TAMPEREEN
AMMATTIKORKEAKOULU

Kriittisen järjestelmän testausautomaatioprosessi

Pekka Seppälä

Opinnäytetyö
Helmikuu 2016
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

PEKKA SEPPÄLÄ:
Kriittisen järjestelmän testausautomaatioprosessi

Opinnäytetyö 61 sivua, joista liitteitä 26 sivua
Helmikuu 2016

Tässä työssä käsitellään kriittisen järjestelmän testauksen automatisointia eri testaustyökalujen ja jatkuvan integroinnin yhteydessä. Tämä työ toteutettiin Insta DefSec Oy:lle Suomen Hätäkeskuslaitokselle toimitettavaan ERICA-projektiin, joka tulee toimimaan valtakunnallisena hätäkeskustietojärjestelmänä.

Työ rakentuu vahvasti Apache Mavenin ympärille ja jo ennalta käytössä olevaan asennusautomaatioon, johon työn kuvaama testausautomaatioprosessi on integroitu. Testausautomaatio koostuu kahdesta osiosta: aloitusprosessista ja lopetusprosessista. Aloitusprosessi hoitaa järjestelmän ja testaustyökalujen asentamisen, sekä määrittelyn, että järjestelmän käynnistyksen ja testauksen aloittamisen. Tulosten keruu- ja analysointiprosessi huolehtii testien sammuttamisesta ja tulosten keräämisestä ja tallettamisesta tulostietokantaan, sekä tulosten analysoinnista ja visuaalisesta esillepanosta.

Asiasanat: testaus, kriittinen järjestelmä, automaatio, jatkuva integrointi

ABSTRACT

Tampere University of Applied Sciences
Information and communications technology
Software engineering

PEKKA SEPPÄLÄ:
The Test Automation Process of a Critical Information System

Bachelor's thesis 61 pages, appendices 26 pages
February 2016

This thesis describes the test automation process of a critical system. The process includes automated testing with different testing software and the continuous integration of an information system. This process was implemented for Insta DefSec Co to a project ERICA delivered to Finnish Emergency Response Center Administration. ERICA is the next national emergency response system to be used in Finland.

The thesis is strongly implemented around of Apache Maven and previously implemented continuous integration of software which itself uses Apache Maven. The test automation process implemented is integrated to this previously implemented continuous integration. The test automation process includes two parts: starting process and finishing process. The starting process consists of installation of the system and testing software as well as configuring them. After installation and configuration the starting process launches the system and triggers the test session into motion. Collecting and analyzing of results process handles the shutdown of tests and collecting of results to a result database and analyzing them to visual inspection.

Key words: testing, critical system, automation, continuous integration

SISÄLLYS

LYHENTEET JA TERMIT

DNS	Domain Name System, joka muuntaa verkkoliikenteessä palvelimien IP-osoitteet helpommin luettavaan ja muistettavaan muotoon
Fail fast	Periaate, jolla tarkoitetaan sitä että keskeytetään toiminta välittömästi vakavan virheen tapahtuessa ja täten estetään mahdolliset lisävahingot mitä virheellinen toiminta aiheuttaisi
GC	Garbage Collection, Java-kielen muistinhallintaominaisuus, joka vapauttaa muistia poistamalla sellaisia objekteja, jotka eivät ole enää käytössä
HTTP	Hypertext Transfer Protocol, hypertekstin siirto protokolla, selainten ja web-applikaatioiden tiedonsiirto protokolla
HTTPS	Hypertext Transfer Protocol Secure, hypertekstin suojattu siirto protokolla. HTTP- ja TLS/SSL–protokollien yhdistelmä, jota käytetään selainten ja web-applikaatioiden suojattuun tiedonsiirtoon
JDBC	Java Database Connectivity, ohjelmointirajapinta joka määrittelee miten sovellus pääsee käsiksi tietokantaan
JVM	Java Virtual Machine, virtuaalikone, jolla Java-sovellus pyörii työasemalla
JMX	Java Management Extensions, Java-teknologia, joka tarjoaa valvonta- ja käsittelytyökaluja Java-ohjelmistoille ja -objekteille
Maven	Apache Maven, rakennusautomaatiota, jota käytetään pääasiassa Java-projektien rakentamiseen
Pattern Match	Shell skriptin ominaisuus, jolla voidaan tunnistaa tekstistä määriteltyjä osia
POM	Project Object Model, määrittelee projektin riippuvuuksia toisiin komponentteihin, sekä projektin nimen ja omistajan, tätä käytetään Apache Mavenin toimesta

REST	Representation State Transfer, HTTP–protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen
RPM	Red Hat Package Manager, pakettien hallintajärjestelmä ja tiedostopääte; voi sisältää useita erilaisia tiedostoja tai esimerkiksi valmiiksi asennettavan version
SOAP	Simple Object Access Protocol, tietoliikenneprotokolla etäkutsujen tekemiseen
SSH	Secure Shell, kryptattu tietoliikenneprotokolla, joka sallii kirjautumisen ja suojatun toiminnan suojaamattomassa verkossa

1 JOHDANTO

Työn tarkoituksena oli laatia kriittisen järjestelmän testaamiseen tarkoitettu testausautomaatioprosessi, jolla helpotetaan muiden käyttäjien suorittamaa testaamista tekemällä prosessista suoraviivainen. Lisäksi prosessi tarjoaa versioituvuutta ja vapauttaa reilusti työaikaa muihin tehtäviin. Testaamisen stabiilius tullee merkittävänä lisänä automaation myötä. Automatisointi vähentää käyttäjävirheiden ja unohduksien riskiä. Testausautomaatio- ja asennusautomaatioprosessi, johon testausautomaatio integroitiin, rakentuvat vahvasti Apache Mavenin ja Jenkinsin ympärille. Automaatiota alettiin toteuttaa Suomen turvallisuusviranomaisille tehdyn ERICA-projektin puitteissa Insta DefSec Oy:ssä 2015 syksyn ja 2016 talven aikana. ERICA on Suomen Hätäkeskuslaitokselle toimitettava seuraava valtakunnallinen hätäkeskustietojärjestelmä.

Kriittisen järjestelmän avainominaisuuksia ovat muun muassa luotettavuus, korkea saatavuus ja skaalautuvuus. Testausautomaatioprosessi integroitiin valmiina olevaan järjestelmän asennusautomaatioon, jonka yhteydessä suoritetaan tarvittavien testityökalujen asentaminen, sekä testiskenaarion valinta. Skenaariot on tarkoitettu testaamaan yllä kuvattuja kriittisen järjestelmän ominaisuuksia. Automaatioon sitoutuu vahvasti myös tulosten kerääminen ja niiden jäsentely, sekä osittainen analysointi. Testisession jälkeen tulokset kerätään käyttäjälle helposti saatavaksi ja niistä muodostetaan myös graafinen esitys josta näkee helposti testauksen lopputuloksen. Tavoitteena siis on automaatio, jolla kyetään suorittamaan kriittisen järjestelmän kokonaisvaltainen asennus ja testaus muutamalla napin painalluksella, eikä testitulostenkaan hakemiseen ja analysoimiseen tarvitse käyttää useita tunteja.

2 TYÖKALUT

Suuren järjestelmän asentaminen on pitkä ja monimutkainen prosessi. Asennukseen oli jo valmiiksi kehitetty asennusautomaatio, joka hoitaa järjestelmän asennuspakettien kokoamisen valitusta versionhallinnan haarasta.
Jotta suurta ja monimutkaista järjestelmää kyetään testaamaan mahdollisimman kattavasti, tulee sitä rasittaa useasta eri aspektista. Tähän ei kykene mikään nykyhetken työkalu vaan parhaiten kattava testiskenaario kyetään rakentamaan käyttämällä useampaa testityökalua.

2.1 Jenkins

Jenkins on helppokäyttöinen ja helposti konfiguroitava, sekä laajennettava jatkuvan integroinnin ja asennusautomaation työkalu. Jenkinsin tarkoitus on tarjota jatkuvaa integrointia ja toimitusta ohjelmistokehitysprojekteihin lukuisilla alustoilla. Esimerkiksi Jenkinsillä kykenet luomaan uuden asennuksen kehitysprojektista ja asentamaan sen haluamaasi paikkaan. Tähän prosessiin kyetään integroimaan lukuisia liitännäisiä hoitamaan esimerkiksi JUnit-testit Java-projekteille tai tarkistamaan että versioinnit ovat yhteensopivia projektien pom–tiedostoissa. (Meet Jenkins.)

Meidän tapauksessamme asennukseen valittiin versionhallinnan haara, josta asennus tehdään, ympäristö ja ajettava testiskenaario, sekä sen kesto. Asennus automaatiolle pystytään myös määrittelemään mitä palveluita asennuksessa asennetaan, mikäli koko järjestelmää ei haluta asentaa. Asennusautomaatio lukee asennettavan projektin pom -tiedostosta jokaisen asennukseen halutun komponentin version ja paketoi tätä vastaavan paketin kustakin komponentista. Pakettien paketoimisen jälkeen kyseiset paketit siirretään testiympäristöön, jossa ne asennetaan halutun ympäristön palvelu-klusteriin. Asennuksen jälkeen palvelut käynnistetään ja järjestelmä on käyttövalmis. Asennus voi keskeytyä lukuisissa vaiheissa, mikäli tapahtuu jokin virhe, joka estää onnistuneen asentamisen, noudatetaan siis niin sanottua fail fast -periaatetta. Esimerkiksi väärä versiotieto pom–tiedostossa, joka on ristiriidassa jonkin muun komponentin version kanssa, tai jos asennuspaketteja ei saada siirrettyä laboratorion puolelle, aiheuttaisivat fail fast -tilanteen. Näitä pom–tiedostoja käytetään Apachen Maven työkalulla, jolla kuvataan miten projekti rakennetaan ja mitkä ovat sen riippuvuudet. Pom–tiedostossa kuvataan

projektin riippuvuudet, ulkoiset moduulit ja komponentit XML muodossa (Apache Maven, POM).

Tämän valmiin asennusautomaation kylkeen alettiin rakentaa testausautomaatiota. Asennuksen alussa voidaan valita, mikäli testausautomaatioprosessi halutaan ottaa käyttöön ja millaisella skenaariolla ja kestolla. Jenkinsiin rakennettiin valintaoptiot edellä mainituille ominaisuuksille, sekä itse testityökalujen asentaminen määritellyille työasemille. Varsinaiset testit käynnistettäisiin vasta siinä tapauksessa, mikäli järjestelmän komponentit ovat asentuneet ja käynnistyneet oikein ja testaaminen olisi mahdollista, sekä tuottaisi haluttua tulosta.

2.2 Testaustyökalut

Järjestelmän kattavaa testaamista varten käyttöön on valittu kolme eri testaustyökalua; JMeter web-käyttöliittymän kuormitus testaamiseen, Squish käyttöliittymien rasitustestaukseen ja vasteaikojen mittaamiseen, sekä korkean saatavuuden testipenkki (Koistinaho 2013, 51), joka on kehittynyt paljon yleiskäyttöisemmäksi testaustyökaluksi kuin vain korkean saatavuuden testaamiseen, jatkossa työkaluun viitataan testipenkkinä.

2.2.1 JMeter

JMeter on Apachen kehittämä täysin Java-pohjainen avoimenlähdekoodin testityökalu, joka oli alun perin tarkoitettu testaamaan web-applikaatioita, mutta on sittemmin laajentunut kokonaisvaltaisemmaksi testityökaluksi. JMeter sopii erityisen hyvin jatkuvaan kuormitustestaukseen, mihin se on alun perin suunniteltu, simuloimalla kuormaa yhdelle tai useammalle palvelimelle, tietoliikenneyhteyksille tai yksittäiselle objektille. Kuormitusta kyetään toteuttamaan lukuisille palvelin- ja protokollatyypeille, kuten HTTP, HTTPS, SOAP, REST, tietokannat JDBC-rajapinnan kautta ja sähköpostiprotokollat. Testaaminen onnistuu useammalla eri työasemalla käyttämällä samaa testisuunnitelmaa, joka jaetaan JMeter-isännältä sen lapsille, jotka toimivat näillä työasemilla. Tämä on yksi tapa simuloida useamman käyttäjän kuormaa testattavalle palvelulle. Usean käyttäjän kuorman simulointi onnistuu myös jakamalla säikeitä omiin ryhmiinsä ja suorittamalla näillä ryhmillä eri toimintoja.

JMeterillä testaaminen tapahtuu luomalla testisuunnitelma, joka sisältää yksittäisiä testejä, jotka puolestaan voivat koostua eri toiminnoista. Näitä testisuunnitelmaan luotuja toimintoja voi käyttää useammassa eri testissä. Testisuunnitelman sisällön voi jakaa edellä mainittuihin säieryhmiin, jotka siten suorittavat tätä kyseistä toimintoa. (JMeter esittely.)

2.2.2 Squish

Squish on hyvin laaja ja joustava käyttöliittymien automaatiotestaustyökalu. Squishille voi määritellä useanlaisia testiskenaarioita ja – askelia esimerkiksi JavaScript–kielellä. Näihin testeihin voidaan sisällyttää tiettyjä vaatimuksia, jotka pitää täyttyä, jotta testi menee hyväksytysti läpi. Käyttöliittymälle ajettavat testiajot koostuvat useasta tällaisesta testiaskeleesta, joiden yhteenlasketuista tuloksista lasketaan ajolle onnistumisprosentti. Täten indikoidaan käyttäjälle selkeästi mikä on käyttöliittymän yleinen tila kyseisellä testiskenaariolla. (Squish Features and Benefits.)

Squish:lle on mahdollista toteuttaa lisää ominaisuuksia, kuten käyttöömme on toteutettu käyttöliittymästä vasteaikojen mittaaminen. Vasteajan mittaus tapahtuu simuloimalla näppäimistön ja hiiren eleitä samoilla mekanismeilla mitä käytetään oikeankin käyttäjän tilanteessa. Hiiren tai näppäimistön painalluksesta mitataan vasteaika ennalta määriteltyyn tulokseen, jota verrataan käyttöliittymän tuottamaan vasteeseen. Esimerkiksi jos simulaatio täyttää hakukenttään hakusanan ja kentän tulisi ehdottaa vasteita kyseiselle sanalle. Squish mittaa kuinka kauan sanan kirjoittamisesta menee siihen että käyttöliittymä tarjoaa määritellyn vasteen. Molemmissa tapauksissa voidaan merkitä testitapaus onnistuneeksi tai epäonnistuneeksi, mikäli tulos oli haluttu. Vasteaikojen mittaaminen ei ole aina suoraviivaista käyttöliittymissä joiden monet eri toiminnallisuudet tuotetaan omissa säikeissään. Tällöin ei voida taata käyttöliittymän valmiutta ja responsiivisuutta kun jokin tietty ominaisuus on käytössä, sillä jokin toinen kriittinen ominaisuus voi olla vielä piirtämättä.

Vasteaikojen mittaamisen lisäksi avainominaisuuksia on testiajoista tehtävät tallenteet, kuvakaappaukset ja itse-toteutettu lisäominaisuus säievedosten ottaminen virhetilanteissa. Tallenteet ja kuvakaappaukset voivat indikoida käyttäjälle mahdollista käyttöliittymäkomponenttien virheellistä toimintaa. Esimerkiksi jos jokin komponentti on näkyvissä hetkellä, jolloin sen ei kuuluisi. Säievedos otetaan vain tilanteissa, joissa testiajo koh-

taa vakavan virheen mikä estää sen hetkisen testiajon jatkamisen. Esimerkiksi säikeen pysähtyminen olisi vakava virhe, koska se estäisi koko sovelluksen käytön. Nämä antavat testaajille ja kehittäjille arvokasta lisätietoa ongelman laadusta ja mahdollisista syistä.

