HAAGA-HELIA
ammattikorkeakoulu

Mr.Goodliving Oy:n palvelinympäristön kehittäminen
CASE: MySQL varmuuskopiointi

Janne Kaistinen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2011

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

26.4.2011

Tietojenkäsittelyn koulutusohjelma

Tekijät Janne Kaistinen	**Aloitusvuosi** 2003
Opinnäytetyön nimi Mr.Goodliving Oy:n palvelinympäristön kehittäminen CASE: MySQL varmuuskopiointi	**Sivu- ja liitesivumäärä** 35 + 65
Ohjaaja Juhani Merilinna	

Tässä opinnäytetyössä toteutettiin Mr.Goodliving Oy:lle sovellus MySQL tietokannan varmuuskopiointi- ja palautusprosessin automatisointia varten. Tavoitteena oli olemassa olevan prosessin laadun ja luotettavuuden parantaminen valitsemalla sopivat sovellukset eri alitehtävien toteutusta varten ja parantamalla raportointia ja prosessin valvontaa. Sovelluksen tavoitteena ei siis ollut varsinaisen varmuuskopioinnin suorittaminen vaan kokonaisprosessin hallinta.

Työ aloitettiin esiselvityksellä, jossa etsittiin opinnäytetyössä toteutettavaa kohdetta. Esiselvityksen perusteella valitun sovelluksen toteutus aloitettiin vaatimusmäärittelyllä ja varmuuskopiointiprosessin manuaalisella mallinnuksella, jonka perusteella viimeisessä vaiheessa toteutettiin varsinainen sovellus. Toteutuskieleksi valittiin Python, joka soveltui hyvin eri sovellusten suorittamiseen ja valvontaan. Varsinainen varmuuskopiointi toteutettiin XtraBackup-sovelluksen avulla, joka mahdollisti MySQL tietokannan vaiheittaisen ja täyden varmistuksen ottamisen tietokannan ollessa käytössä.

Varsinaisen sovelluksen lisäksi opinnäytetyön lopputuloksina syntyi sovelluksen käyttöohje, varmuuskopiointi- ja palautusprosessin kuvaus, testausdokumentaatio ja palvelinympäristön dokumentaatio.

Opinnäytetyön lopputuloksena saatiin aikaiseksi sovellus, joka toteutti sille asetetut vaatimusmäärittelyt. Sovellus otettiin myös onnistuneesti käyttöön tuotannossa ja sen avulla otettiin viikoittaiset varmistukset MySQL tietokannasta.

Kehityskohteina esille nousi sovelluksen julkaiseminen avoimenlähdekoodin lisenssillä. Tavoitteena on, että sovellusta voivat käyttää kohdeyrityksen lisäksi myös muut tahot MySQL tietokannan varmistukseen. Tämä vaatii kuitenkin jatkokehitystä ja sovelluksen rakenteen muokkausta siten, että toiminnallisuuksia voidaan laajentaa ilman sovelluksen lähdekoodin muokkausta.

Asiasanat
varmuuskopionti, tietokannat, relaatiotietokannat, SQL, python

HAAGA-HELIA
ammattikorkeakoulu

Abstract

26 April 2011

Degree programme in Business Information Technology

Author Janne Kaistinen	**Group** 2003
The title of theses Improving Mr.Goodliving Oy server infrastructure: CASE: MySQL backup	**Number of pages and appendices** 35 + 65
Supervisor Juhani Merilinna	

The purpose of this thesis was to implement a solution for the automation of MySQL database backup and restore process for Mr.Goodliving Oy. The objective was to increase the quality and reliability of the existing process by choosing different solutions for the actual tasks required for the backup and restore process. In addition, the quality was improved by better reporting and process control. Therefore, the purpose was not to implement actual backup but to control the whole process.

The work started with a preliminary feasibility study. The MySQL backup solution was selected and the actual project started with a requirement specification and modeling the process manually. In the actual implementation phase, a Python application was developed to automate the result of the manual process. XtraBackup application was selected as the actual program for performing the backup because it could perform incremental and full backups of MySQL database while it was in use.

In addition to the actual application, the end results included a manual for the application, backup and recovery process specification, testing documentation and server documentation.

The final result of this thesis was an application which fulfilled all requirements set for it. The application was taken successfully into production and it was used to take weekly backups of the target MySQL database.

Publishing the solution as open source software was suggested as a development goal. The purpose is that the application could be used by third party users for MySQL backup. However, this would require further development by modifying the source code so that more functionality could be added without modifying the base source code.

Key words
backup, databases, relational databases, SQL, python

Sisällys

1 Johdanto

Tietojärjestelmien toiminta ja luotettavuus ovat olennaisia asioita kaikille IT-järjestelmiä käyttäville yrityksille. Erityisesti tietopalveluita ja immateriaalisia hyödykkeitä tuottavilla yrityksillä tietojärjestelmät ja niihin tallennettu tieto muodostavat suuren osan yrityksen arvosta. Tietojärjestelmiin kohdistuvaa onnettomuutta voikin verrata hyödykkeitä valmistavan yrityksen tilojen, varaston ja työkoneiden tuhoutumiseen. Kummassakin tapauksessa yrityksen toiminta keskeytyy ja pitkittyneestä katkoksesta aiheutuu yritystoiminalle merkittävää haittaa. (Preston 2007, 3 – 12.)

IT-järjestelmät voidaan kuitenkin palauttaa toimintakuntoon suhteellisen pienillä kustannuksilla, jos tietojärjestelmiin kohdistuvilta riskeiltä on suojauduttu etukäteen. Yritysten täytyy myös arvioida omien tietojärjestelmiensä kriittisyys liiketoiminnan kannalta ja pyrkiä löytämään tasapaino onnettomuudesta toipumisen ja tietojen suojauksesta aiheutuvien kustannusten välillä. (Preston 2007, 12.)

Yrityksen oman toiminnan ja toimintaympäristön muuttuessa myös käytettävien tietojärjestelmien arvo muuttuu. Järjestelmien uudelleenarviointi liiketoiminalle aiheutuvien riskien osalta tulisikin olla osa tietojärjestelmienkehittämistä. Arvioinnin perusteella yrityksen tietojen suojaukseen toimenpiteitä ja käytäntöjä pitää tarvittaessa myös muuttaa.

Tässä opinnäytetyössä toteutettiin Mr.Goodliving Oy:lle sovellus MySQL tietokannan varmuuskopiointi- ja palautusprosessin automatisoinniksi. Opinnäytetyö aloitettiin esiselvityksellä palvelinympäristön kehitystarpeista, jonka perusteella valittiin toteutettavaksi sovellus.

Esiselvityksessä todettiin, että yrityksen prosessien ja toimintaympäristön muutokset vaativat MySQL varmuuskopioinnin laadun ja luotettavuuden parantamista. Konkreettisia syitä olivat tietokantaan tallennettavan tiedon määrän nopea kasvu, täyden varmistuksen pitkä ajallinen kesto ja tietokannan käyttöasteen muuttuminen ympärivuorokautiseksi.

1.1 Tavoitteet

Esiselvityksen ja vaatimusmäärittelyn perusteella opinnäytetyössä toteutettavalle sovellukselle asetettiin seuraavat tavoitteet:

Varmistustavat	Sovelluksen pitää pystyä ottamaan täysi ja vaiheittainen varmistus tietokannasta säilyttäen tietokannan eheys.
Tietokantamoottorit	Varmuuskopioinnin on tuettava varmistuksen ottamista MyISAM ja InnoDB tyyppisistä tauluista. Muiden MySQL yhteensopivien tietokantamoottorien tukea ei vaadita.
Käyttöaste	Varmistus on pystyttävä toteuttamaan ilman tietokannan tai siitä riippuvien palveluiden sammuttamista varmuuskopioinnin ajaksi.
Tallennus	Varmuuskopio pitää pystyä tallentamaan joko tietokantapalvelimen paikalliselle levylle tai suoraan varmuuskopiointipalvelimelle ilman väliaikaistiedostoja.
Eheys	Palautetun tietokannan eheydestä on voitava varmistua. Lisäksi varmistuksen on oltava atominen operaatio, jossa varmistus kuvaa tietokannan tilaa varmistushetkellä.
Raportointi	Varmuuskopioinnin lopputulos ja edistyminen pitää raportoida sekä lokitiedostojen ja muiden raportointikeinojen avulla.
Automatisointi ja ajastus	Varmuuskopiointi pitää pystyä automatisoimaan siten, että sen suoritus ei vaadi manuaalisia toimenpiteitä. Lisäksi varmuuskopiointi on pystyttävä ajastamaan suoritettavaksi haluttuina ajankohtina.
Palautus	Palautusprosessin pitää pystyä palauttamaan tietokanta täydestä varmistuksesta ja sen jälkeisistä vaiheittaisista varmistuksista. Palautuksen ei tarvitse kuitenkaan automatisoida muita tietokannan palautukseen liittyviä toimenpiteitä tai esimerkiksi eheyden tarkistusta.

1.2 Käsitteet

InnoDB	InnoDB on yksi MySQL tietokannan tietokantamoottoreista. Se tarjoaa ACID määritystä vastaavan transaktio-ominaisuudet ja viiteavaimet.
Järjestelmän palautus	Järjestelmän palautus tarkoittaa koko prosessia, jonka suoritettua tietojärjestelmä on käytettävissä.
Korkea saatavuus	Korkea saatavuus tarkoittaa sitä, että tietojärjestelmän toimintakyky säilyy suunnitelluista (esimerkiksi laitehuolto) tai suunnittelemattomista (esimerkiksi palvelunestohyökkäys) häiriöistä huolimatta.
Kuuma varmistus	Kuumassa varmistuksessa tietojärjestelmä on käytettävissä koko varmistuksen suorituksen ajan.
Kylmä varmistus	Kylmässä varmistuksessa tietojärjestelmä on suljettu varmuuskopioinnin ajan.
MyISAM	MyISAM on MySQL tietokannan oletus tietokantamoottori ennen versiota 5.5. MyISAM tietokantamoottorin suurin puute on sen rajallinen tuki transaktioille.
Peilaus	Peilauksella tarkoitetaan tietojen jatkuvaa kopiointi kahteen eri fyysiseen tallennuspaikkaan.
Replikointi	Replikointi tarkoittaa saman tiedon hajauttamista moneen paikkaan. Esimerkiksi tietokannan replikoinnissa kaikki tietokantaan tehdyt muutokset kopioidaan jatkuvasti toiseen tietokantainstanssiin.
Tietojen suojaus	Tietojen suojauksella tarkoitetaan kaikkia toimenpiteitä, joilla yrityksen tuottamaa informaatio suojellaan kaikilta sille mahdollisesti tapahtuvilta asioilta.
Tilannevedos	Tiedon lukeminen tai kopiointi yhteen ajankohtaan sidottuna siten, että tiedon eheys säilyy.

Täysi varmistus	Täysi varmistus tarkoittaa kaikkien valittujen tietojen kopioimista. Jokainen täysi varmistus sisältää kaiken varmistettavan tiedon.
Vaiheittainen varmistus	Vaiheittainen varmistus tarkoittaa prosessia, jossa edellisen varmistuksen jälkeen muuttuneet tai lisätyt tiedot tallennetaan.
Valikoiva varmistus	Valikoivassa varmistuksessa kaikki täyden varmistuksen jälkeen muuttuneet tai lisätyt tiedot tallennetaan. Erona vaiheittaiseen varmistukseen on se, että lähtötilanteena on aina täysi varmistus.
Varmuuskopiointi	Varmuuskopiolla tarkoitetaan toimintaa, jossa tieto kopioidaan ja varastoidaan. Jos alkuperäinen tieto häviää, voidaan se palauttaa varmuuskopiosta.

2 Teoriatausta

Opinnäytetyössä toteutettiin sovellus MySQL tietokannan varmuuskopioinnin ja palautuksen automatisointia varten. Laajemmasta näkökulmasta katsoen sovellus on kuitenkin vain yksi osa yrityksen tietojen suojauksen toteutusta.

Tietojen suojauksella tarkoitetaan kaikkia toimenpiteitä, joilla yrityksen tuottamaa informaatiota suojellaan sille mahdollisesti tapahtuvilta asioilta. Tietojen suojaamisen aktiivisiin toimenpiteisiin kuuluvat muun muassa varmuuskopiointi, palautus, arkistointi, tietoturva ja onnettomuudesta toipuminen. Yrityksen täytyy koosta riippumatta arvioida säännöllisesti omat tarpeensa tietojen suojaukselle ja päättää mitä tulokset tarkoittavat käytännön toimenpiteinä. (Preston 2007, 689.)

Yrityksen toiminnasta ja toimialasta riippuen voi tietojen suojaamiseen liittyä myös ulkoisia vaatimuksia ja säädöksiä. Esimerkiksi henkilötietorekisterien ylläpitoa säädellään laissa, joka määrittää henkilötietojen käsittelylle ja tallennukselle omat vaatimukset (Tietosuojavaltuutetun toimisto 2010). Tietojen suojauksessa on otettava huomioon myös nämä vaatimukset.

Tietojen suojausta voidaan myös käyttää muihin tarkoituksiin kuin hävinneen tai korruptoituneen tiedon palautukseen. Tietojen suojaus auttaa hakemaan tietoa, joka on siirretty tai jota tarvitaan eri muodossa kuin se on alun perin tallennettu. Lisäksi se estää tiedon päätymistä vääriin käsiin. (Preston 2007, 689.)

2.1.1 Tietojen suojaus liiketoiminnan näkökulmasta

Liiketoiminnan kannalta tietojen suojauksella pyritään pienentämään riskejä, vähentämään kustannuksia, luokittelemaan tietoa, estämään tietomurtoja sekä nostamaan palveluiden käyttöastetta. Tavoitteista keskeisin on riskienhallinta. Useimmat yritykset tarvitsevat keskeytymätöntä pääsyä kriittiseen liiketoimintatietoon ja esteet tiedon saatavuudessa voivat häiritä tärkeitä toimintoja, kuten tilausten vastaanottoa tai laskutusta. Lisäksi häiriöt voivat vaikuttaa liiketoimintakumppanien ja asiakkaiden toimintaan ja vahingoittaa näin yrityksen imagoa. (Preston 2007, 690 – 692.)

Tietojärjestelmien häiriöt aiheuttavat suoria kustannuksia yrityksen toiminnan keskeytymisestä johtuen. Näiden kustannuksien määrä riippuu liiketoiminnan luonteesta, häiriön kohteesta ja sen kestosta. Suorien kustannusten lisäksi häiriöt saattavat aiheuttaa epäsuoria kustannuksia

esimerkiksi menetettyinä tilauksina. Onnistunut tietojen suojaus minimoi nämä kustannukset. (Preston 2007, 690 – 692.)

Tietojen suojaamisen tavoite on siis ennaltaehkäistä ja vähentää erilaisten häiriötekijöiden vaikutuksia eri liiketoiminnan osa-alueisiin. Haasteena on löytää tasapaino riskien ja niiltä suojautumisesta aiheutuvien kustannusten välillä. Esimerkiksi tietoturvan parantaminen tiukalla käyttöoikeuksien rajaamisella pienentää riskejä ja parantaa tietoturvaa, mutta aiheuttaa kustannuksia kasvaneen byrokratian muodossa. (Preston 2007, 690 – 692.)

2.1.2 Tietojen suojaus teknisestä näkökulmasta

Teknisesti tietojen suojaus vaatii usein monia erilaisia rinnakkaisia ratkaisuja, joka tekee täydellisestä tietojen suojauksesta haastavan. Suurin ongelma on häiriötilojen erilaisuus ja niiden määrä, mistä johtuen yksittäinen ratkaisu ei voi kattaa kaikkia ongelmatilanteita. Laiteviat, ohjelmistoviat, inhimilliset virheet ja verkotetun tiedon tallennuksen sisäänrakennetut riskit vaativat erilaisia lähestymistapoja ongelmien ratkaisuun ja estämiseen. (Preston 2007, 693 – 695.)

Yksittäisistä riskeistä suurin on laitteistovika. Esimerkiksi noin kolme prosenttia kovalevyistä hajoaa kolmen kuukauden aikana käyttöönotosta (Barroso, Pinheiro & Weber 2007, 4). Ottaen huomioon IT-järjestelmien toteuttamiseen tarvittavien laitteiden suuren lukumäärän, laitevian todennäköisyys nousee laitteiden lukumäärän kasvun myötä suureksi.

Haasteita tietojen suojaukselle aiheuttavat myös sisäisten ja ulkoisten uhkien torjuminen. Virukset, madot ja troijalaiset ovat kaikki riskejä, joista toipumiseen tarvitaan usein tietojen palautusta. Myös sisäiset uhat kuten tahaton tai tahallinen tietojen tuhoaminen työntekijän toimesta vaativat tietojen suojaamista. (Preston 2007, 693 – 695.)

2.2 Jatkuvuus- ja toipumissuunnitelma

Jotta yrityksen tietojärjestelmiä kohtaavasta ongelmista voidaan selvitä, tarvitaan jatkuvuus- ja toipumissuunnitelma. Tällä tarkoitetaan kaikkia prosesseja, ohjeistuksia ja suunnitelmia, joiden avulla yrityksen toimintaa pystytään jatkamaan onnettomuuden tapahduttua. Lähtökohtana jatkuvuus- ja toipumissuunnitelmalle on pahin mahdollinen tapahtuma, joka voi koskea IT-järjestelmiä. Esimerkiksi koko tietokonekeskuksen tuhoutuminen voi olla yrityksen kohdalla tämänkaltainen tapahtuma. (Bell, Kindahl, & Thalmann 2010, 410 – 413.)

Ennen suunnitelman tekoa on pystyttävä vastaamaan kysymyksiin, jotka määrittelevät jatkuvuus- ja toipumissuunnitelman lähtökohdat ja tavoitteet. Kysymyksiä voidaan käyttää myös arvioitaessa suunnitelman tehokkuutta. (Bell 2010, 413.)

– Mitkä ovat organisaation aineelliset ja aineettomat uhat?
– Mikä on IT-järjestelmien toiminta-aste, joka tarvitaan yrityksen toiminnan jatkamiseksi?
– Kuinka pitkään yrityksen toiminta voi olla pysähdyksissä?
– Mitä resursseja on käytettävissä jatkuvuus- ja toipumissuunnitelman suunnitteluun ja täytäntöönpanoon?

(Bell 2010, 413.)

Jatkuvuus- ja toipumissuunnitelman tavoitteena on palauttaa yrityksen toimintakyky mahdollisimman nopeasti. Hyvä suunnitelma käsittää koko IT-järjestelmän uudelleen rakentamisen alkaen uuden laitetilan hankkimisesta ja päättyen kaikkien liiketoiminnan kannalta tarpeellisten järjestelmien saamisesta toimintakuntoon halutussa laajuudessa. Lisäksi suunnitelmassa pitää ottaa huomioon tarvittavat rahoitus, henkilöresurssit, tiedottaminen ja vastuunjako. (Bell 2010, 413 – 415.)

Useimmat organisaatiot muodostavat ryhmän jatkuvuus- ja toimintasuunnitelman suunnittelua ja toteutusta varten. Ryhmän jäsenten tulee kattaa koko organisaatio, jotta kaikki liiketoiminnalle kriittiset riskit ja järjestelmät tunnistetaan. Oleellista on myös saada johdon tuki suunnitelmalle, jotta se voidaan tarvittaessa toteuttaa tehokkaasti ja kaikki tarvittavat resurssit ovat saatavissa. (Bell 2010, 415 – 416.)

Jatkuvuus- ja toimintasuunnitelmaa tarvitaan myös koordinoimaan eri palveluiden välisiä riippuvuussuhteita. Yksittäinen IT-järjestelmä on harvoin täysin riippumaton muista palveluista, esimerkiksi nimipalveluiden toimivuus on usein esiehto muiden palveluiden toiminnalle. (Bell 2010, 413- 419.)

Valmista suunnitelmaa pitää myös testata käytännössä. Testauksen tavoitteena on todistaa johdolle, että IT-järjestelmät pystytään palauttamaan toimintakuntoon halutulla toimintatasolla. Jos tähän ei pystytä, suunnitelmaa pitää päivittää ja testaus suorittaa uudelleen. Suunnitelma pitäisi testata vuosittain ja lisäksi aina kun IT-järjestelmien toteutus muuttuu oleellisesti. (Bell 2010, 417.)

2.3 Järjestelmän palautus

Sekä tietojen suojauksen ja jatkuvuus- ja toimintasuunnitelman lopputuloksena on saada aikaan prosessit, joiden avulla IT-järjestelmät saadaan onnettomuuden tapahtuessa palautettua toimintakuntoon mahdollisimman nopeasti (Bell 2010, 414). Yksittäisen järjestelmän kohdalla tämä tarkoittaa järjestelmän palautusta, johon kuuluvat kaikki tehtävät, jotka pitää suorittaa ennen kuin järjestelmä on toimintakuntoinen. Esimerkiksi MySQL palvelimen kohdalla tämä tarkoittaa palvelimen asennusta, asetustiedostojen palautusta, tietokannan palauttamista varmuuskopioista ja palvelimen uudelleen käynnistämistä. (Balling, Lentz, Schwartz, Tkachenko, Zaitsev & Zawodny 2008, 472.)

Järjestelmän palautuksessa varmuuskopioitu tieto pitää pystyä palauttamaan siten, että sen on identtinen alkuperäisen tiedon kanssa. Järjestelmän palautusprosessia on voitava myös seurata, jotta palautetun tiedon eheys voidaan varmistaa ja mahdolliset ongelmat paljastuvat. (Bell 2010, 422.)

2.3.1 Järjestelmän palautuksen mittaaminen

Järjestelmän palautusta voidaan arvioida kahdella suureella, jotka ovat toisistaan riippuvaisia (Bell, 419.)

Palautuksen toiminnallisuustavoite	Palautuksen toiminnallisuustavoite kertoo milloin järjestelmä on tuotantokelpoinen. Käänteisenä arvo kertoo mikä on suurin sallittu toimintakyvyn lasku tai tietojen tuhoutumisen määrä, jolloin järjestelmä on vielä toimintakykyinen.
Palautuksen aikatavoite	Palautuksen aikatavoite määrittää tavoiteajan, jonka kuluessa järjestelmän on oltava toimintakuntoinen häiriön tapahduttua. Aikatavoite voi vaihdella sadasosasekunneista päiviin riippuen tietojärjestelmästä ja sen tehtävästä.

Taulukko 1 Palautuksen mittaaminen (Bell 2010, 419)

Jos kaikki tai suurin osa tiedosta pitää pystyä palauttamaan, tarvitaan enemmän työtä toiminnallisuustavoitteen saavuttamiseksi. Tällöin lyhyt aikatavoite vaatii enemmän laitteisto- ja ohjelmistoinvestointeja. Jos esimerkiksi vaatimuksina on kaiken tiedon palauttaminen alle kolmessa sekunnissa, tarvitaan kattavia korkean saatavuuden ratkaisuja vaatimusten täyttämiseksi. (Bell 2010, 419.)

