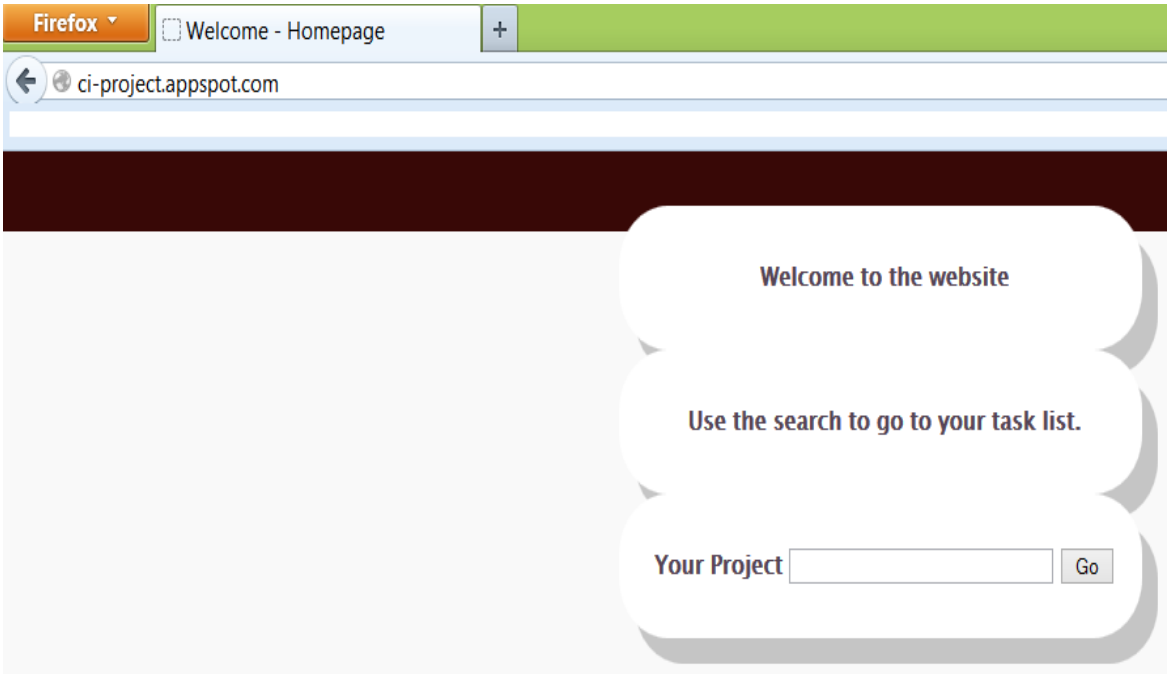
Testiohjelmiston sisäisessä rakenteessa tärkein tiedosto on app.yaml, joka sisältää ohjelmiston rakennetta koskevat tiedot App Enginen käyttöä varten. Tämän tiedoston tietoja lukemalla sovellus ottaa itsensä käyttöön palvelimella. Uuden palvelin instanssin käynnistyessä sovelluksen käyttöön ladataan Settings.py tiedoston muuttujat sekä alustetaan sovellus käyttämään Django ympäristöä main.py tiedoston kautta. App Enginen viimeisin tukema Python versio on 2.7, joten tämä on syytä huomioida myös testiympäristöjen kannalta varsinaisen testauksen aikana.



Kuva 3. Testiohjelmiston rakenne Eclipse-ohjelmointityökalussa.

Sovelluksessa App Enginen käytössä olevaa Bigtable-tietokantaa kutsutaan project-kansion models.py tiedostoon sisältyvissä luokissa. Tests.py luokan yksikkötestit keskittyvät erityisesti näiden Project ja Task-olioiden tietokantakutsujen testaamiseen ja tämän tiedoston tulisi jatkuvan integraatio palveluiden tunnistaa ja ajaa onnistuneesti testien suorittamiseksi.

Verkkosovelluksen käyttöönotto tapahtuu App Enginen kehityskirjaston kautta saatavalla apuohjelmalla, jolla kutsutaan suoraan halutun projektin sijaintia. App.yaml tiedostosta käyttöönotossa luetaan projektin nimi ja muuta oleellista tietoa. Jatkuvan integraation palvelulta odotetaan kykyä käyttää apuohjelmaa ja viemään käyttöönotto loppuun saakka.



Kuva 4. Toimiva testiohjelmisto App Enginen palvelimella.

4.2 Testaussuunnitelma

Tutkimukseen sisältyy kolme palvelua CloudBees, CircleCi ja Travis-ci. Tutkimukseen alunperin suunniteltua ShiningPanda-ci palvelua ei päästy testaamaan sivuston hankalan ilmaismallin tuomien esteiden takia. Tutkimuksen toteutukseen käytetty testaussuunnitelma löytyy liitteestä 1.

Oleellista vertailukelpoisten tulosten saamiseksi oli määritellä tarkkaan mitä tulee testata. Oikean maailman vaatimuksista vastaa ohjelmiston riippuvuus App Engine kirjastoista mikä luo kaikille asennuksen vaiheille lisähaasteita. Tärkeää on nimenomaan löytää ongelmia joihin törmää vasta, kun poistuu ohjekirjojen yksinkertaistetusta maailmasta.

Tärkeimpiä ominaisuuksia, joita voi testata on asentamisen selkeys ja nopeus, koodin ajamisen kesto, Virhetilanteiden aiheuttamien viestien lähteminen ja viestitys vaihtoehtojen määrä, hinta palvelulle sekä hyväksytyn koodin lähettäminen testipalvelimelle. Ongelmaksi voi muodostua tutkimuksen kannalta käytössä olevat ilmaiset kokeilut ja niihin mahdollisesti liittyvät rajoitukset. Hinnoittelua on muutenkin hankalaa arvioida todellisen käytön kannalta. Kuitenkin yleinen suuntaus on nähtävillä

jo vähäiselläkin kokeilulla niin kauan kuin käyttäjälle selkeästi välitetään käytetyt resurssit.

4.3 CircleCi

CircleCi palvelu on keskittynyt tarjoamaan mahdollisimman helposti käyttöönotettavaa jatkuva integraatio ympäristöä. Palvelusta ei ole jatkuvaa ilmaismallia, mutta tarjoalla on kahden viikon ilmainen kokeilu. Kokeilu vastaa palvelun halvinta mallia toiminnoiltaan. CircleCi palvelu tarjoaa ensisijaisesti asiakkailleen SaaS palveluaan, jonka käyttöönotto on avoimesti saatavilla.

CircleCi ympäristö pyörittää itseään Ubuntu 12.04 käyttöjärjestelmässä ja tukee useita ohjelmointikieliä. Ohjelmointikieliin luetellaan muun muassa Ruby, Java, Python, PHP, Clojure, Go, C++ ja Erlang. Palvelussa on käytössä useita sisäisiä palveluita ja tietokanta ratkaisuja. (CircleCi c, 2013.)

CircleCi palvelun malli esittäytyy varsin houkutteleva, koska se lupaa helppokäyttöisyyttä ja käyttötukea palvelua varten. Hinnoittelu ei ole ensisilmäykseltä kovinkaan halvan oloinen, mutta se ei myöskään ole suuri este valinnan kannalta. Muutaman kymmenen euron kuukausittainen maksu siitä, että palvelu toimii ilman ongelmia on kohtuullista.

4.3.1 Asentaminen ja versionhallinnan yhdistäminen

Palveluun rekisteröityessä CircleCi hyödyntää suoraan GitHubia käyttäjätunnuksen tekemiseen ja versionhallinnassa olevien projektien yhdistämiseen. Tunnuksien liittämisen jälkeen CircleCi palvelu ottaa yhteyden GitHubiin ja tarjoaa listan käyttäjällä olevista projekteista. Projekti liitetään jatkuvaan intergraatioon painamalla listauksessa olevaa ”follow”-nappia, jonka jälkeen muutokset versiohallinnassa käynnistävät integraatioprosessin.

CircleCi ensisijaisesti mainostaa yhden napinpainalluksen teknologiaansa, jonka kautta ohjelmistoprojektin pitäisi automaattisesti toimia palvelussa. Kuitenkaan ohjelmisto ei

aluksi tunnistanut muuta kuin Python-ohjelmointikielen käytön ja muu toiminnallisuus oli tarpeellista opastaa CircleCi-palvelun käyttämään circle.yml tiedostoon.