2.2.3 Korkean saatavuuden testipenkki

Testipenkki on Java-pohjainen testaustyökalu, jolla päästään käsiksi suoraan järjestelmän komponenttien rajapintoihin suorittamaan suoria palvelukutsuja tai muokkaamaan suoraan järjestelmän käyttämää tietomallia. Testipenkin ympärille on luotu Miniclient-sovellus, joka koostuu testattavan järjestelmän eri komponenteista. Kyseisten komponenttien kautta suoritetaan testit. Nämä riippuvuudet määritellään pom–tiedostoon. Testipenkki kykenee tallentamaan testituloksia JDBC-yhteyden avulla ja piirtämään testien rakenteen. Rakenteen piirtäminen on hyvinkin hyödyllinen kun suunnitellaan testien toimintaa tai selvitetään mahdollisia virheitä. JDBC-yhteyden avulla puolestaan voidaan tallentaa helposti testituloksia tietokantaan myöhempää analysointia varten (JDBC-API, wikipedia). Testipenkin tietomalli toteutetaan tietokantaan yhteensopivaa koodia generoimalla tietokantakoodi Datanucleuksella, jolloin testipenkin käyttämä tietokanta kyetään vaihtamaan. Tietokannan vaatimuksena on tuki JDBC:lle.

Testipenkki toiminta perustuu testiryhmiin. Ryhmät voivat olla erityyppisiä: tavallisia, malliehdollisia, todennäköisyyteen perustuvia tai satunnaisia (Koistinaho 2013, 22). Näihin ryhmiin lisätyt testit puolestaan hoitavat varsinaisen testaamisen. Näiden ryhmien avulla voidaan rakentaa esimerkiksi tilakone, joka simuloi oikean käyttäjän toimintoja. Tavalliset ryhmät suorittavat annetut testit järjestyksessä. Malliehdolliset testiryhmät toimivat annettujen malliehtojen puitteissa, joiden perusteella suoritetaan ryhmän sisältä yksittäinen testi. Todennäköisyyteen perustuvista testiryhmistä suoritetaan jokin testiryhmän testeistä annetulla prosentuaalisella todennäköisyydellä, kun taas satunnaisuuteen perustuvissa suoritetaan testi satunnaisesti. Yksittäinen testi voi olla myös jokin testiryhmä.

Useita miniclienttejä voidaan käynnistää simuloimaan useamman käyttäjän kuormaa tai vain kuormittamaan järjestelmää laajemmin. Testausautomaatioprosessi luottaa hyvin pitkälti korkean saatavuuden testipenkin avulla luotuihin testeihin ja sen toimitaan. Tes-

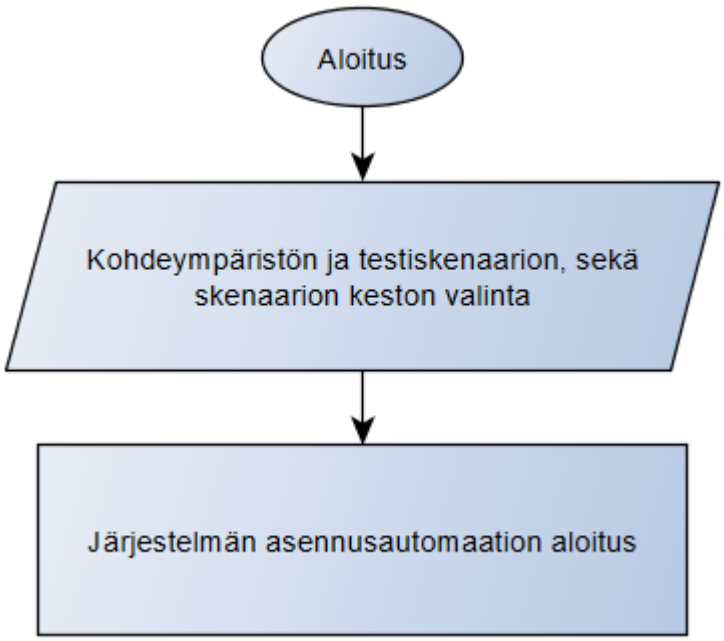
tipenkillä kyetään kuormittamaan järjestelmää kaikkein kattavimmin, sekä monitoroimaan järjestelmän eri palveluiden tilaa ja suorituskykyä.

3 AUTOMAATIORUNKO

Koko prosessia varten luotiin automaatiorunko, joka on jaettu kahteen vaiheeseen; aloitusprosessi ja tulosten keruu- ja analysointiprosessi. Aloitusprosessi kuvastaa järjestelmän ja testityökalujen asennusta, kyseisten asennusten konfigurointia ja järjestelmän, sekä testisession käynnistämistä. Tulosten keruu- ja analysointiprosessi puolestaan nimensä mukaisesti toteuttaa testiskenaarion lopetuksen. Eli testien alas-ajon ja tulosten keräämisen ja esittelyn käyttäjälle.

3.1 Aloitusprosessi

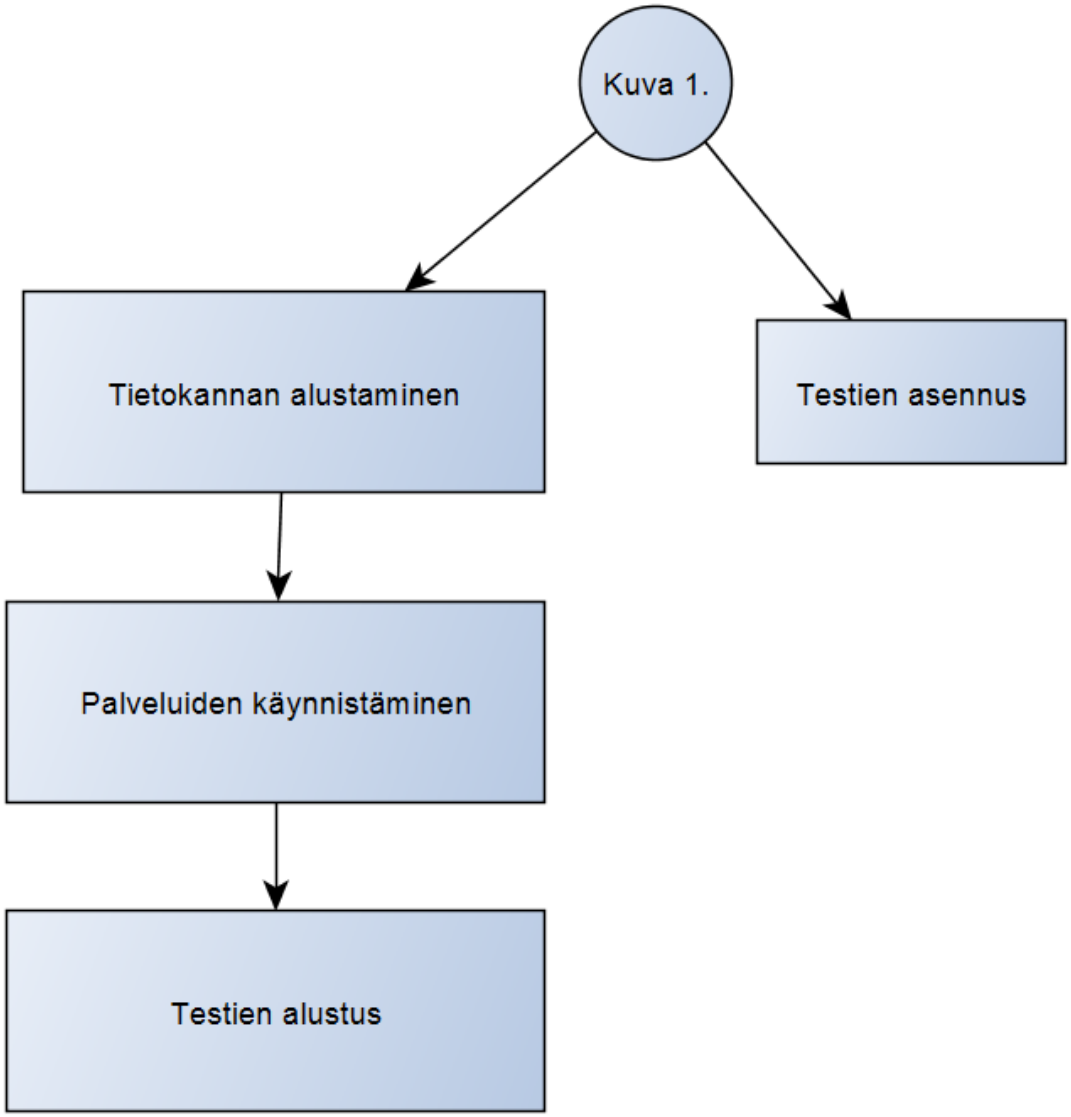
Aloitusprosessissa kartoitetaan testiajon tarpeet ja määrittely. Testausautomaatioprosessi on integroitu järjestelmän asennus- ja integraatioautomaatioon (kuva 1). Prosessi käynnistetään, mikäli käyttäjä antaa tarvittavat parametrit.



KUVA 1. Aloitusprosessin määrittely vaihe

Testiajon tarpeita ovat järjestelmän osat ja versiot, jotka määritellään osana asennusautomaatiota. Asennettava versio määritellään annetun versionhallinnan haaran mukaan. Pom-tiedostosta luetaan komponenttien määritellyt versiot joita käytetään järjestelmän asentamiseen. Testisession määrittelyyn liittyy olennaisesti haluttu testiskenaario, joka sisältää tiedot käytettävistä testityökaluista, työkaluilla ajettavat testit ja niiden määrittely. Testien määrittelyn lisäksi annetaan sessiolle kesto päivinä ja tunteina (kuva

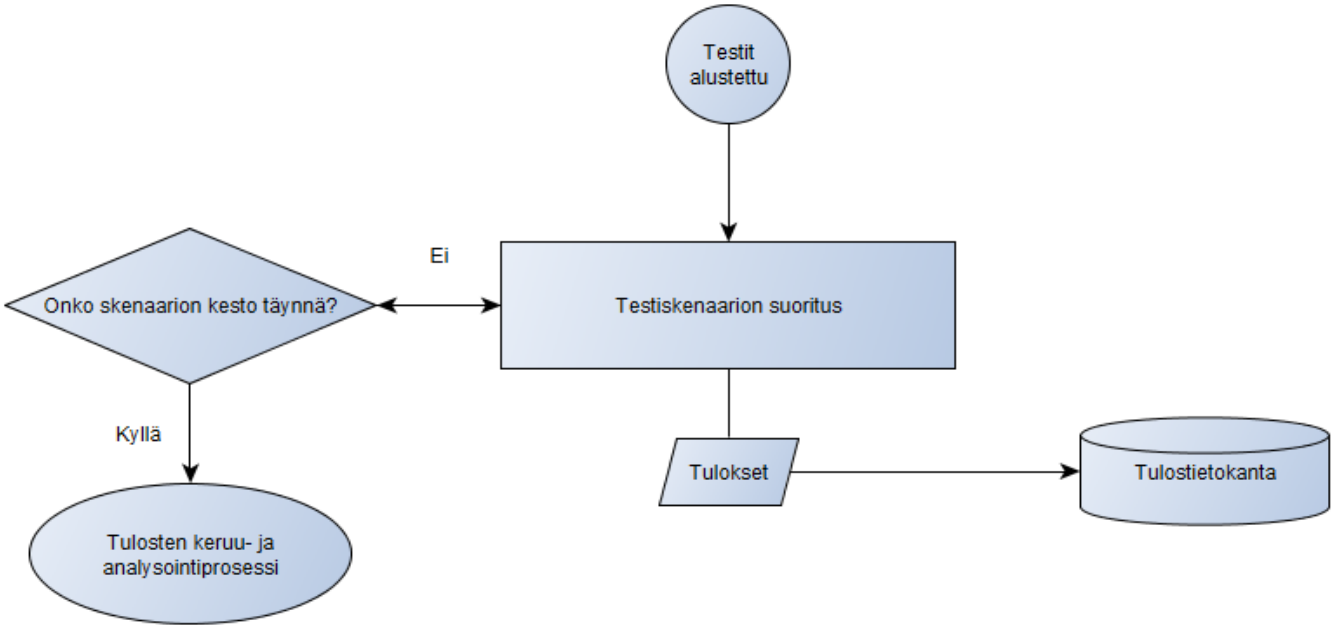
1). Kesto kirjoitetaan palvelimelle (Liite 1), josta sitä seurataan ja ajan täytyttyä aloitetaan lopetusprosessi. Järjestelmän alustaminen on rinnakkaista testien asennuksen kanssa (kuva 2).



KUVA 2. Testien asennus ja alustaminen

Järjestelmän asennusautomaation loppupuolella käynnistetään palveluklusterit juuri asennetussa kohdeympäristössä. Jokaiselle palveluklusterille on määritelty tietyt työasemat jotka näkevät kyseisen palveluklusterin. Palveluiden luotettavan toiminnan vaatimuksena on oikein alustettu tietokanta, joka alustetaan palveluiden käynnistyksen alkupäässä. Mikäli tietokannan alustaminen ei valmistu määräajassa, ei käynnistetä loppuja palveluita eikä testejä, vaan prosessi keskeytetään (Liite 2). Järjestelmän asennusautomaation aikana asennetaan myös halutut testaustyökalut työasemille (kuva 2). Järjestelmän käynnistymisen jälkeen määritellään testeissä käytettävät ympäristökohtaiset parametrit (Liite 3), joita ovat esimerkiksi JMX -valvontaan käytettävät osoitteet.

Kun järjestelmän tietokanta on alustettu ja palvelut käynnistyneet, sekä testityökalujen parametrit on asetettu oikeaksi, voidaan aloittaa testisessio. Testisession aikana tuloksia tallennetaan osittain ajoaikaisesti tulostietokantaan ja osittain jälkikäteen (kuva 3).

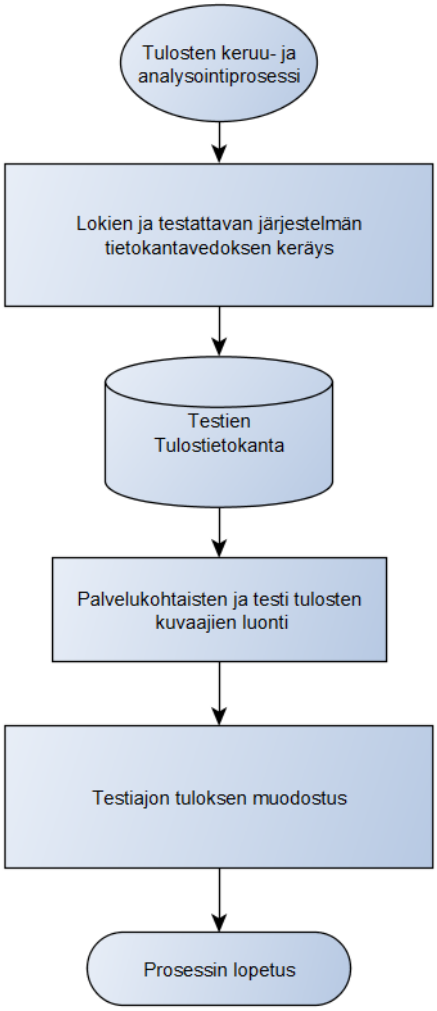


KUVA 3. Testiskenaarion suorittaminen ja aloitusprosessin päättyminen

Testisession aikana testipenkin tuottamia tuloksia tallennetaan suoraan tulostietokantaan (kuva 3). Muiden testityökalujen tuottamat tulokset parsitaan tulosten keruu- ja analysointiprosessin alkuvaiheessa erillisistä tulostiedostoista tietokantaan. Testiskenaarion suorituksen kesto valvotaan palvelimelle talletetusta ajasta, joka aloitusprosessin alkuvaiheessa määriteltiin (kuva 1). Ajan täytyttyä ajetaan testityökalut alas ja käynnistetään lopetusprosessi.

3.2 Tulosten keruu- ja analysointiprosessi

Tulosten keruu- ja analysointiprosessi on pidempi ja kuvaavampi nimi lopetusprosessille. Tässä prosessissa käsitellään tulokset mahdollisimman lukukelpoiseen muotoon henkilötyötuntien säästämiseksi (kuva 4).