2.4 Varmuuskopiointi

Tärkein yksittäinen tehtävä tietojen suojauksessa on varmuuskopiointi. Ilman varmuuskopioita IT-järjestelmän hajoaminen voi aiheuttaa kaiken tallennetun tiedon häviämisen, jolloin myös tietojen suojaus on epäonnistunut. (Bell 2010, 419.)

Varmuuskopiointi pitää huomioida jo tietojärjestelmiä kehittäessä. Jos varmuuskopiointia ei oteta huomioon ajoissa, voivat parhaat tavat varmuuskopioinnille osoittautua mahdottomiksi toteuttaa. Jälkikäteen toteutettu varmuuskopiointi voi myös aiheuttaa yllättäviä ongelmia IT-järjestelmän suorituskyvyn kannalta. (Balling 2008, 472.)

2.4.1 Varmuuskopioinnin luokittelu

Varmuuskopioinnin kohteesta riippuen vaihtoehtoisia toteutustapoja voi olla useita. Eri menetelmien välillä voi olla suuria eroja nopeuden, tarvittavan tilan ja toteutuksen helppouden välillä. Vaikka varmuuskopiointi onkin yksinkertaistettuna vain kopio tallennetusta tiedosta, voi kopion ottaminen IT-järjestelmään tallennetusta tiedosta osoittautua vaikeaksi. (Balling 2008, 477.)

Yksi yleisesti käytetty tapa luokitella varmuuskopiointitapoja on tehdä se varmistettavan tiedon valinnan perusteella (Bell 2010, 422.)

Täysi varmistus	Täydessä varmistuksessa kaikki tietojärjestelmään tallennetut tiedot kopioidaan. Tämä varmuuskopioinnin muoto vie eniten aikaa ja tilaa.
Valikoiva varmistus	Valikoivassa varmistuksessa kopioidaan ainoastaan edellisen täyden varmistuksen jälkeen muuttunut tai lisätty tieto. Valikoiva varmistus on tavallisesti nopeampi ja vie vähemmän tilaa kuin täysi varmistus.
Vaiheittainen varmistus	Vaiheittaisessa varmistuksessa tallennetaan ainoastaan tieto, joka on muuttunut edellisen täyden, valikoivan tai vaiheittaisen varmistuksen jälkeen. Tämä vie tavallisesti vähemmän tilaa ja aikaa kuin valikoiva varmistus tai täysi varmistus.

Taulukko 2 Varmuuskopiointitavat varmistettavan tiedon mukaan (Bell 2010, 422.)

Toinen tapa luokitella varmuuskopiointimenetelmiä on tehdä se IT-järjestelmän toimintakyvyn kannalta. (Balling 2008, 478.)

Kuuma varmistus	Kuumassa varmistuksessa IT-järjestelmä on koko varmuuskopioinnin aikana käytettävissä. Yleisesti suurin ongelma kuumassa varmistuksessa on varmuuskopioitavan tiedon eheyden säilyttäminen. Tämä vaatii sitä, että kaikki tieto voidaan lukita tiettyyn ajanhetkeen, vaikka varmuuskopiointiprosessi kestäisikin useita tunteja. Kuuma varmistus voi myös aiheuttaa IT-järjestelmän toiminnalle häiriötä kuluttamalla palvelimen tai laitteiston resursseja.
Kylmä varmistus	Kylmässä varmistuksessa IT-järjestelmä on varmuuskopioinnin aikana poissa käytössä. Tämä on usein luotettavin ja yksinkertaisin tapa toteuttaa varmistus. Lisäksi järjestelmän kaikki resurssit ovat varmuuskopioinnin käytössä, jolloin varmuuskopiointi voi tapahtua nopeammin kuin kuumassa varmistuksessa. Haittapuolena etenkin isojen järjestelmien kohdalla varmuuskopioinnin pitkä kesto, joka voi liiketoiminnallista syistä johtuen estää kylmän varmistuksen käytön.

Taulukko 3 Varmuuskopiointitavat IT-järjestelmän toimintakyvyn mukaan (Balling 2008, 478).

2.4.2 Peilaus ja replikointi

Peilauksella tarkoitetaan tiedon jatkuvaa kopioimista kahteen eri fyysiseen tallennuspaikkaan. Esimerkki peilauksesta on levyjärjestelmätaso RAID1, jossa kaikki tieto kirjoitetaan kahdelle eri kovalevylle (Vadala 2009, 21). Tietokannoissa vastaava toiminto on replikointi, jossa tietokantaan tehdyt muutokset kirjoitetaan myös toiselle tietokantapalvelimelle (Balling 2008, 343 – 344).

Peilausta ja replikointia pidetään usein varmuuskopiointiratkaisuna, mutta yksinään ne eivät ole riittäviä varmuuskopiointiratkaisuksi. Peilauksen luonteesta johtuen kaikki alkuperäiseen tietoon kohdistuvat muutokset vaikuttavat saman tien peilattuun tietoon. Tämän takia peilatusta tiedosta ei voida esimerkiksi palauttaa vahingossa tuhottua tiedostoa. Niitä voidaan kuitenkin käyttää apuna varmuuskopioinnissa, esimerkiksi tekemällä varmistus peilatusta tiedosta alkuperäisen sijaan. (Balling 2008, 485 – 486.)

Peilaus nostaa kuitenkin järjestelmän saavutettavuutta ja voi monesti ehkäistä tarpeen palauttaa tietoa varmuuskopioista. Lisäksi esimerkiksi tietokannan replikointi varapalvelimelle tarjoaa luontevan ympäristön, johon tietokanta voidaan onnettomuuden tapahtuessa siirtää. (Balling 2008, 485.)

2.4.3 Varmuuskopioiden säilytys

Varmuuskopioinnin suorittamisen lisäksi on tärkeää kiinnittää huomio niiden säilytykseen. Jos varmuuskopiot tuhoutuvat, hukkuvat tai häviävät on koko varmuuskopiointi ollut turhaa. Varmuuskopioiden säilyttämisessä pitää ottaa myös huomioon tila, jossa niitä säilytetään. Jos varmuuskopioita säilytetään samassa tilassa kuin IT-järjestelmiä, voivat varmuuskopiot tuhoutua esimerkiksi tulipalossa. (Preston 2007, 711 – 712.)

Varmuuskopioita tulisikin säilyttää useassa eri tilassa. Laitesalissa voidaan säilyttää yhtä versiota uusimmasta varmuuskopioista, jotta järjestelmä voidaan palauttaa mahdollisimman nopeasti. Lisäksi varmuuskopioista tulee säilyttää kopio myös esimerkiksi yrityksen toimipaikassa, jos se ei sijaitse laitesalin yhteydessä. Kaikissa tapauksessa varmuuskopioita tulisi säilyttää paloturvallisessa ja lukitussa tilassa. (Preston 2007, 711 – 712.)

2.5 Arkistointi

Tiedon suojauksen näkökulmasta arkistointi ja varmuuskopiointi ovat toisiinsa liittyviä, mutta hyvin erilaisia toimintoja. Varmuuskopioinnissa tieto kopioidaan, jotta se voidaan palauttaa onnettomuuden tapahtuessa. Arkistointi puolestaan tarkoittaa tiedon kopiointia tai siirtoa pitkäaikaiseen tallennukseen, josta tietoa voidaan hakea liiketoiminnan tarpeita varten. Varmuuskopioita puolestaan säilytetään yleensä vain niin pitkään, kuin niitä tarvitaan järjestelmän palautusta varten. (Preston 2007, 696 – 697.)

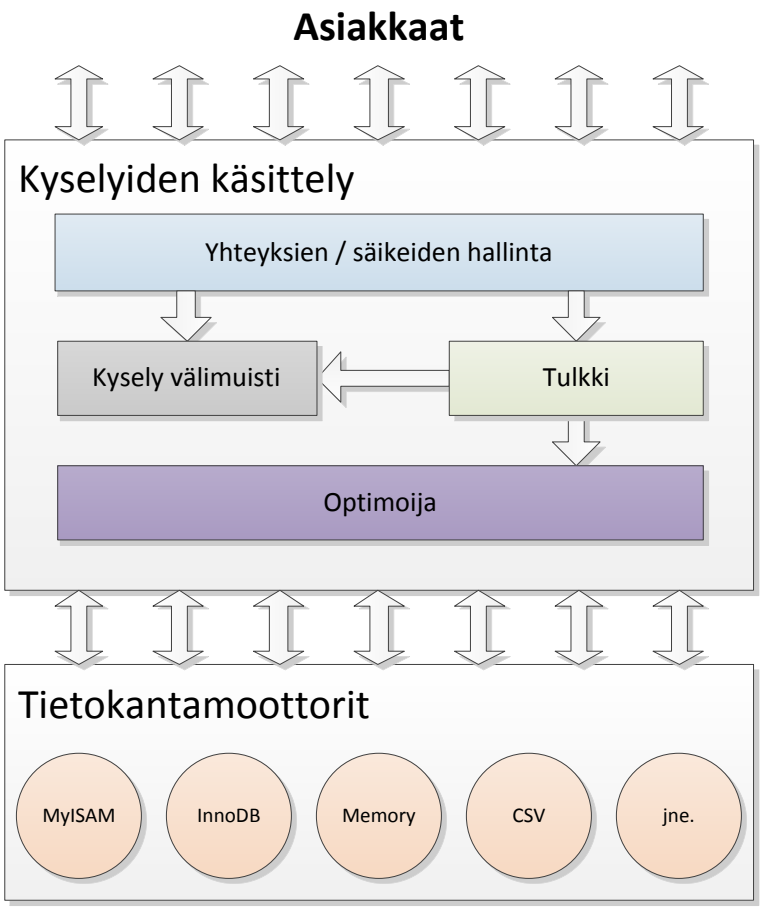
Yritykset ja organisaatiot tarvitsevat usein arkistoja erilaisia käyttötarkoituksia varten. Arkistointia voidaan tarvita esimerkiksi kirjanpidon tarkistusta tai muita lakeihin ja asetuksiin perustuvia säädöksiä varten. Järjestelmän palautuksen näkökulmasta arkistointi tarjoaa toissijaisen lähteen tietojen palauttamiselle. Arkistoitu tieto ei kuitenkaan yleensä ole suoraan palautettavissa, vaan se vaatii manuaalista tiedon muokkausta. (Preston 2007, 696 – 697.)

2.6 MySQL tietokannan varmuuskopiointi

MySQL tietokannan varmuuskopioinnin suunnittelu ja toteutus pohjautuvat aikaisemmin käsiteltyihin tietojen suojauksen ja varmuuskopioinnin perusperiaatteisiin. Tietokannan varmuuskopiointi voidaan kuitenkin suorittaa monella eri tavalla ja seuraavissa aliluvuissa esitellään MySQL tietokannan varmuuskopiointiin liittyviä erikoispiirteitä ja sovelluksia varmuuskopiointia ja palautusta varten.

2.6.1 MySQL tietokannan arkkitehtuuri

MySQL tietokannan poikkeavin ja tärkein ominaisuus on sen arkkitehtuuri, joka erottaa toisistaan kyselyiden käsittelyn tiedon varastoinnista ja hakemisesta. Tämä mahdollistaa taulukohtaisen määrittelyn sille, miten tieto tallennetaan ja mitä ominaisuuksia tiedon käsittelyssä vaaditaan. (Bell 2010, 1 – 2.)



Kuva 1 MySQL tietokannan arkkitehtuuri (Balling 2008, 2).

Varmuuskopioinnin kannalta jokainen tietokantamoottori asettaa rajoitteita ja mahdollisuuksia varmuuskopioinnin toteuttamiselle. Jos käytössä on yhtäaikaisesti useampia tietokantamoottoreita, joudutaan varmuuskopiointi toteuttamaan tavalla, jota kaikki tietokantamoottorit tukevat.

Esimerkiksi MyISAM taulujen varmuuskopiointi vaatii niiden lukitsemista varmuuskopioinnin ajaksi. InnoDB taulujen sisällön sisältäviä tiedostoja ei puolestaan voida kopioida kuin silloin, kun koko tietokanta on suljettu. (Balling 2008, 483 – 485.)

2.6.2 Kylmä ja kuuma varmistus

Helpoin ja turvallisin tapa ottaa varmuuskopio MySQL tietokannasta on sammuttaa tietokanta varmuuskopioinnin ajaksi. Tällöin ei tarvitse huolehtia välimuistissa olevasta tiedosta tai tietojen muuttumisesta varmuuskopioinnin aikana. Lisäksi varmuuskopiointi on nopeampi suorittaa, koska kaikki palvelimen resurssit voidaan käyttää varmuuskopiointiin. (Balling 2008, 478.)

Kylmän varmistuksen tekeminen vaatii kuitenkin tietokannan sammuttamisen ja käynnistämisen, joka voi kestää pitkään – erityisesti jos tietokantapalvelimen käyttöaste on suuri ja käsiteltävää tietoa on paljon. Syitä tähän on esimerkiksi se, että tietokannan sammuttamisen yhteydessä kaikki välimuistissa olevat tiedot pitää kirjoittaa levylle. Tietokanta ei myöskään toimi täydellä kapasiteetilla käynnistyksen jälkeen ennen kuin välimuistiin on tallennettu toiminnan kannalta tärkein tieto. Näistä syistä korkeaa suorituskykyä varten varmuuskopiointi pitää usein suunnitella niin, että tietokantaa ei sammuteta varmuuskopioinnin ajaksi. (Balling 2008, 478.)

Paras tapa toteuttaa kuuma varmistus MySQL tietokannasta riippuu käyttöympäristöstä, tietokannan koosta ja käytetyistä tietokantamoottoreista. Eri mahdollisuuksia ja sovelluksia kuuman varmistuksen toteuttamiseen on esitelty myöhemmin kappaleissa 2.6.5 ja 2.6.6.

2.6.3 Looginen ja tiedostopohjainen varmistus

MySQL tietokannan varmuuskopiointiin on kaksi toisistaan poikkeavaa tapaa. Loogisessa varmistuksessa tietokannan sisältämä tieto tallennetaan tietokantakyselyinä. Tiedostopohjaisessa varmistuksessa puolestaan kopioidaan tiedostot, joihin tieto on tallennettu. (Balling 2008, 479.)

Loogisen varmuuskopioinnin etuna on se, että varmuuskopioitava tieto on tekstimuotoista ja sitä voidaan käsitellä erilaisten komentorivityökalujen avulla. Loogiset varmuuskopiot ovat myös tietokantamoottoreista riippumattomia ja ne eivät ole riippuvaisia tietokantamoottorin sisäisestä tietorakenteesta. Niiden palautus on myös helppoa ja niitä voidaan käyttää jopa tiedon siirtämiseen eri tietokantojen välillä. (Balling 2008, 479 – 480.)

2.6.4 Binaarilokitiedostot

MySQL tietokanta pystyy kirjoittamaan kaikki tietokantaan kohdistuneet kyselyt binaarilokitiedostoihin. Kyselyiden lisäksi tiedostot sisältävät jokaisen kyselyn ajanhetken, joten niiden avulla voidaan suorittaa tietokannan palautus haluttuun ajankohtaan. Vaatimuksena on, että tietokannasta on otettu täysi varmistus ja kaikki sen jälkeiset binaarilokit haluttuun ajankohtaan ovat saatavilla. (Balling 2008, 486.)

2.6.5 Levyjärjestelmän tilannevedokset

Useat nykyaikaiset levyjärjestelmät tai levynhallintajärjestelmät mahdollistavat tiedostojen kopioimisen tilannevedoksena. Pelkkä tilannevedoksen ottaminen ei kuitenkaan takaa sitä, että tietokannan levylle tallentama tieto voidaan palauttaa. Tilannevedosta voidaan kuitenkin käyttää lyhentämään tietokannan lukitukseen kuluvaa aikaa. (Balling 2008, 492.)

Esimerkiksi Linux käyttöjärjestelmän Logical Volume Manager (LVM) mahdollistaa tilannevedoksen ottamisen levyjärjestelmästä kopiointi kirjoitettaessa menetelmällä. Tässä menetelmässä LVM ei varsinaisesti kopioi alkuperäistä tietoa mihinkään, vaan merkitsee ylös levyjärjestelmän tilan. Kun alkuperäistä tiedostoa ollaan muuttamassa, ennen alkuperäisen tiedon ylikirjoitusta kopioidaan tieto talteen ja vasta tämän jälkeen muutetaan tietoa. (Balling 2008, 493.)

Yhdistettynä tietokantamoottorikohtaiseen tapaan varmistaa levylle tallennetun tiedon eheys, voidaan tilannevedoksen avulla suorittaa varmuuskopiointi nopeasti ja luotettavasti. Kaikissa ympäristöissä tilannevedoksen ottaminen ei kuitenkaan ole mahdollista. (Balling 2008, 493.)

2.6.6 MySQL varmuuskopiointisovellukset

MySQL tietokannan varmuuskopiointia varten on olemassa useita sovelluksia. Osa sovelluksista on tietokantamoottorista riippumattomia ja osa on puolestaan kehitetty erityisesti tietyn tietokantamoottorin varmuuskopiointia varten. Tietokannan sisällöstä, varmuuskopioitavasta tiedosta ja varmuuskopioinnille asetetuista vaatimuksista riippuen voi yksittäinen sovellus olla riittävä varmuuskopiointia varten. (Balling 2008, 511.)

mysqldump	Suosituin sovellus MySQL tietokannan varmuuskopiointia varten on mysqldump, jonka avulla voidaan suorittaa looginen varmuuskopiointi tietokannasta. Ohjelma on tietokantamoottorista riippumaton, mutta vaatii taulujen lukitsemisen varmuuskopioinnin ajaksi. (Balling 2008, 511.)
mysqlhotcopy	MyISAM tietokantojen varmuuskopiointia varten MySQL palvelimen mukana tulee mysqlhotcopy sovellus. Sen käyttö on suhteellisen monimutkaista, eikä se nimensä vastaisesti kykene kuumaan varmistukseen, vaan vaatii taulujen lukitsemisen varmuuskopioinnin ajaksi. (Balling 2008, 513.)
InnoDB Hot Backup	InnoDB Hot Backup on alun perin InnoDB tietokantamoottorin kehittämän Innobase yrityksen kaupallinen varmuuskopiointityökalu erityisesti InnoDB tietokantamoottorin kuumaan varmistukseen. Sovelluksella voidaan varmistaa myös MyISAM taulujen tiedot, mutta se vaatii taulujen lukitsemisen varmuuskopioinnin ajaksi. (Balling 2008, 513 – 514.)
mk-parallel-dump	Maatkit on kokoelma erilaisia sovelluksia erityisesti MySQL tietokannan ylläpitoa varten. Yksi sovelluksista on mk-parallel-dump, jonka avulla voidaan suorittaa rinnakkaisesti useita loogisia varmuuskopiointeja. (Balling 2008, 514.)
mylvmbackup	Mylvmbackup on sovellus, joka pyrkii automatisoimaan varmuuskopioiden tekemisen käyttäen hyväksi Linux Volume Manager levynhallintajärjestelmää. Sen avulla voidaan toteuttaa tilannevedoksen vaatima tietokannan lukitus, tilavedoksen ottaminen ja lukituksen vapautus. (Balling 2008, 515.)
XtraBackup	XtraBackup on sovellus, joka on tarkoitettu Percona XtraDB tietokantamoottorin varmuuskopiointia varten. XtraDB on InnoDB johdannainen, joten sovellus kykenee tekemään kuuman varmuuskopion myös InnoDB tauluista. Lisäksi sen avulla voidaan ottaa kopioita myös MyISAM tauluista. (Bell 2010, 432.)

3 Esiselvitys

Tämä opinnäytetyö toteutettiin Mr.Goodliving Oy:lle, joka on RealNetwork konsernin tytäryhtiö. Konsernin näkökulmasta Mr.Goodliving Oy on osa RealArcade D2C organisaatiota, jonka toimenkuvana on pelien julkaisu ja kehittäminen.

Suomessa toiminta on keskittynyt mobiilipelien tekemiseen, markkinointiin ja jakeluun. Opinnäytetyö toteutettiin teknologiaosastolle, joka kehittää ja ylläpitää liiketoimintaa tukevia palveluita, ohjelmistokomponentteja ja sovelluksia. Osaston vastuulla ei ole kuitenkaan perinteisten IT-palveluiden tuottaminen, kuten esimerkiksi toimistosovelluksista, tietoliikenteestä tai sähköpostista vastaaminen.

Liiketoiminnan tueksi on kehitetty useita erilaisia sovelluksia ja palveluita, joilla pyritään vastaamaan mobiilipelien kehityksen ja julkaisun asettamiin erityisvaatimuksiin. Yhteensä sovelluksia ja palveluita on laskentavastasta riippuen noin 40 – 50 kappaletta. Tärkein näistä on Cobra-verkkopalvelu, jonka avulla pelituotantoa hallinnoidaan toteutuksesta markkinointiin ja jakeluun asti. Sovelluksen tietovarastona on MySQL tietokanta, jota hyödynnetään myös muiden sovellusten ja verkkopalveluiden kautta.

Opinnäytetyöprojektin ensimmäisessä vaiheessa tehtiin esiselvitys, jonka tavoitteena oli täydentää yrityksen palvelinympäristön ja verkkopalveluiden dokumentaatiota. Lisäksi esiselvityksessä kartoitettiin opinnäytetyössä toteutettavaksi sopivia kehityshankkeita ja valittiin näistä yksi opinnäytetyössä toteutettavaksi.

3.1 Palvelinympäristö

Mr.Goodliving Oy:n palvelinympäristö pohjautuu Citrix XenServer 5.5 virtuaalipalvelimiin. Fyysisesti palvelimet sijaitsevat palveluntarjoajan laitesalissa. Palvelinten laitteisto koostuu viidestä Hewlett Packarding blade palvelimesta, joissa kaikissa oli 18GB muistia ja 2 kappaletta Intelin Xeon quad core prosessoreita. Levyjärjestelmänä on Hewlett Packarding EVA levypakka, josta Mr.Goodliving Oy:n käyttöön on varattu 6TB tilaa.

Kaikki Mr.Goodliving Oy:n palvelimet ovat virtuaalipalvelimia, joita on yhteensä viisitoista. Virtuaalipalvelimet on hajautettu fyysisille palvelimille niiden käyttämien resurssien perusteella, jotta rasitus kohdistuisi tasaisesti. Palvelimia on myös hajautettu siten, että kahdennetut palvelut sijaitsevat eri fyysisillä palvelimilla. Alla olevassa kuvassa on esitetty eri virtuaalipalvelinten

sijainti, niille annetut resurssit ja palvelinten käyttötarkoitus tai niillä toimivat palvelinsovellukset.