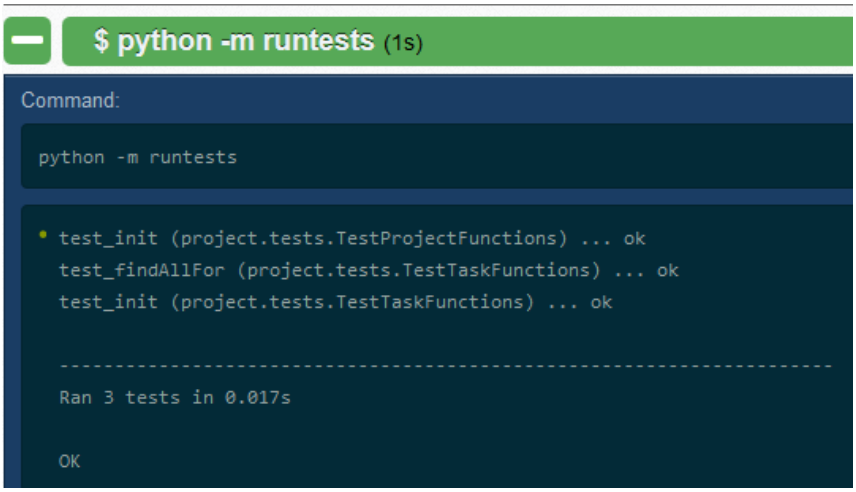
CircleCi tarjoaa asennustiedoille käyttöliittymää internetin kautta. Etuna asennustietojen asettamiselle verkkokäyttöliittymään on selvä, jos ohjelmiston asetuksiin tarvitsee laittaa kirjautumistietoja tai muita vastaavia salaisia tietoja. Asennustietojen pitäminen erillisessä tiedostossa jättää arkaluontoisetkin tiedot jokaisen kehittäjän omalle tietokoneelle mahdollistaen tietojen vuotamisen ulkopuolisille tahoille. Kuitenkin on todennäköisesti pidemmällä tähtäimellä parempi käytäntö laittaa asetukset tiedostoon, jota voi muuttaa kehittäjien tietokoneilla ilman kirjautumista palveluun. Ohjelmiston koodin juuressa oleva tiedosto on monikäyttöisin ja helpoin tapa ylläpitää asennustietoja mikäli projektilla on useampia kehittäjiä. Palvelun testaamiseen käytetty circle.yml tiedosto on nähtävillä opinnäytetyön liitteessä 3.

4.3.2 Testien ajaminen

CircleCi automaattisesti selvittää oleellisia tietoja projektista ja se helpottaa huomattavasti testien ajamista, koska havaittuja asetuksia ei tarvitse asettaa projektille erikseen. Palvelu löysi ohjelmiston testejä ilman avustusta ja yritti huonolla tuloksella ajaa ne. Testit eivät toimi, koska projektin sisäisissä asetuksissa Google App Engine kirjaston tiedostoja ei etsitä käyttäjän kotihakemistosta eikä tätä muutosta välttämättä haluta tehdä ohjelmiston pysyviin asetuksiin.

Kuva 5. Testien ajo CircleCi palvelun oman tunnistuksen mukaan.

Projektin kannalta käytännöllisintä on yliajaa CircleCi-palvelun oma testien tunnistus ja ajo lisäämällä ohjelmiston juureen testien ajoa varten oma luokka ja lisätä tarvittavat tiedostot tämän luokan ajon yhteydessä ohjelmiston kirjastoihin. Näin vältytään hankalalta tilanteelta, jossa jokaisen projektiin osallistuvan ohjelmoijan ympäristön pitäisi olla täsmälleen samalla lailla asetettu kuin testiympäristön. Testit yhdistävän runtests.py luokan tarvitsee tunnistaa käyttäjän kotihakemisto ja etsiä purettu google_appengine hakemisto sieltä. Testien normaalin ajon yliajo tehtiin circle.yml luokassa (katso liite 3).



Kuva 6. Testien ajo onnistuneesti läpi CircleCi palvelussa.

4.3.3 Käyttöönotto

CircleCi -palvelun ollessa käynnissä omassa toimintaympäristössään ei käyttäjä voi tehdä mitään toimintoja koodien ajon aikana mikä estää esimerkiksi salasanojen tai muiden tunnistetietojen syöttämisen. Palvelun omassa dokumentaatiossa opastettiin käyttäjää lähettämään palveluun koodi, jonka ajaminen suorittaisi ohjelmakoodin käyttöönoton App Engineen, mutta sisälsi tarvittavat salasanat ja käyttäjätunnukset. Koodi tulee tässä tapauksessa kuitenkin näkyviin julkiseen versiohallintaan, joten tätä toimenpidettä ei ole kovinkaan mielekästä suorittaa.

Google App Engine tukee OAuth2 kirjautumista. Oleellista kirjautumisen käyttöönotossa on ensin ajaa ohjelmakoodin käyttöönotto Googlen palvelimelle omalta koneeltaan. Käyttäjä lähettää ohjelmakoodin käyttäen --OAuth2 komentoa komentorivillä ja lähettämällä tarvittavat tunnistautumistiedot. Toimenpiteen jälkeen

käyttäjän kotihakemistosta löytyy json-tiedosto nimeltä appcfg_oauth2_tokens. Tiedosto sisältää refresh_token tiedon, joka tulee syöttää mukaan ohjelmiston asennuksen yhteydessä komennon –oauth2_refresh_token perään, jotta ohjelmiston voi asentaa ilman käyttäjätunnuksen syöttämistä. (Google, 2013.)

Käyttöönottoa ajettaessa ilmeni hyvin erikoinen ongelma, kun ohjelmisto lähetti itsensä palvelimelle kaksi kertaa samasta käskystä. Tapahtuma lähinnä kertoisi alustan sisäisistä ongelmista. Ongelma ei kuitenkaan estänyt ohjelmiston lähettämistä eteenpäin App Enginen puolelle, joten ongelmaa ei voi pitää kriittisenä toiminnan kannalta eikä myöskään aiheuta lisäkustannuksia käyttäjälle.

4.3.4 Ilmoitukset

Ilman erillistä asetusten muutosta CircleCi -palvelu automaattisesti lähettää sähköpostin GitHub-tilin sähköpostiosoitteeseen. Viesti lähetetään ohjelmiston rakentamisen epäonnistuessa sekä ensimmäisellä kerralla, kun koodin kokoaminen onnistuu virhetilan jälkeen.

Perustoiminnallisuuden lisäksi palvelun tapahtumista lähtevien viestejä voi asettaa palvelun circle.yml tiedostossa tai vaihtoehtoisesti verkkosivuilla. Palvelusta voi asettaa viestit lähtemään sähköpostin lisäksi myös Campfire, Flowdock ja HipChat palveluihin.

4.3.5 Hinnoittelu

CircleCi palvelusta on ilmaiseksi tarjolla vain 14 päivän kokeilu, jonka päätteeksi käytöstä olisi maksettava kuukausittaisesti. Alin kuukausimaksu on 19 euroa ja ylimmästä voi palveluntarjoajan kanssa sopia erikseen. Korkeammilla kuukausimaksuilla maksuun sisällytetään suurempia määriä versiohallintaprojekteja, joita palvelu seuraa. Korkeampaan kuukausimaksuun sisällytetään myös muita palvelun ominaisuuksia, kuten välitöntä käyttötukea. (CircleCi a, 2013)

Hinnoittelun vertailun kannalta on hyvä huomioida ettei palvelu ota kantaa sovelluskehittäjien määrään, mutta ohjelmistoprojekteja ollessa kolme kappaletta pitää

hinnastosta valita ensimmäistä kuukausimaksua korkeampi 49 euron paketti, johon kuuluu 10 projektin rakentamisen rajaus.

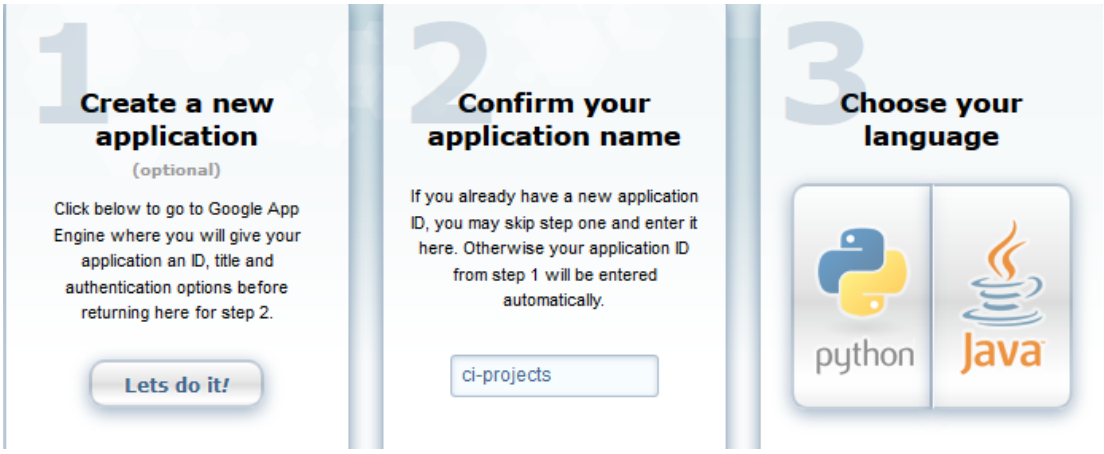
4.4 CloudBees

Cloudbees -palvelu tarjoaa kattavan valikoiman erilaisia pilvimallisia ratkaisuja sekä valmiita palvelupohjia, joita kehitetään yhteisön voimalla. Cloudbees on vertailun ainoa PaaS alusta eli alusta, joka ensisijaisesti antaa käyttäjälle mahdollisuuden omatoimisesti ottaa käyttöönsä laajan valikoiman erilaisia ohjelmistoja. Tämä helposti johtaa ongelmiin palvelun käyttöönotossa, koska palvelu ei keskity toimittamaan vain yhtä ominaisuutta ja ohjelmien tai niiden lisäosien asennus jätetään pitkälti käyttäjän omille harteille. Etuna tässä on mahdollisuus keskittää mahdollisia muita palveluita samaan käyttöympäristöön sekä parempi kontrolli ympäristöstä.

Cloudbees palvelu on kaikista testattavista palveluista käyttäjän kannalta monipuolisin, mutta sen kautta myös monimutkaisin. Palvelu ei ohjaa käyttäjää samalla tavalla kuin yhteen asiaan omistautuneet palvelut. Hankalaa käyttöönottoa ei helpota myöskään palvelun Java keskeinen ajattelu vaikka muitakin ohjelmointikieliä on mahdollista käyttää projekteissaan.