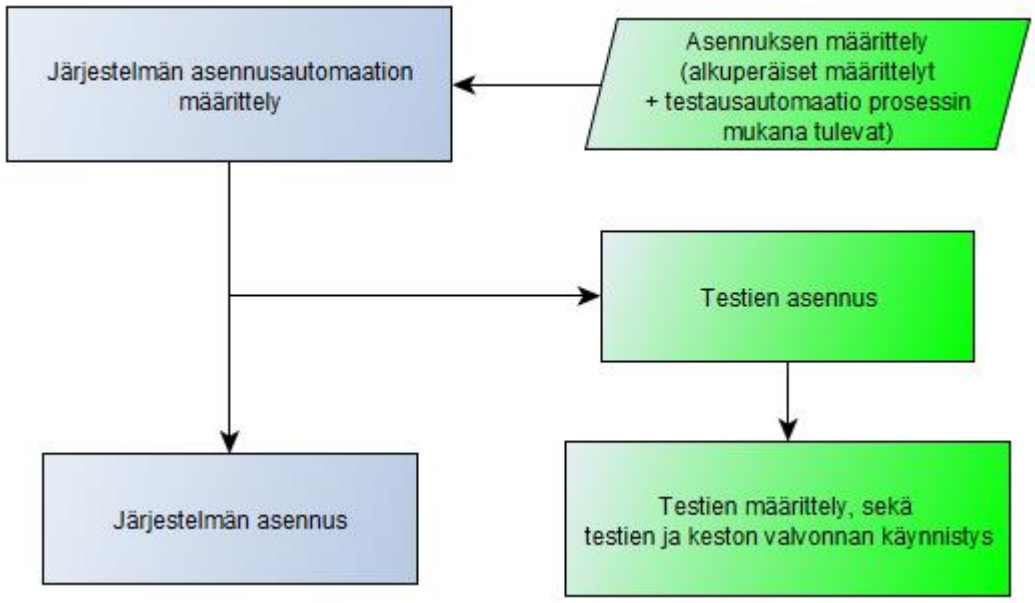


KUVA 4. Tulosten keruu- ja analysointiprosessi

Testiautomaation loppuosa hoitaa tulosten keruu- ja analysointiprosessi jonka tarkoituksena on kerätä testityökalujen tuottamat tulokset ja lokit talteen käyttäjän analysoitavaksi. Analysoinnin helpottamiseksi osana prosessia muodostetaan valmiita kuvaajia ajon tuloksista, täten nopeuttaen työskentelyä ja vapauttaen työaikaa sellaisiin tehtäviin, joita ei ole voitu automatisoida. Testityökalut tuottavat tuloksia eri muodoissa, joten näiden kerääminen manuaalisesti lukuisista eri paikoista on työlästä. Itse tulosten analysointiin kuluu myös helposti useampi tunti, joten on helpompaa, kun kaikki tarjolla oleva materiaali tarjotaan yhdessä paketissa ja osa siitä jo valmiiksi visuaalisesti jäsenneltynä. Tulokset myös tallennetaan, jotta niitä voi tarkastella jälkeenpäin ja vertailla muiden ajojen tuloksiin. Tallennussijaintina toimii tulostietokanta, jossa on ympäristökohtaiset taulut, jonne jo osa tuloksista on valmiiksi tallennettu niin sanotussa raakamuodossa ennen kuin niitä on analysoitu.

4 ALOITUSPROSESSIN VAIHEET

Tässä luvussa esitetään testiautomaation aloitusprosessiin kuuluvat vaiheet tarkemmalla ja teknisemmällä tasolla. Luvussa käsitellään uuden toteutuksen lisäksi myös vanhaa toteutusta, johon testausautomaatioprosessi on integroitu (kuva 5).



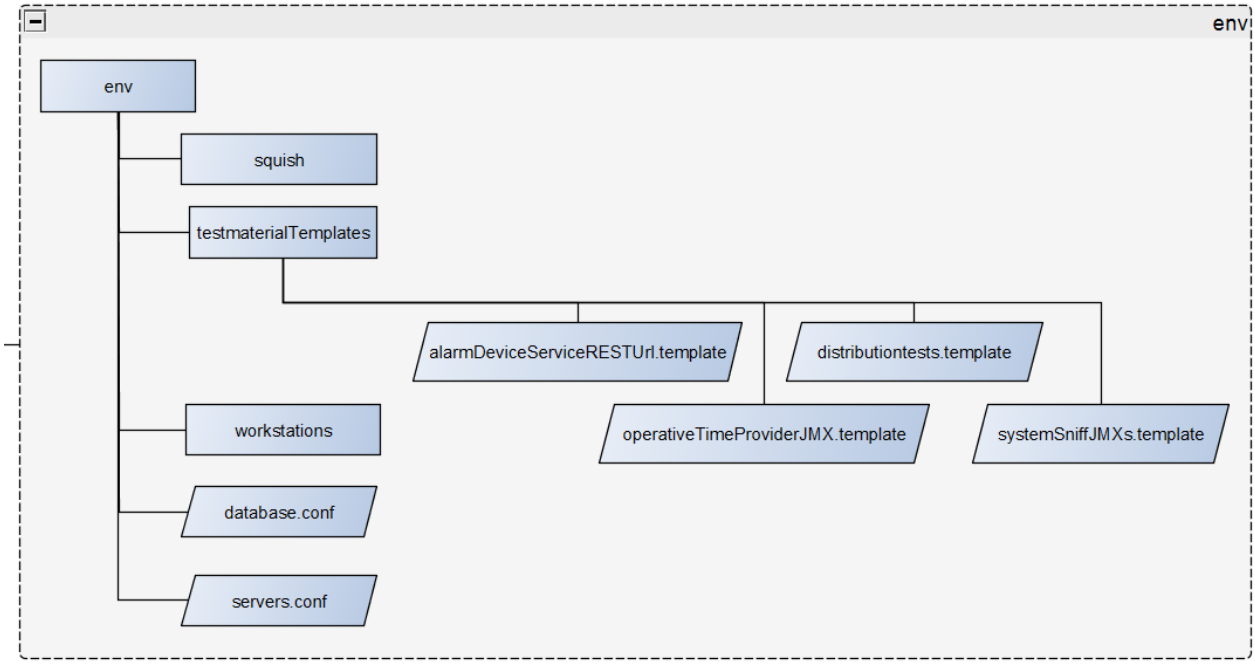
KUVA 5. Jenkinsillä tapahtuva määrittely, oikealla olevat kolme laatikkoa ovat testiautomaatioprosessin mukana tulleita lisäyksiä

4.1 Määrittely

Niin testiskenaarion kuin järjestelmäasennuksen määrittely tapahtuu Jenkinsillä. Alla kuvataan kukin määrittely tarkemmin. Järjestelmäasennus suoritetaan käyttäjän antamasta haarasta haluttuun ympäristöön, jonka palvelimille palvelut asennetaan ympäristökohtaisen määrittelyn mukaan. Asennukseen voidaan määritellä komponenttikohtaisesti tuleeko kyseinen komponentti mukaan asennukseen. Rajoittamalla komponenttien määrää, nopeutetaan komponenttien paketointi ja asennus vaihetta.

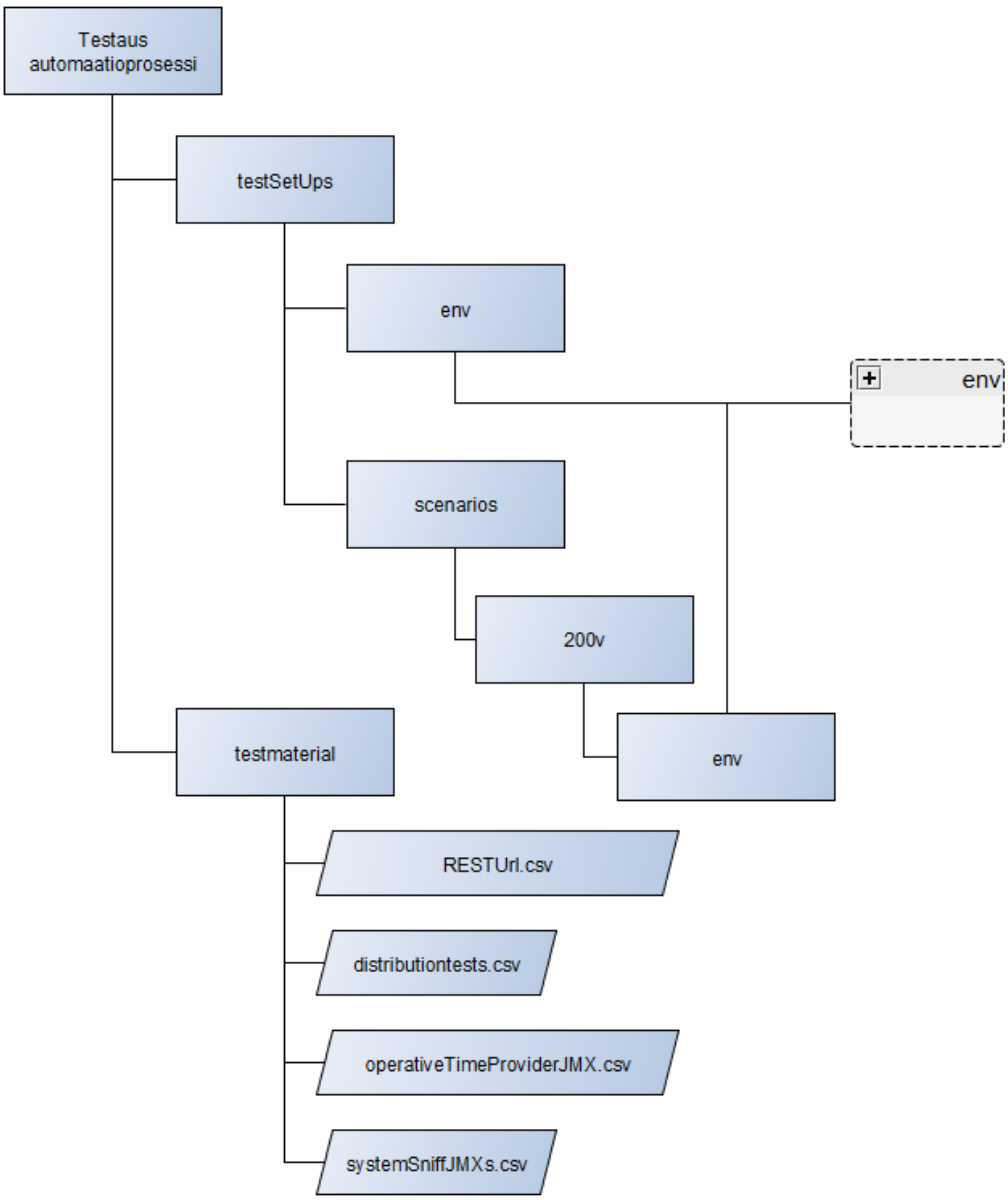
4.1.1 Testiskenaarion valinta

Jokainen testiskenaario määritellään ympäristökohtaisesti, joten ympäristön valinnan yhteydessä pitää olla myös varma, että kyseiselle ympäristölle on luotu testiskenaario (kuva 6).



KUVA 6. Ympäristökohtainen määrittely

Ympäristökohtaiseen määrittelyyn sisältyviä tiedostoja käytetään yliajamaan ympäristöille olevat vakioidut määrittelyt (kuva 7). Näitä määrittelyjä ovat tietokantayhteydet, palvelinkonfiguraatiot, testimateriaalit ja työasemat.



KUVA 7. Testiskenaarioiden kansiorakenne

Ympäristökohtainen määrittely (kuva 6) sisältää kyseiseen ympäristöön ja skenaarioon ajettavan testauksen ja asennuksen tarkempaa määrittelyä. Asennuksen tarkempaan määrittelyyn käytetään ympäristön kansiosta löytyvää tiedostoa (kuva 6, servers.conf), joka sisältää ympäristöön asennettavat ja käynnistettävät palvelut. Tämä tiedosto korvaa normaalisti käynnistettävän määrittelytiedoston, samalla yliajetaan oletus asetukset workstations-kansiosta (Liite 4). Testauksen tarkempi määrittely luetaan lopuista kansioista ja tiedostoista. Työasemakohtaiset testikonfiguraatiot (kuva 6, workstations- ja squish-kansiot), ympäristökohtaiset testimateriaalit (kuva 6, testmaterialTemplates) ja tietokantayhteyden määrittely (kuva 6, database.conf) yliajavat aiemmat määrittelyt (Liite 3).

Testiskenaariolla tarkoitetaan testien ja testityökalujen yhdistelmää, jota käytetään testausautomaatioprosessissa. Skenaarioita on tarkoitus olla useita, joilla voidaan kuormittaa järjestelmän eri osa-alueita vaatimuksien rajoissa. Kriittisen järjestelmän on erittäin tärkeää säilyttää toimintakykynsä pitkien ajanjaksojen ylitse ja normaalia raskaamman, jopa pitkäkestoisen kuorman alla. Skenaario määritellään ympäristökohtaisesti (kuva 7) ympäristön työasemille (kuva 6), määrittely tapahtuu työasemakohtaiseen properties – tiedostoon johon kirjoitetaan millä testaustyökalulla käynnistetään mitäkin testejä ja montako kappaletta. Työasemakohtaiset määrittelytiedostot on nimetty työaseman DNS-nimen mukaan, täten automaatio tietää ottaa etäyhteyden ssh:n ylitse kyseiselle työasemalle. Määrittelytiedostosta (esimerkki 1) luetaan shell–skriptillä jokainen rivi ja rivin sisällön indikoimat testit käynnistetään (Liite 5).

```
console\\do Stuff=3
service\\stress Service=1
jmeter=env11
squish=config.properties
```

ESIMERKKI 1. Työasemakohtainen testien määrittely tiedosto: keijo.properties

Oikean testin tunnistaminen tapahtuu pattern matchin avulla. Testipenkin testit määritellään muodossa hakemisto\\testin nimi=kpl, jossa hakemisto on polku testipenkin bin–hakemiston alla. Testin nimi viittaa testin parametreihin, jotka luetaan testin nimen mukaisesta properties–tiedostosta. Käynnistettävän testin määrän puolestaan antaa kpl-merkintä yhtäsuuruusmerkin jälkeen. JMeterin käynnistäminen luetaan muuten samalla tavalla, mutta sille ei tarvitse antaa käynnistyspolkua, sillä se on sama kuin asennuspolku, joka on määritelty asennusautomaatioon. Squishille puolestaan määritellään määrittelytiedosto, josta luetaan Squish-testiajolle määritellyt parametrit. Näiden parametrien mukaan parsitaan kokoon erillinen tiedosto, johon kirjoitetaan määritellyt testiprofiilin mukaiset parametrit (Liite 6). Tätä parsittua tiedostoa käytetään Squish-testiajon käynnistämiseen.

4.1.2 Testisession keston määrittely

Testisession kesto määritellään Jenkinsillä koko prosessin alussa. Kesto annetaan päivinä ja tunteina joka kirjataan ylös palvelimelle (Liite 1), josta sitä verrataan reaaliaikaan (Liite 7). Testisession keston määrittely on hyvin tärkeää, sillä se toimii toisena indi-

kaattorina testien käynnistämiselle (Liite 8). Mikäli testipenkkiä ole paketoitu tai testisessiolle ei ole annettu kestoa, ei myöskään testejä käynnistetään (Liite 8). Tällä menettelyllä ehkäistään käyttäjävirheistä mahdollisesti sattuvia ongelmia. Tällainen ongelma on esimerkiksi tilanne, jossa palvelujen käynnistäminen ei jostain syystä onnistu asennusautomaation skripteillä. Tässä tapauksessa mikäli testausautomaatio on aloitettu, se odottaa, että tietokantojen alustus on käynnistetty ja mikäli ei ole, automaatio sammuttaa palveluklusterit, tyhjentää ja luo tietokannat uudelleen sekä käynnistää palveluklusterin (Liite 2).