Virtuaalipalvelin	Blade-palvelin	Käyttöjärjestelmä	Muistia (GB)	Levytilaa (GB)	Palvelut / sovellukset
api	1	Debian	2	4+0	Apache2, PHP, Memcached
dns1	1	Debian	1	4+0	Bind9 / DNS palvelin
enduser	1	Debian	2	4+15	Apache2, PHP, Memcached
svn	2	Debian	4	4+30	Apache2, Subversion
dns2	2	Debian	1	4+0	Bind9 / DNS palvelin
cobra	2	Debian	4	4+286	Apache2, PHP, Tomcat, Memcached
gamedb	3	Debian	16	4+2048	MySQL
slave	3	Debian	4	4+1024	MySQL
shell	3	CentOS	2	4+0	Apache2, PHP, MySQL
mail	4	CentOS	2	4+0	MySQL, Courier-IMAP, Spamassassin, Postfix, Amavisd, Apache2
enduserdb	4	Debian	8	4+15	MySQL
game	4	Debian	2	4+15	Apache2, PHP, Memcached
mobile-tools	5	Debian	4	4+15	Apache2, PHP
tools	5	Debian	2	4+15	Apache2, PHP, Python, Java, MySQL, Trac, Instiki
dev	5	CentOS	4	4+99	Kehitys- ja testipalvelin

Kuva 2 Palvelinten resurssijako ja käyttötarkoitus

Osa palvelinten resursseista on jätetty käyttämättä, jotta virtuaalipalvelimia voidaan siirtää ongelmatilanteista eri palvelinten välillä. Lisäksi resursseja voidaan tarvittaessa antaa lisää, jos tarvetta esiintyy.

Esiselvityksessä toteutetussa palvelinympäristön dokumentaatiossa yllä olevan tiedon lisäksi esiteltiin myös tarkemmin eri palvelimilla toimivat sovellukset. Tavoitteena dokumentaatiolla oli antaa yleiskuvaus palvelinympäristöstä. Opinnäytetyön kannalta palvelinympäristön dokumentaatiolla pyrittiin myös tunnistamaan kehityskohteita, jotka voisivat soveltua opinnäytetyössä toteutettavaksi.

3.2 Kehityshankkeiden kartoitus

Palvelinympäristön dokumentoinnin lisäksi esiselvitysvaiheessa kartoitettiin ja arvioitiin erilaisten projektien soveltuvuutta toteutettavaksi opinnäytetyössä. Soveltuvia projekteja löytyi yhteensä seitsemän kappaletta, joista jokainen arvioitiin seuraavilla kriteereillä:

- projektin arvo liiketoiminnan kannalta
- projektin vaatima työmäärä
- projektin toteuttamiseen vaaditut resurssit
- riippuvaisuus muista hankkeista
- projektin toteutusaikataulun kiireellisyys

Projekteja arvioitiin myös niiden soveltuvuudelta opinnäytetyössä toteutettavaksi seuraavilla kriteereillä:

- Täyttääkö projekti opinnäytetyölle esitetyt vaatimukset yleisestä hyödynnettävyydestä?
- Sisältääkö projekti salattavaa aineistoa ja aiheutuuko tästä ongelmia opinnäytetyön julkisuusvaatimusten osalta?

Kehityshankkeiden kartoitus löytyy kokonaisuudestaan liitteestä 1.

3.3 Esiselvityksen lopputulos

Esiselvityksen perusteella ohjauskokouksessa päätettiin projektissa toteuttaa sovellus MySQL tietokannan varmuuskopiointi- ja palautusprosessin automatisointia varten. Valinnassa painottuivat erityisesti mahdollisuus toteuttaa sovellus ilman riippuvaisuuksia muista hankkeista tai liiketoiminnan kannalta asettuja aikatauluja. Lisäksi lopputulos ei sisällä salaista aineistoa ja on hyödynnettävissä myös yrityksen ulkopuolisten tahojen osalta.

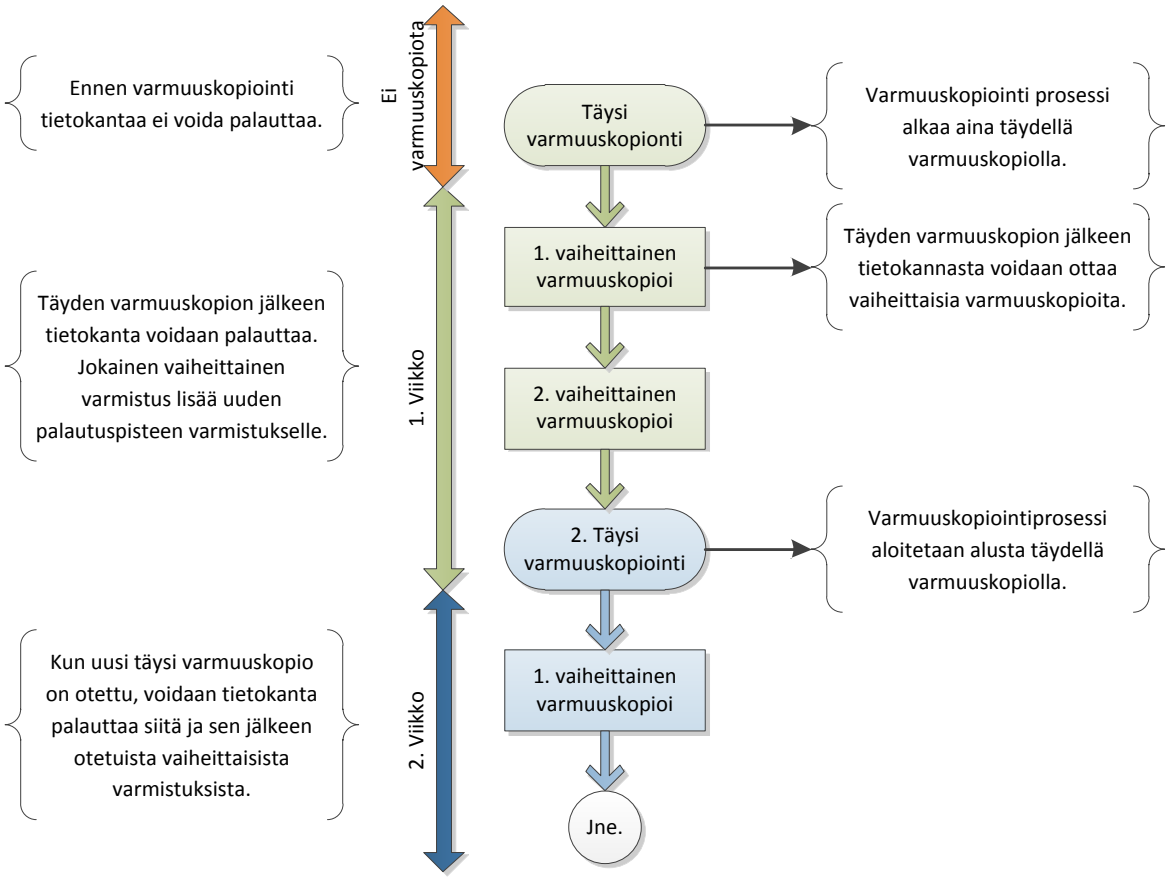
4 Sovelluksen esittely

Opinnäytetyössä toteutettiin sovellus MySQL tietokannan varmuuskopiointi- ja palautusprosessin automatisoinniksi. Tavoitteena oli varmuuskopiointiprosessin laadun ja luotettavuuden parantaminen. Sovelluksen tarkoituksena ei ollut varmuuskopioinnin tai muiden prosessiin liittyvien tehtävien suorittaminen. Sen sijaan sovellus vastaa muiden sovellusten käynnistämisestä, niiden tulosten valvonnasta ja tiedon välityksestä prosessin vaiheiden välillä. Erityisesti ajallisesti eri aikoina tapahtuvien tehtävien välinen tiedonsiirto on oleellista prosessin luotettavuuden varmistamiseksi.

Tärkeimpänä keinoa laadun parantamiseksi oli varmuuskopioinnin toteuttavan sovelluksen vaihtaminen mysqldump sovelluksesta Percona XtraBackup sovellukseen. Lisäksi toteutettu sovellus mahdollisti paremman raportoinnin ja varmuuskopioinnin valvonnan. Koska sovelluksen tavoitteena oli varmuuskopiointi- ja palautusprosessin automatisointi, on sovelluksen toiminta helpointa kuvata prosessikuvauksella.

4.1 Varmuuskopiointiprosessi

Varmuuskopiointiprosessi koostuu useasta tehtävästä, jotka suoritetaan ajallisesti toisistaan erillään. Prosessi alkaa aina täydellä varmistuksella, jonka jälkeen voidaan suorittaa vaiheittaisia varmistuksia. Kuvassa 2 on esitetty kaavio varmuuskopiointiprosessista.



Kuva 3 Varmuuskopiointiprosessi

Varmuuskopiointiprosessi on siis toistuva prosessi, joka alkaa aina täydellä varmistuksella ja jatkuu mahdollisesti vaiheittaisilla varmistuksilla. Yleinen tapa prosessin ajoittamiseksi on täyden varmistuksen ottaminen viikonloppuna ja vaiheittainen varmistus muina viikonpäivinä. Mikään ei kuitenkaan estä varmuuskopiointiprosessin suorittamista esimerkiksi joka päivä ottamalla öisin täysi varmistus ja vaiheittainen varmistus esimerkiksi tunnin välein.

4.1.1 Varmuuskopiointi käytännössä

Jokainen varmuuskopiointitehtävä koostuu yhdestä tai useammasta alitehtävästä, jotka vastaavat käytännössä yhtä komentorivillä suoritettavaa käskyä. Esimerkiksi täyden varmistuksen suorittaminen voi koostua varmuuskopion ottamisesta ja tiivisteen laskemisesta halutuista tietokannan tauluista varmuuskopion eheyden tarkistusta varten. Varmuuskopiointi itsessään tapahtuu suorittamalla mybackup.py sovellus ja antamalla halutun tehtävän nimi parametrina alla olevan esimerkin mukaisesti.

```
mybackup.py -t fullbackup
```

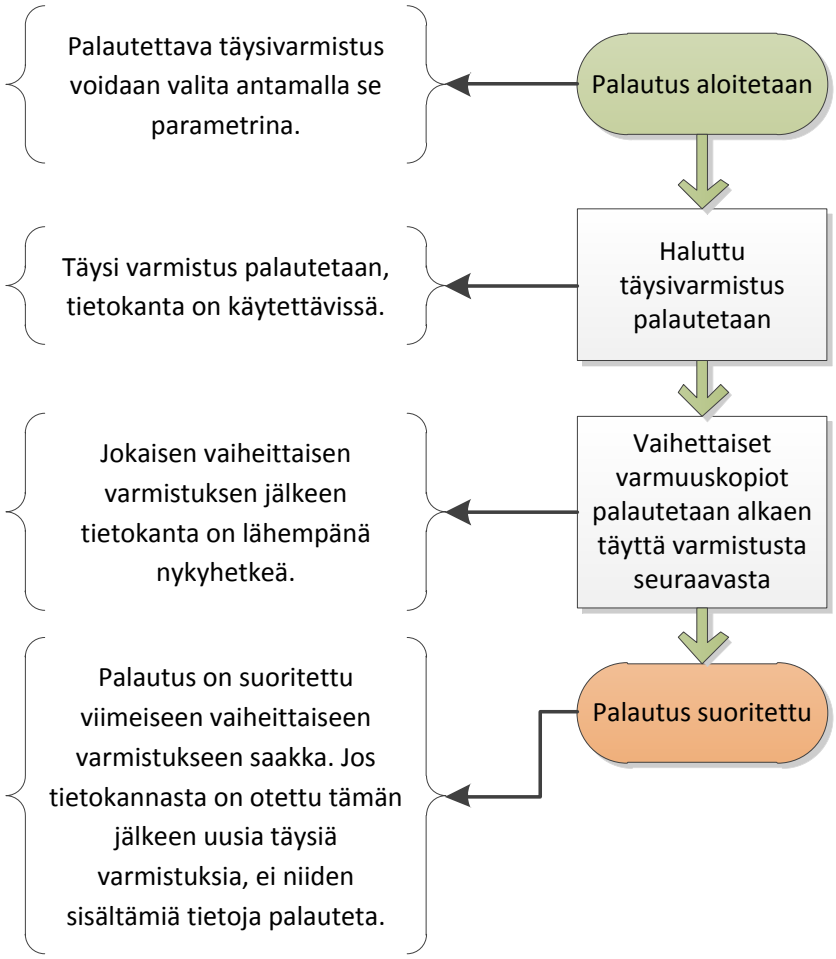
Esimerkkiä vastaava määrittelytiedosto puolestaan koostuu asetuksista koko tehtävälle ja kaikille suoritettaville alitehtäville. Alitehtävät määritellään pilkulla erotettuina päätehtävän asetuksissa. Esimerkin mukainen määritystiedosto on seuraavanlainen:

```
[fullbackup]
type:                   full
tasks:                  local_backup,checksum
basedir:                /mybackuptest/backups/
loglevel:               debug


[local_backup]
type:                   local
defaults-file:          /etc/my.cnf

[checksum]
type:                   checksum
filename:               checksum.txt
databases:              mysql,mybackuptest
ignore-databases:       test
tables:
ignore-tables:          mysql.user
dbconfig:               db

[db]
host:                   localhost
port:                   3306
user:                   root
password:
```

Asetustiedostojen ja sovellusten parametrien tarkempi ohjeistus löytyy liitteestä 4, käyttöohje.

4.2 Palautusprosessi

Palautusprosessin kulku alkaa aina täyden varmistuksen palauttamisella. Tämän jälkeen palautetaan kaikki sen jälkeiset vaiheittaiset varmistukset alkaen vanhimmasta ja päätyen uusimpaan. Esivaatimuksena palautukselle on se, että MySQL palvelinohjelmisto on asennettu ja sen asetukset ovat tietokantaa vastaavat. Palautusprosessissa varmuuskopioitu tietokanta kopioidaan kohdehakemistoon, jonka pitää olla ennen palautusta tyhjä. Palautuksen jälkeen MySQL tietokantaohjelmisto voidaan käynnistää uudestaan. Kuvassa 3 on esitetty palautusprosessin vaiheet.



Kuva 4 Palautusprosessi

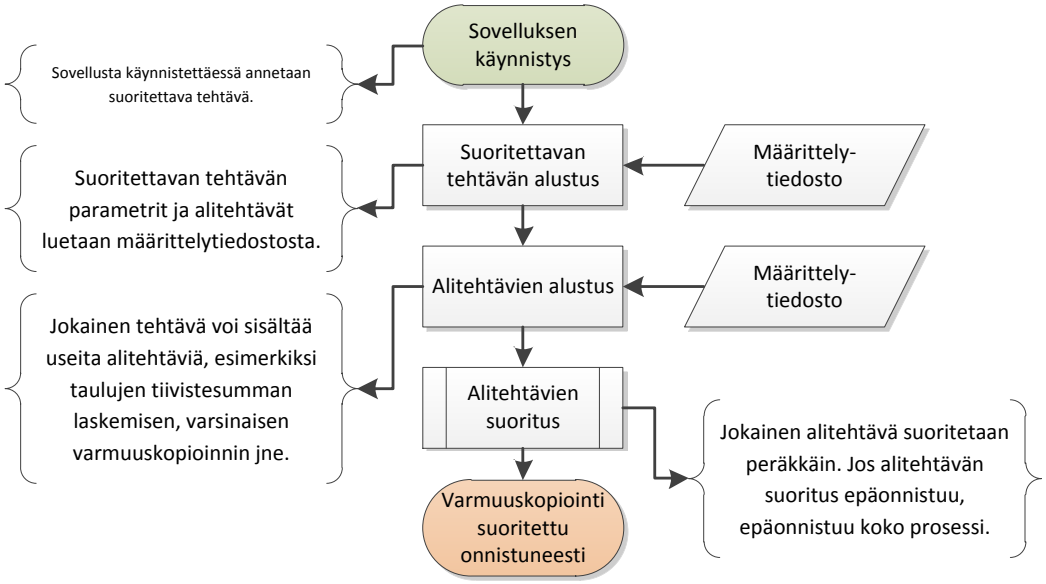
Varmuuskopioinnista poiketen ei palautusprosessia tarvitse määritellä etukäteen. Kaikki palautusprosessissa tarvittava tieto kirjoitetaan varmuuskopiointiprosessin aikana tiedostoihin, joista ne luetaan tietokantaa palautettaessa. Tarvittavat parametrit sovellukselle annetaan komentoriviltä alla olevan esimerkin mukaisesti, jossa palautetaan backup hakemistosta varmuuskopio data hakemistoon. Koska eri käyttöjärjestelmät suorittavat MySQL palvelinohjelmiston eri käyttäjä ja ryhmä tunnuksilla, pitää myös nämä parametrit antaa sovellukselle.

```
myrestory.py –d /mysql/data –s /backup/2010-09-19_16-54-06/ -g mysql –u mysql
```

4.2.1 Sovelluksen toiminta

Jokainen sovelluksen suorituskerralla suoritetaan yksi tehtävä. Tehtävä voi olla esimerkiksi täyden varmistuksen ottaminen tai tietokannan palautus halutusta varmuuskopiosta. Tehtävä puolestaan koostuu erilaisista alitehtävistä, kuten tiivistesumman laskemisesta halutuista tietokantatauluista tai varsinaisesta varmuuskopion suorittamisesta. Yksittäisen suorituskerran logiikka

on kuvattu seuraavassa kuvassa:



Kuva 5 Sovelluksen toimintaperiaate

Toiminnan kannalta sovellus voidaan jakaa kahteen osaan. Varsinainen ohjelmakoodi huolehtii eri tehtävien suorittamisesta ja raportoinnista. Eri tehtävillä on puolestaan muuttujia, joiden tietoja voidaan muokata määritystiedoston avulla. Esimerkiksi varmuuskopion tallennusmetodi ja sijainti voidaan määritellä erikseen joka tehtävällä. Tämä mahdollistaa varmuuskopioiden ottamisen eri ympäristöissä. Tarkempi kuvaus näistä määritystiedostoista ja muuttujista löytyy liitteestä 4, käyttöohje.

5 Toteutus

Sovellus toteutettiin kolmessa eri vaiheessa. Ensimmäisenä määriteltiin toteutettavan sovelluksen reunaehdot ja toiminnalliset tavoitteet. Tämän jälkeen mallinnettiin varmuuskopiointi- ja palautusprosessi manuaalisesti. Viimeisenä vaiheena toteutettiin ja testattiin varsinainen sovellus tietokannan varmuuskopioinnin- ja palautusprossin automatisoinniksi.

5.1 Määrittely

Sovelluksen suunnittelu aloitettiin vaatimusmäärittelyllä, jossa dokumentoitiin sovelluksen käyttö- ja kehitysympäristö, toteutustapa, testausmenetelmät ja ominaisuudet. Projektinhallinnan kannalta vaatimusmäärittelyn perusteella piti pystyä myös arvioimaan toteutukseen vaadittavat resurssit, jotka hyväksyttiin ohjauskokouksessa määrittelyvaiheen jälkeen.

5.1.1 Lähtötilanne

Lähtökohtana suunnittelulle oli olemassa oleva varmuuskopiointiprosessi, joka pohjautui mysqldump-sovelluksella tehtyyn loogiseen varmistukseen. Prosessina varmuuskopiointi oli hyvin yksinkertainen ja se perustui mysqldump-sovelluksen ajastettuun suorittamiseen yksinkertaisen BASH skriptan avulla. Palautus puolestaan tapahtui alustamalla uusi tietokanta manuaalisesti varmuuskopioidulla tiedolla.

Suurimpana motivaationa uuden sovelluksen kehittämiselle olivat mysqldump sovelluksen rajoitukset, joista tärkein oli tuen puuttuminen vaiheittaiselle varmistukselle. Yhdistettynä kohdetietokannan yli teratavun kokoon ja varmistuksen pitkään kestoon, ei varmistuksia voitu suorittaa kuin viikonloppuisin. Muina päivinä usean tunnin käyttökatkos ei ollut mahdollista tietokannan ympärivuorokautisen käyttöasteen takia.

Aikaisempi varmuuskopiointiprosessi ei myöskään sisältänyt automaattista raportointia, joten varmistusten seuranta perustui kehitystietokannan ajoittaiseen päivittämiseen varmuuskopiosta. Ottaen huomioon tietokannan tärkeyden liiketoiminnan kannalta, tarpeena oli myös parantaa luotettavuutta raportoimalla varmistuksen onnistuminen.

5.1.2 Palvelinympäristö

Mr.Goodliving Oy:n palvelinympäristö koostuu virtuaalisista Linux palvelimista, jotka pohjautuvat joko Debian tai CentOS Linux jakeluihin. Vaatimuksena oli, että sovelluksen on toimitta-

va kummankin Linux jakelun kanssa. Tukea muille UNIX-pohjaisille käyttöjärjestelmille ei asetettu vaatimukseksi, mutta asia toivottiin otettavan huomioon toteutuksessa.

5.1.3 MySQL versiot

MySQL tietokannasta tuotannossa oleva versio oli 5.0.51a. Tulevaisuutta varten sovelluksen pitää tukea myös MySQL versiota 5.1.

5.1.4 Ohjelmointikielet

Sovelluksen päätettiin toteuttaa käyttäen Bash ja Python skriptakieliä tarpeen mukaan. Syynä valintaan oli sovelluksen toimintaperiaate, joka perustui muiden ohjelmien käynnistämiseen varsinaista varmuuskopiointia ja muita alitehtäviä varten. Kumpikin kieli mahdollistaa tämäntapaisten tehtävien toteuttamisen helposti. Skriptakielinä ne eivät myöskään vaadi sovelluksen kääntämistä, mikä helpottaa käyttöönottoa ja siirrettävyyttä käyttöjärjestelmien välillä.

5.1.5 Käytettävät sovellukset

Varsinaisen varmuuskopioinnin toteuttaminen päätettiin tehdä Percona XtraBackup sovelluksen avulla. Perusteluina sovelluksen käyttämiselle olivat sen tuki sekä MyISAM että InnoDB taulujen varmuuskopioinnille ja mahdollisuus ottaa InnoDB tauluista vaiheittaisia varmuuskopioita. Sovellusta oli myös kokeiltu aikaisemmin varmuuskopioinnissa ja kokemukset sen toimivuudesta olivat olleet hyviä. Vaihtoehtona olisi myös ollut käyttää InnoDB Hot Backup sovellusta, jonka ominaisuudet ovat hyvin samankaltaiset XtraBackup sovelluksen kanssa. Ongelmana oli kuitenkin palvelinkohtainen 390€ vuotuinen lisenssimaksu, mikä olisi aiheuttanut rajoituksia sovelluksen kehittämisessä sekä testauksessa.

MySQL tietokannan analysointia ja ylläpitoa varten on kehitetty Maatkit niminen kokoelma sovelluksia. Näiden sovellusten avulla on mahdollista esimerkiksi laskea tarkistussumma tietokannan tauluista ja tarkistaa tietokannan tila palautuksen jälkeen. Koska sovelluskokoelma oli saatavilla sekä CentOS ja Debian yhteensopivina paketteina, hyväksyttiin sovelluksen käyttö osana sovelluksen vaatimuksia.