4.4.1 Asentaminen ja versionhallinnan yhdistäminen

Cloudbees palveluun tarvitsee luoda omat tunnuksensa. Palvelu ei opasta käyttäjää tekemään mitään varsinaista toimintoa palvelun sisällä vaan käyttäjän on itse löydettävä haluamansa toiminnot palvelun sisältä. Ympäristö tekee monet asiat ensisijaisesti Javan ominaisuuksia ajatellen ja tekee muiden ohjelmointikielien käytön ympäristössä vaivalloiseksi. Cloudbees tarjoaa onneksi automaattista App Engine projektin muodostamista myös Python ohjelmointikielelle. ClickStart nimiset yhteisön kehittämät toiminnot on mahdollista ajaa järjestelmässä ja niiden tarkoitus on mahdollisimman paljon helpottaa palveluiden käyttöönotossa.



Kuva 7. App Engine kohtaisen ClickStart projektin aloittaminen Cloudbees palvelussa.

ClickStart toiminnon käyttö ei ollut hankalaa ja toiminto suoritettiin minuuteissa. Lopputulos ei kuitenkaan ollut odotettu ja projektin rakentaminen tuotti erikoisen tuloksen. App Enginen puolelle oli lähetetty tyhjä projekti, joka oli perustettu Cloudbees käyttäjätunnuksen perusteella.

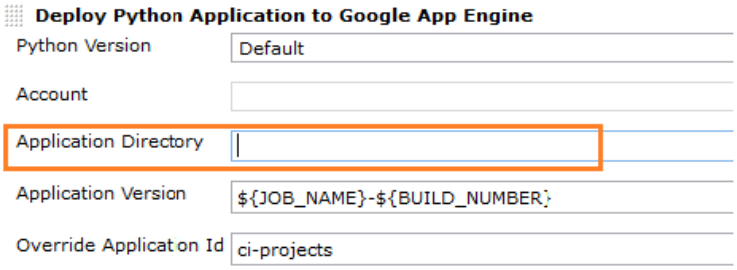
Console Output

```
Started by user
Building remotely on s-23a44d38 in workspace /scratch/jenkins/workspace/ci-project
Checkout:ci-project / /scratch/jenkins/workspace/ci-project - hudson.remoting.Channel@4f928a17:s-23a44d38
Using strategy: Default
Cloning the remote Git repository
Cloning repository origin
Fetching upstream changes from https://github.com/ci-project/ci-project.git
Seen branch in repository origin/HEAD
Seen branch in repository origin/master
Commencing build of Revision 1e6b767463660677fd86efa282fc044f868541a7 (origin/HEAD, origin/master)
Checking out Revision 1e6b767463660677fd86efa282fc044f868541a7 (origin/HEAD, origin/master)
No change to record in branch origin/HEAD
No change to record in branch origin/master
Finished: SUCCESS
```
Kuva 8. Ensimmäinen ajo oli palvelun mukaan onnistunut.

Käytössä oleva ClickStart toiminto ei siis toiminut suoraan, koska monia tietoja oli ohjattu vääriin paikkoihin. Toiminto oli olettanut esimerkiksi käyttäjänimen olevan sama GitHubissa kuin itse palvelussa ja ohjelmiston sijaitsevan Java mallisessa tiedostopolussa. Silti ensimmäinen ajo oli Jenkinsin mielestä onnistunut vaikka mitään testejä ei ollut ajettu ja App Engineen oli lähetetty tyhjä projekti.

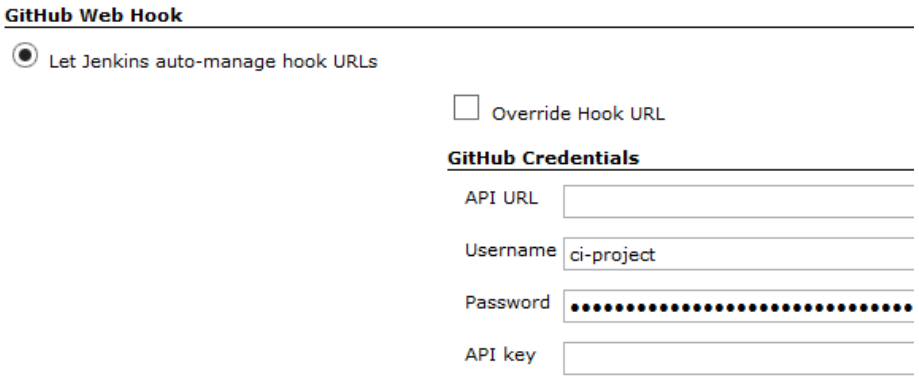
Projektin sijainti oli ohjattu kansioon src/webapp ja GitHub osoite asetettiin projektin nimen ja Cloudbees tunnuksien mukaisesti. Palveluun rekisteröityessä ei kuitenkaan

pyydetty GitHub -palvelussa käytettyjä tietoja vaan palvelu itse pyrkii tekemään projektin GitHub osoitteet itse.



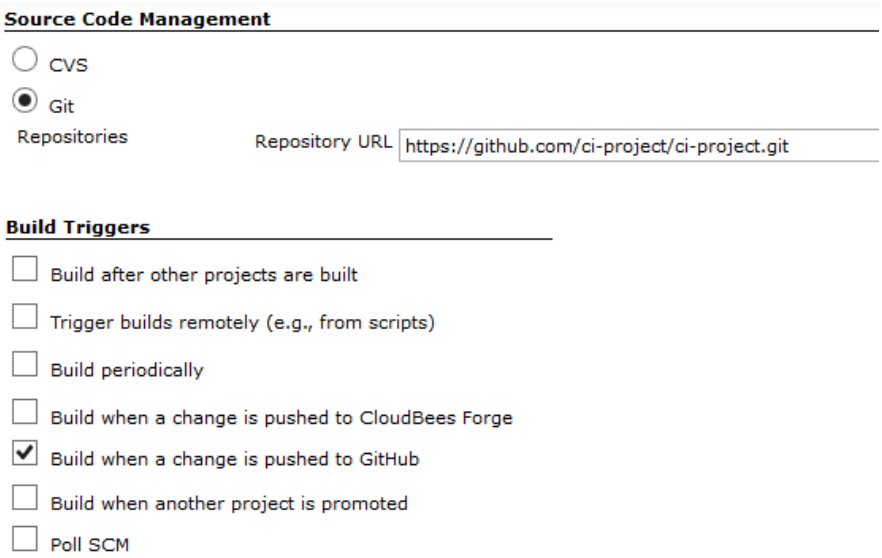
Kuva 9. Cloudbees palvelussa ohjelmiston sijainti korjaantui tyhjentämällä kenttä.

Toimiakseen oikein Jenkins ympäristöön on tarpeellista asentaa GitHub plugin ja Jenkins Git plugin lisäosat. Lisäosia voi asentaa manage plugins valikosta Jenkinsin asetuksissa. Lisäosan asennettua se pitää vielä yhdistää GitHub projektiisi Jenkinsin yleisissä asetuksissa lisäämällä GitHub tunnus ja salasana sen käyttöön.



Kuva 10. GitHub plugin -lisäosan asetuksien lisääminen Jenkinsin asetuksissa.

Lisäosien asettamisen jälkeen on vielä tarpeen käydä muuttamassa aiemmin ClickStart ominaisuudella luodun projektin asetuksia ja kertoa mistä seurattava projekti löytyy. Huomiotavaa on myös asettaa projekti rakentumaan, kun GitHub tiedoissa tapahtuu muutoksia. Tämän jälkeen, kun versionhallintaan tulee muutoksia Cloudbees ympäristö aloittaa muutosten testaamisen.

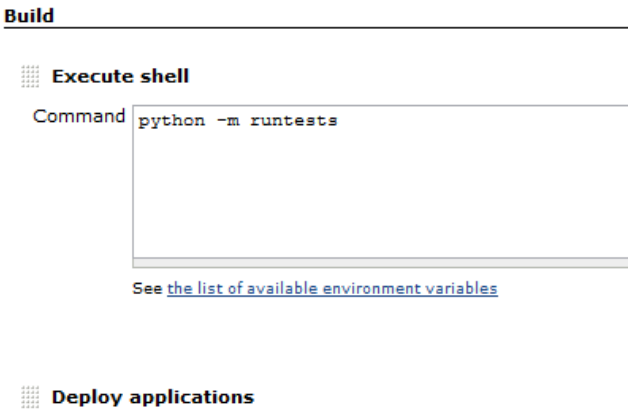


Kuva 11. Jenkins projektin asetuksien asettaminen lukemaan muutoksia GitHub projektissa.

4.4.2 Testien ajaminen

Automaattisesti asetettu NoseTest järjestely aiheutti ongelmia ja oli syytä yliajaa asettamalla asetuksiin oma koodi testejä varten. Palvelun toiminnallisuuksien kannalta mitään ei tässä kuitenkaan yliajeta, koska mitään ei normaalisti tapahdu ilman ohjeita. Ohjelmiston rakentamisen katsotaan epäonnistuneen mikäli yksikin vaihe epäonnistuu itsenäisesti, joten käyttäjän on itse huolehdittava tapahtumien järjestyksestä ja virheiden välittämisestä Jenkinsille.