4.2 Asennus

Prosessin sisältämä asennusosuus jakautuu kahteen kokonaisuuteen: jo ennalta rakennettuun järjestelmän asennusautomaatioon ja tähän integroituvaan testien asennukseen. Asennusprosessin osana on myös niin palveluiden kuin testityökalujen määrittelyjen muokkaaminen ympäristöön sopiviksi. Näitä määrittelyjä ovat esimerkiksi palvelukutsujen osoitteet ja palvelujen portit, sekä ympäristökohtaiset ominaisuudet. Järjestelmän asennusautomaatio noudattaa fail fast -periaatetta, jolloin jonkin onnistuneen asennuksen vaarantavan virheen sattuessa koko prosessi keskeytetään lisävahinkojen ehkäisemiseksi.

4.2.1 Järjestelmä

Määrittelyn jälkeen käynnistetään asennus, joka ensiksi tarkistaa kaikki komponenttiversiot ehkäistäkseen ristiriitaisia riippuvuuksia, jotka estävät komponenttien toiminnan. Tämän jälkeen komponenteista tehdään rpm -paketit, jotka kopioidaan ja asennetaan kohdeympäristön palvelimille konfiguraation mukaan. Samalla rinnakkaisesti kohdeympäristön työasemille asennetaan sovellukset järjestelmän käyttöä varten. Molemmat asennusvaiheet toteutetaan vuorotellen ja vaiheittain jokaiselle palvelimelle. Ensimmäiseksi poistetaan edellinen asennus ja tämän jälkeen asennetaan uusi. Molemmissa nojataan ympäristölle määriteltyihin työasemamäärittelyihin (esimerkki 2) ja palvelinmäärittelyihin (esimerkki 3).

tarantino
keijo
batman

Esimerkki 2. Ympäristökohtainen työasemien määrittelytiedoston sisältö, jossa työasemien DNS-nimet

palvelin1/11.222.3.44 palvelu-A palvelu-B palveluC/down
palvelin2/55.666.7.88 palvelu-D palvelu-E

Esimerkki 3. Ympäristökohtainen palvelimien palveluiden määrittely

Asennuksen jälkeen käynnistetään tarvittavat palvelut ja alustetaan tietokanta, jonka jälkeen käynnistetään loputkin palvelut. Mikäli kannan alustus ei valmistu neljässä tunnissa, keskeytetään prosessi ja loppuja palveluita ei käynnistetä automaattisesti. Palvelinten määrittely tiedostoon voidaan myös määritellä, mikäli jonkin tietyn palvelun ei haluta käynnistyvän tietokannan alustuksen jälkeen - tällöin palvelun perään tulee kirjoittaa /down (esimerkki 3, palvelu-C).

4.2.2 Testaustyökalut

Testaustyökalujen asentaminen nojaa automaatioprosessissa mukana oleville työasemille määriteltyihin testeihin (esimerkki 1). Määrittelyn mukaan työasemille asennetaan testaustyökalut. Testipenkki on paketoitu asennusautomaation yhteydessä ja asennus tapahtuu vain siirtämällä paketti kohdetyöasemalle ja purkamalla se. Testipenkki käyttää ympäristökohtaisia parametreja (kuva 6, testMaterialTemplates -kansio) ja ne ylikirjoitetaan testimateriaaleihin (kuva 7, testmaterial –kansio) järjestelmän käynnistymisen jälkeen.

JMeter (Liite 9 & 10) ja Squish (Liite 11 & 12) työkalut noudattavat samanlaista asennusmallia kuin testipenkki. Mikäli työasemalle on määritelty jommankumman työkalun testejä, asennetaan kyseessä oleva työkalu työasemalle. JMeter löytyy laboratoriotyökalujen versionhallinnasta samoin kuin Squish-automaation käyttämät skriptit. Molempien asennus suoritetaan kopioimalla pakatut tiedostot työasemalle ja purkamalla paketti. JMeterin käyttämä testisuunnitelma kopioidaan myös tässä vaiheessa JMeter-testejä suorittavalle työasemalle (Liite 9). Squish-testiajon parametrit kopioidaan järjestelmän asennuksen aikana (Liite 13).

4.3 Järjestelmän käynnistyksen tilan seuranta

Testaustyökalujen ja järjestelmän asennuksen jälkeen seuraa palveluiden käynnistäminen ja tietokannan alustaminen, mikäli näin on määritelty (Liite 8). Testaamista luonnollisesti ei voida aloittaa ennen kuin tietokanta on alustettu, mikäli testaus riippuu siitä. Kaikkien palveluiden tulee olla päällä, sekä kaikkien palveluklusterien on oltava synkronoitunut keskenään (Liite 8). Ensimmäisessä avainasemassa on tietokannan alustaminen, jota testausautomaatioprosessi seuraa, mikäli kanta on kuitenkin jo alustettu, ja prosessille ei ole määritelty uudelleenalustusta, voidaan tämä vaihe ohittaa (Liite 2). Tietokannan alustusta seurataan siten, että ensimmäiseksi tulee havaita tietokannan alustavan palvelun käynnissä olo, tämä tapahtuu kyselemällä kyseisen palvelun palvelimelta onko tietokannan alustavan palvelun nimellä olevaa prosessia pyörimässä. Mikäli prosessi on käynnistynyt, odotetaan samaisen prosessin sammumista, joka tarkoittaa että tietokanta on alustettu (Liite 8). Tämän jälkeen voidaan käynnistää loput palvelut. Jos tietokannan alustava palvelu ei kuitenkaan ole käynnistynyt ennalta määritellyn aikarajan kuluessa, käynnistetään uudelleen kaikki palveluklusterit, tyhjätään ja alustetaan tietokannat uudelleen (Liite 8). Jos tietokannan alustaminen kestää pidempään kuin kolme tuntia, suoritetaan samainen toimenpide. Tässä testausautomaatioprosessi eroaa järjestelmän asennusautomaatioprosessista siten, että asennusautomaatio vain keskeyttää koko asennus prosessin, mikäli tietokanta ei ole alustettu määräaikaan mennessä.

Tietokannan alustuksen jälkeen tulee vielä muidenkin määriteltyjen palveluiden käynnistyä. Loppujen palveluiden käynnistyminen varmennetaan tarkastamalla jokaiselta palvelimelta niiden palvelukonfiguraation mukaisesti palveluiden tilat. Jokaiselta palvelulta kysytään sovelluksen tilatietoa, joka kuvastaa palvelun toimintakuntoa (Liite 8). Mikäli tila on epätosi, ei testausta voida suorittaa vajavaisen toiminnallisuuden vuoksi.

4.3.1 Testiskenaarion käynnistys

Kun prosessissa on edetty siihen vaiheeseen, jossa kaikki palvelut ovat päällä, voidaan aloittaa myös testaaminen jokaisella ympäristön työasemalla (Liite 5). Testipenkillä suoritettavat testit käynnistetään palvelimelta käsin antamalla ssh:n ylitse komento, jossa on määritelty testi, testien määrä (esimerkki 1) ja ympäristön nimi parametreina. JMeter-testit puolestaan käynnistetään testipenkin mukana tulevalla Windows-skriptillä,

joka työasemalla olevasta ympäristökohtaisesta kansiosta käynnistää testaustyökalun asennuksen mukana tulleella testisuunnitelmalla (Liite 9).

Squish-testiajo käynnistetään puolestaan generoimalla start-testing – tekstitiedosto, jota Windowsin ajastettujen tehtävien työkalu odottaa. Start-testing – tekstitiedoston sisältö saadaan testipenkin avulla, joka lukee testimateriaaliksi annetusta tiedostosta aiemmin parsitut Squish-testiajon parametrit (Liite 6). Käytännössä nämä parametrit kopioidaan vain start-testing – tekstitiedostoon joka laukaisee Squish-testiajon käynnistyksen. Parametrit, jotka on ennen prosessin käynnistystä määritelty, kopioidaan näin monen askeleen kautta juurikin Windowsin ajastettujen tehtävien työkalun toiminnan takia. Squish-testiajojen käynnistys on valmis oma prosessinsa, joka on ollut käytössä ennen testausautomaatioprosessia. Täten siis on saatu Squish-testausautomaation käynnistys prosessi integroitua testausautomaatioprosessiin.

4.3.2 Testiajon keston seuranta

Palvelimelle talletettu testisession kesto, joka on prosessin käynnistyshetkellä annettu, sijaitsee jokaiselle prosessille luotavassa kansiossa. Kansion nimeen liitetään ympäristö, jossa testaus suoritetaan, sekä Jenkins-työn numero (Liite 1). Näillä yksilöidään jokainen testiajo. Varsinainen keston seura tapahtuu omalla itsenäisellä Jenkins-työllä (Liite 14), joka tarkastaa palvelimelta kaikki testausautomaatioprosessin kestoa varten luodut kansiot ja niihin talletetun loppumisajankohdan (Liite 7). Jokaista loppumisajankohtaa verrataan sen hetkiseen aikaan. Mikäli aika on ylittynyt, käytetään kansioon kopioituja skriptejä, joilla lopetetaan testaaminen ympäristön työasemilla (Liite 15 & 16) ja kerätään näille tallennettuja testituloksia (Liite 17). Samalla myös kaikilta ympäristön palvelimilta käydään keräämässä palvelujen tuottamat lokitiedostot, sekä mahdollisesti syntyneet muistivedokset (Liite 18), joita JVM tuottaa muistin loppuessa. Kun nämä skriptit on ajettu ja tulosten keruu- ja analysointiprosessi on käynnistetty, poistetaan palvelimelta tämä testiautomaation kestoa varten luotu kansio, jotta palvelin ei täyttyisi vanhoista kansioista, ja jottei tulosten keruu- ja analysointiprosessia käynnistetä uudestaan turhaan (Liite 15).

5 LOPETUSPROSESSIN VAIHEET

Jälkimmäinen osuus automaatiosta, eli tulosten keruu- ja analysointiprosessi, toteutettiin vain osittain projektiaikataulujen takia. Tästä osuudesta toteutettiin tulosten kerääminen ja niiden analysoinnille luotiin vain suunnitelma, jonka avulla analysointivaiheet kyetään toteuttamaan aikataulun salliessa.

Tulosten kerääminen toteutettiin hyödyntäen jo olemassa olevia skriptejä, sekä luomalla muutamia uusia. Tulosten keräämisen laukaisee Jenkins-työ, joka sammuttaa myös testit kohdetyöasemilla. Tulosten keräys on toteutettu asynkronisesti, sillä keruu järjestyksellä ei ole väliä tulosten kannalta. Tulosten keruu alkaa käskyttämällä jokaista palvelinta ja prosessia aloittamaan tulosten paketoiminen taustaprosessina, jolloin ssh-yhteys ei jää odottamaan komennon valmistumista (Liite 19). Kun paketointi on käynnistetty kaikilla työasemilla ja palvelimilla, jäädään odottelemaan että kyseiset paketit valmistuvat, jolloin pakattu tiedosto siirretään verkkolevylle (Liite 20 & 21). Kun työasema- ja palvelinkohtaiset tulokset on siirretty verkkolevylle, aloitetaan tietokanta vedosten ottaminen (Liite 22). Tietokantavedokset otetaan ja siirretään verkkolevylle viimeisenä, koska niiden ottamisessa kestää aina sitä pidempään, mitä isompi tietokanta on. Työasema- ja palvelinkohtaiset tiedostot ovat myös tavallaan tärkeämpiä kuin tietokantavedos, sillä ne antavat nopeimman palautteen testisessiosta. Näiden tulosten tarkastelun perusteella siis tiedetään, mikäli tietokantavedoksia tulee tarkastella.

Palvelimilla pyörivien palveluiden lokitiedostot kootaan skriptillä, joka samalla ottaa talteen mahdolliset muistivedokset. Työasemakohtaiset tulokset puolestaan kootaan työasemalla olevasta ympäristökohtaisesta kansiosta, jonka alle ennalta määrättyihin rakenteisiin Miniclient ja JMeter tuottavat tuloksensa. Niin palvelimien kuin työasemien tulokset paketoidaan ja siirretään verkkolevylle (Liite 20 & 21), josta käyttäjä voi hakea valmiin paketin tarkastelua varten.

5.1 Testien alas-ajo

Testien alas-ajo (Liite 15.) suoritetaan, kun palvelimelle tallennettu aikaleima on saavutettu tai ylitetty. Squish-testit lopettavat toimintansa itsenäisesti, kun Squish-ajolle määritelty kesto on täyttynyt. JMeter:llä ja testipenkillä suoritettavat testit puolestaan lopetetaan terminoimalla kaikki Java-prosessit testiajoa koskevilta työasemilta (Liite 16). Tä-

mä siksi että Miniclient-testeille ei ole mitään controller–tyyppistä sovellusta, jolla testit voitaisiin ajaa hallitusti alas. Sama pätee JMeteriin, sillä se on käynnistetty komentoriviltä eikä graafisen käyttöliittymän toiminnot ole käytettävissä.

Kovinkaan kaunistahan prosessien tappaminen ei ole kun sille ei ole varsinaista syytä, kuten prosessin jumiutuminen tai muu vastaava. Kuitenkaan toiminnalla ei sinänsä ole väliä, sillä testisession päätyttyä todennäköisesti seuraava asennus asentaa uudelleen palvelut ja alustaa tietokannat, sekä asentaa testaustyökalut uudelleen. Puolestaan jos testausta halutaan jatkaa, ei sille ole estettä, sillä testit eivät kuitenkaan jätä järjestelmää epäeheään tilaan.

5.2 Palvelukohtaiset tulokset

Palvelukohtaisilla tuloksilla tarkoitetaan JMX:n yli tapahtunutta palvelun monitorointia. Palvelun monitorointi tuottaa tuloksenaan muun muassa palvelun muistinkäytöstä, prosessorikuormasta, GC-statistiikasta ja palvelukutsujen vaste-, sekä suoritusajasta tietoa CSV–tiedostoihin. Datasta kootaan asetusten mukaan koontitiedosto, jonka parsimalla kyetään luomaan Excelillä pivot-kaavio, josta nähdään graafisesti mittarien arvot per palvelu. Kaikkea tietoa ei kuitenkaan koontitiedostoon sisällytetä, vaan nämä loput tiedot ovat käyttäjän saatavilla manuaalista tarkastelua varten.

JMX-monitoroinnin toteuttaa Miniclient, joten jatkon kannalta on suunniteltu monitoroinnin tulosten tallentamista suoraan testipenkin käyttämään ympäristökohtaiseen tietokantaan. Täten käyttäjän ei tarvitsisi käsitellä erillisiä tiedostoja vaan tuloksiin pääsisi käsiksi tietokannan avulla.

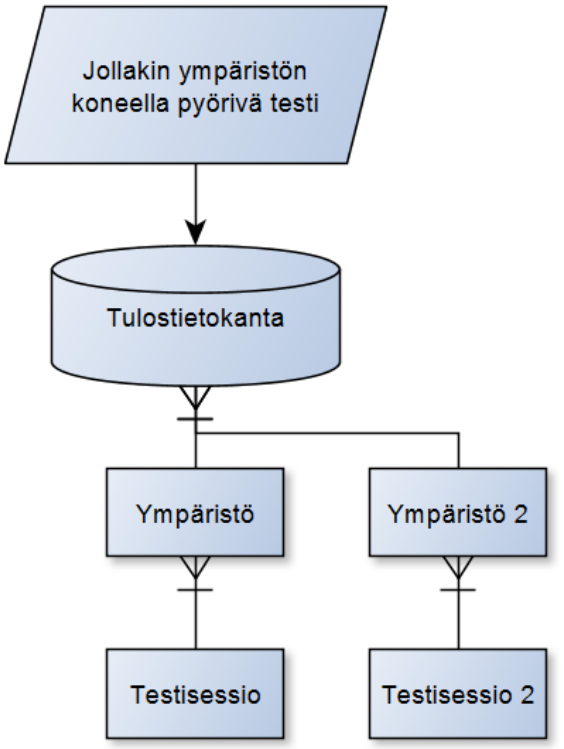
5.3 Työasemakohtaiset tulokset

Työasemakohtaisia tuloksia ovat Miniclientin ja JMeterin tuottamat tulokset. Miniclient tuottaa tuloksia tällä hetkellä vain ennalta määrätyistä toiminnoista, joista se mittaa vasteajan. Jokaisesta toiminnosta dokumentoidaan lopputulos CSV–tiedostoon, sillä tietokantaan tallentamista ei ole vielä otettu käyttöön tietokannan puuttumisen takia. Lopputulos sisältää aikaleiman, tuloksen ja vasteajan, sekä mahdolliset lisähuomiot. Tuloksella tarkoitetaan testin onnistumista tai epäonnistumista.