Muiden sovellusten osalta päädyttiin käyttämään mahdollisuuksien mukaan Linux jakeluiden mukana yleisesti saatavia sovelluksia. Näihin kuuluivat esimerkiksi tar, ssh, gzip ja cat. Tavoitteena oli pitää riippuvaisuudet muista erikoisemmista sovelluksista mahdollisimman pieninä ja varmistua että tarvittavat sovellukset ovat saatavilla paketinhallinnan kautta.

5.1.6 Vaatimusmäärittelyn lopputulos

Vaatimusmäärittely sisälsi projektin toteutuksen yleisten reunaehtojen lisäksi myös toiminnallisia vaatimuksia lopputuloksena syntyvälle sovellukselle. Vaatimusten tavoitteena oli luoda sekä kokonaiskuva toteutettavasta sovelluksesta ja tarkistuslista, johon lopullista toteutusta voitiin verrata. Sovelluksen toiminnallisiksi vaatimuksiksi määriteltiin seuraavat asiat:

Täysi varmistus	Sovelluksen pitää pystyä tekemään koko tietokannasta tilavedos tietyllä ajanhetkellä.
Kuuma varmistus	Varmuuskopiointi on pystyttävä tekemään ilman tietokannan sulkemista varmuuskopioinnin ajaksi.
Vaiheittainen varmistus	Tietokannasta on pystyttävä ottamaan täyden varmistuksen jälkeen vaiheittaisia varmuuskopioita.
Replikointi	Sovelluksen ottamista varmuuskopioista on pystyttävä käynnistämään uusi palvelin replikointia varten.
Varmuuskopion eheys	Tietokannan eheydestä palautuksen jälkeen on pystyttävä varmistautumaan. Tämä voidaan todentaa esimerkiksi laskemalla tietokannan sisällöstä tarkistussumma ennen varmuuskopion alkua ja vertaamalla sitä palautuksen jälkeiseen tilaan.
Varmuuskopion sijainti	Varmuuskopio pitää voida tallentaa sekä tietokantapalvelimelle että verkon yli toiselle palvelimelle.
Lokitiedot	Varmuuskopiointi prosessin pitää kirjoittaa edistymisen tilanne ja toimenpiteiden tulokset järjestelmän lokiin.
Raportointi	Sovelluksen pitää kirjoittaa virhetilanteissa raportti järjestelmän lokiin. Lisäksi raportoinnin pitää tarjota mahdollisuus lähettää varmuuskopioinnista raportti esimerkiksi sähköpostilla tai SMS viesteillä.
Ajastus ja synkronointi	Varmuuskopiointi pitää pystyä ajastamaan käyttäen esimerkiksi cronia tai muuta vastaavaa palvelua. Jos varmuuskopiointi prosessi vaatii eri palvelinten välistä synkronointia, pitää sovelluksen tarjota ratkaisu tähän.

Vaatimusmäärittely on luettavissa kokonaisuudessaan liitteessä 2.

5.2 Kehitysympäristö ja tarvittavien sovellusten asennus

Tietokantapohjaisen sovelluksen kehittämisessä tarvitaan usein kehitysympäristö ja erillinen tietokanta, jonka sisältö voidaan helposti alustaa testausta varten uudestaan. Koska vaatimusmäärittely ei vaatinut palvelinympäristöltä erityisiä vaatimuksia, voitiin kehitysympäristönä käyttää virtuaalikonetta. Virtuaalisointiratkaisuna käytettiin VMWare player sovellusta. Koska sovelluksen piti tukea Debian ja CentOS Linux jakeluita, asennettiin http://www.thoughtpolice.co.uk/vmware/ sivustolta saatavissa olevien levykuvien avulla kaksi eri virtuaalikonetta (Thoughtpolice 2010).

Suurimmat ongelmat kehitysympäristön asentamisessa ilmenivät CentOS pakettihallinnan kautta asennettava MySQL palvelinohjelmiston ja Percona XtraBackup paketin epäyhteensopivuudesta. Ratkaisuna oli käyttää http://www.mysql.com/ sivuston kautta ladattavia RPM-paketteja, jotka olivat tarkoitettuja CentOS käyttöjärjestelmälle.

Toinen ongelma kehitysympäristössä ilmeni Python tulkin version kanssa. CentOS paketinhallintajärjestelmää ei ole vielä muokattu toimimaan Python versiota 2.4 uudempien julkaisuiden kanssa. Tämän takia alkuperäinen ajatus toteuttaa Python 3.0 yhteensopiva sovellus ei ollut mahdollista.

5.3 Prototyyppi

Sovelluksen toteutus aloitettiin mallintamalla varmuuskopiointi- ja palautusprosessi manuaalisesti. Käytännössä tämä tarkoitti sitä, että aikaisemmin kuvattu varmuuskopiointi- ja palautusprosessi toteutettiin suorittamalla prosessissa tarvittavat sovelluksen käsin. Manuaalisen prototyypin tarkoituksena oli sekä varmistaa prosessin toteuttamiskelpoisuus sekä kirjata ylös eri sovellusten komentoriviparametrit, joita prosessin toteuttamiseksi tarvittiin. Prototyypin avulla saatiin testattua käytännössä seuraavien käyttötapausten toiminta:

- täyden varmistuksen ottaminen paikalliselle levylle
- varmuuskopion palauttaminen
- varmuuskopion eheyden varmistus tiivisteiden avulla
- vaiheittaisen varmuuskopion ottaminen
- vaiheittaisen ja sitä edeltävän täyden varmistuksen palauttaminen
- varmuuskopion ottaminen suoraan toiselle palvelimelle

Prototyyppiä tehdessä kirjattiin ylös myös huomioita, jotka piti ottaa huomioon toteutuksessa. Esimerkiksi hakemistojen käyttöoikeuksista pitää huolehtia prosessin aikana. Prototyypin avulla dokumentoitiin myös tieto siitä, mitä tietoa eri prosessin vaiheet tarvitsivat aikaisemmin suoritetuista tehtävistä. Esimerkiksi edellisen varmuuskopion tila pitää pystyä välittämään parametrina vaiheittaiselle varmuuskopioinnille. Prototyyppi löytyy kokonaisuudessaan liitteestä 3.

5.4 Toteutus

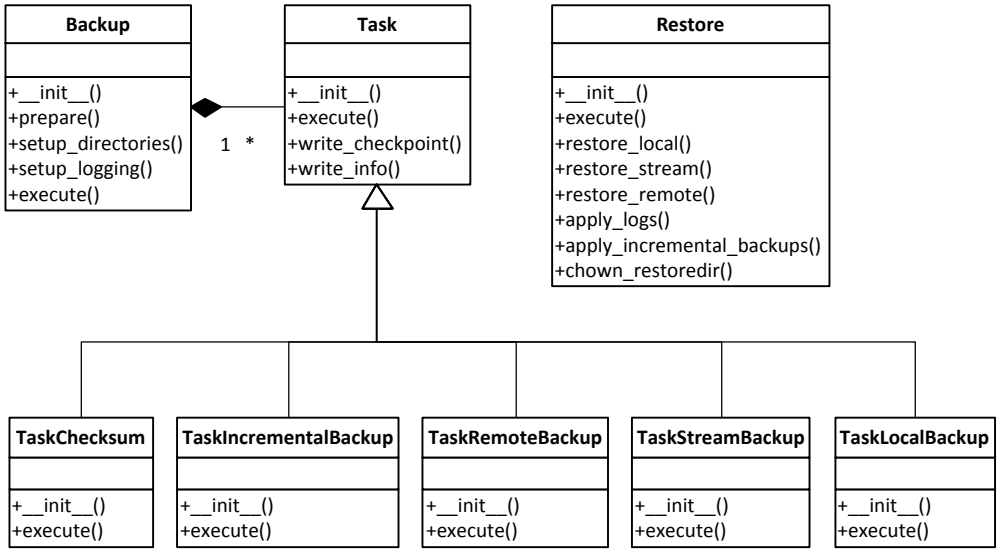
Määrityksen ja prototyypin perusteella oli sovelluksen rakenne ja toimintalogiikka yksinkertaista suunnitella. Prototyypin perusteella oli myös selvinnyt, että varmuuskopiointi ja palautus ovat erillisiä prosesseja, jotka voidaan eriyttää erillisiksi sovelluksiksi.

Varmuuskopiointisovellus koostuu loogisella tasolla varsinaisesta sovelluksesta ja sitä ohjaavasta määrittelytiedostosta. Tavoitteena oli, että määrittelytiedoston avulla voidaan parametrisoida kaikki varmuuskopioinnissa tarvittavat muuttujat, kuten esimerkiksi hakemistopolut, varmuuskopiointi tapa jne. Sovellus itsessään ei siis sisällä vakioita, joita käyttöönotossa joutuisi muuttamaan.

5.4.1 Luokkakaavio

Kuten aikaisemmin sovelluksen esittelyssä todettiin, koostuu varmuuskopiointi useasta alitehtävästä. Sovelluksen toteutuksessa tämä mallinnettiin siten, että varmuuskopiointi koostui yhdestä hallinnoivasta luokasta ja alitehtävät periytettiin kaikki Task-luokasta. Pythonin dynaamisen luonteen vuoksi näin ei olisi ollut pakko tehdä. Sen sijaan olisi voitu vain vaatia, että tehtävän suorittava luokka toteuttaa execute metodin. Selkeyden vuoksi toteutuksessa päätettiin kuitenkin käyttää perintää.

Palautusprosessi on varmistukseen verrattuna suoraviivainen. Sen tehtävänä on palauttaa varmuuskopio lähdehakemistosta kohdehakemistoon. Kaikki palautuksessa tarvittava metatieto kirjoitetaan varmuuskopioinnin aikana ylös, joten vastaavaa parametrisointia kuin varmuuskopioinnissa ei myöskään tarvita. Tästä syystä palautussovelluksessa on ainoastaan yksi luokka.

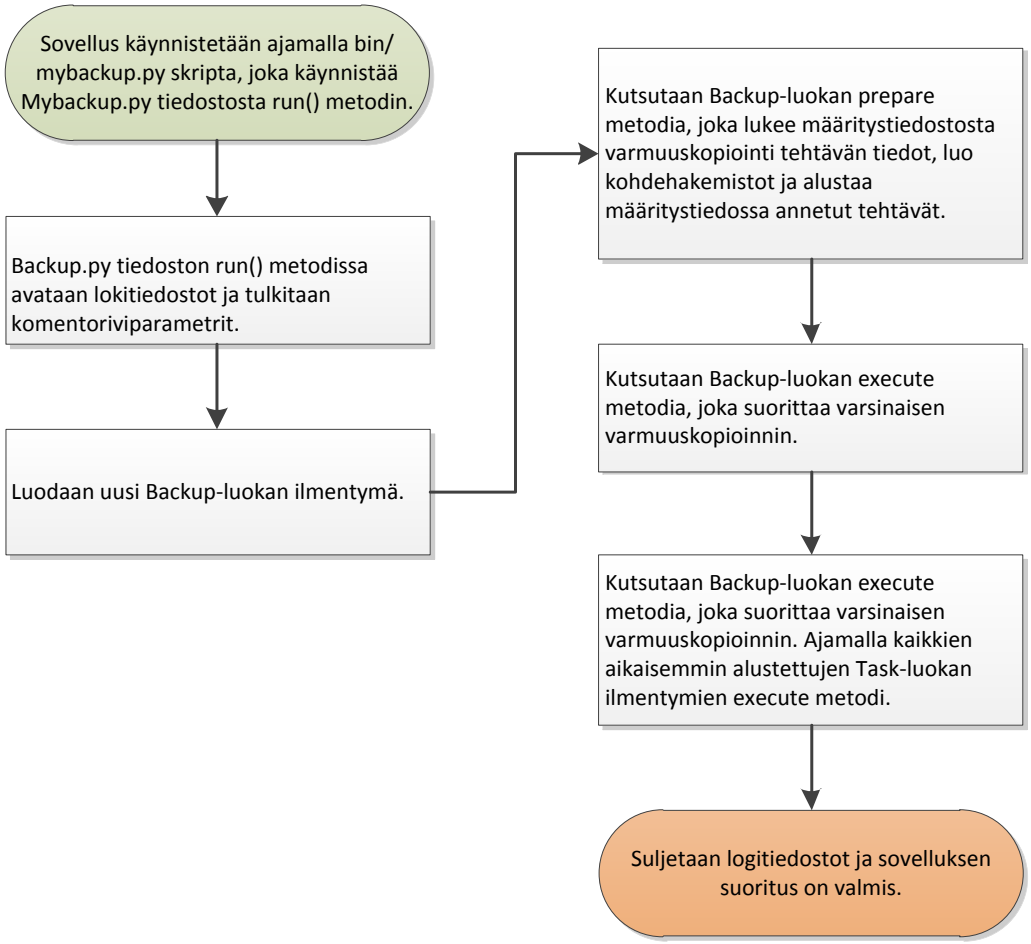


Kuva 6 Luokkakaavio

Sovelluksen koko lähdekoodi löytyy opinnäytetyön liitteestä 6. Liite sisältää myös sovelluksen käynnistämistä varten toteutetut suoritettavat ohjelmat, testauksessa käytetyt skriptat ja sovelluksen määritystiedostot.

5.4.2 Varmuuskopiointi

Varmuuskopiointisovelluksen toiminta voidaan yksinkertaistaa alla olevan vuokaavion mukaiseksi. Kuvassa ei ole otettu huomioon virheentarkistusta ja se kuvaa onnistunutta varmuuskopiointiprosessia.



Kuva 7 Varmuuskopioinnin toteutus

5.4.3 Varmuuskopiointitavat

Varmuuskopiointi voidaan jakaa joko täyteen varmistukseen tai vaiheittaiseen varmistukseen. Tieto varmuuskopiointitavasta on määritelty sovelluksen määritystiedostoon. Esivaatimuksena vaiheittaiselle varmistukselle on aikaisemmin suoritettu täysi varmistus.

Toinen varmuuskopiointiin liittyvä määritys on varmistuksen tallentamistapa. Varmuuskopio voidaan tallentaa suoraan paikalliselle levylle tai siitä voidaan muodostaa tar paketti uudelleen-ohjauksen avulla. Tämä paketti voidaan tallentaa joko samalle palvelimelle tai suoraan toiselle palvelimelle. Paketti voidaan myös valinnaisesti pakata käyttäen gzip ohjelmaa.

5.4.4 Raportointi

Raportointi varmuuskopiointiprosessin tilasta tapahtuu käyttämällä hyväksi Pythonin syslog moduulia. Moduuli tarjoaa kattavat ominaisuudet raportoida yhtä aikaa useisiin eri kohteisiin, mukaan lukien käyttöjärjestelmäloki ja sovelluksen oma lokitiedosto.

Eri tallennuskohteiden lisäksi moduuli mahdollisti myös vaihtoehtoiset tasot raportoinnin määrälle. Oletuksena sovellus tukee viittä eri tasoa: debug, info, warning, error ja critical. Lisäksi sovellus on mahdollista suorittaa hiljaisesti, jossa se ei kirjoita stdout-virtaan mitään. Tämä on hyödyllinen ominaisuus ajastetussa suorituksessa, jossa ihminen ei ole suoraan seuraamassa varmuuskopiointiprosessia.

Vaatimusmäärittelyssä esitetty sähköpostiraportointi tapahtuu lähettämällä sovelluksen lopuksi kirjoitettu lokitiedosto sähköpostilla määrittelytiedossa annettuun sähköpostiosoitteeseen. Tarkempi tieto raportoinnin määrittelystä löytyy liitteestä 4, käyttöohje.

5.5 Testaus

Alkuperäisessä vaatimusmäärittelyssä todettiin, että testaus toteutetaan yksikkö- ja integraatiotestauksella. Yksikkötestausta oli tarkoitus käyttää yksittäisen prosessin osien testaamiseen ja integraatiotestausta koko prosessin toiminnan testaukseen. Alkuperäinen testaussuunnitelma löytyy liitteestä 5.

Yksikkötestaus oli suunniteltu toteutettavaksi PyUnit ympäristön avulla (PyUnit 2010). Toteutuksen aikana huomattiin kuitenkin, että sovelluksen toimintaperiaate on hyvin lähellä yksikkötestausta. Jokainen yksittäinen varmuuskopiointitehtävän tulos pitää tarkistaa sovelluksen toimesta ja virheiden tapahtuessa raportoida niistä. Myös Pythonin ominaisuudet ja standardikirjastot osoittautuivat niin kattaviksi, että erilaisia apufunktioita ei tarvinnut toteuttaa. Näistä syistä johtuen yksikkötestaus jätettiin pois.

Integraatiotestausta sen sijaan käytettiin koko sovelluksen kehittämisen aikana. Sovelluksen kehityksen aikana jokainen käyttötapaus testattiin aina kun uusi käyttötapaus oli toteutettu. Testaus tapahtui käyttämällä yksinkertaisia skriptoja, jotka suorittivat varmuuskopioinnin eri parametreilla ja palauttivat varmuuskopioidun tietokannan. Tämä osoittautui tehokkaaksi tavaksi löytää ongelmia ja varmistaa että koko prosessi voitiin suorittaa luotettavasti.

Kehityksenaikaisesta testauksesta ei täytetty testauslomakkeita tai pidetty muuten erityisesti kirjaa. Kehitystyön yhteydessä huomatut virheet korjattiin sen sijaan aina ennen uusien ominaisuuksien kehittämistä. Näin sovelluksen toiminnasta voitiin varmistua ja uusien ominaisuuksien myötä tapahtuneet virheet pystyttiin korjaamaan tehokkaasti.

Kun sovellus oli saatu valmiiksi, tehtiin vielä täysi testauskierros. Tavoitteena tällä testauksella oli varmistaa sovelluksen toimivuus ennen sen käyttöönottoa tuotantoympäristössä.

5.6 Käyttöohje

Viimeisenä vaiheena toteutuksessa kirjoitettiin sovelluksesta käyttöohje, jonka tarkoituksena on mahdollistaa sovelluksen käyttö ulkopuolisten tahojen osalta. Käyttöohje on tarkoitettu Linux ylläpitäjien luettavaksi, joten se ei pyri kattamaan kaikkea MySQL palvelimen asennukseen tai Linux ylläpitoon kuuluvia asioita. Kokonaisuudessaan käyttöohje löytyy liitteestä 4.

6 Yhteenveto

Opinnäytetyön tavoitteena oli Mr.Goodliving Oy:n palvelinympäristön kehittäminen. Työ aloitettiin esiselvityksellä, jossa kartoitettiin palvelinympäristön kehitystarpeita. Esiselvityksen perusteella päätettiin opinnäytetyössä toteuttaa sovellus MySQL tietokannan varmuuskopiointi- ja palautusprosessin automatisointia varten.

6.1 Projektin tulokset

Projektin tärkeimpänä tuloksena on Python kielellä toteutettu sovellus, jonka avulla voidaan hallinnoida MySQL tietokannan varmuuskopiointi- ja palautusprosessia ja suorittaa automatisoituja varmuuskopioita. Sovelluksen lisäksi projektin lopputuloksena saavutettiin täydennetty dokumentaatio palvelinympäristöstä, jota on hyödynnetty palvelinympäristön ylläpidossa ja tiedon välittämisessä eri yhteistyötahoille.

Projektinäkökulmasta toteutusvaihe edistyi hyvin ja se saatiin suoritettua asetettujen tavoitteiden mukaisesti. Opinnäytetyön kirjallista osiota ei kuitenkaan aikataulutettu osaksi projektisuunnitelmaa, mikä oli osittain syynä siihen, että koko opinnäytetyön loppuunsaattaminen kesti pitkään. Toisena suurena syynä oli se, että omat työtehtävät vaihtuivat toteutusvaiheen loppuessa, jolloin aikaa kului paljon uusien työtehtävien opettelemiseen, eikä ylimääräistä aikaa ollut käytettävissä opinnäytetyön tekemiseen. Kolmantena syynä pitkään kestoon oli työn hajanaisuus, mikä vaati asioiden kertaamista ennen varsinaisen työn aloittamista.

Tärkeimmät oppimiskokemukset opinnäytetyön kannalta olivat projektin aikataulutuksen onnistuminen ja toteutettavan kokonaisuuden laajuuden onnistunut määritys. Tämä antoi vahvistusta omalle arviointikyvylle ohjelmistoprojektien toteutusta suunniteltaessa. Opinnäytetyön teoriaosuus puolestaan vahvisti tietojen varmistuksen ja erityisesti MySQL tietokanan varmuuskopioinnin erilaisten vaihtoehtojen ymmärrystä. Käytännön tasolla Python-kielen osaaminen

6.2 Kokemukset sovelluksesta

Käytännössä sovellusta käytettiin sen alkuperäiseen tarkoitukseen ja se osoittautui toimivaksi. Varmistusten palauttamista ei jouduttu käyttämään varsinaisen tietokannan palauttamiseksi, mutta sitä voitiin hyödyntää testitietokannan asentamisessa. Sovellus ei myöskään vaatinut jatkuvaa ylläpitoa ja alkuperäisen asennuksen jälkeen sen toiminta oli automatisoitu.

6.3 Muita vaihtoehtoja

Toiminnaltaan toteutettua sovellusta vastaavia ohjelmia löytyy useita. Esimerkiksi Pythonilla toteutettu DoIT Automation Tool tarjoaa myös mahdollisuuden luoda ja määritellä suoritettavia tehtäviä (DoIT Automation Tool, 2010). Toinen esimerkki on Cedar Backup, joka on tarkoitettu paikallisten ja etäkäytettävien tietokoneiden varmuuskopiointiin, mutta mahdollistaa myös erilaisten tehtävien määrittelyn (Cedar Backup, 2010).

Suurimpana etuna oman sovelluksen toteuttamisessa oli mahdollisuus räätälöidä sekä sovelluksen ja tehtävien parametrisointi sopimaan MySQL tietokannan varmuuskopiointia varten. Lisäksi tiedonvälitys esimerkiksi edellisen tehtävän suoritusajankohdasta ja tietokannan tilanteesta oli helpompi toteuttaa omalla ratkaisulla. Ottaen huomioon sovelluksen pienuuden osoittautui oman sovelluksen kehittäminen oikeaksi ratkaisuksi.

6.4 Ehdotukset jatkotoimenpiteiksi

Ominaisuuksien osalta sovellus täytti kaikki vaaditut toiminnallisuudet eikä yrityksen näkökulmasta tarvetta uusien ominaisuuksien jatkokehitykseen esiintynyt. Sen sijaan oli toivottavissa, että sovellusta voitaisiin hyödyntää myös muiden yritysten toimesta. Tämä olisi mahdollista, jos sovellus julkaistaan esimerkiksi avoimenlähdekoodin lisenssillä. Tavoitteena olisi saada palautetta sovelluksen toimivuudesta eri ympäristöissä ja lisäksi saada palautetta sovelluksen mahdollisista ongelmista.