App Engine kirjasto oli jo asennettu järjestelmään, mutta ohjelmisto ei sitä voi käyttää tietämättä missä se sijaitsee. Ohjelmoijien kannalta kuitenkin käytännöllisintä on, että testit ajetaan oikein riippumatta ohjelmiston omista asetuksista. Tarpeellisten kirjastojen sijainnin selvittämisen jälkeen tieto lisättiin runtests testiluokkaan. Luokan ajoa varten automaattisesti lisätty NoseTest koodi yliajettiin yksinkertaisella kutsulla testiluokkaa varten. Muutosten jälkeen testit menivät onnistuneesti läpi.

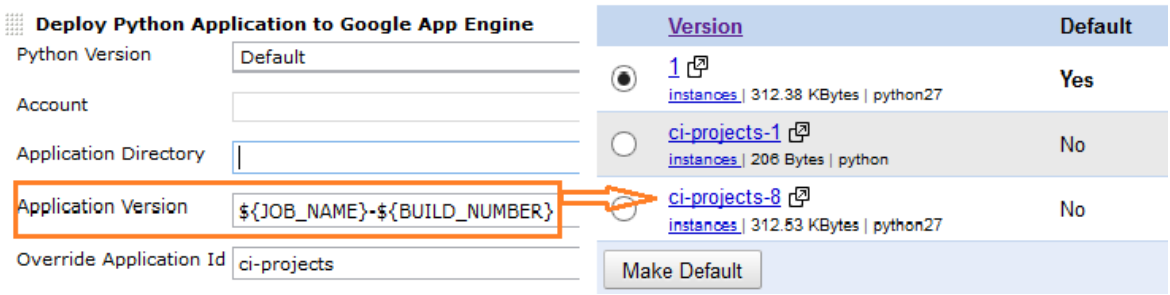


Kuva 12. Cloudbees palvelussa testaamista varten tehtyä luokkaa kutsutaan NoseTest järjestelyn sijaan.

4.4.3 Käyttöönotto

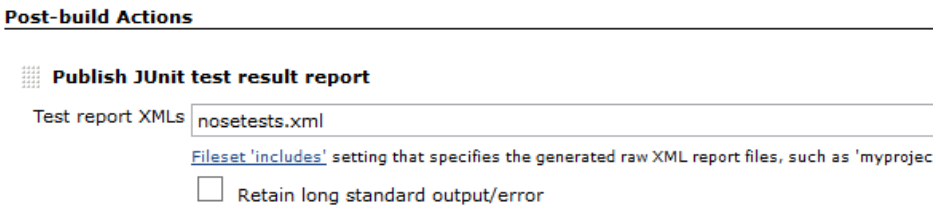
Käyttöönotto oli yllättävän helppoa sisäänrakennetun App Engine kirjautumisen avustuksella. Taustalla Jenkins installaatiolle on ajettu OAuth2 tunnistautuminen, jonka käyttäjä erikseen sallii Googlen palvelun puolella.

Ohjelmiston käyttöönotot menevät omiin versioihinsa App Enginen puolella ja täyttävät hyvin nopeasti 10 yhtäaikaisen version rajan. Tämän toiminnon voi muuttaa asetuksissa kohdassa Application Version asettamalla siihen esimerkiksi muuttumattoman tekstin jolloin kaikki Cloudbeesin kautta tulevat käyttöönotot ovat omassa versiossaan.



Kuva 13. Cloudbees palvelun asetuksien yhteys App Enginelle lähetettyyn versioon.

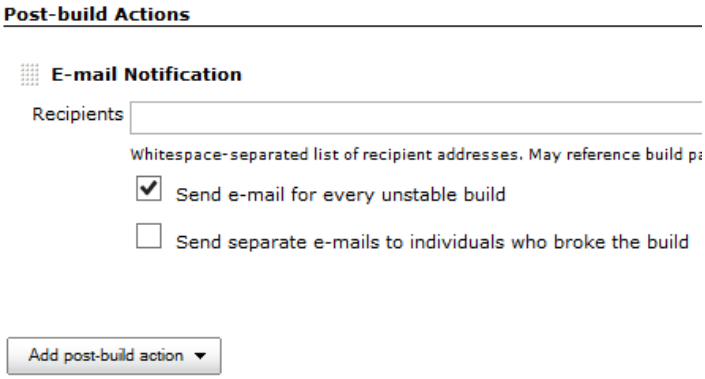
Käyttöönotto suoritettiin automaattisesti oikein, mutta Jenkins silti ilmoitti projektin epäonnistuneen. Ongelman syyksi selvisi erikoinen osio asetuksissa, joka oli tarpeellista poistaa ennen kuin projekti meni hyväksytysti loppuun saakka.



Kuva 14. Cloudbees asetus, joka aiheutti prosessin epäonnistumisen.

4.4.4 Ilmoitukset

Projektin perusasetuksilla ei viestejä sähköpostiin tullut lainkaan. Ongelman selvittäminen oli yllättävän hankalaa. Palvelun ohjeistuksissa käytiin lävitse vain maven projektin perustamista, jonka asennuksessa ilmoitukset voi asettaa tulemaan sähköpostiin. Myöskään mikään varsinainen palvelun lisäosa tai asetus ei aluksi tuntunut muuttavan tilannetta. Hieman epäloogisesti ominaisuus täytyi lisätä post-build toimintoihin. Tämän jälkeen sähköpostiviestit saapuivat onnistuneesti perille.



Kuva 15. Cloudbees palvelussa sähköpostin lähettämisen lisääminen.

4.4.5 Hinnoittelu

Cloubees palvelun ilmaismalli on jatkuvasti käytössä ja kuukausittain projekteille annetaan 300 minuuttia ilmaista työaikaa, jota voi käyttää projektinsa ohjelmistojen ajamiseen (Cloudbees, 2013.). Testiasetelman 5 testaajaa ja 3 projektia kuitenkin käyttäisivät ilmaiset minuutit jo muutamassa päivässä.

Cloudbees palvelun ensimmäinen maksullinen versio maksaa 15 euroa kuukaudessa ja antaa kuukausittain 1000 minuuttia ohjelmistojen ajamisen käyttöön sekä 0,106 euroa jokaiselta ylimenevältä tunnilta (Cloudbees, 2013). Mikäli jokainen ohjelmoija lähettää uutta ohjelmakoodia keskimäärin viisi kertaa päivässä ja mitään ongelmia ei ilmaannu voidaan kuukausittaiset kulut arvioida olevan noin 18 euron luokkaa.

4.5 Travis-ci

Travis-ci on jatkuva integraatio palvelu, joka toimii pitkälti lahjoitusten ja sponsoreiden avulla. Travis-ci palvelun käyttö on ilmaista, mutta se on saatavilla vain avoimen lähdekoodin projekteille. Palvelusta on suunnitteilla pro-versio, jonka tulisi mahdollistaa maksulliset suljetut projektit järjestelmässä. Travis-ci palvelua on hinnoittelun kannalta hankalaa verrata muihin palveluihin, mutta sen pois jättäminen olisi ollut myös huono vaihtoehto. Se tarjoaa hyviä mahdollisuuksia mikäli projektin lähdekoodi voi olla avoimesti nähtävillä.

4.5.1 Asentaminen ja versionhallinnan yhdistäminen

Travis-ci käyttää GitHubin OAuth palvelua, jonka avulla oman ohjelmakoodinsa yhdistäminen Travis-ci palveluun on helppoa. Kirjautumisen jälkeen palvelun kotisivulla näkyy viimeaikaisten koodiajojen tilan kenen tahansa palvelua käyttävän ohjelmistoille.

Oikean yläkulman tilipainikkeen alta löytyvä Accounts ohjastaa käyttäjän sivulle, jolla luodaan yhteys omaan GitHub repositorioon. Sync now napin kautta Travis-ci hakee avoimet GitHub projektisi ja näyttää niistä listan. Oleellista tässä listassa on pieni 'on/off' painike oikeassa reunassa, jonka jättäminen 'on' tilaan kertoo Travis-ci-palvelulle, että tämän ohjelmakoodin tilaa halutaan seurata.

Jos ohjelmakoodia päivittää nyt versiohallinnassa Travis-ci yrittää ajaa ohjelmistoon liitetyt koodit, mutta epäonnistuu täysin, koska oletuksena Travis-ci olettaa ohjelmiston olevan Ruby-ohjelmointikielellä tehty. Travis-ci tarvitsee ohjeita toimiakseen oikein ja nämä ohjeet kirjoitetaan ohjelmiston juureen travis.yml tiedostoon.

```
Using worker: ruby5.worker.travis-ci.org:travis-ruby-1

$ git clone --depth=100 --quiet --branch=master git://github.com/ci-project/ci-p
$ cd ci-project/ci-project
$ git checkout -qf 43adf7327e0672d6015711f90c1ad1d68ca082d5
$ rvm use default --install --binary --fuzzy
Using /home/travis/.rvm/gems/ruby-1.9.3-p327
$ ruby --version
ruby 1.9.3p327 (2012-11-10 revision 37606) [i686-linux]
 $ gem --version
 1.8.24
 $ rake
 rake aborted!
 No Rakefile found (looking for: rakefile, Rakefile, rakefile.rb, Rakefile.rb)

 (See full trace by running task with --trace)

 Done. Build script exited with 1
```

Kuva 16. Ilman tarkempaa konfiguraatiota Travis-ci palvelu olettaa ohjelmiston olevan Ruby ohjelmointikielellä tehty

Kun Travis-ci palvelu käynnistää ohjelmistokoodin rakentamisen se lukee tarvittavat tiedot travis.yml tiedostosta. Palvelun asennuksen yhteydessä on tärkeää huomioida yml-tiedoston oikea muoto, koska palvelu automaattisesti yrittää käyttää koodin ajamiseen Ruby asetuksia ilman oikeita ohjeita tai oikein rakennettua yaml-tiedostoa.