JMeter tuottaa testiensä tulokset myös CSV–tiedostoihin. Kukin tulos sisältää suoritetun toiminnon, aikaleiman ja palvelimen vastauksen. Tulokset tallennetaan kahteen erilliseen tiedostoon, joista toiseen tulee kaikki tulokset testin loppu tulemasta riippumatta ja toiseen vain epäonnistuneet lopputulokset. JMeter tuottaa testituloksensa joko XML -tai CSV -muotoon, joten nämä tulokset tulee parsia erikseen tulostietokantaan. Tähän parsintaan on suunniteltu käytettävän Miniclienttiä, joka lukisi CSV–tiedoston, ja tallentaisi tulokset tietokantaan samoja mekanismeja käyttäen kuin se tekee omissa testeissään. Samanlainen parsinta on suunniteltu Squishin tuottamalle tulos xml:n parsimiselle.

5.4 Tulostietokanta

Testituloksille tallennuspaikaksi suunniteltiin tulostietokanta, jossa on ympäristökohtaiset taulut (kuva 8), jonne tallennettaisiin Miniclient-, JMeter- ja Squish-testaustyökalujen tuottamat tulokset. Tulostietokannan ideana on keskittää kaikki tulokset saataville ympäristökohtaisesti, täten voidaan vertailla testitulosten eroavaisuuksia menneisiin testiajoihin. Lisäksi tietokannan käyttö on paljon yksinkertaisempaa kuin erillisten tiedostojen, joita tarvitsisi itse avata esimerkiksi Excelillä ja sitä kautta piirtää kuvaajia.

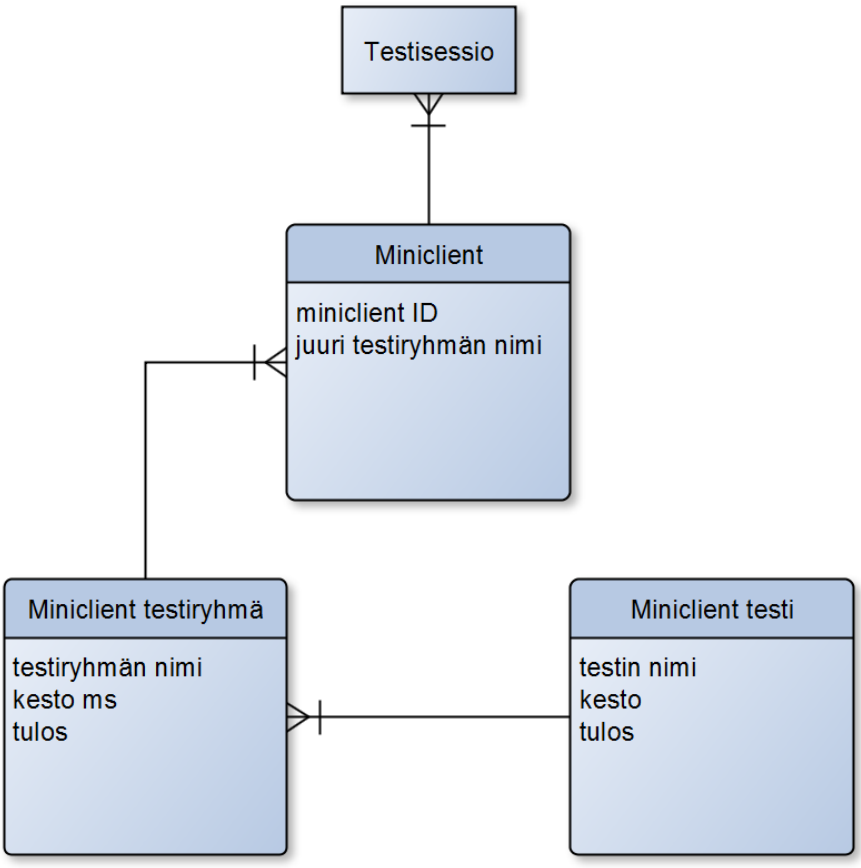


KUVA 8. Tulostietokannan ympäristökohtaisuus

Kaikki testausympäristöt ovat yhteydessä tulostietokantaan ja käyttävät siellä omaa ympäristökohtaista taulua testitulosten tallentamiseen (kuva 8). Tietokannasta kyetään vertailemaan edellä mainitulla tavalla tuloksia testisessioiden välillä eri ja samoissa ympäristöissä.

5.4.1 Miniclient testitulokset

Miniclientin suorittamien varsinaisten testien tulokset tallennetaan Miniclient kohtaisesti omiin tauluihinsa (kuva 9), sillä testejä toistetaan monta kertaa jolloin niiden tulokset on helposti niputettavissa.

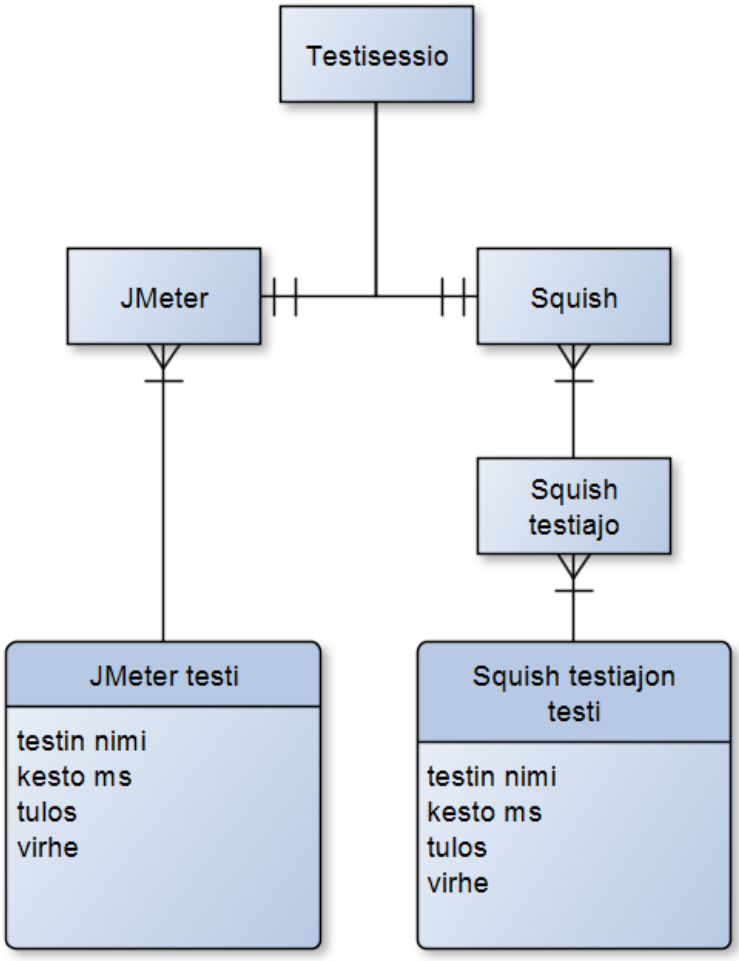


KUVA 9. Miniclientin tulokset tulostietokannassa

Miniclient muodostaa kustakin yksittäisestä testistä tuloksen sitä mukaan tuottaako testin toiminta poikkeuksen vai ei. Kukin testi kuuluu johonkin testiryhmään, jonka kaikkien testien tuloksista muodostetaan testiryhmän tulos. Tämä ketju toistuu aina niin pitkälle kuin juuri testiryhmään asti, jonka tulos määritellään kaikkien näiden alatulosten perusteella. Jos yksittäinen testi heittää poikkeuksen, merkitään se epäonnistuneeksi. Täten merkitään myös testiryhmä, johon testi kuului, epäonnistuneesti ja aina rekursiivisesti juureen asti.

5.4.2 JMeter ja Squish testitulokset

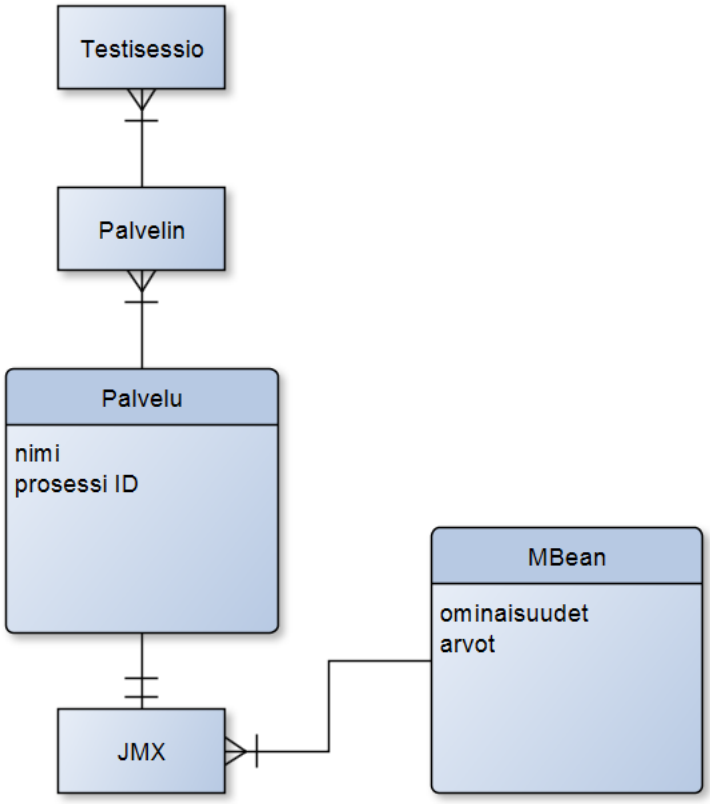
JMeter suorittaa kokoajan samaa testiajoa testisuunnitelman mukaisesti. Kun JMeter pääsee testisuunnitelman loppuun, aloitetaan uusi iteraatio. Täten samaisia testejä suoritetaan useampia kertoja, jolloin on järkevintä niputtaa ne samaan tauluun (kuva 10). Squish puolestaan suorittaa useampia testiajoja, jotka koostuvat tietyistä testeistä skenaarion mukaan. Jokaista testiä toistetaan ajon sisällä lukuisia kertoja, joten samaiset testit on hyvä laittaa omaan tauluunsa Squish-testiajon alle (kuva 10). Squishin-testiajot on hyvä yksilöidä, sillä se helpottaa mahdollisien laatupoikkeamien seuraamista. Molemmat tulokset on suunniteltu parsittavaksi testaustyökalujen tuottamista tulostiedostoista vasta testisession päättymisvaiheessa.



KUVA 10. JMeter- ja Squish-testien tulokset tulostietokannassa

5.4.3 Palvelukohtaisen JMX monitoroinnin tulokset

Palvelut on jaettu useammalle palvelimelle jokaisessa ympäristössä. Täten jokaiselle palvelimelle muodostetaan taulu sen DNS-nimestä (kuva 11). Tauluun kuuluvat puolestaan palvelimella pyörivien palveluiden taulut. Kustakin palvelusta on monitoroitu kaikki mahdolliset JMX:n yli saatavat tiedot, niin Javan omia virtuaalikoneen tuottamia tietoja, kuin palvelun itsensä. Monitorointi hakee tietojen arvot minuutin välein, jotta voidaan muodostaa kattava kuva järjestelmän tilasta pitkien aikajaksojen yli menettämättä tiedon tarkkuutta.



KUVA 11. Palvelukohtaiset JMX-tulokset tulostietokannassa

6 TULOSTEN GRAAFINEN ESITTÄMINEN

Tulosten analysoinnin helpottamiseksi ja nopeuttamiseksi tulostietokannassa olevista tuloksista muodostetaan kuvaajia osa-aluekohtaisesti. Nämä tulokset tullaan esittämään jokaisesta testisessiosta ympäristökohtaisesti tulostietokannan datan avulla. Kuvaajien muodostukseen on suunniteltu käytettäväksi jotakin selainpohjaista sovellusta, esimerkiksi Grafanaa. Grafana on selainkäyttöinen työkalu, jolla kyetään muodostamaan eri tietolähteistä kuvaajia. Kuvaajilla voi olla useita Y-akseleita, joille on useita eri formaatteja kuten bitit, millisekunnit ja niin edelleen, ja ne voidaan voi skaalata logaritmisesti. Jokaisen kuvaajaan pystyy viemään selaimesta png-muotoon. Näkymiä ja kuvaajia pystyy muokkaamaan, ja niitä pystyy siirtelemään näkymissä oman mielen mukaan. (Features, Grafana.)

Testisessiosta muodostetaan yksittäisiä osa-aluekohtaisia tuloksia, joiden pohjalta määritellään myös koko testisession lopputulos. Osa-aluekohtaisille tuloksille määritellään tietyt rajat, joilla rajataan tuloksen onnistuminen tai epäonnistuminen. Tällaisia rajoja ja osa-alueita ovat muun muassa GC:n kesto ja määrä, muistinkäyttö ja prosessorikuorma. Osa-aluekohtaiset tulokset siis muodostetaan palvelujen tiloista, jotka on JMX:n yli tapahtuvan monitoroinnin kautta saatu tietoon. Osa-aluekohtaisia tuloksia muodostetaan myös kunkin testaustyökalun tuottamien tuloksien kokonaiskuvasta. Testaustyökalujen tuottamia kokonaisuuksia ovat testien kesto ja onnistuminen, joita mitataan testipenkillä, sekä REST-rajapinnan ja käyttöliittymän responssivisuus, jotka mitataan JMeterillä ja Squishilla.

7 POHDINTA

Työtä tehdessäni opin erittäin paljon testaamisen eri muodoista ja tavoista toteuttaa niitä. Opin myös ohjelmistotuotannosta ja arkkitehtuurista runsaasti. Suurimpana oppina työssä kuitenkin oli skriptaaminen. Suurin osa skriptaamisesta tapahtui shell:llä, mutta työssä käytettiin myös jonkin verran Windows-skriptejä. Työn toteutus lähti jokseenkin hitaasti liikkeelle syksyllä 2015, mutta kehitys sai vauhtia mitä lähemmäs vuoden vaihdetta päästiin. Kaikkea mitä työn toteutukseen oli suunniteltu, ei keritty toteuttamaan projektin aikataulun ja prioriteettien takia. Kun huomattiin ajan loppuminen kesken, luotiin suunnitelmia tulosten keruu- ja analysointiprosessin toteutusta varten, jotta myöhemmin kun toteutus keritään tekemään, se kyettäisiin tekemään nopeasti.

Eniten työn toteutuksessa jäi harmittamaan se, etten viestinyt tarpeeksi voimakkaasti projektijohdolle työn positiivista vaikutusta päivittäiseen työskentelyyn. Edellä on kuvattu kuinka paljon tämä automaatio säästää aikaa lähes päivittäisessä työskentelyssä, kun tuloksia on useassa eri ympäristössä ja täten vielä useammilla palvelimilla ja työasemilla. Automaation tuoman avun huomasi heti tammikuun alussa, kun ei tarvinnut käsin hakea kaikkea vaan kykeni käynnistämään asynkronisen tulosten keräämisen. Tulosten keruussa kului noin 12 minuuttia, kun taas käsin siihen saattoi kulua kaksi tuntia. Lisäksi tuloksille suunniteltu tulostietokanta ja sitä kautta saatavat valmiit graafiset esitykset vähentävät entisestään työaikaa. Tällöin käsin tehtäväksi työksi jää lokien selaaminen kun kaikki kuvaajat ovat valmiina eikä niitä tarvitse itse tehdä esimerkiksi Excelin pivot tauluilla. Lokien selaamiseen ja jäsentelyyn mietittiin myös joitakin ratkaisuja suunnittelun alkuvaiheessa, mutta ne nopeasti hylättiin, sillä toteutukseen menevä aika ei välttämättä korvaisi itseään lokien selaamisessa.