Vaikka sovellus dokumentoitiin kattavasti oli sen käyttötarkoitus kuitenkin suunniteltu Mr.Goodliving Oy:n tarpeiden perusteella. Tärkein tavoitteista olisi toteuttaa Python moduuliasennus. Tämä mahdollistaisi sovelluksen asentamisen helposti eri ympäristöihin.

Toiminnallisuuksista dynaaminen alitehtävien suorittaminen mahdollistaisi erilaisten kolmansille osapuolille helpomman tavan toteuttaa eri varmuuskopiointitehtäviä. Tämä poistaisi myös tehtävien rippuvaisuuden tehtävien suorittamisesta vastaavan logiikan osalta. Raportointi toiminnallisuus olisi myös hyvä eriyttää omaksi kokonaisuudeksi. Tämä mahdollistaisi erilaisten raportointitapojen kehittämisen ja tekisi sovelluksesta helpommin muokattavan.

Näiden muutosten avulla sovellus olisi helppo ottaa käyttöön ja muokata erilaisia varmuuskopiointitarpeita vasten. Samalla sovelluksen rakenne selkiytyisi entisestään ja sen ylläpito muuttuisi helpommaksi.

Lähteet

Painetetut

Balling,D., Lentz, A., Schwartz, B., Tkachenko, V., Zaitsev, P. & Zawodny, D. 2008. High Performance MySQL, Second Edition. O'Reilly, Sebastopol.

Barroso, L., Pinheiro, E. & Weber, W., 2007. Failure trends in a large disk drive population. Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST 2007).

Bell, C., Kindahl, M, & Thalmann, L. 2010. MySQL High Availability. O'Reilly, Sebastopol.

Preston, W. 2007. Backup & Recovery. O'Reilly, Sebastopol.

Vadala, D., 2009. Managing RAID on Linux. O'Reilly, Sebastopol.

Painamattomat

DoIT Automation Tool 2010. About. Luettavissa http://python-doit.sourceforge.net/index.html. Luettu 22.10.2010.

Cedar Backup 2010. Cedar Backup. Luettavissa http://cedar-backup.sourceforge.net/. Luettu 22.10.2010.

PyUnit 2010. PyUnit - the standard unit testing framework for Python. Luettavissa http://pyunit.sourceforge.net/. Luettu 20.10.2010.

Thoughtpolice 2010. Some VMware images. Luettavissa http://www.thoughtpolice.co.uk/vmware/. Luettu 22.10.2010.

Tietosuojavaltuutetun toimisto 2010. Tietoa rekisterinpitäjälle. Luettavissa http://www.tietosuoja.fi/1698.htm. Luettu 20.10.2010.

Liitteet

Liite 1. Kehityshankkeiden kartoitus

Mr.Goodliving development plans

1 Version history

Date	Author	Comment
25.05.2010	Janne Kaistinen	Initial version
25.05.2010	Janne Kaistinen	Approved by steering group

2 Buildmaster refactoring

2.1 Business case

The buildmaster as it currently works is relatively crude tool for handling the building of game SKUs. The management of skus to build and/or rebuild is limited to large batches. In addition, it doesn't support any ad hoc builds of any kind.

Another problem with buildmaster is the fact that it is somewhat resource hog – especially updating the build queue. Part of this is a result of the GameDB design, which has been addressed in another project proposal.

From UI point of view, managing the build queue and setting up build sets is currently cumbersome and unintuitive. To fix this would require thinking about how the build sets are configured and how test results are handled.

2.2 Project scope

The scope of the project is large and requires quite a lot rethinking if done right. The project can be divided into following parts:

1. Redesigning buildsets and buildset inheritance
2. Designing the UI to support the new redesigned buildset structure
3. Rewriting of the build queue updater
4. Rewriting the build master clients

2.3 Estimated resources

Estimated time rewriting, designing, documenting and testing all of the project tasks would take at least 1 month of full time development. In addition, if this large of change would be made, opinions and requirements should be gathered from all stake holders.

2.4 Other issues

Since this would be a major task to implement, other solutions should be though too. There might be frameworks available for distributed task management which might be suitable. Initial review of at least the software listed here should be done:
http://en.wikipedia.org/wiki/List_of_build_automation_software

2.5 Suitability for thesis

Not suitable because of large scope and lot of unknown issues. In addition, the project results might not be worth of the expense.

3 Memcached server infrastructure

3.1 Business case

Using memcached instead of direct MySQL access allows a lot of the services to scale massively. In addition, memcached is resource wise a lot more efficient solution compared to MySQL for many tasks. Currently there are several local memcache instances, but one or more distributed servers would be ideal.

The big question is however is there any point in starting to investigate how to setup a memcached server infrastructure. RealNetworks and the Fusion team are using memcached already and can probably provide help setting up things.

3.2 Project scope

The project would consist of two separate tasks:

1. Testing and installing a dedicated memcached server
2. Creating a software library and procedure to prevent key conflicts

3.3 Suitability for thesis project

The server requirements and the fact that RealNetworks has already the required expertise available for configuring a memcached server infrastructure make this a bad choice for a thesis project.

4 Improving Cobra Excel support

4.1 Business case

Several pages in Cobra would be helped from a real integration between Cobra and Excel. There is currently an option available for exporting data into Excel XML-format, but no way to import data into Cobra.

Another place where Excel integration could be useful are in reports. Some of the Cobra pages have a dual functionality as reporting pages and data editing pages. These could and should be separated for easier maintenance.

The functionality has been requested several times, but it has never been a priority since the current Excel import does work – even if it is not the best one possible. The best library for importing and exporting Excel data has been found, so no comparison of different libraries is necessary.

4.2 Project scope

The project can be divided into three separate tasks:

1. Creating required software components for easy exporting of data.
2. Creating required software components for easy importing and reviewing of data.
3. Converting the existing pages to new exporting where needed.

4.3 Estimated resources

Setting the importing of data and creating the required software components would probably take 3 days. Importing data would take about 5 days. Converting existing pages into new one would take about a week, if just the most important pages would be converted. Total for the project would be around 2 weeks of full time development.

4.4 Suitability for thesis project

This could be a suitable project for a bachelor thesis project. However, this is mostly refactoring code and rather mechanical work. The only real design issue is how to handle importing of data back to Cobra.

5 GameDB database optimization

5.1 Business case

There are several cases in the GameDB design, where the database has been normalized, but which causes problems with writing efficient queries. A solution for this problem is to flatten or cache the current status information from these dependant relations into the main table.

One of the most expensive cases is the platform status information. The platform parent / child informatio is stored in a tree format, which requires a recursive query for finding the parent of a platform in the worst case. Writing the parent information directly into platform table would allow fast queries in the build queue updater and a lot of other places where SKUs are located for a platform.

5.2 Project scope

The project would consist of two phases:

1. Editing the database and modifying the platform edit pages in Cobra
2. Rewriting slow and/or resource expensive code to take advantage of the new optimized query.

5.3 Estimated resources

The editing of the database and modifying the platform edit pages in Cobra would take 2-3 working days. Changing build queue updater to take advantage of the new system would probably take about a week, but would also allow the opportunity to refactor the code. In total the time taken to do this project could be about one to a maximum of three weeks of coding, if all places would be changed to take advantage of the code.

5.4 Expected results

Performance wise this would remove one of the biggest slowdown factors in build queue updated. However, since we are not really pushing the internal tools resource limits, optimizing this is not necessary a high priority project.

5.5 Suitability for thesis project

This could be a suitable project for the bachelor thesis. However, since this is so code intensive project, not a lot of documentation or general usage information would be gathered.

6 Database upgrade and replication

6.1 Business case

Currently we are running all of our MySQL databases on version 5.0. The 5.1 branch is however stable and contains several interesting changes which could help us manage and scale things. The most important changes are:

1. Partitioning – this would help manage the 1TB GameDB a lot easier.

2. Row-based replication – the experiments with large binary files in MySQL and replication have failed consistently on 5.0 release. This together with the difficulty of starting a new slave has rendered replication useless for GameDB.

3. Server log tables - would reduce the performance hit from logging things. Enables turning logging on and off dynamically.

In addition, upgrading the database version is inevitable and will happen in the future.

6.2 Project scope

The project would consist of two phases:

1. Setting up a new server with MySQL 5.1
2. Upgrading existing databases
3. Planning and implementing a test harness for the upgraded database

6.3 Estimated resources

If everything would go according to plan and no major problems would be found, this would be about one and half week project. Setting up the server and installing databases would take maybe one day. Designing and running tests would take maybe 5-7 days depending on problems and how extensively the testing would be done.

6.4 Suitability for thesis project

Somewhat problematic. Requires server space and outside resources a lot.

7 DB backup process automation & documentation

7.1 Business case

The process for moving and taking backups from the GameDB has proved to be a hard problem. When the Mr.Goodliving server hosting last changed from Aldata to Academica, it was the GameDB transfer that took over a month before it was successfully running in new server. Even after the transfer complete, the sheer size of the database (1TB) has proven to be problematic for backups. Traditional dumps lock the database for way too long time and automating the backups doesn't work without custom software.

It is probable that the server hosting partner will change again soon and for that we need to the transfer process again. Even if this process is now known and tested, it is error prone. considering the large amounts of data this can mean that each mistake may mean one day delay easily. Therefore automatization and documenting the procedure is needed.

7.2 Project scope

The project is a traditional server side software project. The purpose is to create wrapper scripts for automating all tasks defined in a process document for taking a backup / transferring a database.

The process can be divided into the following phases:

1. Developing the process for making a backup and validating it.
2. Documenting and modeling the process.
3. Automating the tasks that can be automated – including testing and verification.
4. Doing a test run with either taking a backup at current hosting site or transferring the GameDB to new hosting place.

7.3 Estimated resources

Developing and testing the process for taking a backup should take about 2 days. Documenting and modeling the process will take about 3 days. Writing the scripts and testing the automatization process will take about a week. Doing a test run will take maybe one full day. Finalizing the project for external use should take a few days.

All in all the project would take about three weeks for full implementation.

7.4 Suitability for thesis project

This project is a good candidate for a bachelor thesis project, because it is independent of the Mr.Goodliving technology and the result is not company secret. Therefore all material can be used and NDA problems minimized.

Liite 2. Vaatimusmääritys

Requirement specification for MySQL backup process

1 Version history

Date	Author	Comment
14.06.2010	Janne Kaistinen	Initial version
15.06.2010	Janne Kaistinen	Approved by steering group.

2 Overview

The purpose of this document is to specify the requirement specification for an automated process of taking backups of large MySQL databases. In addition of just taking backups, the process must take into account other related tasks, such as backup verification and starting new slaves from backups.

This document is not a technical specification and doesn't include any implementation details. The technical aspects do however include things such as environment, planned programming languages and software and testing methodology.

3 Environment

The production environment for the process and and software is a Linux server running either Debian or CentOS distribution. The production servers virtual servers running under Xen virtualization environment, but the software must work on a standalone server too.

Other Linux distributions and UNIX like operating systems are a secondary consideration. The purpose is to provide a portable solution not dependent on any operating system specific issues.

The MySQL versions supported include 5.0 and 5.1 versions. These are currently the two most common versions provided by most stable distribution versions of Linux, including Debian and CentOS.

3.1 Programming languages

Automation of the process consists of writing wrapper scripts and programs for other tools. The languages used for this process are bash and Python. External tools may require other

languages such as Perl to be installed on the machines. However, considering that most standard installations include all of these by default, this is not a problem.

3.2 External tools

The external tools that the project depends include the following ones:

Percona xtrabackup

This is a set of scripts and software which allows making full and incremental backups from MySQL databases.

Maatkit tools

Maatkit is a set of scripts to query metadata from MySQL databases. This includes scripts for calculating hash sums from tables etc.

rsync

rsync is a tool for synchronizing data between two locations. It can be used to do incremental copies of directory structures.

In addition, other tools might be used to perform other tasks. This will depend on the implementation and whether the process can be automated using just the tools described above or not.

3.3 Development environment

Testing and developing the process is done using Mac OS X machine with a local MySQL database. In addition, the development and testing environment includes a server running both CentOS and Debian. In all of these environments the same set of tools and utilities will be available as in the production environment.

4 Requirements

The main requirements for the end result of the automated MySQL backup process have been described below. The actual process will include several sub processes which can be combined together to provide the results. These task will be modeled as the first task in the implementation phase.

4.1 Full database backup

The main requirement for the process is taking a full snapshot of the database. This means that the backup must be taken so that it represents the state of the full database at a single point in time.

4.2 Hot backup

An additional requirement for the backup process is that it must be a hot backup. In other words, the database must be available for use while the backup process is happening. This requirement comes from the fact that there are always users using the database. Another reason is that a cold database backup is trivial if we can ignore the downtime requirement and the whole process is not required anymore.

4.3 Incremental backup

An alternative option to snapshot backup strategy is to support incremental backups. The incremental backups may require making a snapshot backup first, but additional smaller backups can be taken later with only the changed data. There is still a requirement that the restored backup represents the database at a single point in time.

4.4 Preparing for replication

One of the requirements for the backup operation is to support creating a backup for starting a slave server. This requires specifying the server configuration for both master and slave server and making sure that the process can handle the additional requirements.

4.5 Verification of backup integrity

Just making a backup is not enough to be sure that the process has succeeded. In addition, there is a requirement that the backup can be restarted and that the data has not been corrupted. Therefore additional checks like data verification needs to be part of the process.

4.6 Backup locations

The process must support two main methods for taking backup. First is taking backup to a local drive available as a normal file system. Another requirement is that the backup can be done over network transfer, for example through an ssh pipe. This allows flexible locating of the backup server and removes physical server location requirements in some cases. This is not a mandatory requirement, but should be enabled if at all possible.

4.7 Logging

The database backup process must log each action and the results into a system log. This allows monitoring of the backup process and makes debugging errors easier.

4.8 Error reporting

At minimum the backup process must write each error case into system error log. In addition, the process should allow hooks for specifying additional reporting mechanism. The reporting mechanism may include for example sending emails or SMS messages in case of error.

4.9 Scheduling and synchronization

Scheduling the backups using cron or some other mechanism is one of the most important requirements for the process. If the process requires synchronization between several servers, a simple time based scheduling might not be enough. In these cases the process must have a method of continuing the backup process at another server using some kind of synchronization.

4.10 Restarting database from backup

This is the final requirement for the backup process. If the database can not be started from the backup, the whole process has failed.

5 Testing

Developing a software which depends on several different pieces of software requires extensive automated testing framework. One of the reasons is that whenever a software component changes the whole process must be tested. Doing this manually is both error prone and time consuming. Therefore developing a test framework is an integral part of the testing.
For this project, testing can be divided into two parts:

1. Unit tests for individual parts of the process
2. Testing that the whole process works and a secondary server can be started

5.1 Unit testing

Each part of the process will process some data and output other data. Therefore devising test data and test results allows developing unit tests for each individual part of the process. The test framework planned for this project is PyUnit, a Python version of the JUnit framework.

The PyUnit framework supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections and independence of the tests from the reporting framework. Considering that the main development language for the automated process will be Python, the framework is a natural match.

5.2 Integration testing

Whenever there is a new version of one of the underlying pieces of the backup software, running a full backup process and testing the results is required before upgrading production servers. To automate this an suite of scripts will be developed to easily setup a system with known configuration. This system is then backed up and restored and the end result is compared to the original database.

Another purpose for this is to enable faster development turnaround by minimizing the manual work for setting up database servers and resetting their state into a known configuration.

Liite 3. Prototyyppi

Backup process

1 Version history

Version	Date	Author	Description
0.1	19.9.2010	Janne Kaistinen	Initial version of the manual.
1.0	28.9.2010	Janne Kaistinen	Approved by steering group.

2 Overview

This document describes the process of taking backups using xtrabackup backup tool. Each of the following chapters describes one of the processes that the MyBackup tool is required to handle.

Another purpose of the manual process is to recognize patterns in the different use cases. These findings have are described in the chapter 6.

3 Full snapshot backup with verification

A full snapshot backup of the database contains all of the data at a single point in time. Using xtrabackup we can get a full snapshot of the InnoDB tables without locking full tables, but MyISAM backups require locking the tables while taking a backup.

In addition to backing up the data the process uses mk-table-checksum to calculate checksums of the data for verification. In production environment problems can arise in situation where data is continuously inserted into the database. In these cases we need to be able to select and/or exclude databases and tables from the checksum calculations.

In this case the backup is written to a local directory /mybackuptest/backup and the MySQL database is running with the database data at /mybackuptest/data

3.1 Database checksum

The mk-table-checksum program writes a file with checksums calculated for each database and table.

```
mk-table-checksum h=localhost > /mybackuptest/backup/checksum1.txt
```

3.2 Take full backup

Taking a full backup of the InnoDB and MyISAM tables can be done using the innobackupex-1.5.1 program. Adding –no-timestamp option allows specifying the exact backup location.

```
innobackupex-1.5.1 --no-timestamp /mybackuptest/backup/data
```

3.3 Prepare the backup for restore

The xtrabackup program writes log file for all transactions that happened between starting and finishing the backup. Before a new instance of the database can be run using the backup, these changes must be migrated to the backup.

```
innobackupex-1.5.1 --apply-log /mybackuptest/backup/data
```

3.4 Restore the backup

Before restarting the backup we must first stop the MySQL, remove existing data and then copy the database backup to original location. The innobackupex-1.5.1 provides an option to do the copying using the –copy-backup option. Before restarting the MySQL service the file owner and group must be changed to the MySQL user.

```
/etc/init.d/mysql stop
rm -rf /mybackuptest/data/*
innobackupex-1.5.1 –copy-back /mybackuptest/backup/data
chown –R mysql:mysql /mybackuptest/data
/etc/init.d/mysql start
```

3.5 Verify the backup

To verify that the backup was successful the mk-table-checksum is run again and the checksum files compared.

```
mk-table-checksum h=localhost > /mybackuptest/backup/checksum2.txt
diff /mybackuptest/backup/checksum2.txt /mybackuptest/backup/checksum2.txt
```

4 Incremental backup

Incremental backups extend the functionality of full backup by allowing taking backups of only the changes after last full backup. This is a lot faster than taking a full backup – especially for large databases. The incremental backups can be run multiple times with each previous incremental backup as the starting point.

Another option to simulate incremental backups is to copy the binary logs and use these to apply changes made after full backup when restoring the backup. There can be problems with this approach, which rise from the issue that the statements stored in the binary logs are run again and for example triggers and functions might not return the same data as originally.

4.1 Take an incremental backup

The location of last backup directory must be provided using the –incremental-basedir options. The xtrabackup program reads the xtrabackup_checkpoints file from this directory which contains the log sequence numbers for the last backup. The backup location can be the last incremental backup location or the last full backup location. The incremental backup is written to the directory given with –target-dir option.

The example below runs two incremental backups. The first one uses the full backup as starting point and the second the previous incremental backup.

```
xtrabackup --backup --target-dir=/mybackuptest/backup/incremental1 --incremental-
basedir=/mybackuptest/backup/data
```

```
xtrabackup --backup --target-dir=/mybackuptest/backup/incremental2 --incremental-
basedir=/mybackuptest/backup/data
```

4.2 Restoring incremental backup

The changes from the incremental backup must be migrated into the backup before it can be used. The first step is to apply the changes after the full backup changes have applied (step 2.3). The incremental backups must be restored in the same order they were taken.

```
xtrabackup --prepare --target-dir=/mybackuptest/backup/data --incremental-
dir=/mybackuptest/backup/incremental1
```

```
xtrabackup --prepare --target-dir=/mybackuptest/backup/data --incremental-
dir=/mybackuptest/backup/incremental2
```

After this the restore operation continues as in the full backup from step 2.4.

5 Stream backups

In addition to writing the backup directly to files, the xtrabackup program can output the data as a tar archive. This is outputted to stdout and can be redirected to a file or using ssh to another machine. This allows taking backups even when there is not enough space for another copy at the database machine.

Another thing to take into account is that the incremental backups require that the xtrabackup_checkpoints file can be read from database. Since all of the output is written inside the tar archive the backup utility must be able to parse this from the output of the backup tool.

The examples below show different options for taking a stream backup and how pipes can be used to for example compress the backup.

```
# Simple stream backup which outputs the backup into a file
innobackupex-1.5.1 --stream=tar ./ > backup.tar

# Same but with compression
innobackupex-1.5.1 --stream=tar ./ | gzip - > backup.tgz

# Using compression and writing the output to a remote host
innobackupex-1.5.1 --stream=tar ./ | gzip - | ssh root@localhost "cat - >
/home/jannek/backup/backupssh.tgz"

# Compression, throttling and writing to remote host
innobackupex-1.5.1 --stream=tar ./ | gzip - | pv -q -L10m | ssh
root@localhost "cat - > /home/jannek/backup/backupssh.tgz"
```

6 Notes about backup process

6.1 Owner and group of the MySQL directories

The owner and group of the restored backup data directory must be changed to the user and group of the MySQL server. For example on CentOS these must be set changed recursively to mysql.

6.2 Using the --copy-back options

When restoring from database backup, the MySQL data directory must exist but be empty. The backup tools won't work if there is data already on the directories.

6.3 The backup location

If not using the --no-timestamp option, innobackupex-1.5.1 script creates a new directory with date and time. This makes it harder to automate the process. A better way is to use some global naming system for the whole backup process.

6.4 The backup binlog position

The backup scripts use the log sequence numbers for incremental backups and they are read from xtrabackup_checkpoints file. If using stream backups this file is inside the tar archive and might be hard to extract – especially when writing to a remote host. Another option is to capture the log position from the output of the innobackupex-1.5.1 program.

6.5 SSH keys

The use of SSH for streaming the backup to remote host needs public key authentication to be setup. This removes the need to enter passwords when the backup is not run interactively.

6.6 Verifying data

Running the mk-table-checksum is best when run immediately after the backup has finished. This minimizes the changes between the backup finish and checksum. It is best to limit the checksum calculation to include only tables which have relatively slow change speed. This minimizes false alarms when comparing the backup checksums after the database has been restored.

6.7 Defaults file

By default the xtrabackup and innobackupex-1.5.1 scripts try to load mysql configuration file from my.cnf. On debian the default location is however /etc/mysql/my.cnf so the file location needs to configurable.

7 Design for automating the backup process

Analyzing the different backup commands it becomes obvious that the backup process consists of different tasks run one after another. The tasks can be run at different times so they must have some way to communicate information what has been done before and in what state the backup process is.

Each backup process consists of one full backup and several incremental backups. The restore operation doesn't need to be separated into several tasks, since there is no need to stop the process and continue it at a later time.

7.1 Task types

The backup process can be divided into the following tasks:

- Take a full backup writing to a directory
- Take a full backup using stream output
- Run an incremental backup
- Restore backup
    - Apply logs to the backup
    - Apply all incremental backups in the original order for latest to newest
    - Change the owner
- Run checksum

7.2 Communication between separate backup tasks

The communication between separate backup tasks requires that information about each back task is written to a common file. The required data includes the target directory for the previous backup and the backup task type. This information is required for the restore process.