Ohjelmiston toiminnan kannalta on tarpeellista asentaa jokainen ohjelmiston kannalta oleellinen asia palvelun virtuaaliseen ympäristöön jokainen kerta, kun ohjelmiston kokoaminen ajetaan. Tiedoston sisältöön pitää asettaa ohjeet Google App Engine -tiedostojen lataamiseen ja purkamiseen sekä ohjeistaa palvelua ajamaan koodi, joka testaa ohjelmiston sisältä löytyvät yksikkötestit.

4.5.2 Testien ajaminen

Travis-ci palvelu ei yritä ajaa testejä ilman tietoa siitä miten ne tulee ajaa. Tiedot asetetaan travis.yml tiedostoon. Testien ajoa varten ei kannata kutsua jokaista erillistä yksikkötestiä erikseen. Käytäntö olisi hankala myös kirjastojen asettamisessa jokaiselle erilliselle testille oikein. Erillisten kutsujen sijasta testit kannattaa ajaa yhdestä luokasta, joka varmistaa, että ympäristö on oikein asetettu ennen kuin testit ajetaan.

```
18 $ wget http://googleappengine.googlecode.com/files/google_appengine_1.7.4.zip -nv
19 2013-03-24 11:22:19 URL:http://googleappengine.googlecode.com/files/google_appengi
20 $ unzip -q google_appengine_1.7.4.zip
21 Please override the script: key in your .travis.yml to run tests.
22
23 The command "false" exited with 1.
24
25 Done. Your build exited with 1.
```

Kuva 17. Testien ajo ilman täysin konfiguroitua travis.yml tiedostoa.

Testejä varten tehty runtests.py luokka lisää ensin järjestelmän käyttöön kotihakemistoon ladatun App Engine kirjaston ja suorittaa sen jälkeen ohjelmakoodista löytämänsä testit. Luokan ajoa varten yliajettu komento travis.yml tiedostossa näkyy liitteessä 1. Oleellista on myös huomioida tarve testiluokassa palauttaa oikea virhetilaa kuvaava viesti järjestelmälle, jotta Travis-ci tietää keskeyttää ohjelmiston ajon tarvittaessa.

```
12 $ wget http://googleappengine.googlecode.com/files/google_appengine_1.7.4.zip -nv
13 2013-02-16 18:12:23 URL:http://googleappengine.googlecode.com/files/google_appengin
14 $ unzip -q google_appengine_1.7.4.zip
15 $ python -m runtests
16 test_init (project.tests.TestProjectFunctions) ... ok
17 test_findAllFor (project.tests.TestTaskFunctions) ... ok
18 test_init (project.tests.TestTaskFunctions) ... ok
19
20 ----------------------------------------------------------------------
21 Ran 3 tests in 0.023s
```

Kuva 18. Testien ajo onnistuneesti läpi Travis-ci Palvelussa.

4.5.3 Käyttöönotto

Travis-ci palvelu ajetaan omassa eristetyssä ympäristössään, joten koodin ajamisen aikana ei sen suorittamiseen voi vaikuttaa. Google App Engine kuitenkin vaatii ohjelmiston päivitykseen käyttäjätunnuksen ja salasanan. Ilman käyttäjän toimenpiteitä kirjautumiseen tarvitaan OAuth2 refresh_token. Tokenin saadakseen käyttäjän tulee ajaa ohjelmiston käyttöönotto App Engine palvelimelle omalta koneeltaan --OAuth2 komennon kanssa. Toiminnon ajon jälkeen käyttäjän omassa kotihakemistossa on tiedosto nimeltä appcfg_oauth2_tokens. Tiedosto sisältää json-koodia, jonka sisältä löytyy refresh_token tieto. Tiedon syöttäminen --oauth2_refresh_token käskyn perään mahdollistaa ohjelmiston päivityksen ilman käyttäjätunnuksen syöttämistä. (Google, 2013.)

Tiedostojen päivityksessä täytyy vielä ottaa huomioon, että aiemmin ladattu kansio google_appengine on nyt ohjelmistokoodin juuressa ja komento yrittää ladata koko kansion mukanaan. Tämä ei onnistu, koska kyseisessä kansiossa on liikaa tiedostoja, joten App Engine ei salli sen asentamista loppuun saakka. Ongelman ratkaisuksi ohjelmiston app.yaml tiedostoon on lisättävä rivi, joka kertoo asennukselle ettei kyseistä kansiota tule ottaa mukaan ohjelmakoodiin.

```
09:15 PM Cloned 10000 files.
09:15 PM Cloned 12000 files.
09:15 PM Rolling back the update.
Error 400: --- begin server output ---
Max number of files and blobs is 10000.
--- end server output ---
```

Kuva 19. App Engine asentaa omia kirjastojaan ohjelmiston mukana Travis-ci palvelussa.

4.5.4 Ilmoitukset

Heti ohjelmiston asennuksen jälkeen kaikki virheelliset yritykset ajaa ohjelmakoodi tai tehdä käyttöönotto lähetetään GitHub-tiliin yhdistettyyn sähköpostiosoitteeseen. Ilmoitus saapuu myös ensimmäisellä kerralla, kun ohjelmisto kääntyy oikein virhetilan jälkeen. Toimintoa voi muokata monipuolisesti travis.yml tiedostossa mahdollistaen viestien lähetyksen esimerkiksi jokainen kerta, kun ohjelmiston rakentaminen ajetaan. Viestien sisältöön on myös mahdollista vaikuttaa. Ilmoituksen voi asettaa tulemaan myös Irc, Campfire, Flowdock, HipChat palveluiden avulla tai lähettää ilmoituksen haluamaansa verkko-osoitteeseen.

4.5.5 Hinnoittelu

Travis-ci on tutkimuksen ainoa täysin ilmainen palvelu, joten hinnoittelu muiden palveluiden kanssa ei ole vertailukelpoinen. Palvelun käytön ehtona kuitenkin on, että käytetty ohjelmistokoodi on avoimesti kaikkien nähtävillä mikä voi olla monien yritysten kannalta huono asia. Travis-ci palvelun ilmainen versio kuitenkin antaa kuvan siitä millaista olisi käyttää myös mahdollisesti tulevaa pro-versiota ja voi hyvinkin olla, että se on erittäin kilpailukykyinen palvelu hinnoittelullaan.

5 Tutkimustulokset ja niiden tulkinta

5.1 CircleCi

CircleCi vastasi hyvin tarkkaan alkuperäisen hypoteesin mukaista palvelua ja käyttöönotto oli testattavista palveluista kevyin, vaikka tosin ongelmat olivat pitkälti samat kuin Travis-ci palvelun kanssa. Ongelmien ilmaantuessa valittavissa oli useampia erilaisia toimintoja ja ratkaisumahdollisuuksia, mikä auttoi huomattavasti ratkaisujen löytämistä. Palvelun oma asennusjärjestelmä ei toiminut palvelun oman mainostuksen mukaisella yksinkertaisuudella, mutta ongelma oli kokonaisuuden kannalta varsin vähäinen. Ongelmia palvelussa oli testien ajamisen ja ohjelmiston käyttöönoton kanssa. Testien ajossa palvelu ei toiminut, koska palvelun oma testijärjestely ei osannut hyödyntää App Enginen kehityskirjastoa suoraan ja korjaukseksi oli tarpeellista tehdä ohjelmistoon erillinen luokka testien ajamista varten. Käyttöönoton kanssa ongelman tuotti ohjelmakoodin lähetys eteenpäin ja tunnistetietojen syöttäminen prosessin aikana.

CircleCi palvelussa ohjelmiston testauksen ja käyttöönoton ajaminen kesti yli viisi minuuttia. Palvelun hitauden perustan voisi olettaa perustuvan yksinkertaisesti palvelun järjestäjän pyrkimyksestä rajaamaan yksittäisen käyttäjän kerrallaan käytössä olevaa resurssimäärää mahdollisimman paljon, koska sitä ei käytetä palvelussa laskutuksen perusteena. Useista vastaavista palveluista poiketen hinnoittelu ei perustu ohjelmien rakentamiseen käytettyihin laskentaresursseihin. Hinnoittelu on kuitenkin kiitettävän suoraviivainen ja perustuu pääasiassa seurattavien projektien määrään, mutta toisaalta hyppy ensimmäiseltä asteelta toiselle on kohtuullisen suuri. Aloittelevan yrityksen kannalta, jolla on viisi sovelluskehittäjää ja kolme erillistä projektia hinnaksi nousee 49 euroa kuukaudessa.

CircleCi palvelun käyttö on suositeltavaa, jos haluaa selkeästi asialle omistautuneen palveluntarjoajan alustan käyttöönsä selkeällä hinnoittelulla. Muihin palveluihin nähden palvelussa on myös selkein käyttöympäristö sekä ohjeistus. Hinnoittelu on tutkimuksessa olevista palveluista korkein, mutta on edelleen suhteellisen pieni kuluerä ohjelmistoprojektin kannalta.

5.2 Cloudbees

Cloudbees palvelu oli ainakin osittain alkuperäisen hypoteesin mukainen. Asennuksessa oli selkeitä hankaluuksia odottamattomilta osin, mutta kuitenkaan mitään ei jäänyt lopussa toteuttamatta. Ongelmia oli alusta lähtien versionhallinnan yhdistämisessä, testien ajamisessa ja ilmoitusten lähettämisessä. Ainoana poikkeuksena palvelun testauksen ongelmien kannalta oli ohjelmiston käyttöönotto App Enginen palveluun, joka toimi suoraan ClickStart asennuksen ansiosta.