Helmikuun alussa olleella kollegan pitämällä tietoiskulla koneopista heräsi uusi ajatus. Tämän ajatuksen synnytti tietoiskulla esimerkkinä ollut toteutus usean eri palvelimen lokien jäsentelyyn käytetystä algoritmista. Algoritmi etsi eroavaisuuksia lokitiedostoista ja muodosti näistä lineaarisella regressiolla kuvaajan, jossa näkyi selkeästi nippu lokeja, jotka olivat virheettömiä. Kuitenkin regressio viiva ei kulkenut keskeltä tätä rypästä, vaan reippaasi ohitse, sillä sivummalta löytyi kaksi palvelinta, joiden lokeissa oli hyvin paljon virheitä. Samankaltaisen algoritmin käyttömahdollisuuksia voisi selvittää, josko niistä olisi hyötyä palvelujen lokien läpikäymiseen.

LÄHTEET

Meet Jenkins. Luettu 23.12.2015
https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Esittely. Luettu 23.12.2015
http://jmeter.apache.org/index.html

JDBC-API. Luettu 23.12.2015
https://en.wikipedia.org/wiki/Java_Database_Connectivity

Apache Maven. POM. Luettu 23.12.2015
https://en.wikipedia.org/wiki/Apache_Maven#Project_Object_Model

Squish Features and Benefits. Luettu 23.12.2015
http://www.froglogic.com/squish/gui-testing/features/index.php

Koistinaho Jarkko. Korkean saatavuuden järjestelmän automaattinen testaus. Huhtikuu 2013
https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/21601/koistinaho.pdf?sequence=1

Features. Luettu 31.01.2016
http://grafana.org/features/

LIITTEET

Liite 1. Määritellyn testisession keston kopiointi palvelimelle

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3
HA_DAYS=$4
HA_HOURS=$5

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

if [ "$ENVIRONMENT" == "" ]; then
    echo "Installation environment has not been defined, no tests will
be executed!"
    exit 0
elif [ $HA_DAYS -eq 0 ] && [ $HA_HOURS -eq 0 ]; then
    echo "HA_DAYS and HA_HOURS are both zero, no tests will be execut-
ed!"
    exit 0
elif [ "$HA_DAYS" == "" ] && [ "$HA_HOURS" == "" ]; then
    echo "HA_DAYS and HA_HOURS are not defined, no tests will be exe
cuted!"
    exit 0
fi

PROJECT_DIR=/var/www/$PROJECT
TEST_AUTOMATION_PROCESS_DIR=$PROJECT_DIR/$JOB/scripts/test_automation_
process
WORKSTATIONS_DIR=$TEST_AUTOMATION_PROCESS_DIR/workstations
TEST_SESSION_MONITORING_DIR=$TEST_AUTOMATION_PROCESS_DIR/testSessionMo
nitoring

for ENV in "${ENV_LIST[@]}"; do
    ONGOING_TESTS_DIR=$PROJECT_DIR/ongoing_tests_$ENV-$JOB

    if [ -d $ONGOING_TESTS_DIR ]; then
        rm -rf $ONGOING_TESTS_DIR
    fi

    mkdir $ONGOING_TESTS_DIR

    if [ ! -d $ONGOING_TESTS_DIR/workstations ]; then
        mkdir $ONGOING_TESTS_DIR/workstations
    fi

    cp $WORKSTATIONS_DIR/* $ONGOING_TESTS_DIR/workstations/
    cp $TEST_SESSION_MONITORING_DIR/* $ONGOING_TESTS_DIR

    CURRENT_TIME=`date +%s`
    DURATION_SEC=$((HA_DAYS*24*60*60+HA_HOURS*60*60))
    END_TIME=$((CURRENT_TIME+DURATION_SEC))

    echo "Test session ends in $HA_DAYS days and $HA_HOURS hours ($DU-
RATION_SEC seconds). Current time $CURRENT_TIME, end time $END_TIME
(EPOC)"
    echo $END_TIME > $ONGOING_TESTS_DIR/end_time.properties
done
```

Liite 2. Tietokannan alustuksen seuranta 1 (2)

```
KEY=$1
REMOTE=$2
JOB=$3
PROJECT=$4
ENVIRONMENT=$5

POLL_DB_INIT_STATUS="ps -fU response -ww | grep DatabaseInitService"
GREP_DB_INIT_LOG="less /home/usr/usr/log/service-database-
init/debug.log | grep \"DataInitializer - Data initialization done.\""
CLUSTER_RESTART_CLEAN="sh /opt/ scripts/cluster_services.sh restart --
clean"
SHOULD_INSTALL_PORTAL=0

DB_INIT_MAX_TIME=21600
DB_INIT_TOTAL_ELAPSED_TIME=0
DB_INIT_SHOULD_HAVE_STARTED=360
DB_INIT_ELAPSED_TIME=0

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do

    WAS_INIT_DONE_BEFORE=`ssh -n -i $KEY $REMOTE "$GREP_DB_INIT_LOG"`
|| true
    if [ "$WAS_INIT_DONE_BEFORE" != "" ]; then
        echo "It seems that database has been initialized before, mov-
ing on..."
        exit 0
    fi

    while true; do
        echo "Polling database-init startup in $ENV..."
        echo "Total elapsed time: $DB_INIT_TOTAL_ELAPSED_TIME seconds"
        DB_INIT_STATUS=`ssh -n -i $KEY $REMOTE "$POLL_DB_INIT_STATUS"`
|| true

        if [ "$DB_INIT_STATUS" == "" ]; then
            if [ $DB_INIT_ELAPSED_TIME -gt
$DB_INIT_SHOULD_HAVE_STARTED ]; then
                echo "Database-init was not started in 6 minutes!"
                echo "Restarting service cluster and reseting data-
base!"
                ssh -n -i $KEY $REMOTE $CLUSTER_RESTART_CLEAN
                DB_INIT_ELAPSED_TIME=0
                SHOULD_INSTALL_PORTAL=1
            fi
        elif [ "$DB_INIT_STATUS" != "" ]; then
            echo "Database-init started!"
            break;
        fi

        if [ $DB_INIT_TOTAL_ELAPSED_TIME -gt $DB_INIT_MAX_TIME ]; then
            echo "Database is not initialized in 3 hours, exiting pro-
cess!"
            exit 1
        fi

        sleep 5
        DB_INIT_ELAPSED_TIME=$(( $DB_INIT_ELAPSED_TIME + 5 ))
```

jatkuu 2 (2)

```
 DB_INIT_TOTAL_ELAPSED_TIME=$(( $DB_INIT_TOTAL_ELAPSED_TIME + 5 ))
    done

    while true; do
        echo "Polling database-init in $ENV..."
        echo "Total elapsed time: $DB_INIT_TOTAL_ELAPSED_TIME seconds"
        DB_INIT_STATUS=`ssh -n -i $KEY $REMOTE "$POLL_DB_INIT_STATUS"`
|| true

        if [ "$DB_INIT_STATUS" == "" ]; then
            echo "Database-init done!"
            break
        fi

        if [ $DB_INIT_TOTAL_ELAPSED_TIME -gt $DB_INIT_MAX_TIME ]; then
            echo "Database is not initialized in 3 hours, exiting pro-
cess!"
            break
        fi

        sleep 5
        DB_INIT_TOTAL_ELAPSED_TIME=$(( $DB_INIT_TOTAL_ELAPSED_TIME + 5
))
    done

    if [ $SHOULD_INSTALL_PORTAL -eq 1 ]; then
        sh installPortal.sh $JOB $PROJECT $ENV
    fi
done

exit 0
```

Liite 3. Ympäristökohtaisen testimateriaalin yliajaminen työasemalle

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3

SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
TEST_MATERIAL=$SCRIPTSDIR/test_automation_process/testMaterialToUse/
WORKSTATION_CONFS=$SCRIPTSDIR/test_automation_process/workstations
KEY=/folder/subfolder/subsubfolder/subsubsubfolder

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do
    CMD_MC_TEST_MATERIAL_LOCATION="cd $ENV/miniclient/testmaterial/"

    IFS=$'\r\n' TMP_LIST=($(cat
$SCRIPTSDIR/environments/$ENV/workstations.conf | sed -e
'1s/^\xef\xbb\xbf//'))
    for ITEM in "${TMP_LIST[@]}"; do
        WORKSTATIONS[$I]=$ITEM
        I=$(( $I + 1 ))
    done

    for WS in ${WORKSTATIONS[@]}; do
        for FILE in `ls $TEST_MATERIAL`; do
            if [ -f $WORKSTATION_CONFS/$WS.properties ]; then
            CLIENT=jenkins@$WS
            sftp -b /dev/stdin -o "PubkeyAuthentication yes" -o "Iden-
tityFile=$KEY" $CLIENT << EOF
            $CMD_MC_TEST_MATERIAL_LOCATION
            put $TEST_MATERIAL/$FILE
            exit
EOF
            fi
        done
    done
done
```

Liite 4. Testiskenaarion yliajaminen testi automaatio prosessiin

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3
SCENARIO=$4

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))
SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
WORKSTATION_CONFS=$SCRIPTSDIR/test_automation_process/workstations
KEY=/folder/subfolder/subsubfolder/subsubsubfolder

for ENV in "${ENV_LIST[@]}"; do
    SCENAR-
IO_CONFS=$SCRIPTSDIR/test_automation_process/testSetUps/scenarios/$SCE
NARIO/$ENV/
    if [! -d $SCENARIO_CONFS ]; then
        echo "Alternative configuration not found for $SCENARIO in
$ENV => ABORT!!"
        exit 1
    fi

    # Overwrite alternative servers.conf
    if [ -f $SCENARIO_CONFS/servers.conf ]; then
        SERVERS_CONF=$SCRIPTSDIR/environments/$ENV/servers.conf
        echo "Overwriting servers.conf for $ENV..."

        echo "PREVIOUS:"
        cat $SERVERS_CONF

        echo "NEXT:"
        cat $SCENARIO_CONFS/servers.conf

        cp $SCENARIO_CONFS/servers.conf $SERVERS_CONF

        echo "AFTER copy-paste:"
        cat $SERVERS_CONF
    fi

    # Overwrite test properties for workstations
    if test "$(ls -A "$SCENARIO_CONFS/workstations/")"; then
        echo "Overwriting previous workstation configurations..."
        cp $SCENARIO_CONFS/workstations/* $WORKSTATION_CONFS
    fi

    # Overwrite Squish configuration
    if [ -d $SCENARIO_CONFS/squish ]; then
        if [ -f $SCENARIO_CONFS/squish/configureSquish.properties ];
then
            echo "Overwriting squish configuration..."
            cp $SCENARIO_CONFS/squish/configureSquish.properties
$SCRIPTSDIR/test_automation_process/squish/
        fi
    fi

done

# Copy environment specific test material to temporary folder, from
where it will be inserted to workstations
cp
$SCRIPTSDIR/test_automation_process/testSetUps/scenarios/$SCENARIO/"${
ENV_LIST[0]}"/testmaterialTemplates/*
$SCRIPTSDIR/test_automation_process/testMaterialToUse/
```

Liite 5. Testien käynnistys

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3

SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
WORKSTATION_CONFS=$SCRIPTSDIR/test_automation_process/workstations
KEY=/folder/subfolder/subsubfolder/subsubsubfolder
COMMENT_LINE="^#.*$"
HAS_SQUISH="^\s*squish\s*=\s*(.*)\s*$"
HAS_JMETER="^\s*jmeter\s*=\s*(.*)\s*$"
HAS_TESTS="^\s*(.*)\s*=\s*(.*)\s*$"

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))
WORKSTATIONS=()
I=0
for ENV in "${ENV_LIST[@]}"; do
    echo "Starting tests in $ENV"
    MINICLIENT_DIR=c:/folder/$ENV/miniclient/bin/
    START_JMETER="cmd cd /c cd \ & cd folder & cd $ENV & cd minicli-
ent\\bin\\console & start call start_JMeter_with_HA_200v_SingleTest-
plan.cmd"
    START_SQUISH="cmd cd /c cd \ & cd folder & cd $ENV & cd minicli-
ent\\bin & start run_multiple_tests.bat console\\Generate start test-
ing file for Squish.properties 1 $ENV"
    START_MINICLIENTS="cmd cd /c cd \ & cd folder & cd $ENV & cd mini-
client\\bin & start run_multiple_tests.bat"
    IFS=$'\r\n' TMP_LIST=($(cat
$SCRIPTSDIR/environments/$ENV/workstations.conf | sed -e
'1s/^\xef\xbb\xbf//'))
    for ITEM in "${TMP_LIST[@]}"; do
        WORKSTATIONS[$I]=$ITEM
        I=$(( $I + 1 ))
    done
    for WS in ${WORKSTATIONS[@]}; do
        if [ -f $WORKSTATION_CONFS/$WS.properties ]; then
            FILE=$WS.properties
            while read LINE || [[ -n "$LINE" ]]; do
                if [[ ! $LINE =~ $COMMENT_LINE ]]; then
                    CLIENT=jenkins@$WS
                    if [[ $LINE =~ $HAS_SQUISH ]]; then
                        echo "Starting squish on $WS"
                        ssh -n -i $KEY $CLIENT $START_SQUISH
                        break
                    elif [[ $LINE =~ $HAS_JMETER ]]; then
                        echo "Starting JMeter on $WS"
                        ssh -n -i $KEY $CLIENT $START_JMETER
                    elif [[ $LINE =~ $HAS_TESTS ]]; then
                        echo "Starting ${BASH_REMATCH[2]} minicli-
ent(s) with test ${BASH_REMATCH[1]} on $WS"
                        ssh -n -i $KEY $CLIENT "$START_MINICLIENTS
\"${BASH_REMATCH[1]}.properties\" ${BASH_REMATCH[2]} $ENV"
                    else
                        echo "No tests defined on $WS (Line: $LINE)!"
                    fi
                else
                    echo "Skipped commented line: $LINE on $WS"
                fi
            done < $WORKSTATION_CONFS/$FILE
        fi
    done
done
```

Liite 6. Squish parametrien kokoaminen 1 (2)

```
ENVIRONMENT=$1
HAS_SQUISH="^\s*squish\s*=\s*(.*)\s*$"
WORKSTATION_NAME="workstations/(.*)\.properties"
COMMENT_LINE="^#.*$"
TESTCASE="^\s*TESTCASES\s*=\s*(.*)\s*$"
PARAM_KEY="^\s*(.*)\s*="
SQUISH_CONF_DIR=workstationSquishConfig

if [ "$ENVIRONMENT" == "" ]
then
    echo "Environment not provided!"
    exit 1
fi

rm -rf $SQUISH_CONF_DIR/*

for FILE in workstations/*.properties
do
    if [[ $FILE =~ $WORKSTATION_NAME ]]
    then
        NAME=${BASH_REMATCH[1]}
        CURRENT_FILE=`head -n 1 $FILE`
        if [[ $CURRENT_FILE =~ $HAS_SQUISH ]]
        then
            SQUISH_CONFIG=${BASH_REMATCH[1]}
            mkdir -p $SQUISH_CONF_DIR/$NAME
            touch $SQUISH_CONF_DIR/$NAME/squishParameters.txt
            if [ -r squish/$SQUISH_CONFIG ]
            then
                FILE=()
                FILE[0]="ENVIRONMENT=$ENVIRONMENT"
FILE[1]="JAVABIN=%ProgramFiles(x86)%\\folder\\$ENVIRONMENT\\folder\\jre\\bin\\"
                j=2
                while read LINE || [[ -n "$LINE" ]]; do
                    if [[ ! $LINE =~ $COMMENT_LINE ]]
                    then
                        if [[ $LINE =~ $TESTCASE ]]
                        then
                            TESTCASE_PARAM=${BASH_REMATCH[1]}
                        fi
                        FILE[$j]=$LINE
                        j=$((j+1))
                    fi
                done < squish/$SQUISH_CONFIG
                if [ -r squish/squish$TESTCASE_PARAM.template ]
                then
                    TESTSET=`head -n 1 $FILE`
                fi
                while read LINE || [[ -n "$LINE" ]]; do
                    if [[ $LINE =~ $PARAM_KEY ]]
                    then
                        MATCH_FOUND=0
                        KEY=${BASH_REMATCH[1]}
                        for FILELINE in ${FILE[@]}
                        do
                            if [[ $FILELINE =~ $PARAM_KEY ]]
                            then
                                if [ "$KEY" == "${BASH_REMATCH[1]}" ]
                                then
```

jatkuu

```
echo -e "$FILELINE\r" >> $SQUISH_CONF_DIR/$NAME/squishParameters.txt
                                        MATCH_FOUND=1
                                        break
                                    fi
                                fi
                            done
                            if [ $MATCH_FOUND -eq 0 ]
                            then
                                echo $LINE >>
$SQUISH_CONF_DIR/$NAME/squishParameters.txt
                            fi
                        fi
                    done < squish/squishParameters.template
                fi
            fi
        fi
done
```