The most consistent solution is to use one base directory for each backup process with subdirectories for each task.

7.3 Configuring backup tasks

Each of the backup tasks requires configuration which specifies the command line variables. Since the MySQL server configuration is in a key - value format with sections, it is a reasonably good format for specifying configuration options for the backup system.

Liite 4. Käyttöohje

Mybackup Manual

1 Version history

Version	Date	Author	Description
0.1	19.9.2010	Janne Kaistinen	Initial version of the manual.
0.2	27.9.2010	Janne Kaistinen	Updated manual with Debian installation instructions.
1.0	28.9.2010	Janne Kaistinen	Approved by steering group.

2 Overview

2.1 The reason behind mybackup

The purpose of the MyBackup software is to provide a robust framework for automating MySQL backups. The initial reason for the software arose from a situation where the author needed to take backups from a 1TB database which couldn't be shut down for backups. In addition, the server environment didn't support taking snapshots from the file system. There are several commercial solutions available for taking backups, but for various reasons they were not a good match for all the requirements. Therefore the decision was made to write a new tool for automating the backup process.

2.2 Features

- Taking snapshots of the full database.
- Supporting hot backup without downtime.
- Incremental backups between full backup cycles.
- Support for both MyISAM and InnoDB tables.
- Starting a new slave server from the backup.
- Verifying the integrity of the backups.
- Local storage of the backup.
- Direct writes to remote server without using temporary storage.
- Logging of the backup process.
- Reporting success and/or failure of the backup process.

• Timed backups using cron or some other mechanism.
• Support for both CentOS and Debian distributions.
• Using only free tools.
• Simple to use and configure.
• No special hardware or file system requirements.

2.3 Components

The software consists of the following components:

• A backup tool (bin/mybackup.py) for taking full and incremental backups.
• A restore tool (bin/restore.py) for restoring backups.
• Utilities for creating a test environment and populating database.

2.4 Directory structure overview

• bin - Binaries for starting the MyBackup tools.
• config - Configuration files for the backup and tests.
• data - Data files for populating the test database.
• documentation - Documentation for the tools including this.
• mybackup - The actual python code for the software
    o backup - Source code for backup and backup tasks
    o common - Utility code used by the actual tools
    o restore - Source code for restoring backups
    o tests - Source code for creating test environment.

2.5 Supported operating systems

The MyBackup tools have been tested and developed on CentOS 5.4 and Debian Lenny distributions. It should work on all Linux distributions which have the required software available, but no guarantees are made. Windows support would require a lot of modification, because the software depends on the availability of the GNU toolkits and system services such as syslog.

2.6 Required software

The following programs must be available to use all features of the MyBackup software:

• Python runtime version 2.4 – 2.7
• Perl 5.8

• MySQL server 5.0 or 5.1
• Percona xtrabackup 1.2.132 or later
• Innobackupex-1.5.1

Other standard software required for the program include tar, pv, ssh client and gzip.

3 Overview of the backup process

3.1 Configure backup task

Before starting the backup process, you must first configure the backup tasks. If the configuration file is not given as an argument to the mybackup.py program, it is loaded by default from the file config/backup.cfg. You can edit this file or write a new configuration file from scratch. The configuration file consists of sections, which group options together and key value pairs. The following snippet from the configuration file is an example how to configure the database connection information:

```
[db]
host:        localhost
port:        3306
user:        root
```

Detailed documentation about configuration the backup options are available in chapter 3.

3.2 Run full backup task

After configuring the backup settings a full backup task can be run. The full backup task is run by starting the mybackup.py program with the backup task name as argument:

```
bin/mybackup.py -t fullbackup
```

The basic full backup configuration contains first the global backup settings and the actual full backup task configuration:

```
[fullbackup]
type:                   full
tasks:                  fullbackup_task
basedir:                /mybackuptest/backups/
loglevel:               debug

[fullbackup_task]
type:                   local
defaults-file:          /etc/my.cnf
```

The basedir configuration file specifies the directory where the backup data is written to. Each full backup task creates a new subdirectory under it with name set as the current time. This allows making multiple backups of the same database without overwriting the old backup information. The backup directory name is given in the YYYY-MM-DD_HH_MM_SS format so that incremental backups can find the latest full backup and continue it without explicitly specifying the last backup directory.

After running the full backup task, the directory contains the following data:

• mybackup.log
Log output from the backup process.

• myback_restore
Information about the backup process for restore and incremental backups.

• data/
Directory which contains the data from full backup. The actual data can be located at a remote server if using the remote streaming method and in this case the directory contains just the xtrabackup_checkpoints file.

• data/xtrabackup_checkpoints
File containing the log sequence information for incremental backups.

3.3 Run incremental backup task

After a full backup task has been run it is possible to take incremental backups containing only the changes after the last full backup. Similar to the full backup, the incremental backup is started with the bin/mybackup.py program.

```
bin/mybackup.py –t incremental
```

The basic incremental backup task configuration consists of the backup settings and the incremental backup task configuration:

```
[incremental]
type:                   incremental
tasks:                  incremental_task
basedir:                /mybackuptest/backups/
loglevel:               debug

[incremental_task]
type:                   incremental
defaults-file:          /etc/my.cnf
```

When starting the incremental backup it iterates through all the subdirectories in the directory configured in the basedir and finding out the latest one. After that it reads the mybackup_restore file in the file to find out if there have been previous incremental directories and where the log sequence information can be found.

The incremental backup data is stored in a subdirectory named YYYY-MM-DD_HH_MM_SS format. In addition, the mybackup_restore file is updated to include information required by the restore process.

3.4 Restore backup

Restoring a backup doesn't require any changes to the backup configuration file. All of the data required for the restoration process is stored in the mybackup_restore file at backup directory.

The myrestore.py program requires that source and destination directories are given as argument to the program. In addition the user and group name of the MySQL service must be given as argument so that the permissions of the restored database can be changed.

```
bin/myrestore.py
-d /mybackuptest/restore
-s /mybackuptest/backups/2010-09-19_16-54-06/
-g mysql
-u mysql
```

The restore program first writes the backup data into the destination directory and then applies all changes from incremental backups. Finally it changes recursively the owner and group of the restore directory.

4 Configuring backup tasks

If no configuration file is specified as an argument the configuration file is read from config/backup.cfg. For an example of how to configure backup tasks see config/backup.cfg file.

4.1 General task configuration

Each backup task is defined in a single section of backup configuration. This name is given as argument to the mybackup.py program. The actual operations done by the backup are configured in sections which are listed in tasks key. This allows multiple sub tasks to be run for the same backup task.

Option	Value
type	This option specifies the type of backup to run. The allowed values are full and incremental.
tasks	List of tasks to run as part of the backup. The list is a comma separated list of task names. Each of the task names must have their own section in the configuration file.
basedir	The base directory for backups. The actual backup data is written in a sub directory.
loglevel	Defines the logging verbosity for the program. Allowed values are debug, info, warning, error or critical.

4.2 Local backup

Local backup writes the data directory to the local machine. This simplest of the backup operations.

Option	Value
type	local
defaults-file	Specifies the operating system specific location of the MySQL configuration file. Usually found at /etc/my.cnf
basedir	The base directory for backups. The actual backup data is written in a sub directory.

3.3 Stream Backup

Stream backup creates a tar-archive instead of writing the data directly into disk.

Option	Value
type	local
defaults-file	Specifies the operating system specific location of the MySQL configuration file. Usually found at /etc/my.cnf
compression	Compression level for the backup. 0 to disable 1-9 to specify the compression level from fastest to slowest. Slower compression means better compression.
filename	Name of the output filename. You can use the following parameters as part of the filename: - %date% converted to YYYY-MM-DD - %time% converted to HH-MM-SS - %datetime% converted to YYYY-MM-DD_HH-MM-SS For example: stream_%datetime%.tgz

4.4 Remote backup

Remote backup extends the stream backup by allowing the data to be written into another machine.

Option	Value
type	local
defaults-file	Specifies the operating system specific location of the MySQL configuration file. Usually found at /etc/my.cnf
compression	Compression level for the backup. 0 to disable 1-9 to specify the compression level from fastest to slowest. Slower compression means better compression.
filename	Name of the output filename. You can use the following parameters as part of the filename: - %date% converted to YYYY-MM-DD - %time% converted to HH-MM-SS - %datetime% converted to YYYY-MM-DD_HH-MM-SS For remote backup it is usually best to provide an absolute path to the file-

	name, for example: /backups/servername/%datetime%.tgz
throttle	If a value is specified for throttling, the output stream is piped through pv command. Allowed values are the settings specified for the –L option at man pv. For example 10m or 1g.
connection	SSH connection string. Example with explicit key file specification: ssh root@localhost –i /root/.ssh/.id_rsa

4.5 Incremental backup

Incremental backup looks for the latest backup directory under the basedir configuration setting and reads the backup status from mybackup_restore file.

Option	Value
type	incremental
defaults-file	Specifies the operating system specific location of the MySQL configuration file. Usually found at /etc/my.cnf

4.6 Checksum

Checksum task calculates a checksum of the database data which can be later compared to the output of another checksum task for data verification.

Option	Value
type	checksum
filename	checksum.txt
databases	Names of the databases from which checksum is calculated. A list of database names separated by comma.
ignore-database	Names of the databases to exclude from checksum calculation. A list of database names separated by comma.
tables	List of table names separated by comma to include into the checksum calculation. Full names must be used: mysql.user
ignore-tables	List of table names separated by comma to exclude from the checksum calculation. Full names must be used: mysql.user
dbconfig	Section name where the database connection information is read from.

4.7 Database connection information

Currently only the checksum task requires database connection.

Option	Value
host	Host information for database connection
port	Port information for database connection.
user	Username for database connection.
password	Password for database connection.

4.8 Example configuration file

A full example of the database configuration file is shown below. It contains all the backup task types described in the previous chapters.

```
########################################
# Database connection information
########################################
[db]
host:                   localhost
port:                   3306
user:                   root
password:

########################################
# Local backup task
########################################
[local]
type:                   full
tasks:                  local_backup,local_checksum
basedir:                /mybackuptest/backups/
loglevel:               debug

[local_checksum]
type:                   checksum
filename:               checksum-before.txt
databases               mysql,mybackuptest
ignore-databases:       test
tables:
ignore-tables:          mysql.user
dbconfig:               db

[local_backup]
type:                   local
defaults-file:          /etc/my.cnf

########################################
# Local stream backup
########################################
[stream]
type:                   full
tasks:                  stream_backup
basedir:                /mybackuptest/backups/
loglevel:               debug

[stream_backup]
type:                   stream
defaults-file:          /etc/my.cnf
compression: 1
```

```
filename:                stream_%datetime%.tgz

########################################
# Remote stream
########################################
[remote]
type:                    full
tasks:                   remote_backup
basedir:                 /mybackuptest/backups/
loglevel:                debug

[remote_backup]
type:                    remote
defaults-file:           /etc/my.cnf
filename:                /root/backup/mybackup_%datetime%.tgz
compression: 1
throttle:                10m
connection:              ssh root@localhost -i /root/.ssh/id_rsa

########################################
# Incremental backup task
########################################
[incremental]
type:                    incremental
tasks:                   incremental_backup
basedir:                 /mybackuptest/backups/
loglevel:                debug

[incremental_backup]
type:                    incremental
defaults-file:           /etc/my.cnf
```

5 Centos installation

The first step is to download a VMware image for CentOS 5.4. There are several ready images available, but the image used for this test was downloaded from http://www.thoughtpolice.co.uk/vmware/. After downloading and starting the virtual machine, the next step was to change system settings like keyboard and updating the available packages. Some links with information about configuring CentOS can be found at:

- http://www.thoughtpolice.co.uk/vmware/howto/1-minute-guide.html#centos5
- http://www.centos.org/docs/5/html/5.4/Technical_Notes/

5.1 Configuring yum repositories

By default CentOS has has only a limited set of enabled repositories and software available. Since we are going to be installing several packages, it is necessary to enable more software options. However, since we are going to be enabling third party repositories, it is recommended to first install yum-priorities package. This is not exactly required, but might prevent some accidental mistakes. In addition, the extras, centosplus and contrib repositories should be enabled if they have been disabled. A short list of what to do:

• install yum-priorities
• enable the extra, centosplus and contrib repositories
• add the rpmforge repository
• set priorities for the default repositories

For more detailed instructions how to configure yum repositories see the following pages:

• http://wiki.centos.org/AdditionalResources/Repositories/RPMForge

5.2 Installing MySQL on CentOS

The default MySQL packages available from the CentOS repositories are still based on MySQL 5.0 repositories. In addition, they are not compatible with the packages from Percona. Therefore the best solution is to download packages directly from MySQL.com website.

After downloading the rpm packages, installing them can be done as follows:

```
rpm -ihv MySQL-client-community-5.1.48-1.rhel5.x86_64.rpm
rpm -ihv MySQL-devel-community-5.1.48-1.rhel4.x86_64.rpm
rpm -ihv MySQL-shared-compat-5.1.48-1.rhel5.x86_64.rpm
rpm -ihv MySQL-server-community-5.1.48-1.rhel5.x86_64.rpm
```

After installing the server rpm, mysql server is automatically started. To check that the installation worked, you can run the mysql client program.

5.3 Installing maatkit

Maatkit is a toolkit for open source databases. It includes several scripts that can be used for replication, analyzing and other tasks. The latest rpm version for maatkit can be downloaded from http://www.maatkit.org.

The maatkit toolkit depends on Perl DBD::mysql and Term::ReadKey libraries, so installing the must be done first.

```
yum install perl-DBD-MySQL perl-TermReadKey
```

After downloading, installing the toolkit can be done as follows:

```
rpm -ihv maatkit-6839-1.noarch.rpm
```

5.4 Installing Percona XtraBackup

```
rpm -ihv xtrabackup-1.2-22.rhel5.x86_64.rpm
```

For more information, read the following pages:

- http://www.percona.com/docs/wiki/percona-xtrabackup:start
- http://www.percona.com/downloads/XtraBackup/

5.5 Installing Python MySQL library

```
yum install MySQL-python
```

5.6 Install support for throttling

To use the throttling option pv package needs to be installed.

```
yum install pv
```

6 Debian installation

The first thing to do is to download a Debian virtual image. The image used for testing the MyBackup test was downloaded from http://www.thoughtpolice.co.uk/vmware/.

6.1 Configuration

Installing required software for Debian is relatively straightforward. Most of the required packages are available through the Debian repository and only xtrabackup requires using third party repository. Below is an example how to configure the VMWare image downloaded from the thoughtpolice web site for use with the MyBackup software.

```
# Reconfigure keyboard
dpkg-reconfigure console-data

# Install vmware tools
mount /dev/cdrom /cdrom
cp /cdrom/VMWareTools-8.4.3-282343.tar.gz /tmp
cd /tmp
tar xvzf VMWareTools-8.4.3-282343.tar.gz
vmware-tools/vmware-install.pl

# Reboot after installing vmware
reboot

# Edit the /etc/apt/sources.list
# The sources list should have main repository settings set to
# something like this, without commented out
# deb http://ftp.fi.debian.org/debian/ lenny main contrib non-free
# deb-src http://ftp.fi.debian.org/debian/ lenny main

# Update packages
aptitude update
aptitude upgrade

# Install required basic software
aptitude install sudo python python-mysqldb

# Install mysql client and server
aptitude install mysql-server mysql-client

# Install maatkit
aptitude install maatkit

# Install Percona keyfile
wget http://www.percona.com/downloads/RPM-GPG-KEY-percona
apt-key add RPM-GPG-KEY-percona

# Add Percona repository to /etc/apt/source.list
# deb http://repo.percona.com/apt lenny main
# deb-src http://repo.percona.com/apt lenny main

# Install xtrabackup
apt-get install xtrabackup
```

Liite 5. Testaussuunnitelma

Testing Plan

1 Version history

Version	Date	Author	Description
0.1	19.9.2010	Janne Kaistinen	Initial version of the testing plan.
1.0	28.9.2010	Janne Kaistinen	Approved by steering group.

2 Overview

This document describes testing of the MyBackup software. The document describes how the tests were selected, what is tested and how the test results are verified.

3 Test targets

Testing of the software is done using integration testing. The individual tasks are relatively simple, but the interaction of the tasks is more complicated and error prone. By using integration testing the full backup process can be validated.

The following processes were selected for integration testing. They cover all of the functionality of the software:

- Local backup and restoration
- Stream backup and restoration
- Remote backup and restoration

4 Test process

All of the tests consist of running two shell scripts. The first script creates a backup using different backup methodology and another script which restores the backup.

4.1 Backup script process

There are three different backup scripts which test local, stream and remote backup functionality. The scripts are bash scripts which perform the following tasks:

1. Stop the MySQL server
2. Remove all files and directories under /mybackuptest directory
3. Create a test database using the test_create_environment.py script
4. Populate the database using test_populate_database.py script
5. Write all files under the MySQL data directory to /mybackuptest/files1.txt
6. Run the mybackup.py script with local, stream or remote argument
7. Run incremental backup
8. Run incremental backup a second time

4.2 Restore script process

The restore script tries to restore the database from the backup and verify that the database hasn't been changed during the backup process. The process for the restore script is the same and doesn't depend on the backup type.

1. Stop the MySQL server
2. Remove the original database located at /mybackuptest/data
3. Replace /etc/my.cnf file with a version using /mybackup/restore as data directory
4. Run myrestory.py to restore the backup to /mybackup/restore directory
5. Start MySQL server
6. Run checksum task and output the result to /mybackuptest/restore/checksum.txt
7. Compare the checksum files taken while running backup and in previous step
8. Write a list of all files under /mybackuptest/restore to /mybackuptest/files2.txt
9. Compare the output of file listings
10. Launch mysql client program to verify that we can access the database

5 Testing methods and timing

Integration testing is done when a new backup type has been developed and it has passed developer testing. At this point the previously finished components and the new functionality are tested again.

Liite 6. Sovelluksen lähdekoodi

bin\mybackup.py

```
#! /usr/bin/python

# This makes sure that users don't have to set up their environment
# specially in order to run these programs from bin/.
import sys
import os
import string

if string.find(os.path.abspath(sys.argv[0]), os.sep+'mybackup') != -1:
            sys.path.insert(0,
os.path.normpath(os.path.join(os.path.abspath(sys.argv[0]), os.pardir, os.pardir)))

if hasattr(os, "getuid") and os.getuid() != 0:
            sys.path.insert(0, os.path.abspath(os.getcwd()))

# Run the create test environment
from mybackup.backup.backup import run
run()
```

bin\myrestore.py

```
#! /usr/bin/python

# This makes sure that users don't have to set up their environment
# specially in order to run these programs from bin/.
import sys
import os
import string

if string.find(os.path.abspath(sys.argv[0]), os.sep+'mybackup') != -1:
            sys.path.insert(0,
os.path.normpath(os.path.join(os.path.abspath(sys.argv[0]), os.pardir, os.pardir)))

if hasattr(os, "getuid") and os.getuid() != 0:
            sys.path.insert(0, os.path.abspath(os.getcwd()))

# Run the create test environment
from mybackup.restore.restore import run
run()
```

bin\test_create_environment.py

```
#! /usr/bin/python

# This makes sure that users don't have to set up their environment
# specially in order to run these programs from bin/.
import sys
import os
import string

if string.find(os.path.abspath(sys.argv[0]), os.sep+'mybackup') != -1:
            sys.path.insert(0,
os.path.normpath(os.path.join(os.path.abspath(sys.argv[0]), os.pardir, os.pardir)))

if hasattr(os, "getuid") and os.getuid() != 0:
            sys.path.insert(0, os.path.abspath(os.getcwd()))
```

```
# Run the create test environment
from mybackup.tests.test_create_environment import run
run()
```

bin\test_populate_database.py

```
#! /usr/bin/python

# This makes sure that users don't have to set up their environment
# specially in order to run these programs from bin/.
import sys
import os
import string

if string.find(os.path.abspath(sys.argv[0]), os.sep+'mybackup') != -1:
            sys.path.insert(0,
os.path.normpath(os.path.join(os.path.abspath(sys.argv[0]), os.pardir, os.pardir)))

if hasattr(os, "getuid") and os.getuid() != 0:
            sys.path.insert(0, os.path.abspath(os.getcwd()))

# Run the create test environment
from mybackup.tests.test_populate_database import run
run()
```

config\ backup.cfg

```
########################################
# Database connection information
########################################
[db]
host:                   localhost
port:                   3306
user:                   root
password:

########################################
# Checksum task
########################################
[checksum]
type:                   checksum
filename:               checksum.txt
databases:              mysql,mybackuptest
ignore-databases:       test
tables:
ignore-tables:          mysql.user
dbconfig:               db

########################################
# Local backup task
########################################
[local]
type:                   full
tasks:                  local_backup,checksum
basedir:                /mybackuptest/backups/
loglevel:               debug

[local_backup]
type:                   local
defaults-file:          /etc/my.cnf
```

```
########################################
# Local stream backup
########################################
[stream]
type:                   full
tasks:                  stream_backup,checksum
basedir:                /mybackuptest/backups/
loglevel:               debug

[stream_backup]
type:                   stream
defaults-file:          /etc/my.cnf
compression: 1
filename:               stream_%datetime%.tgz

########################################
# Remote stream
########################################
[remote]
type:                   full
tasks:                  remote_backup,checksum
basedir:                /mybackuptest/backups/
loglevel:               debug

[remote_backup]
type:                   remote
defaults-file:          /etc/my.cnf
filename:               /root/backup/mybackup_%datetime%.tgz
compression: 1
throttle:               10m
connection:             ssh root@localhost -i /root/.ssh/id_rsa

########################################
# Incremental backup task
########################################
[incremental]
type:                   incremental
tasks:                  incremental_backup,checksum
basedir:                /mybackuptest/backups/
loglevel:               debug

[incremental_backup]
type:                   incremental
defaults-file:          /etc/my.cnf

########################################
# Diff task
########################################
[diff]
type:                   diff
file1:                  checksum-before.txt
file2:                  checksum-after.txt
```

config\ test_config.cfg

```
# MySQL connection configuration
#
# Directory entries must be given without trailing slash: /data not /data
#
[db]
host:                                   localhost
port:                                   3306
user:                                   root
```

```
password:

[common]
basedir:                                /mybackuptest
datadir:                                data
innodb_data_home_dir:                   data
innodb_log_group_home_dir: data
init_script:              /etc/init.d/mysql
testuser:                               mybackup
testpassword:             mybackup

[centos]
user:                                   mysql
group:                                  mysql
mycnf:                                  /etc/my.cnf
port:                                   3306
socket:                                 /var/lib/mysql/mysql.sock
pid:                                    /var/lib/mysql/mysql.pid
defaults_file:                          /etc/my.cnf

[debian]
user:                                   mysql
group:                                  mysql
mycnf:                                  /etc/mysql/my.cnf
port:                                   3306
socket:                                 /var/run/mysqld/mysqld.sock
pid:                                    /var/run/mysqld/mysqld.pid
defaults_file:                          /etc/mysql/my.cnf

# The values allowed for engine are myisam, innodb or both
[testdb]
database:                               mybackuptest
tables:                                 2
rows:                                   10
size:                                   1k
engine:                                 both
```