Cloudbees palvelussa oli hankalaa saada tulosta lopullisesta rakentamisen kestosta, koska toiminnon aloitusaika vaihteli huomattavasti ja rakentamisesta seurautuneista kuluista ei raportoitu selkeästi. Minuutteja kuitenkin käytetään palvelussa laskutuksen perusteena ja käyttäjälle näytetään kokonaislukuina rakentamista varten jäljellä olevat minuutit. Palvelun kuluja tulee valitun maksupaketin mukaisesti ja viiden kehittäjän ja kolmen projektin yrityksessä on tarpeellista valita 15 euron kuukausipohjainen hinnoittelu. Paketissa saa 1000 minuuttia aikaa rakentamisen suorittamiseen ja ylimenevät minuutit laskutetaan tämän päälle. Aloittelevan yrityksen kannalta, jolla on viisi ohjelmiston kehittäjää ja kolme projektia ja, jotka lähettävät arkipäivisin keskimäärin viisi kertaa päivässä muutoksia ohjelmakoodiin hinnaksi muodostuu noin 18 euroa kuukaudessa.

Cloudbees palvelun käyttö on suositeltavaa, jos haluaa enemmän mahdollisuuksia lisätä ja muokata palvelun käytössä olevia ominaisuuksia ja on valmis käyttämään enemmän aikaa toimintojen kanssa. Palvelussa on huomattavasti enemmän ominaisuuksia, mutta niiden käyttäminen on täysin palvelun käyttäjän taitojen varassa ja ohjeistusta on melko huonosti saatavilla.

5.3 Travis-ci

Travis-ci vastasi hyvin alkuperäistä hypoteesia ja käyttöönotossa ei ollut montaa ongelmallista vaihetta. Palvelun käyttämisen kannalta tärkein yksittäinen huomiotava tekijä on ohjelmakoodin vaatimus olla avoimen lähdekoodin versionhallintaprojektissa. Ongelmat palvelussa ilmentyivät testien ajossa sekä ohjelmiston käyttöönotossa.

Palvelussa ei yritetty tehdä minkäänlaista automaattista tunnistusta ohjelmiston ominaisuuksille ja kaikki oleelliset tiedot oli tarpeellista opastaa palvelun käyttöön erillisen tiedoston kautta. Testien ajamista varten palvelua piti ohjeistaa käyttämään Python-ohjelmointikieltä ja ajamaan erillinen luokka, joka osasi löytää App Enginen kehityskirjastot. Käyttöönoton kanssa ongelmia syntyi ohjelmakoodin lähetyksessä App Enginelle, kun ohjelmiston käyttöönoton aikana oli tarpeellista syöttää tunnistetietoja.

Travis-ci oli hieman yllättäen tutkimuksen palveluista nopein toiminnaltaan. Tämän voisi olettaa johtuvan siitä, että mitään syytä rajoittaa käyttäjien resursseja ei ole. Palvelu toimii lahjoituksien kautta eikä selkeästi yritä toimia yritysperiaatteiden mukaisesti. Voisi helposti olettaa, että kaikkia mahdollisia kuluja olisi karsittu pois, mutta silti ohjelmiston käyttöönotto oli selkeää ja hyvin opastettua.

Travis-ci palvelun käyttöä voi suositella, mikäli ohjelmiston voi asettaa julkiseen versionhallintaan. Palvelun ominaisuudet vastasivat hyvin maksullisia palveluita ja käyttöympäristöä oli selkeästi mietitty. Aloittelevan yrityksen kannalta palvelun käyttämisestä ei synny muita kuluja kuin asennukseen käytetty aika.

5.4 Vertailu

Alustavasti palveluita valitessa ei ilmennyt syytä olettaa, että palvelut olisivat oleellisesti toisistaan erilaisia. Kuitenkin palveluiden erot ilmenivät hyvinkin pian niiden käyttöönoton yhteydessä. Travis-ci ja CircleCi mallisten SaaS palveluiden käyttöönotto oli miellyttävän mutkatonta, koska niiden yhteisö ja ohjeistus keskittyi palvelun käytön ympärille eikä käyttäjältä vaadittu oleellista ymmärrystä palvelun taustalla toimivista järjestelmistä. Tutkimuksen PaaS ratkaisusta vastannut Cloudbees palvelu oli monimutkaista saada toimimaan edes samoilla perustoiminnoilla kuin muut vertailtavat palvelut. Tämä ei silti tehnyt PaaS ratkaisua huonoksi, koska käyttäjän kannalta tärkeitä ominaisuuksia pystyi itse lisäämään tarpeensa mukaan eikä käyttäjä ole samalla tavalla täysin riippuvainen palveluntarjoajasta saadakseen lisää ominaisuuksia alustaansa. Travis-ci palvelu ja CircleCi palvelut eroavat toisistaan erityisesti siinä miten paljon palveluiden käytöstä koituu kuluja ja millaisen projektin palvelussa voi ottaa käyttöönsä.

	CircleCi	**Cloudbees**	**Travis-ci**
Palvelun malli.	SaaS	PaaS	SaaS
Arvioitu kustannus /kk.	49,00 €	18,00 €	0 €
Ongelmien hankaluus yht.	5	9	5
Lopullinen ajon kesto.	5 min. 18s.	1-2 min.	1 min. 12 s.
Kokeiluversion malli.	14 päivää.	Rajoitettu käyttö.	Ilmainen.

Taulukko 1. Palveluiden välisten ominaisuuksien vertailu.

Vertailussa löytyy odottamattomia eroavaisuuksia. Cloudbees palvelun pitäisi olla palveluista nopein, koska sen järjestelmässä ei ole tarpeellista ladata App Enginen kirjastoja erikseen ajon yhteydessä ja osittain tulos voikin selittyä hankalasti ilmoitetulla ajon kestolla. Puolestaan CircleCi ja Travis-ci ajavat lähes samanlaisen toteutuksen läpi lopullisena ajona, mutta CircleCi palvelun ajo kestää yli neljä kertaa kauemmin. Ajamiseen voisi olettaa eniten vaikuttavan ympäristön käytössä olevien palvelimen resurssien määrä ja sen myötä on yllättävää todeta ilmaisen Travis-ci palvelun toimivan tehokkaimmin.

6 Loppupäätelmät ja suositukset

Vertailun kannalta tuloksista ei löydy selkeää voittajaa, jonka ratkaisumalli toimisi paremmin kuin muiden tarjoamat vaihtoehdot. Palvelun valinta tulisi tehdä omien tarpeiden ja resurssien mukaisesti ja siinä mielessä ei myöskään negatiivista vaihtoehtoa tullut esiin. Mikäli ohjelmistoprojektin ei tarvitse perustua suljettuun lähdekoodiin ja budjetti on valmiiksi erittäin rajattu niin Travis-ci palvelun ilmainen ratkaisu riittää hyvin. Kuitenkin ottaen huomioon sen miten pienellä summalla pystyy saamaan käyttöönsä täysimittaisen Jenkins-alustan tai asialle omistautuneen palveluntarjoajan ratkaisun ei kynnys siirtyä maksulliseen palveluun ole suuri. Palvelu kulueränä ei ole minkään tasoinen este jatkuvan integraation käyttöönottamiselle yrityksen ohjelmistoprojekteissa. CircleCi tarjoaa käyttäjälle helppokäyttöista ja erikoistunutta palvelua. Travis-ci puolestaan tarjoaa ilmaista järjestelmää mikäli ohjelmistoprojekti muutoin vastaa vaatimuksia. Cloudbeesin tarjoama monimutkaisempi, mutta monipuolisempi vaihtoehto on hyvä varsinkin mikäli käyttäjä haluaa käyttää muitakin tarjolla olevia palveluita.

Mitään hyvää syytä olla toteuttamatta jatkuvaa integraatiota omassa ohjelmistoprojektissaan ei tutkimuksen perusteella voi sanoa olevan. Jatkuva Integraatio ei ole massiivinen komentoketjujen syöttämisen kurimus jollaiseksi sen voisi helposti mieltää. Selkeästi positiivisena tuloksissa esiintyivät SaaS -pohjaiset ratkaisut, jotka pyrkimät mahdollisimman hyvin vähentämään käyttäjän suorittamaa vaivalloista asetuksien läpikäyntiä. Mikäli ohjelmistoprojekti tekee niin kuin kaikkien ohjelmistoprojektien ehdottomasti pitäisi eli käyttää kehityksen aikana yksikkötestausta ja integraatiotestausta on jatkuvan integraation liittäminen projektiin suositeltavaa.

Tulosten luotettavuuden kannalta on tärkeää ottaa huomioon muutama asia. Testauksen tuloksien hankkimisen aikana olisi ollut hyvä paremmin varmistaa, että testaukset lähtisivät aina samalta viivalta. Testien järjestyksellä on voinut olla pientä vaikutusta löydettyihin tuloksiin. Tällöin tulokseen on saattanut vaikuttaa aiemmin keksitty ratkaisu, jonka vaikutusta ei enään huomioida seuraavan testin ajon aikana. Vaikutukset tuloksiin kuitenkin ovat vähäiset ja ongelmaa on pyritty välttämään mahdollisimman hyvin.