Liite 7. Testisession keston seuranta

```
ROOTDIR=/var/www
PROJECTS=(project 1 project 2 project 3)

for PROJECT in ${PROJECTS[@]}; do
    echo "Polling project: $PROJECT"
    for TEST in $ROOTDIR/$PROJECT/ongoing_tests_*; do
        echo "Polling ongoing: $TEST"
        if [ -f $TEST/end_time.properties ]; then
            echo -n "$TEST "
            cat $TEST/end_time.properties
        fi
    done
done
```

Liite 8. Järjestelmän käynnityksen tilan seuranta ja testien käynnistys 1 (3)

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3

SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
START_TESTING_DIR=$SCRIPTSDIR/test_automation_process/startTesting
KEY=$SCRIPTSDIR/folder/file.someFile

POLL_DB_INIT_STATUS_SH=$START_TESTING_DIR/pollDatabaseInitStatus.sh
POLL_MODEL_SEARCH_STATUS_SH=$START_TESTING_DIR/pollModelSearchStatus.s
h
POLL_MODEL_SEARCH="sh /opt/response-scripts/pollModelSearchStatus.sh"

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

#Check if ha-packages are present
if [ -d $SCRIPTSDIR/ha_packages ]; then
    echo "$SCRIPTSDIR/ha_packages is present!"
    if [ ! -f $SCRIPTSDIR/ha_packages/miniclient-*.zip ]; then
        echo "No response miniclient in ha_packages!"
        exit 0
    fi
    for ENV in "${ENV_LIST[@]}"; do
        if [! -d /var/www/$PROJECT/ongoing_tests_$ENV-$JOB ]; then
            echo "No ongoing_tests directory found for env \"$ENV\"
and job \"$JOB\"!"
            exit 0
        fi
    done
    echo "Miniclient package and ongoing_tests directory found, start-
ing testing process!"
else
    echo "$SCRIPTSDIR/ha_packages wasn't found!"
    exit 0
fi

MAX_TRIES=0
DB_INIT_COMPLETED=0
MS_EXPORTING_COMPLETED=0

#DB-init polling
for ENV in "${ENV_LIST[@]}"; do

    if [ $DB_INIT_COMPLETED -eq 0 ]; then
        SERVER_LINE=`cat $SCRIPTSDIR/environments/$ENV/servers.conf |
sed -e '1s/^\xef\xbb\xbf//' | grep database-init` || true
        if [ "$SERVER_LINE" == "" ]; then
            echo "Couldn't find database-init in $ENV, continuing..."
            echo "Continuing to next environment in list:
${ENV_LIST[@]}"
        elif [ "$SERVER_LINE" == "*/down" ]; then
            echo "Database-init was specified down, moving on..."
            DB_INIT_COMPLETED=1
        else
            ADDRESS=`echo $SERVER_LINE | cut -d' ' -f1`
            if [[ $ADDRESS == */* ]]; then
                    IP=${ADDRESS%/*}
            else
            IP=$ADDRESS
            fi
            REMOTE=admin@$IP
```


jatkuu

```
            sh $POLL_DB_INIT_STATUS_SH $KEY $REMOTE $JOB $PROJECT $ENV
            DB_INIT_RESULT=$?
            if [ $DB_INIT_RESULT -gt 0 ]; then
                if [ $DB_INIT_RESULT -eq 1 ]; then
                    echo "Database-init didn't start properly!"
                elif [ $DB_INIT_RESULT -eq 2 ]; then
                    echo "Database-init didn't complete in 3 hours!"
                else
                    echo "UNKNOWN ERROR!"
                fi
                continue
            else
                DB_INIT_COMPLETED=1
            fi
        fi
    fi

    if [ $DB_INIT_COMPLETED -eq 1 ]; then
        echo "Database-init checked successfully, moving on..."
        break
    fi

done

#Poll services appHealths
for ENV in "${ENV_LIST[@]}"; do

    IFS=$'\n' SERVERS=`cat
/var/www/$PROJECT/$JOB/scripts/environments/$ENV/servers.conf | sed
's/\// /g' | sed -e '1s/^\xef\xbb\xbf//' | awk '{print  $1}'`
    for SERVER in $SERVERS; do
        REMOTE=admin@$SERVER
        FORMAT=`ssh -n -i $KEY $REMOTE "if ls /usr/bin/service-* 2>
/dev/null | grep -qc service-; then echo \"format1\";  else echo
\"format2 \"; fi"`
        SERVER_LINE=`cat $SCRIPTSDIR/environments/$ENV/servers.conf |
sed -e '1s/^\xef\xbb\xbf//' | grep $SERVER`

        IFS=$' ' SERVICES=($(echo "$SERVER_LINE"))
        for SERVICE in "${SERVICES[@]}"; do

            if [[ $SERVICE == service-* ]] && [[ $SERVICE != */down ]]
&& [[ $SERVICE == service-database-init ]]; then
                COMMAND="${FORMAT}${SERVICE} health"
                STATUS=`ssh -n -i $KEY $REMOTE $COMMAND`
                while [  $STATUS -eq 1 ]; do
                    echo "$SERVICE application health isn't OK yet at
$SERVER!"
                    sleep 5
                    STATUS=`ssh -n -i $KEY $REMOTE $COMMAND`
                done
            else
                echo "$SERVICE is either server dns-name/ip or speci-
fied down or is database-init"
            fi

        done
    done

done

#Start testing
for ENV in "${ENV_LIST[@]}"; do
```

```
    #Copy environment specific test material to workstations
    sh $START_TESTING_DIR/initEnvSpecificTestmaterial.sh $JOB $PROJECT
$ENV

    #Start testing
    sh $START_TESTING_DIR/startTesting.sh $JOB $PROJECT $ENV
done
```

Liite 9. JMeter asennuspaketin kopiointi työasemalle 1 (2)

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3
SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
KEY=/folder/subfolder/subsubfolder/subsubsubfolder
HAS_JMETER="^\s*jmeter\s*=\s*(.*)\s*$"
WORKSTATION_NAME="workstations/(.*)\.properties"
JMETER_FILEPATH=/var/www/html/install
IN-
STALL_JMETER_BAT=$SCRIPTSDIR/test_automation_process/installJMeter.bat
WORKSTATION_CONFS=$SCRIPTSDIR/test_automation_process/workstations

JMETER_CONF_FOLDER=c:\JMeter
JMETER=apache-jmeter-2.11.7z
TESTPLAN=HA_200v_Testplan.jmx

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

WORKSTATIONS=()
I=0
for ENV in "${ENV_LIST[@]}"; do
    MKDIR_JMETER="cmd /c cd \ & cd folder & cd $ENV & IF NOT EXIST
JMeter mkdir JMeter"
    MKDIR_JMETER_LOGS="cmd /c cd \ & cd folder & cd $ENV\\JMeter & IF
NOT EXIST logs mkdir logs"
    MKDIR_JMETER_TESTPLAN="cmd /c cd \ & cd folder & cd $ENV\\JMeter &
IF NOT EXIST testplan mkdir testplan"
    CHECK_JMETER_INSTALLATION="cmd /c cd \ & cd folder & IF EXIST
$ENV\\JMeter\\apache-jmeter* echo 1"
    JMETER_INSTALLATION="cmd /c cd \ & cd folder & cd $ENV & cd JMeter
& installJMeter.bat $JMETER"

    IFS=$'\r\n' TMP_LIST=($(cat
$SCRIPTSDIR/environments/$ENV/workstations.conf | sed -e
'1s/^\xef\xbb\xbf//'))
    for ITEM in "${TMP_LIST[@]}"; do
        WORKSTATIONS[$I]=$ITEM
        I=$(( $I + 1 ))
    done

    for WS in ${WORKSTATIONS[@]}; do
        if [ -f $WORKSTATION_CONFS/$WS.properties ]; then
            FILE=$WS.properties
            while read LINE || [[ -n "$LINE" ]]; do
                if [[ $LINE =~ $HAS_JMETER ]]; then
                    REMOVE_PREVIOUS_CONF_IF_EXIST="IF EXIST del con-
fig.csv"
                    SET_JMETER_CONF="echo
${BASH_REMATCH[2]},,user,https,,Jokitie 3,Lahti >> config.csv"

JMETER_TESTPLAN_FOLDER=c:\folder\$ENV\JMeter\testplan
                    CLIENT=jenkins@$WS
                    IS_JMETER_INSTALLED=`ssh -n -i $KEY $CLIENT
$CHECK_JMETER_INSTALLATION`

                    if [[ $IS_JMETER_INSTALLED != 1* ]]
                    then
                        ssh -n -i $KEY $CLIENT $MKDIR_JMETER
                        sftp -b /dev/stdin -o "PubkeyAuthentication
yes" -o "IdentityFile=$KEY" $CLIENT << EOF
                            cd $ENV
```

```
                        cd JMeter
                        put $JMETER_FILEPATH/$JMETER
                        put $INSTALL_JMETER_BAT
                        exit
EOF
                            ssh -n -i $KEY $CLIENT $JMETER_INSTALLATION
                    fi

                    ssh -n -i $KEY $CLIENT $MKDIR_JMETER_LOGS
                    ssh -n -i $KEY $CLIENT $MKDIR_JMETER_TESTPLAN

                    sftp -b /dev/stdin -o "PubkeyAuthentication yes" -
o "IdentityFile=$KEY" $CLIENT << EOF
                    cd $JMETER_CONF_FOLDER
                    $REMOVE_PREVIOUS_CONF_IF_EXIST
                    $SET_JMETER_CONF
                    cd $JMETER_TESTPLAN_FOLDER
                    put $SCRIPTSDIR/test_automation_process/$TESTPLAN
                    exit
EOF
                    break
                fi
            done < $WORKSTATION_CONFS/$FILE
        fi
    done
done
```

Liite 10. JMeter asennus työasemalla

```
set JMETERFILE=%1
set SEVENZIP="C:\Program Files\7-Zip\7z.exe"
set SEVEN86ZIP="C:\Program Files (x86)\7-Zip\7z.exe"

if NOT exist %SEVENZIP% set SEVENZIP=%SEVEN86ZIP%

%SEVENZIP% x %JMETERFILE% -aoa
```

Liite 11. Squish asennuspaketin kopiointi työasemalle

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3
SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
KEY=/folder/subfolder/subsubfolder/subsubsubfolder
HAS_SQUISH="^\s*squish\s*=\s*(.*)\s*$"

EX-
TRACT_SYSTEMTESTING_BAT=$SCRIPTSDIR/test_automation_process/extractSys
temtesting.bat

WORKSTATION_CONFS=$SCRIPTSDIR/test_automation_process/workstations

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

WORKSTATIONS=()
I=0
for ENV in "${ENV_LIST[@]}"; do
    RMDIR_SYSTEMTESTING="cmd /c cd \ & cd folder & cd $ENV & IF EXIST
systemtesting rmdir /s /q systemtesting"
    SYSTEMTESTING_EXTRACTION="cmd /c cd \ & cd folder & cd $ENV & ex-
tractSystemtesting.bat"

    IFS=$'\r\n' TMP_LIST=($(cat
$SCRIPTSDIR/environments/$ENV/workstations.conf | sed -e
'1s/^\xef\xbb\xbf//'))
    for ITEM in "${TMP_LIST[@]}"; do
        WORKSTATIONS[$I]=$ITEM
        I=$(( $I + 1 ))
    done

    for WS in ${WORKSTATIONS[@]}; do
        if [ -f $WORKSTATION_CONFS/$WS.properties ]; then
            FILE=$WS.properties
            while read LINE || [[ -n "$LINE" ]]; do
                if [[ $LINE =~ $HAS_SQUISH ]]; then
                    CLIENT=jenkins@$WS
                    ssh -n -i $KEY $CLIENT $RMDIR_SYSTEMTESTING
                    sftp -b /dev/stdin -o "PubkeyAuthentication yes" -
o "IdentityFile=$KEY" $CLIENT << EOF
                    cd $ENV
                    put $SCRIPTSDIR/../systemtesting*.zip
                    put $EXTRACT_SYSTEMTESTING_BAT
                    exit
EOF
                    ssh -n -i $KEY $CLIENT $SYSTEMTESTING_EXTRACTION
                    break
                    fi
            done < $WORKSTATION_CONFS/$FILE
        fi
    done
done
```

Liite 12. Squish asennus työasemalla

```
set SEVENZIP="C:\Program Files\7-Zip\7z.exe"
set SEVEN86ZIP="C:\Program Files (x86)\7-Zip\7z.exe"

if NOT exist %SEVENZIP% set SEVENZIP=%SEVEN86ZIP%
%SEVENZIP% x systemtesting*.zip -aoa
```

Liite 13. Squish-parametrien kopiointi työasemalle

```
JOB=$1
PROJECT=$2
ENVIRONMENT=$3

SCRIPTSDIR=/var/www/$PROJECT/$JOB/scripts
WORKSTA-
STA-
TION_DIRS=$SCRIPTSDIR/test_automation_process/workstationSquishConfig

KEY=/folder/subfolder/subsubfolder/subsubsubfolder

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

WORKSTATIONS=()
I=0
for ENV in "${ENV_LIST[@]}"; do
    IFS=$'\r\n' TMP_LIST=($(cat
$SCRIPTSDIR/environments/$ENV/workstations.conf | sed -e
'1s/^\xef\xbb\xbf//'))
    for ITEM in "${TMP_LIST[@]}"; do
        WORKSTATIONS[$I]=$ITEM
        I=$(( $I + 1 ))
    done

    for WS in ${WORKSTATIONS[@]}; do
        if [ -f $WORKSTATION_DIRS/$WS/squishParameters.txt ]
        then
            SQUISH_DIR=$WORKSTATION_DIRS/$WS/squishParameters.txt
            CLIENT=jenkins@$WS
            sftp -b /dev/stdin -o "PubkeyAuthentication yes" -o "Iden-
tityFile=$KEY" $CLIENT << EOF
            cd $ENV
            cd miniclient/testmaterial
            put $SQUISH_DIR
            exit
EOF
        fi
    done
done
```

Liite 14. Testisession keston seuranta Jenkinsiltä käsin

```
HOST=11.222.3.44
TARGET=jenkins@$HOST

SOURCE_DIR=test_automation_process/testSessionMonitoring
SCRIPT=pollOngoingTests.sh

TARGET_DIR=/var/www/test_monitoring

if [ -d stop_needed ]; then
    rm -rf stop_needed
fi
mkdir stop_needed

if [ -f results.txt ]; then
    rm -f results.txt
fi

touch results.txt

scp -B $SOURCE_DIR/$SCRIPT $TARGET:$TARGET_DIR
ssh -n $TARGET "sh $TARGET_DIR/$SCRIPT" >> results.txt
LINE_PATTERN="^(.*)\s([0-9]+)$"