config\ test_my_template.cnf

```
[client]
port                            = %port%
socket                          = %socket%

[mysqld]
port                            = %port%
socket                          = %socket%
pid                             = %pid%
datadir                         = %datadir%

skip-locking
key_buffer_size = 16K
max_allowed_packet              = 1G
sort_buffer_size                = 64K
read_buffer_size                = 256K
read_rnd_buffer_size            = 256K
net_buffer_length               = 2K
thread_stack                    = 128K
log_bin
             = binlog

innodb_data_home_dir            = %innodb_data_home_dir%
innodb_data_file_path           = ibdata1:10M:autoextend
innodb_log_group_home_dir       = %innodb_log_group_home_dir%
innodb_buffer_pool_size         = 128M
```

```
innodb_additional_mem_pool_size = 2M
innodb_log_file_size            = 32M
innodb_log_buffer_size          = 32M
innodb_flush_log_at_trx_commit  = 1
innodb_lock_wait_timeout        = 50

[mysqldump]
quick
max_allowed_packet              = 16M

[mysql]
no-auto-rehash

[myisamchk]
key_buffer_size                 = 8M
sort_buffer_size                = 8M

[mysqlhotcopy]
interactive-timeout
```

mybackup\backup\ backup.py

```
import os
import sys
import ConfigParser
import logging
import syslog
import getopt
import datetime
import traceback
import Task
from mybackup.common.utility import *

################################################################################
# Class for performing multiple tasks for backup or restoring a backup
################################################################################
class Backup:

    ################################################################################
    # Initialize class options
    ################################################################################
    def __init__(self):
        global G_CONFIG
        global G_SETTINGS

        # Backup task configuration common to all tasks
        G_SETTINGS["verbosity"] = ""
        G_SETTINGS["basedir"] = ""
        G_SETTINGS["backupdir"] = ""
        G_SETTINGS["datadir"] = ""
        G_SETTINGS["targetdir"] = ""

        # Task to run, ordered dict would be nice but not available in Python 2.4
        self.tasknames = list()
        self.tasks = dict()

        # If no config file is given, use the default one
        if G_SETTINGS["configfile"] == "":
            G_SETTINGS["configfile"] = os.path.dirname(__file__) +
"/../../config/backup.cfg"

    ################################################################################
    # Load options from the config file
    ################################################################################
```

```
    def prepare(self):
        global G_CONFIG
        global G_SETTINGS

        try:
            # Make sure we have a task to process
            if G_SETTINGS["task"] == "":
                raise Error("No backup task name given, aborting backup")

            # Make sure the config file exists
            if os.path.isfile(G_SETTINGS["configfile"]) == False:
                raise Error("The config file %s doesn't exist, aborting backup")

            # Load the configuration file
            G_CONFIG.read(G_SETTINGS["configfile"])

            # Check that a section for the task exists in the config file.
            if G_CONFIG.has_section(G_SETTINGS["task"]) == False:
                raise Error("No task named %s in the config file %s" %
(G_SETTINGS["task"], G_SETTINGS["configfile"]))

            # Check and set the logging verbosity level and backup type
            G_SETTINGS["loglevel"] = get_param_enum(G_SETTINGS["task"], "loglevel",
G_LOGLEVELS)
            G_SETTINGS["type"] = get_param_enum(G_SETTINGS["task"], "type", G_TASKTYPES)

            # Create and setup the backupdir and targetdir
            self.setup_directories()

            # Setup logging
            self.setup_logging()
            logging.info("Starting backup task %s" % G_SETTINGS["task"])

            # Load the backup sub tasks
            self.tasknames = get_param_list(G_SETTINGS["task"], "tasks", False)

            for task in self.tasknames:
                if G_CONFIG.has_section(task) == False:
                    raise Error("No backup subtask named %s in the backup.cfg file" %
task)

                # Create a new insta
                tasktype = get_param(task, "type").lower()
                if tasktype == "checksum":
                    self.tasks[task] = Task.TaskChecksum(task)
                elif tasktype == "local":
                    self.tasks[task] = Task.TaskLocalBackup(task)
                elif tasktype == "stream":
                    self.tasks[task] = Task.TaskStreamBackup(task)
                elif tasktype == "remote":
                    self.tasks[task] = Task.TaskRemoteBackup(task)
                elif tasktype == "incremental":
                    self.tasks[task] = Task.TaskIncrementalBackup(task)
                else:
                    raise Error("Unknown backup type for task %s " % task)

        except ConfigParser.Error, msg:
            raise Error("Failed parsing config file %s , error message: %s " %
(G_SETTINGS["configfile"], msg))

    ##########################################################################
    # Create and setup the backup information and target directory
    ##########################################################################
    def setup_directories(self):
        global G_CONFIG
        global G_SETTINGS
```

```
        # Set the base directory for backups
        G_SETTINGS["basedir"] = get_param_dir(G_SETTINGS["task"], "basedir", True)

        # The backup is a full backup
        if G_SETTINGS["type"] == "full":

            # Try to create the backup basedir if it doesn't exist
            if os.path.exists(G_SETTINGS["basedir"]) == False:
                try:
                    os.makedirs(G_SETTINGS["basedir"])
                except OSError, msg:
                    raise Error("Failed to create the base directory '%s': %s" %
(G_SETTINGS["basedir"], msg))

            # Setup the backup directory under basedir
            if "forcedir" in G_SETTINGS:
                G_SETTINGS["backupdir"] = add_ending_slash(G_SETTINGS["forcedir"])
            else:
                G_SETTINGS["backupdir"] = datetime.datetime.today().strftime("%Y-%m-
%d_%H-%M-%S/")

            # Make sure that the target directory doesn't exist
            if os.path.exists(get_full_backupdir()):
                raise Error("The backup directory %s already exists in the filesystem,
aborting backup." % get_full_backupdir())

            # Try to create the backup directory
            try:
                os.makedirs(get_full_backupdir())
            except OSError, msg:
                raise Error("Failed to create backup directory %s : %s " %
(get_full_backupdir(), msg))

            # Set the target directory
            G_SETTINGS["targetdir"] = "data/"

        # Incremental backup
        else:
            # Make sure that the path in the config file exists
            if os.path.exists(G_SETTINGS["basedir"]) == False:
                raise Error("The backup base directory %s doesn't exist, aborting" %
G_SETTINGS["basedir"])

            # Find the latest backup directory
            G_SETTINGS["backupdir"] = find_latest_dir(G_SETTINGS["basedir"], list())
            if G_SETTINGS["backupdir"] == "":
                raise Error("No backup directories were found under %s" %
G_SETTINGS["basedir"])

            G_SETTINGS["backupdir"] = add_ending_slash(G_SETTINGS["backupdir"])

            if os.path.isdir(get_full_backupdir() + "data") == False:
                raise Error("No data directory found at backup directory %s " %
get_full_backupdir())

            # The datadir and targetdir are set in the constructor of
TaskIncrementalBackup
            G_SETTINGS["datadir"] = ""
            G_SETTINGS["targetdir"] = ""

    ################################################################################
    # setup logging,
    ################################################################################
    def setup_logging(self):
        global G_CONFIG
        global G_SETTINGS
```

```
        logging.basicConfig(
            level = G_LOGLEVELS[G_SETTINGS["loglevel"]],
            format = '%(asctime)s %(name)-12s %(levelname)-8s %(message)s',
            datefmt = '%Y-%m-%d %H:%M',
            filename = get_full_backupdir() + "mybackup.log")

        # Enable logging to stdout if the verbose option has been given as commandline
paramater
        if G_VERBOSE:
            console = logging.StreamHandler()
            console.setLevel(G_LOGLEVELS[G_SETTINGS["loglevel"]])
            formatter = logging.Formatter('%(name)-12s: %(levelname)-8s %(message)s')
            console.setFormatter(formatter)
            logging.getLogger('').addHandler(console)

    ###########################################################################
    # Executes all the subtasks loaded from the configuration file for the backup
    ###########################################################################
    def execute(self):
        for task in self.tasknames:
            logging.info("Starting backup subtask %s" % task)
            self.tasks[task].execute()
            logging.info("Finished backup subtask %s" % task)

###############################################################################
# Shows a help message for configuration options.
###############################################################################
def help_message():
    print "This script takes a backup from the mysql database. It reads configuration
data "
    print "from config/backup.cfg file. If the program is run without any parameters,
the "
    print "settings from [default] group are used. For more options about configuring
the "
    print "backup settings read the documentation."

    print "Usage: mybackup.py [OPTIONS]..."
    print ""
    print "-c   --config    Full path to the configuration file from which to load the
information."
    print "                 If not given, the default is to use the config file at
config/backup.cfg"
    print "-f   --forcedir  Force backupdir to be the one"
    print "-t   --task      Name of the backup task to run"
    print "-q   --quiet     Quiet, do not output progress data into stdout"
    print "-h   --help      Displays this help and exit"
    print ""

###############################################################################
# Entry point called from the bin/backup.py script
###############################################################################
def run():
    global G_VERBOSE
    global G_SETTINGS

    # Show help if we don't have any arguments except program name
    if len(sys.argv) < 2:
        print >>sys.stderr, "ERROR: The backup task name must be given"
        print ""
        help_message()
        return 0

    try:
        # Open syslog for doing the minimun logging for backups, more verbose logs are
        # written under the backup directory.
        syslog.openlog("mybackup")
        syslog.syslog("Starting mybackup with arguments: %s " % " ".join(sys.argv))
```

```
        # Parse the commandline options
        try:
            opts, args = getopt.getopt(sys.argv[1:], "t:c:f:hv", ["task=", "config=",
"force=", "help", "verbose"])
        except getopt.error, msg:
            raise Usage("Failed to parse options, %s" % msg)

        # Make sure that these are set in the global settings config always
        G_SETTINGS["task"] = ""
        G_SETTINGS["configfile"] = ""

        # Process the commandline options
        for opt, arg in opts:
            if opt in ("-q", "--quiet"):
                G_VERBOSE = False
            elif opt in ("-c", "--config"):
                G_SETTINGS["configfile"] = arg.strip()
            elif opt in ("-t", "--task"):
                G_SETTINGS["task"] = arg.strip().lower()
            elif opt in ("-f", "--force"):
                G_SETTINGS["forcedir"] = arg.strip().lower()
            elif opt in ("-h", "--help"):
                help_message()
                return 0
            else:
                raise Usage("Unknown option %s" % opt)

        # Start the backup process
        backup = Backup()
        backup.prepare()
        backup.execute()
        syslog.syslog("Finished mybackup task %s" % G_SETTINGS["task"])

    except Usage, err:
        syslog.syslog("Error: %s" % err.msg)
        logging.critical(err.msg)
        return 3

    except Error, err:
        syslog.syslog("Error: %s" % err.msg)
        logging.critical(err.msg)
        return 2

    except Exception:
        syslog.syslog("Unknown error:\n\n%s" % traceback.format_exc())
        logging.critical("Unknown error:\n\n%s" % traceback.format_exc())
        return 1

    syslog.syslog("Backup process finished succesfully")
    return 0
```

mybackup\backup\ task.py

```
import os
import re
import sys
import logging
import subprocess
import traceback
from mybackup.common.utility import *

################################################################################
# Base class for backup tasks
```

```
##############################################################################
class Task(object):

    ##########################################################################
    # Constructor the base task
    ##########################################################################
    def __init__(self, name, requireDB):
        global G_CONFIG
        global G_SETTINGS

        self.name = name
        self.db = DatabaseConfig()
        self.info = ConfigParser.ConfigParser()

        # Load previous restore information if it exists
        filename = get_full_backupdir() + "mybackup_restore"
        if os.path.isfile(filename):
            self.info.read(filename)

        # Load database config if it has been specified for the task
        if G_CONFIG.has_option(self.name, "dbconfig"):
            dbconfig = get_param(self.name, "dbconfig")
            self.db.initialize(
                get_param(dbconfig, "host"), get_param_int(dbconfig, "port"),
                get_param(dbconfig, "user"), get_param(dbconfig, "password"))

        if requireDB and self.db.is_initialized() == False:
            raise Error("The database connection information must be specified for task
%s" % self.name)

    ##########################################################################
    # All tasks have a execute function
    ##########################################################################
    def execute(self):
        raise Error("Can't call baseclass Task execute method")

    ##########################################################################
    # Write the checkpoint file
    ##########################################################################
    def write_checkpoint(self, logstart, logend):
        global G_CONFIG
        global G_SETTINGS

        filename = get_full_targetdir() + "xtrabackup_checkpoints"
        logging.info("Writing xtrabackup_checkpoints file to %s " % filename)

        try:
            out = open(filename, "w")
            out.write("backup_type = full-backuped\n")
            out.write("from_lsn = %s\n" % logstart)
            out.write("to_lsn = %s\n" % logend)
            out.close()
        except Exception:
            raise Error("Failed to write xtrabackup_checkpoints file %s\n\n%s" %
(filename, traceback.format_exc()))

    ##########################################################################
    # Write restore configuration information
    ##########################################################################
    def write_info(self):
        global G_CONFIG
        global G_SETTINGS

        filename = get_full_backupdir() + "mybackup_restore"
        logging.info("Writing restore information to %s " % filename)

        try:
```

```
            out = open(filename, "w")
            self.info.write(out)
            out.close()
        except Exception:
            raise Error("Failed to write mybackup_restore file %s\n\n%s" % (filename,
traceback.format_exc()))

##############################################################################
# Task for doing a checksum
##############################################################################
class TaskChecksum(Task):

    ##########################################################################
    # Constructor for the checksum task
    ##########################################################################
    def __init__(self, name):
        global G_CONFIG
        global G_SETTINGS

        # Call the base class constructor
        Task.__init__(self, name, True)

        # Checksum specific options
        self.databases = get_param(self.name, "databases")
        self.ignoredatabases = get_param(self.name, "ignore-databases")
        self.tables = get_param(self.name, "tables")
        self.ignoretables = get_param(self.name, "ignore-tables")
        self.filename = get_full_targetdir() + get_param(self.name, "filename")

        # Commandline for the task
        self.cmdline = "mk-table-checksum h=%s,P=%s,u=%s,p=%s " % (
            self.db.get_host(), self.db.get_port(), self.db.get_user(),
self.db.get_password())

        if self.databases != "":
            self.cmdline += "--databases=%s " % self.databases
        if self.ignoredatabases != "":
            self.cmdline += "--ignore-databases=%s " % self.ignoredatabases
        if self.tables != "":
            self.cmdline += "--tables=%s " % self.tables
        if self.ignoretables != "":
            self.cmdline += "--ignore-tables=%s " % self.ignoretables

        self.cmdline += "> %s" % self.filename

    ##########################################################################
    # Execute the checksum task
    ##########################################################################
    def execute(self):
        global G_CONFIG
        global G_SETTINGS

        try:
            logging.info("Running checksum task with arguments: %s" % self.cmdline)

            # Run the checksum task and write the stderr output to log
            p = subprocess.Popen(self.cmdline, shell=True, stderr=subprocess.PIPE)

            for line in p.communicate()[1].split("\n"):
                if line.strip() != "":
                    logging.error(line)

            # Check that the command was run succesfully
            if p.returncode != 0:
                raise Error("Error running checksum task %s, returncode was: %s" %
(self.name, p.returncode))
```

```
            # Read and validate the output from checksum
            out = open(self.filename, "r")
            data = out.readlines()
            if len(data) <= 0:
                raise Error("The output of checksum task %s didn't contain any data" %
self.name)
            line = data[0].split()
            if line[0] != "DATABASE":
                raise Error("The output of checksum task %s didn't start with DATABASE"
% self.name)
            out.close()

        except Error:
            raise

        except Exception:
            raise Error("Failed to run maatkit checksum task %s:\n\n%s" % (self.name,
traceback.format_exc()))

################################################################################
# Task for doing a local backup into a directory
################################################################################
class TaskLocalBackup(Task):

    ################################################################################
    # Constructor for the local backup task
    ################################################################################
    def __init__(self, name):
        global G_CONFIG
        global G_SETTINGS

        # Initialize common options for tasks
        Task.__init__(self, name, False)

        # Local backup specific options
        self.defaultsfile = get_param(self.name, "defaults-file")

        # Commandline for backup
        self.cmdline = "innobackupex-1.5.1 --no-timestamp "
        if self.defaultsfile != "":
            self.cmdline += "--defaults-file=%s " % self.defaultsfile
        self.cmdline += get_full_targetdir() + " 2>&1"

        # Info for restoration
        self.info.add_section("restore")
        self.info.set("restore", "type", "local")
        self.info.set("restore", "targetdir", G_SETTINGS["targetdir"])
        self.info.set("restore", "incremental", 0)

    ################################################################################
    # Execute the local backup task
    ################################################################################
    def execute(self):
        global G_CONFIG
        global G_SETTINGS

        try:
            logging.info("Running localbackup task with arguments: %s" % (self.cmdline))

            # Run the process
            p = subprocess.Popen(self.cmdline, shell=True, stdout=subprocess.PIPE)

            # Log the output from stdout and stderr to log file
            for line in p.communicate()[0].split("\n"):
                logging.debug(line)

            # Validate the returncode from the process
```

```
            if p.returncode != 0:
                raise Error("Error running localbackup task %s, returncode was: %s" %
(self.name, p.returncode))

            # Write information needed for restoring the backup
            self.write_info()

        except Error:
            raise

        except Exception:
            raise Error("Unexceptect exception while running localbackup task %s:\n\n%s"
% (self.name, traceback.format_exc()))

################################################################################
# Task for doing a a local stream backup
################################################################################
class TaskStreamBackup(Task):

    ############################################################################
    # Constructor for the local stream backup
    ############################################################################
    def __init__(self, name):
        global G_CONFIG
        global G_SETTINGS

        # Initialize common options for tasks
        Task.__init__(self, name, False)

        # Stream backup specific options
        self.defaultsfile = get_param(self.name, "defaults-file")
        self.compression = get_param_int(self.name, "compression")
        self.filename = get_param(self.name, "filename")

        # Commandline for backup
        self.cmdline = "innobackupex-1.5.1 --no-timestamp --stream=tar "
        if self.defaultsfile != "":
            self.cmdline += "--defaults-file=%s " % self.defaultsfile
        self.cmdline += "./ "
        if self.compression != 0:
            self.cmdline += "| gzip -%s " % str(self.compression)
        self.cmdline += "> %s%s" % (get_full_targetdir(), self.filename)

        # Construct the info for restore
        self.info.add_section("restore")
        self.info.set("restore", "type", "stream")
        self.info.set("restore", "targetdir", G_SETTINGS["targetdir"])
        self.info.set("restore", "filename", self.filename)
        self.info.set("restore", "defaults-file", self.defaultsfile)
        self.info.set("restore", "compression", self.compression)
        self.info.set("restore", "incremental", 0)

    ############################################################################
    # Execute the local stream backup
    ############################################################################
    def execute(self):
        global G_CONFIG
        global G_SETTINGS

        try:
            logging.info("Running stream backup task with arguments: %s" % self.cmdline)

            os.mkdir(get_full_targetdir())
            p = subprocess.Popen(self.cmdline, shell=True, stderr=subprocess.PIPE)

            logposition = -1
            for line in p.communicate()[1].split("\n"):
```

```
                logging.debug(line)
                match = re.match(r"xtrabackup: The latest check point \(for
incremental\): '(\d+)'", line)
                if match:
                    logposition = match.group(1)
                    logging.info("Found the check point position for incremental
backups: %s" % logposition)

            if p.returncode != 0:
                raise Error("Error running streambackup task %s, returncode was: %s" %
(self.name, p.returncode))

            if logposition == -1:
                raise Error("No log position found for streambackup task %s, aborting."
% self.name)

            self.write_checkpoint(0, logposition)
            self.write_info()

        except Error:
            raise

        except Exception:
            raise Error("Failed to run stream backup task %s:\n\n%s" % (self.name,
traceback.format_exc()))

################################################################################
# Task for doing a remote stream backup
################################################################################
class TaskRemoteBackup(Task):

    ################################################################################
    # Constructor for the remote backup task
    ################################################################################
    def __init__(self, name):
        global G_CONFIG
        global G_SETTINGS

        # Initialize the base class
        Task.__init__(self, name, False)

        # Additional configuration for stream backups
        self.defaultsfile = get_param(self.name, "defaults-file")
        self.compression = get_param_int(self.name, "compression")
        self.filename = get_param(self.name, "filename")
        self.throttle = get_param(self.name, "throttle")
        self.connection = get_param(self.name, "connection")

        self.cmdline = "innobackupex-1.5.1 --no-timestamp --stream=tar"
        if self.defaultsfile != "":
            self.cmdline += (" --defaults-file=%s " % self.defaultsfile)
        self.cmdline += " ./ "
        if self.compression != 0:
            self.cmdline += ("| gzip -%s " % str(self.compression))
        if self.throttle != "0":
            self.cmdline += ("| pv -q -L%s " % self.throttle)
        self.cmdline += ("| %s 'cat - > %s'" % (self.connection, self.filename))

        # Info for restore
        self.info.add_section("restore")
        self.info.set("restore", "type", "remote")
        self.info.set("restore", "targetdir", G_SETTINGS["targetdir"])
        self.info.set("restore", "filename", self.filename)
        self.info.set("restore", "defaults-file", self.defaultsfile)
        self.info.set("restore", "compression", self.compression)
        self.info.set("restore", "connection", self.connection)
        self.info.set("restore", "incremental", 0)
```

```
    ###########################################################################
    # Execute the backup
    ###########################################################################
    def execute(self):
        global G_CONFIG
        global G_SETTINGS

        try:
            logging.info("Running stream backup task with arguments: %s" % self.cmdline)

            os.mkdir(get_full_targetdir())
            p = subprocess.Popen(self.cmdline, shell=True, stderr=subprocess.PIPE)

            logposition = -1
            for line in p.communicate()[1].split("\n"):
                logging.debug(line)
                match = re.match(r"xtrabackup: The latest check point \(for
incremental\): '(\d+)'", line)
                if match:
                    logposition = match.group(1)
                    logging.info("Found the check point position for incremental
backups: %s" % logposition)

            if p.returncode != 0:
                raise Error("Error running streambackup task %s, returncode was: %s" %
(self.name, p.returncode))

            if logposition == -1:
                raise Error("No log position found for streambackup task %s, aborting."
% self.name)

            self.write_checkpoint(0, logposition)
            self.write_info()

        except Error:
            raise

        except Exception:
            raise Error("Failed to run stream backup task %s:\n\n%s" % (self.name,
traceback.format_exc()))


###############################################################################
# Task for doing an incremental backup
###############################################################################
class TaskIncrementalBackup(Task):

    ###########################################################################
    # Constructor
    ###########################################################################
    def __init__(self, name):
        global G_CONFIG
        global G_SETTINGS

        # Call the base constructor
        Task.__init__(self, name, False)

        # Initialize and validate the source and destination directories for backup
        G_SETTINGS["datadir"] = get_param("restore", "targetdir", self.info)
        G_SETTINGS["targetdir"] = datetime.datetime.today().strftime("%Y-%m-%d_%H-%M-
%S/")

        if os.path.exists(get_full_datadir()) == False:
            raise Error("The source directory %s for incremental backup doesn't exist,
aborting." % get_full_datadir())
```

```
        if os.path.exists(G_SETTINGS["targetdir"]) == True:
            raise Error("The incremental backup target directory %s exists, aborting." %
get_full_targetdir())

        # Additional configuration for stream backups
        self.defaultsfile = get_param(self.name, "defaults-file")

        # Command line for doing incremental backup
        self.cmdline = "xtrabackup --backup --incremental-basedir=%s --target-dir=%s " %
(strip_ending_slash(get_full_datadir()), strip_ending_slash(get_full_targetdir()))
        if self.defaultsfile != "":
            self.cmdline += "--defaults-file=%s " % self.defaultsfile
        self.cmdline += "2>&1"

        # Info for restore
        incremental = get_param_int("restore", "incremental", self.info)
        section = "incremental" + str(incremental)

        self.info.set("restore", "incremental", incremental + 1)
        self.info.add_section(section)
        self.info.set(section, "targetdir", G_SETTINGS["targetdir"])


    ##############################################################################
    # Execute the incremental backup task
    ##############################################################################
    def execute(self):
        try:
            logging.info("Running incremental backup task: %s" % self.cmdline)

            p = subprocess.Popen(self.cmdline, shell=True, stdout=subprocess.PIPE)

            for line in p.communicate()[0].split("\n"):
                logging.debug(line)

            if p.returncode != 0:
                raise Error("Error running incrementalbackup task %s, returncode was:
%s" % (self.name, p.returncode))

            self.write_info()

        except Error:
            raise

        except Exception:
            raise Error("Failed to run incremental backup task %s:\n\n%s" % (self.name,
traceback.format_exc()))
```