Opinnäytetyön tekemisen aikana on tullut selkeästi esille myös pilvipalveluiden jatkuva kehitys. Muutaman vuoden sisällä nämä palvelut voivat kehittyä erittäin paljon pidemmälle nykyisestä tilanteestaan. Pilvipalveluiden tuottamiseen on tarjolla monia erilaisia vaihtoehtoja ja on palveluntarjoajan sekä kuluttajan yhteinen etu saavuttaa pilvipalveluiden rakenteesta mahdollisimman suuri hyöty. Tracis-Ci palvelusta tuleva pro-versio tai uusi palvelu, joka haluaa tarjota markkinoille paremmin suunniteltua palvelua voi hyvinkin muuttaa tilannetta nopeasti. Tulosten tulkinnan kannalta on siis hyvä huomioida, että jo muutaman vuoden kuluttua tutkimuksen valmistumisesta tilanne on todennäköisesti muuttunut jo huomattavasti. Mikäli tutkimuksen palvelut ovat edelleen muutaman vuoden päästä tarjolla voidaan päätellä niiden palvelun tason olleen myös tarpeeksi korkea ylläpitämään yrityksen jatkuvaa toimintaa.

Palveluissa on kuluttajien kannalta vielä hankalasti esitettyjä piirteitä. Cloudbees palvelussa ollut ClickStart ominaisuus oli selkeästi askel oikeaan suuntaan. Ominaisuus teki sovelluksen käyttöönoton käyttötapauksesta erittäin miellyttävän kokemuksen, koska käyttäjän ei tarvinnut hankkia tietoutta mistään erillisestä mekaniikasta saadakseen palvelun toimimaan haluamallaan tavalla. Käytännössä tämän tuloksena näkyviin tulee selkeitä piirteitä siitä miten ideaalinen palvelu toimisi. CircleCi palvelun automaattinen sovelluksen ominaisuuksien tunnistus ja Cloudbees palvelun tapa yhdistää suoraan App Enginen palvelimelle olisi tässä tutkimuksessa ollut ehdottomasti voittava yhdistelmä. Mikäli projektiinsa voisi vielä liittää kehityskirjastoja suoraan verkkokäyttöliittymän kautta olisi lopputulos pystynyt ajamaan koko ohjelmiston onnistuneesti heti yhdistämisen jälkeen. Tämä tietenkin toimisi täydellisesti vain käytetyn testiohjelmiston kannalta mikä lieneekin syy siihen etteivät palvelut halua käyttää liikaa aikaa yksittäisten käyttötapausten parantamiseen. Erikoistuminen kuitenkin todennäköisesti ohjaisi kuluttajat käyttämään palvelua, joka tekee toiminnan mahdollisimman kivuttomaksi ja varmaksi.

Lähteet

Armbrust, Michael. Fox, Armando. Griffith, Rean. Joseph D. Anthony. Katz, Randy. Konwinski, Andy. Lee, Gunho. Patterson, David. Rabkin, Ariel. Stoica, Ion. Zaharia, Matei. 2009. Above the Clouds: A Berkeley View of Cloud Computing. UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

Buyya, Rajkumar. Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2008. Cloud Computing and Emerging IT Platforms:
Vision, Hype, and Reality for Delivering Computing as the 5th Utility. The University of Melbourne, Australia & Manjrasoft Pty Ltd & Vienna University of Technology.

CircleCi a. 2013. Circle – Continuous Integration made easy.
<https://circleci.com/account/plans> Luettu 2.3.2013.

CircleCi b. 2013. Circle – Continuous Deployment to Google App Engine.
<https://circleci.com/docs/deploy-google-app-engine> Luettu 2.3.2013.

CircleCi c. 2013. Test Environment – CircleCi.
<https://circleci.com/docs/environment> Luettu 2.3.2013.

Cloudbees. 2013. Cloudbees: The Java Paas Company.
<http://www.cloudbees.com/platform/pricing/devcloud.cb> Luettu 2.3.2013.

Duvall, Paul M. Matyas, Steve & Glover, Andrew. 2007. Continuous Integration. Improving software quality and reducing risk. Pearson Education.

Fowler, Martin. 2006. Verkkodokumentti. Continuous Integration.
<http://www.martinfowler.com/articles/continuousIntegration.html> 1.5.2006. Luettu 28.1.2013.

Google. 2013. Verkkodokumentti. Uploading, Downloading and Managing a python App.

<https://developers.google.com/appengine/docs/python/tools/uploadinganapp?hl=en> Luettu 24.2.2013.

Hunt, Andy. Thomas, Dave. 2003. Pragmatic Unit Testing in Java with Junit. The Pragmatic Programmers, LLC.

O'Sullivan, Bryan. 2009. Mercurial: The Definitive Guide. O'Reilly Media.

Peter Mell & Timothy Grance. 2011. The NIST Definition of Cloud Computing (Luonnos). NIST Erityisjulkaisu 800-145.

Liitteet

Liite 1. Testaussuunnitelma

HAAGA-HELIA
ammattikorkeakoulu

Testaussuunnitelma

Panu Salmi

Testaussuunnitelma

Tietotekniikan koulutusohjelma

24.4.2013

1 Yleistä testauksesta

1.1 Johdanto

Tutkimuksessa testataan jatkuvan integraation käyttöönottoa eri pilvipalveluissa käyttäen testausta varten valmistettua ohjelmistoa. Testaus kohdistuu siihen mitkä toiminnot ovat käyttäjän kannalta suoraan toiminnassa ja mitkä vaativat muutoksia palvelun järjestelmässä tai itse ohjelmistossa. Tuloksissa näkyy erityisesti se miten palveluntarjoaja on halunnut toimintonsa tarjota ja minkä tason ymmärrystä käyttäjältä vaaditaan palveluiden käyttöönottamiseen.

1.2 Testauksen kohteet

Testaus kohdistuu kolmeen eri pilvipalveluun. Testauksen palveluiksi valittiin CircleCi, Cloudbees ja Travis-ci. Perusteluina palveluiden valinnalle oli käyttäjälle tarjotut erilaiset hinnoittelu ja palvelumallit. Travis-ci palvelu toimii ilmaisena ja on siksi hyvä kohde tutkimukselle selvittää mitä käyttäjälle tarjotaan ilman kustannuksia. Cloudbees tarjoaa laajaa palvelukokonaisuutta ja sen myötä käyttäjä voi olla kiinnostunut rakentamaan monimutkaisempaakin ohjelmistoprojektia palvelussa. CircleCi tarjoaa erikoistunutta jatkuva integraatio palvelua maksullisena mikä luo paineita tarjota laadukasta ratkaisua.

1.3 Testattavat ominaisuudet

Pääasiallisesti testaus keskittyy käyttötapauksiin ja niiden hyväksymiseen. Testattavat käyttötapaukset ovat versionhallinnan yhdistäminen, testien ajaminen, ohjelmiston käyttöönotto ja ilmoitusten lähteminen. Mikäli järjestelmä on onnistuneesti otettu käyttöön mitataan myös lopullinen järjestelmän ajoon kulunut aika ja sen perusteella arvioidaan kuukausittaista kuluerää palvelusta testitapaukselle. Tulokset koostetaan käyttötapauksittain omiin taulukkoihinsa.

1.4 Ominaisuudet joita ei testata

Testaus sisältää vain testiohjelmiston käyttöönoton palveluissa. Versionhallinnan puolelta testauksessa on vain Git-pohjainen ratkaisu. Testeissä ei huomioida muita ohjelmointikieliä kuin Python tai käyttöönotossa muita alustoja kuin App Engine. Palveluiden ilmoituksista testataan ainoastaan sähköpostien lähtemistä.

1.5 Lähestymistapa

Testausta varten valmistettava testiohjelmisto on Python-ohjelmointikielellä tehty ja ohjelmistolla on käytössään Google App Enginen kehityskirjastot. App Engine itsessään on myös pilvipalvelu, joka tarjoaa palvelussaan mahdollisuutta ylläpitää monipuolisia verkkosovelluksia omassa järjestelmässään. Ennen testien suorittamista varmistetaan ohjelmiston olevan toiminnallinen verkkosovellus, jonka voi käyttöönottaa App Enginen palvelussa. Ohjelmistoon sisällytetään muutama yksikkötesti, jotka palveluiden tulee pystyä onnistuneesti ajamaan. Ilmoitusten testausta varten varmistetaan, että testit epäonnistuvat tarvittaessa ja ilmoitus käyttäjälle saapuu.

Palveluiden testaus etenee järjestelmällisesti alkaen versionhallinnan liittämisestä, testien ajamisesta, ohjelmiston käyttöönotosta ja päättyen ilmoitusten testaamiseen. Testaamisen aikana kirjataan ylös ongelmakohdat ja etsitään niihin mahdolliset ratkaisut. Testin käyttötapauksen toiminta merkataan taulukkoon ensin ilman käyttäjän tekemiä muutoksia järjestelmän asetuksissa tai ohjelmistossa. Mikäli palvelu ei ole toiminut odotetulla tavalla listataan havaitut ongelmat. Ongelman estäessä siirtymisen seuraavaan käyttötapaukseen listataan käyttötapaus testitapaukseksi uudestaan ja käydään läpi tehdyt korjaukset. Toiseen vaiheeseen lisätään myös arvio korjauksen hankaluudesta asteikolla yhdestä viiteen. Testit ovat riippuvaisia toisistaan lukuunottamatta ilmoitusten testausta. Ennen kuin aiempi käyttötapaus on onnistuneesti suoritettu ei seuraavaan käyttötapaukseen voi siirtyä.

Palveluiden eroista johtuen projekti perustetaan ensin eli suoritetaan tarvittavat toimenpiteet järjestelmän asennukseen. Toimenpiteinä on yleisesti palvelun tunnusten luominen ja esimerkiksi alustapohjaisissa palveluissa myös projektin perustaminen palvelun sisällä. App Enginen kirjastojen yhdistäminen palvelun ympäristöön on myös osana asennusta.