CURRENT_TIME=`date +%s`

I=1

while read LINE; do
    if [[ $LINE =~ $LINE_PATTERN ]]; then
        WORK_DIR=${BASH_REMATCH[1]}
        STOP_TIME=${BASH_REMATCH[2]}
        if [ $CURRENT_TIME -ge $STOP_TIME ];  then
            echo "directory=$WORK_DIR" >
stop_needed/stop.$I.properties
            I=$(( I + 1 ))
        fi
    fi
done < results.txt
```

Liite 15. Testien sammuttaminen

```
WORK_DIR=$1

if [ -f $WORK_DIR/end_time.properties ]; then
       rm -f end_time.properties
fi

ENV_PATTERN="ongoing_tests_(.*)-[0-9]+"
ENV=""
if [[ $WORK_DIR =~ $ENV_PATTERN ]]; then
    ENV=${BASH_REMATCH[1]}
fi

echo "Killing all Java instances on workstations in use"

WS_NAME="([^\/]+)\.properties$"
for WS in $WORK_DIR/workstations/*.properties; do
        if [[ $WS =~ $WS_NAME ]]; then
                echo "Killing Java instances at $WS"
                sh $WORK_DIR/killJavaProcesses.sh ${BASH_REMATCH[1]}
$ENV
        fi
done

rm -rf $WORK_DIR
```

Liite 16. Työaseman Java-prosessien tappaminen

```
WORKSTATION=$1
ENV=$2

if [ "$WORKSTATION" == "" ]; then
    echo "ERROR: Workstation was not given as parameter!"
    exit 1
elif [ "$ENV" == "" ]; then
    echo "ERROR: Environment was not given as parameter!"
    exit 1
fi

TMP_FILE=output.txt

COMMAND1="wmic process where \"commandline like '%$ENV%%miniclient%'\"
get processid,commandline"
COMMAND2="wmic process where \"commandline like '%$ENV%%run_tests%'\"
get processid,commandline"

KEY=/folder/subfolder/subsubfolder/subsubsubfolder

REMOTE=jenkins@$WORKSTATION

ssh -n -i $KEY $REMOTE $COMMAND1 > $TMP_FILE
ssh -n -i $KEY $REMOTE $COMMAND2 >> $TMP_FILE

JAVA_COMMAND="java\.exe"
CMD_COMMAND="cmd\.exe"
LAST_NUMBER="([0-9]+)\s*.?$"
CMD_PIDS=()
JAVA_PIDS=()
C=0
J=0
while read LINE; do
    if [[ $LINE =~ $CMD_COMMAND ]]; then
        if [[ $LINE =~ $LAST_NUMBER ]]; then
            CMD_PIDS[$C]=${BASH_REMATCH[1]}
            C=$(( C + 1 ))
        fi
    elif [[ $LINE =~ $JAVA_COMMAND ]]; then
        if [[ $LINE =~ $LAST_NUMBER ]]; then
            JAVA_PIDS[$J]=${BASH_REMATCH[1]}
            J=$(( J + 1 ))
        fi
    fi
done < $TMP_FILE
SORTED_JAVA_PIDS=$(echo "${JAVA_PIDS[@]}" | tr ' ' '\n' | sort -u | tr
'\n' ' ')
SORTED_CMD_PIDS=$(echo "${CMD_PIDS[@]}" | tr ' ' '\n' | sort -u | tr
'\n' ' ')

for PID in ${SORTED_JAVA_PIDS[@]}; do
    echo "Killing java with PID $PID"
    ssh -n -i $KEY $REMOTE "taskkill /PID $PID"
done

sleep 2

for PID in ${SORTED_CMD_PIDS[@]}; do
    echo "Killing cmd with PID $PID"
    ssh -n -i $KEY $REMOTE "taskkill /PID $PID"
done
```

Liite 17. Työasemakohtaisten tulosten kerääminen

```
setlocal enabledelayedexpansion
SET ENVIRONMENT=%1%
SET WS=%2%

SET RESPONSE_FOLDER=c:\response
SET RE-
SULTS_FOLDER=%RESPONSE_FOLDER%\workstation_dump_%ENVIRONMENT%_%WS%
SET MINICLIENT_FOLDER=%RESULTS_FOLDER%\miniclient
SET JMX_FOLDER=%RESULTS_FOLDER%\jmx
SET JMETER_FOLDER=%RESULTS_FOLDER%\JMeter
SET MINICLI-
ENT_LOGS_FOLDER=%RESPONSE_FOLDER%\%ENVIRONMENT%\miniclient\logs
SET MINICLI-
ENT_RESULTS_FOLDER=%RESPONSE_FOLDER%\%ENVIRONMENT%\miniclient\miniclie
ntTestResults
SET MINICLI-
ENT_JMX_RESULTS_FOLDER=%RESPONSE_FOLDER%\%ENVIRONMENT%\miniclient\moni
toring
SET JMETER_RESULTS_FOLDER=%RESPONSE_FOLDER%\%ENVIRONMENT%\JMeter\logs

SET ZIPNAME_MINICLIENT_LOGS=miniclientLogs.zip
SET ZIPNAME_MINICLIENT_TESTRESULTS=miniclientTestResults.zip
SET ZIPNAME_CLIENT_LOGS=clientLogs.zip
SET ZIPNAME_CLIENT_HEAPDUMPS=clientHeapdumps.zip
SET ZIPNAME_JMX_RESULTS=JMXMonitoring.zip
SET ZIPNAME_JMETER_RESULTS=JMeterResults.zip

mkdir %RESULTS_FOLDER%
echo zipping in progress... >> %RESPONSE_FOLDER%/status.txt
mkdir %MINICLIENT_FOLDER%
mkdir %JMX_FOLDER%
mkdir %JMETER_FOLDER%

SET M=1024
cd %RESPONSE_FOLDER%\%ENVIRONMENT%\miniclient\logs\
FOR /D %%d in (*) do (
    if exist %%d\client\softphone (
        set size=0
        for /f "tokens=*" %%x in ('dir /s /a /b') do set /a
size+=%%~zx
        echo !size!
        if (%size% gtr (40 * %M%))
            rmdir %%d\client\softphone /s /q
    )
)

"c:\Program Files\7-Zip\7z.exe" a %MINICLI-
ENT_FOLDER%\%ZIPNAME_MINICLIENT_LOGS% %MINICLIENT_LOGS_FOLDER%
"c:\Program Files\7-Zip\7z.exe" a %MINICLI-
ENT_FOLDER%\%ZIPNAME_MINICLIENT_TESTRESULTS% %MINICLI-
ENT_RESULTS_FOLDER%
"c:\Program Files\7-Zip\7z.exe" a %JMX_FOLDER%\%ZIPNAME_JMX_RESULTS%
%MINICLIENT_JMX_RESULTS_FOLDER%
"c:\Program Files\7-Zip\7z.exe" a
%JMETER_FOLDER%\%ZIPNAME_JMETER_RESULTS% %JMETER_RESULTS_FOLDER%
"c:\Program Files\7-Zip\7z.exe" a %RE-
SPONSE_FOLDER%\workstation_dump_%ENVIRONMENT%-%WS%.zip %RE-
SULTS_FOLDER%

rmdir /S /Q %RESULTS_FOLDER%
exit
```

Liite 18. Palvelinkohtaisten tulosten kerääminen

```
ENVIRONMENT=$1

LATEST_VANILLA=`ls -dt /folder/subfolder/subsubfolder/[0-9]*/ | head -
n 1`
KEY=$LATEST_VANILLA/folder/subfolder/file.someFile

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do
   LATEST_CONF="$LATEST_VANILLA/folder/subfolder/$ENV/servers.conf"

   IFS=$'\n' SERVERS=`cat $LATEST_CONF | sed 's/\// /g' | sed -e
'1s/^\xef\xbb\xbf//' | awk '{print  $1}'`
   for IP in $SERVERS; do
     REMOTE=admin@$IP
     ssh -n -i $KEY $REMOTE "nohup sh /opt/response-
scripts/zip_heapdumps_n_logs.sh > nohup.out 2>nohup.out < /dev/null &"
   done
done
```

Liite 19. Tulosten keräämisen aloittaminen

```
ENVIRONMENT=$1
JOB=$2

LATEST_VANILLA=`ls -dt /var/www/response-vanilla/[0-9]*/ | head -n 1`
KEY=$LATEST_VANILLA/folder/subfolder/file.someFile

SCRIPTSDIR=/folder/subfolder/subsubfolder/
DUMP_TARGET_DIR="$JOB-$ENVIRONMENT-dump"

# Dump service logs & heapdumps on servers
echo "====================================="
echo "| Dumping service logs & heapdumps! |"
echo "====================================="
sh $SCRIPTSDIR/storage/dump_services_logs.sh $ENVIRONMENT

# Dump workstation logs & heapdumps on workstations
echo "=============================="
echo "| Dumping workstation dumps! |"
echo "=============================="
sh $SCRIPTSDIR/workstation/dump_workstation.sh $ENVIRONMENT

echo
"=============================================================================
======================"
echo "| Dumping initialized, now waiting for them to end and then mov-
ing dumps to mohave-export |"
echo
"=============================================================================
======================"

sh $SCRIPTSDIR/workstation/moveWorkstationDumpsToMohaveExport.sh $EN-
VIRONMENT $DUMP_TARGET_DIR/workstation
sh $SCRIPTSDIR/storage/moveServiceLogsToMohaveExport.sh $ENVIRONMENT
$DUMP_TARGET_DIR/service-logs

echo "Dump is found here: @mohave-export: /mohave-
export/$DUMP_TARGET_DIR"

# Dump database dump to mohave-export
echo "=========================="
echo "| Dumping databse dumps! |"
echo "=========================="
sh $SCRIPTSDIR/storage/dump_database.sh $ENVIRONMENT
$DUMP_TARGET_DIR/database
```

Liite 20. Palvelinkohtaisten tulosten siirtäminen verkkolevyllä

```
ENVIRONMENT=$1
DUMP_TARGET_DIR=$2

LATEST_VANILLA=`ls -dt /folder/subfolder/subsubfolder/[0-9]*/ | head -
n 1`

GZIP_PATTERN="^\s*(.*)\s*zip_heapdumps_n_logs.sh\s*(.*)\s*$"

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do
        LAT-
EST_CONF="$LATEST_VANILLA/scripts/environments/$ENV/servers.conf"
        KEY=$LATEST_VANILLA/scripts/storage/admin.private

        IFS=$'\n' SERVERS=`cat $LATEST_CONF | sed 's/\// /g' | sed -e
'1s/^\xef\xbb\xbf//' | awk '{print  $1}'`
        for IP in $SERVERS; do
                echo "==================================[START]
$IP======================================="
                REMOTE=admin@$IP
                ZIPPING_STATUS=`ssh -n -i $KEY $REMOTE ps aux | grep
zip`
                while [[ $ZIPPING_STATUS =~ $GZIP_PATTERN ]]; do
                        echo "Target zip not found from $IP in $ENV,
sleeping for 30 seconds"
                        sleep 30
                        ZIPPING_STATUS=`ssh -n -i $KEY $REMOTE ps aux
| grep zip`
                done

                ssh -n -i $KEY admin@mohave-export "mkdir -p
/mohave_export/$DUMP_TARGET_DIR"
                ssh -n -i $KEY $REMOTE "scp -o StrictHostKeyCheck-
ing=no -i /opt/response-scripts/admin.private /opt/response-
scripts/*tar.gz admin@mohave-export:/mohave_export/$DUMP_TARGET_DIR/"
                ssh -n -i $KEY $REMOTE "sudo rm -f /opt/response-
scripts/nohup.out"
                ssh -n -i $KEY $REMOTE "sudo rm -f /opt/response-
scripts/*tar.gz"
                echo "==================================[STOP]
$IP======================================="
        done
done
```

Liite 21. Työasemakohtaisten tulosten siirto verkkolevylle

```
ENVIRONMENT=$1
WORKSTATIONS_DUMP_DIR=$2

LATEST_VANILLA=`ls -dt /folder/subfolder/subsubfolder/[0-9]*/ | head -
n 1`

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do
    LAT-
EST_CONF="$LATEST_VANILLA/scripts/environments/$ENV/workstations.conf"
    KEY=$LATEST_VANILLA/scripts/storage/admin.private
    WORKSTATION_KEY=/home/jenkins/keys/response

    IFS=$'\n' WORKSTATIONS=`cat $LATEST_CONF | sed 's/\// /g' | sed -e
'1s/^\xef\xbb\xbf//' | awk '{print  $1}'`
    for WS in $WORKSTATIONS; do
        echo "==================================[START]
$WS======================================="
        echo "Start moving packages from $WS in $ENV to mohave-
export..."
        REMOTE=jenkins@$WS
        WORKSTATION_DUMP=workstation_dump_$ENV-$WS.zip
        IS_DUMP_DONE="cmd /c cd \ & cd response & cd work-
station_dump_$ENV_$WS & IF EXIST status.txt echo sleep"
        REMOVE_WORKSATION_DUMP="cmd /c cd \ & cd response & del $WORK-
STATION_DUMP"

        STATUS=`ssh -n -i $KEY $REMOTE $IS_DUMP_DONE`
        while [ "$STATUS" == "sleep" ]; do
            echo "Dump on $WS in $ENV is not ready yet, sleeping for
30 seconds..."
            sleep 30
            STATUS=`ssh -n -i $KEY $REMOTE $IS_DUMP_DONE`
        done

        sftp -b /dev/stdin -o "PubkeyAuthentication yes" -o "Identi-
tyFile=$WORKSTATION_KEY" $REMOTE << EOF
        get $WORKSTATION_DUMP
        exit
EOF
        du -h $WORKSTATION_DUMP
        ssh -n -i $KEY admin@mohave-export "mkdir -p
/mohave_export/$WORKSTATIONS_DUMP_DIR"
        scp -o StrictHostKeyChecking=no -i $KEY $WORKSTATION_DUMP ad-
min@mohave-export:/mohave_export/$WORKSTATIONS_DUMP_DIR/
        ssh -n -i $WORKSTATION_KEY $REMOTE $REMOVE_WORKSATION_DUMP
        echo "==================================[STOP]
$WS======================================="
    done
done
```

Liite 22. Tietokanta vedoksen ottaminen ja siirto verkkolevylle

```
ENVIRONMENT=$1
DUMP_DB_DIR=$2

LATEST_VANILLA=`ls -dt /folder/subfolder/subsubfolder/[0-9]*/ | head -
n 1`

IFS=$',' ENV_LIST=($(echo "$ENVIRONMENT"))

for ENV in "${ENV_LIST[@]}"; do
   LAT-
EST_CONF="$LATEST_VANILLA/scripts/environments/$ENV/database.conf"
   KEY=$LATEST_VANILLA/folder/subfolder/file.someFile

   IFS=$'\n' SERVERS=`cat $LATEST_CONF | sed 's/\// /g' | sed -e
'1s/^\xef\xbb\xbf//' | awk '{print  $1}'`
   for IP in $SERVERS; do
      REMOTE=admin@$IP
      scp -o StrictHostKeyChecking=no -i $KEY $KEY $RE-
MOTE:/opt/response-scripts/
      ssh -n -i $KEY $REMOTE "sudo chmod 400 /opt/response-
scripts/admin.private"
      CREATED_DUMPS=`ssh -n -i $KEY $REMOTE "sh /opt/response-
scripts/database_dump.sh postgres --remove-obsolete" | grep "Database
dump completed" | sed 's/^.*: //g'`
      ssh -n -i $KEY admin@mohave-export "mkdir -p
/mohave_export/$DUMP_DB_DIR"
      for DUMP in $CREATED_DUMPS; do
        ssh -n -i $KEY $REMOTE "scp -o StrictHostKeyChecking=no -i
/opt/response-scripts/admin.private $DUMP admin@mohave-
export:/mohave_export/$DUMP_DB_DIR/"
      done
      break
   done
   break
done
```