1.6 Läpipääsykriteerit

Testin voidaan katsoa menneen onnistuneesti läpi, kun palvelun käytössä voidaan siirtyä seuraavaan vaiheeseen. Ongelmien estäessä testauksessa etenemisen joudutaan testaus palvelun osalta keskeyttämään ja tulokseksi merkata epäonnistunut palvelun käyttöönotto.

2 Testitapausten suunnittelu

2.1 Versionhallinnan yhdistäminen

Versiohallinnan yhdistämisessä palveluun käytetään aiemmin testausohjelmistoa varten tehtyä Git-versionhallintaprojektia ja lisätään se testattavan palvelun käyttöön. Samalla varmistetaan, että versionhallissa tapahtuvien muutoksien jälkeen jatkuvan integraation prosessi käynnistyy yhdistetyssä palvelussa. Prosessin käynnistämisen testaamista varten tehdään muutoin tyhjä muutos versionhallintaan.

Käyttötapauksen kohdalla odotetaan, että versionhallinnan yhdistäminen palveluun onnistuu suoraan palvelun omasta järjestelmästä. On olettavaa, että käyttäjän tulee käynnistää yhdistäminen palvelussa olevan prosessin kautta ja valita haluamansa versionhallintaprojekti. Palvelun ominaista prosessia ei voi laskea ongelmaksi tälle käyttötapaukselle. Ongelmaksi voidaan laskea esimerkiksi Git-palvelun käytön puuttuminen prosessin valinnoista.

2.2 Testien ajo

Toisena käyttötapauksena on ohjelmistoon tehtyjen yksikkötestien ajaminen palvelussa. Versionhallinnan liittämisen jälkeen versionhallintaan suoritetaan muutoksia, jotka kertovat palvelulle, että testit on tarpeellista ajaa. Testien suorittaminen hyväksytysti edellyttää, että testit löytävät ympäristöstään App Enginen käyttämän kehityskirjaston ja suorittavat itsensä onnistuneesti.

Odotettu tulos testien ajamiselle palvelussa on, että testit ajetaan automaattisesti oikein, kun versionhallinta on yhdistetty palveluun ja ohjelmistoon tehdään muutoksia. Ongelmia voi olettaa tulevan palvelun järjestelmän sisäisestä rakenteesta ja itse testien löytämisestä ohjelmiston sisältä.

2.3 Ohjelmiston käyttöönotto

Kolmantena käyttötapauksena on ohjelmiston käyttöönotto Google App Engineen. Käyttötapauksessa testien ajamisen suorittamisen jälkeen pyritään ohjelmisto lähettämään App Enginen järjestelmään testattavana olevan palvelun sisältä. Palvelun pitäisi kyetä tunnistamaan ohjelmisto App Enginen järjestelmän mukaiseksi ja pyrkiä ottamaan ohjelmistokoodi käyttöön.

Odotettu tulos ohjelmiston käyttöönotossa on, että ohjelmiston käyttöönotto tapahtuu automaattisesti, kun testit on ajettu onnistuneesti. Ongelmaksi voisi olettaa muodostuvan lähetyksen tarpeen tunnistaminen palvelun sisältä.

2.4 Ilmoitusten lähettäminen

Ilmoitukset testataan varmistamalla palvelun meneminen virhetilaan. Testauksen osalta huomioidaan ainoastaan sähköpostin lähettäminen palvelun käyttäjälle ongelmatilanteissa, vaikka muitakin vaihtoehtoja olisi tarjolla.

Odotettu tulos käyttötapaukselle on, että virhetilan sattuessa palvelusta lähtee automaattisesti käyttäjälle sähköpostiviesti. Ongelmia ei odoteta esiintyvän tälle käyttötapaukselle.

3 Testiraportit

3.1 Versionhallinnan yhdistäminen

Versionhallinnan yhdistäminen			
	Varsinainen tulos	Ongelman kuvaus	Hyväksytty
CircleCi	Versiohallinta on yhdistettävissä.	Ei ongelmia.	Ok
Cloudbees	Versiohallinta ei ole yhdistettävissä.	Projekti ei tarjoa Git-palvelua vaihtoehdoksi.	Ei
Travis-ci	Versiohallinta on yhdistettävissä.	Ei ongelmia.	Ok

Taulukko 1. Versionhallinnan yhdistäminen ilman muutoksia.

Versionhallinnan yhdistäminen konfiguroinnin jälkeen				
	Tehdyt muutokset	Saavutettu tulos	Hyväksytty	Arvio
Cloudbees	- Jenkins ympäristöön lisättiin Jenkins GitHub plugin sekä GitHub plugin. -Asetettiin GitHub tunnukset lisäosien käyttöön.	Versionhallinta on yhdistettävissä.	OK	4

Taulukko 2. Versionhallinnan yhdistäminen muutoksien jälkeen.

3.2 Testien ajaminen

Testien ajaminen			
	Varsinainen tulos	Ongelman kuvaus	Hyväksytty
CircleCi	Testien ajaminen ei onnistu.	Palvelu ei löydä App Enginen kirjastoa.	Ei
Cloudbees	Testien ajaminen ei onnistu.	Palvelun automaattinen NoseTest koodi ei osaa ajaa testejä oikein.	Ei
Travis-ci	Testien ajaminen ei onnistu.	Palvelu ei yritä ajaa testejä.	Ei

Taulukko 3. Testien ajo ilman muutoksia.

Testien ajaminen konfiguroinnin jälkeen				
	Tehdyt ratkaisut	Saavutettu tulos	Hyväksytty	Arvio
CircleCi	Normaali testikäytäntö yliajettu testiluokalla, joka tietää App Enginen kirjaston sijainnin.	Testit ajettu onnistuneesti.	Ok	2
Cloudbees	Normaali testikäytäntö yliajettu testiluokalla, joka tietää App Enginen kirjaston sijainnin.	Testit ajettu onnistuneesti.	Ok	2
Travis-ci	Lisätty konfiguraatioon tiedot testiluokasta, joka tietää App Enginen kirjaston sijainnin.	Testit ajettu onnistuneesti.	Ok	2

Taulukko 4. Testien ajo muutoksien jälkeen.

3.3 Ohjelmiston käyttöönotto

Ohjelmiston käyttöönotto			
	Varsinainen tulos	Ongelman kuvaus	Hyväksytty
CircleCi	Ohjelmiston käyttöönottoa ei tapahdu.	Ohjelmistoa ei yritetä lähettää App Engineen.	Ei
Cloudbees	Ohjelmiston käyttöönotto onnistui.	Huomaa ottaa pois JUnit koodi ajon lopusta.	OK
Travis-ci	Ohjelmiston käyttöönottoa ei tapahdu.	Ohjelmistoa ei yritetä lähettää App Engineen.	Ei

Taulukko 5. Ohjelmiston käyttöönotto ennen muutoksia.

Ohjelmiston käyttöönotto konfiguroinnin jälkeen				
	Tehdyt ratkaisut	Saavutettu tulos	Hyväksytty	Arvio
CircleCi	- Lisättiin OAuth 2 kirjautuminen App Engineen. - Käyttöönotto App Engineen refresh_token tietoa hyödyntäen.	Ohjelmiston käyttöönotto onnistui.	Ok	3
Travis-ci	- Lisättiin OAuth 2 kirjautuminen App Engineen. - Käyttöönotto App Engineen refresh_token tietoa hyödyntäen.	Ohjelmiston käyttöönotto onnistui.	Ok	3

Taulukko 6. Ohjelmiston käyttöönotto muutoksien jälkeen.

3.4 Ilmoitusten lähettäminen

Ilmoitusten lähettäminen			
	Varsinainen tulos	Ongelman kuvaus	Hyväksytty
CircleCi	Sähköposti saapui.	Ei ongelmia.	Ok
Cloudbees	Sähköposti ei saapunut.	Sähköpostin lähettäminen pitää asettaa erikseen toimintaan.	Ei
Travis-ci	Sähköposti saapui.	Ei ongelmaa.	Ok

Taulukko 7. Ilmoitusten lähettäminen ennen muutoksia.

Ilmoitusten lähettäminen konfiguroinnin jälkeen				
	Tehdyt ratkaisut	Saavutettu tulos	Hyväksytty	Arvio
Cloudbees	Projekin asetuksiin lisätty post-build toiminto lähettämään ilmoituksia.	Sähköposti saapui	Ok	3

Taulukko 8. Ilmoitusten lähettäminen muutoksien jälkeen.

Liite 2. Travis.yml tiedosto.

```
language: python

before_script:
- wget http://googleappengine.googlecode.com/files/google_appengine_1.7.4.zip -nv
- unzip -q google_appengine_1.7.4.zip

script:
- python -m runtests
- google_appengine/appcfg.py --
oauth2_refresh_token=1/5dpDAWoe3xecNf15jr7GpmfuvfAA8yDS7lUnxqHX0-w
update .
```

Liite 3. Circle.yml tiedosto.

```
dependencies:
  pre:
    - curl -O
https://googleappengine.googlecode.com/files/google_appengine_1.7.4.zip
    - unzip -d $HOME google_appengine_1.7.4.zip

test:
  override:
    - python -m runtests

deployment:
  appengine:
    branch: master
    commands:
      - $HOME/google_appengine/appcfg.py --
oauth2_refresh_token=1/5dpDAWoe3xecNf15jr7GpmfuvfAA8yDS7lUnxqHX0-w
update .
```