mybackup\common\ utility.py

```
import os
import logging
import datetime
import ConfigParser

##############################################################################
# Logging settings
##############################################################################
G_VERBOSE = True
G_LOGLEVELS = {
    "debug":    logging.DEBUG,
    "info":     logging.INFO,
    "warning":  logging.WARNING,
    "error":    logging.ERROR,
```

```
    "critical": logging.CRITICAL
    }

################################################################################
# Backup configuration file and settings
################################################################################
G_CONFIG = ConfigParser.ConfigParser()
G_SETTINGS = dict()
G_TASKTYPES = [ "full", "incremental" ]
G_BACKUPTYPES = [ "local", "stream", "remote" ]

################################################################################
# Invalid commandline arguments
################################################################################
class Usage(Exception):
    def __init__(self, msg):
        self.msg = msg

################################################################################
# General error class
################################################################################
class Error(Exception):
    def __init__(self, msg):
        self.msg = msg

################################################################################
# Show a confirmation promp
################################################################################
def confirm(prompt, defaultAnswer):
    while True:
        ans = raw_input(prompt)
        if not ans:
            return defaultAnswer
        if ans not in ['y', 'Y', 'n', 'N']:
            print 'please enter y or n.'
            continue
        if ans == 'y' or ans == 'Y':
            return True
        if ans == 'n' or ans == 'N':
            return False

################################################################################
# Gets a variable from config file
################################################################################
def get_param(section, name, config=None):
    global G_CONFIG

    if config == None:
        param = G_CONFIG.get(section, name).strip()
    else:
        param = config.get(section, name).strip()

    return replace_param(param)

################################################################################
# Gets a variable from config file which must be one of the ones given
################################################################################
def get_param_enum(section, name, enums, config=None):
    param = get_param(section, name, config).lower()
    if param not in enums:
        raise Error("The parameter %s at section %s was not one of the following params:
%s " % (name, section, " ".join(enums)))
    return param

################################################################################
# Returns a list separated by "," character
################################################################################
```

```
def get_param_list(section, name, allowEmpty, config=None):
    params = get_param(section, name, config).split(",")
    if allowEmpty == False and len(params) == 0:
        raise Error("Empty set of parameters found for option %s at section %s" %
(section, name))
    return params

################################################################################
# Returns an int param
################################################################################
def get_param_int(section, name, config=None):
    global G_CONFIG

    if config == None:
        return G_CONFIG.getint(section, name)
    else:
        return config.getint(section, name)

################################################################################
# Returns a directory path ending in /
################################################################################
def get_param_dir(section, name, absolute, config=None):
    param = get_param(section, name, config)

    if absolute == True and os.path.isabs(param) == False:
        raise Error("The path %s was not absolute for config param %s : %s" % (param,
section, name))

    if absolute == False and os.path.isabs(param) == True:
        raise Error("The path %s was absolute for config param %s : %s" % (param,
section, name))

    return add_ending_slash(param)

################################################################################
# Replace a parameter with value
################################################################################
def replace_param(param):
    global G_SETTINGS

    param = param.replace("%date%", datetime.datetime.today().strftime("%Y-%m-%d"))
    param = param.replace("%time%", datetime.datetime.today().strftime("%H-%M-%S"))
    param = param.replace("%datetime%", datetime.datetime.today().strftime("%Y-%m-%d_%H-
%M-%S"))

    # If the parameters have been read from config file, this can be used too
    if "basedir" in G_SETTINGS:
        param = param.replace("%basedir%", G_SETTINGS["basedir"])
    if "backupdir" in G_SETTINGS:
        param = param.replace("%backupdir%", G_SETTINGS["backupdir"])
    if "datadir" in G_SETTINGS:
        param = param.replace("%datadir%", G_SETTINGS["datadir"])

    return param


################################################################################
# Returns the full path to backupdir
################################################################################
def get_full_backupdir():
    global G_SETTINGS
    return G_SETTINGS["basedir"] + G_SETTINGS["backupdir"]

################################################################################
# Returns the full path to targetdir
################################################################################
def get_full_targetdir():
```

```
    global G_SETTINGS
    return G_SETTINGS["basedir"] + G_SETTINGS["backupdir"] + G_SETTINGS["targetdir"]

################################################################################
# Returns the full path to datadir
################################################################################
def get_full_datadir():
    global G_SETTINGS
    return G_SETTINGS["basedir"] + G_SETTINGS["backupdir"] + G_SETTINGS["datadir"]

################################################################################
# Find the latest file in a directory, optionally directories can be excluded
################################################################################
def find_latest_dir(path, exclude):
    dirs = list()

    for name in os.listdir(path):
        if name not in exclude and os.path.isdir(path + name):
            dirs.append(name)

    if len(dirs) == 0:
        return ""

    dirs.sort(reverse=True)

    return dirs[0]

################################################################################
# Strips the ending slash from path
################################################################################
def strip_ending_slash(path):
    if path[-1] == "/":
        return path[:-1]
    return path

################################################################################
# Adds an slash to the end of path if it doesn't have one
################################################################################
def add_ending_slash(path):
    if path[-1] != "/":
        return path + "/"
    return path

################################################################################
# Database connection settings
################################################################################
class DatabaseConfig:
    def __init__(self):
        self.host = ""
        self.port = ""
        self.user = ""
        self.password = ""
        self.initialized = False

    def initialize(self, host, port, user, password):
        self.host = host
        self.port = port
        self.user = user
        self.password = password
        self.initialized = True

    def get_host(self):         return self.host
    def get_port(self):         return self.port
    def get_user(self):         return self.user
    def get_password(self):     return self.password
    def is_initialized(self):   return self.initialized
```

mybackup\restore\restore.py

```
import os
import sys
import ConfigParser
import logging
import syslog
import getopt
import datetime
import subprocess
import traceback
import pwd
import grp
from mybackup.common.utility import *

################################################################################
# Writes message to stdout if quiet option hasn't been specified
################################################################################
def log(msg):
    global G_VERBOSE
    if G_VERBOSE:
        print ">>> " + msg
    syslog.syslog(msg)

################################################################################
# Class for performing multiple tasks for backup or restoring a backup
################################################################################
class Restore:

    ############################################################################
    # Constructor
    ############################################################################
    def __init__(self):
        pass


    ############################################################################
    # Prepare and validate all required parameters
    ############################################################################
    def execute(self):
        try:
            # Validate source and destination directories
            if G_SETTINGS["dstdir"] == "":
                raise Error("The backup destination directory must be given")
            if G_SETTINGS["srcdir"] == "":
                raise Error("The backup source directory must be given")

            # Make sure the src directory exists and the destination directory doesn't
            if os.path.isdir(G_SETTINGS["srcdir"]) == False:
                raise Error("The backup source directory %s doesn't exist, aborting
restore." % G_SETTINGS["srcdir"])

            if os.path.exists(G_SETTINGS["dstdir"]) == True:
                raise Error("The backup destination directory %s exists, aborting
restore." % G_SETTINGS["dstdir"])

            G_SETTINGS["srcdir"] = add_ending_slash(G_SETTINGS["srcdir"])
            G_SETTINGS["dstdir"] = add_ending_slash(G_SETTINGS["dstdir"])
            G_SETTINGS["configfile"] = G_SETTINGS["srcdir"] + "mybackup_restore"

            if os.path.exists(G_SETTINGS["configfile"]) == False:
                raise Error("The backup source directory %s doesn't contain
mybackup_restore file, aborting restore." % (G_SETTINGS["srcdir"] + "mybackup_restore"))

            G_CONFIG.read(G_SETTINGS["configfile"])
```

```
            if G_CONFIG.has_section("restore") == False:
                raise Error("The %s doesn't have restore section, aborting restore" %
G_SETTINGS["configfile"])

            G_SETTINGS["targetdir"] = get_param("restore", "targetdir")
            G_SETTINGS["type"] = get_param_enum("restore", "type", G_BACKUPTYPES)
            G_SETTINGS["incremental"] = get_param_int("restore", "incremental")

            # Validate the group and user settings
            if G_SETTINGS["user"] == "" or G_SETTINGS["group"] == "":
                raise Error("User or group argument not given, aborting restore")

            try:
                pwd.getpwnam(G_SETTINGS["user"])
                grp.getgrnam(G_SETTINGS["group"])
            except KeyError, err:
                raise Error("No user/group found matching %s:%s, check the given user
and group" % (G_SETTINGS["user"], G_SETTINGS["group"]))

            # Restore the backup to restore directory
            if G_SETTINGS["type"] == "local":
                self.restore_local()
            elif G_SETTINGS["type"] == "stream":
                self.restore_stream()
            elif G_SETTINGS["type"] == "remote":
                self.restore_remote()
            else:
                raise Error("Unknown backup type %s, unable to continue" %
G_SETTINGS["type"])

            # Apply binary logs and incremental backups
            self.apply_logs()
            self.apply_incremental_backups()
            self.chown_restoredir()

        except Error:
            raise

        except ConfigParser.Error, err:
            raise Error("Failed to read configuration from mybackup_restore file %s :
%s" % (G_SETTINGS["configfile"], err))

        except Exception:
            raise Error("Unknown error while trying to restore backup\n\n%s" %
traceback.format_exc())


    ################################################################################
    # Restore a local backup
    ################################################################################
    def restore_local(self):
        global G_CONFIG
        global G_SETTINGS

        cmdline = "cp -r %s %s" % (G_SETTINGS["srcdir"] + G_SETTINGS["targetdir"],
G_SETTINGS["dstdir"])

        try:
            log("Restoring local backup: %s" % cmdline)
            if subprocess.call(cmdline, shell=True) != 0:
                raise Error("Failed to copy from %s to %s" % (G_SETTINGS["srcdir"],
G_SETTINGS["dstdir"]))

        except Error:
            raise

        except Exception:
```

```
            raise Error("Unknown exception when restoring backup:\n\n%s" %
traceback.format_exc())

    ###########################################################################
    # Restore a stream backup
    ###########################################################################
    def restore_stream(self):
        global G_CONFIG
        global G_SETTINGS

        compression = get_param_int("restore", "compression")
        filename = G_SETTINGS["srcdir"] + G_SETTINGS["targetdir"] + get_param("restore",
"filename")

        cmdline = "tar -xvzf "
        if compression != 0:
            cmdline = "tar -xvzif "
        cmdline += "%s -C %s" % (filename, G_SETTINGS["dstdir"])

        try:
            log("Restoring stream backup: %s" % cmdline)
            os.mkdir(G_SETTINGS["dstdir"])
            if subprocess.call(cmdline, shell=True) != 0:
                raise Error("Failed to restore stream archive %s to %s" % (filename,
G_SETTINGS["dstdir"]))

        except Error:
            raise

        except Exception:
            raise Error("Unknown exception when restoring stream backup\n\n%s" %
traceback.format_exc())

    ###########################################################################
    # Restore a remote backup
    #  ssh root@localhost -i /root/.ssh/id_rsa 'cat /root/backup/mybackup_2010-09-19_15-
38-53.tgz' | gunzip > tmp.tar
    ###########################################################################
    def restore_remote(self):
        global G_CONFIG
        global G_SETTINGS

        connection = get_param("restore", "connection")
        compression = get_param_int("restore", "compression")
        filename = get_param("restore", "filename")

        cmdline = "%s 'cat %s' " % (connection, filename)
        if compression != 0:
            cmdline += "| tar xvzi "
        else:
            cmdlind += "| tar xvi "
        cmdline += "-C %s" % G_SETTINGS["dstdir"]

        try:
            log("Restoring remote backup: %s" % cmdline)
            os.mkdir(G_SETTINGS["dstdir"])
            if subprocess.call(cmdline, Shell=True) != 0:
                raise Error("Failed to restore remote archive %s to %s" % (filename,
G_SETTINGS["dstdir"]))
        except Error:
            raise

        except Exception:
            raise Error("Unknown exception when restoring remote backup\n\n%s" %
traceback.format_exc())

    ###########################################################################
```

```
    # Apply log
    ###############################################################################
    def apply_logs(self):
        global G_SETTINGS

        cmdline = "innobackupex-1.5.1 --apply-log %s " % G_SETTINGS["dstdir"]
        if G_SETTINGS["memory"] != "":
            cmdline += "--use-memory=%s " % G_SETTINGS["memory"]
        log("Applying logs: %s" % cmdline)
        if subprocess.call(cmdline, shell=True) != 0:
            raise Error("Failed to apply logs to %s" % (G_SETTINGS["dstdir"]))

    ###############################################################################
    # Apply incremental backups
    ###############################################################################
    def apply_incremental_backups(self):
        global G_SETTINGS

        incremental = G_SETTINGS["incremental"]
        for i in range(0, incremental):
            section = "incremental" + str(i)
            incrementaldir = G_SETTINGS["srcdir"] + get_param(section, "targetdir")
            cmdline = "xtrabackup --prepare --target-dir=%s --incremental-dir=%s" %
(strip_ending_slash(G_SETTINGS["dstdir"]), strip_ending_slash(incrementaldir))

            log("MYRESTORE: Applying incremental backup: %s" % cmdline)

            # Run the command and validate the returncode
            if subprocess.call(cmdline, shell=True) != 0:
                raise Error("Failed to apply logs to %s" % (G_SETTINGS["dstdir"]))

    ###############################################################################
    # Change owner and user
    ###############################################################################
    def chown_restoredir(self):
        global G_SETTINGS

        cmdline = "chown -R %s:%s %s" % (G_SETTINGS["user"], G_SETTINGS["group"],
G_SETTINGS["dstdir"])
        log("Changing the user and group for the restore directory: %s" % cmdline)
        if subprocess.call(cmdline, shell=True) != 0:
            raise Error("Failed to change the ownership of directory tree %s" %
cfg_basedir)


###############################################################################
# Shows a help message for configuration options.
###############################################################################
def help_message():
    print "This script restores a backup into the given directory. "
    print ""
    print "Usage: myrestory.py [OPTIONS]..."
    print ""
    print "-d   --dstdir    Full path to the directory where mysql is to be restored"
    print "-s   --srcdir    Full path to the backup directory which should be restored"
    print "-u   --user      User to chown the restored backup dir (mysql or mysqld,
depends on OS)"
    print "-g   --group     Group to chown the restored backup dir (mysql or mysqld,
depends on OS)"
    print "-m   --memory    Define the amount of memory to allocate when applying logs."
    print "                 80% of available memory at server can be used for speeding
the apply logs process"
    print "-q   --quiet     Quiet, no not output progress data into stdout"
    print "-h   --help      Displays this help and exit"
    print ""

###############################################################################
```

```
# Entry point called from the bin/backup.py script
################################################################################
def run():
    global G_VERBOSE
    global G_SETTINGS

    # Initialize the global settings
    G_SETTINGS["dstdir"] = ""
    G_SETTINGS["srcdir"] = ""
    G_SETTINGS["user"] = ""
    G_SETTINGS["group"] = ""
    G_SETTINGS["memory"] = ""

    # Set the default verbosity level to be true
    G_VERBOSE = True

    # Show help if we don't have any arguments except program name
    if len(sys.argv) < 2:
        print >>sys.stderr, "ERROR: Not enough arguments"
        print ""
        help_message()
        return 0

    try:
        # Open syslog for doing the minimun logging for backups, more verbose logs are
        # written under the backup directory.
        syslog.openlog("mybackup")
        log("Starting myrestore with arguments: %s " % " ".join(sys.argv))

        # Parse the commandline options
        try:
            opts, args = getopt.getopt(sys.argv[1:], "d:s:u:g:m:hq", ["dstdir",
"srcdir", "user", "group", "memory", "help", "quiet"])
        except getopt.error, msg:
            raise Usage("Failed to parse options, %s" % msg)

        # Process the commandline options
        for opt, arg in opts:
            if opt in ("-q", "--quiet"):
                G_VERBOSE = False
            elif opt in ("-d", "--dstdir"):
                G_SETTINGS["dstdir"] = arg.strip()
            elif opt in ("-s", "--srcdir"):
                G_SETTINGS["srcdir"] = arg.strip()
            elif opt in ("-u", "--user"):
                G_SETTINGS["user"] = arg.strip()
            elif opt in ("-g", "--group"):
                G_SETTINGS["group"] = arg.strip()
            elif opt in ("-m", "--memory"):
                G_SETTINGS["memory"] = arg.strip()
            elif opt in ("-h", "--help"):
                help_message()
                return 0
            else:
                raise Usage("Unknown option %s" % opt)

        # Start the backup process
        restore = Restore()
        restore.execute()

        # We are done
        log("Finished myrestore")

    except Usage, err:
        log("Error: %s" % err.msg)
        return 3
```

```
    except Error, err:
        log("Error: %s" % err.msg)
        return 2

    except Exception:
        log("Unknown excption:\n\n%s" % traceback.format_exc())
        return 1

    return 0
```

mybackup\tests\ test_create_environment.py

```
import os
import sys
import pwd
import grp
import shutil
import getopt
import ConfigParser
import MySQLdb
from mybackup.common.utility import *

########################################################################
#
########################################################################
def helpmsg():
    print "This script creates a new test environment for testing mysql backup."
    print "The script removes all data under the basedir configuration variable set "
    print "in config.cfg and overwrites the operating system my.cnf file."
    print ""
    usage()

########################################################################
#
########################################################################
def usage():
    print "Usage: create_test_environment.py [OPTION]..."
    print ""
    print "-o   --os=[centos|debian] what operating system configuration should be read"
    print "-h   --help display this help and exit"


########################################################################
#
########################################################################
def run():
    argv = sys.argv
    try:
        try:
            # Defaults for variables
            osname = ""
            configfile = os.path.dirname(__file__) + "/../../config/test_config.cfg"
            templatefile = os.path.dirname(__file__) +
"/../../config/test_my_template.cnf"

            # Make sure that the script is run as root
            if os.getuid() != pwd.getpwnam("root")[2]:
                raise Error("This script must be run as root")

            # Parse command line options
            try:
                opts, args = getopt.getopt(argv[1:], "o:h", ["help", "os="])
            except getopt.error, msg:
                raise Usage(msg)
```

```
            for opt, arg in opts:
                if opt in ("-o", "--os"):
                    osname = arg
                elif opt in ("-h", "--help"):
                    helpmsg()
                    return 0
                else:
                    raise Usage("Unknown option %s" % opt)

            if osname == "":
                raise Usage("The operating system option (-o or --os=) must be
specfied")

            # Load the configuration file
            config = ConfigParser.SafeConfigParser()
            config.read(configfile)

            # Get the configuration variables, this causes errors if we are missing some
variable
            cfg_basedir         = config.get("common", "basedir")
            cfg_init_script     = config.get("common", "init_script")
            cfg_datadir         = cfg_basedir + "/" + config.get("common", "datadir")
            cfg_innodb_datadir  = cfg_basedir + "/" + config.get("common",
"innodb_data_home_dir")
            cfg_innodb_groupdir = cfg_basedir + "/" + config.get("common",
"innodb_log_group_home_dir")
            cfg_test_user       = config.get("common", "testuser").strip()
            cfg_test_password   = config.get("common", "testpassword").strip()

            cfg_pid     = config.get(osname, "pid")
            cfg_port    = config.get(osname, "port")
            cfg_socket  = config.get(osname, "socket")
            cfg_mycnf   = config.get(osname, "mycnf")
            cfg_user    = config.get(osname, "user")
            cfg_group   = config.get(osname, "group")

            # Confirm that the script should be run
            print "This script will delete everything under %s and install a new my.cnf
file at %s" % (cfg_basedir, cfg_mycnf)
            if not confirm("Are you sure you wish to continue? (y/N): ", False):
                print "Script finished, no data was replaced"
                return 0

            # Stop the mysql daemon
            print "Stopping mysql daemon"
            if os.system("/etc/init.d/mysql stop") != 0:
                raise Error("Failed to stop mysql daemon, exiting")

            # Read the template my.cnf file and replace variables from config file
            mydata = []
            mytemplate = open(templatefile, 'r')
            for line in mytemplate.readlines():
                line = line.replace("%pid%", cfg_pid)
                line = line.replace("%port%", cfg_port)
                line = line.replace("%socket%", cfg_socket)
                line = line.replace("%datadir%", cfg_datadir)
                line = line.replace("%innodb_data_home_dir%", cfg_innodb_datadir)
                line = line.replace("%innodb_log_group_home_dir%", cfg_innodb_groupdir)
                mydata.append(line)
            mytemplate.close()

            # Write the config file
            mycnf = open (cfg_mycnf, 'w')
            mycnf.writelines(mydata)
            mycnf.close()
```

```
            print "Wrote a my.cnf file to %s" % cfg_mycnf

            # Create the directory tree
            if os.path.exists(cfg_basedir):
                print "Removing directory tree %s" % cfg_basedir
                shutil.rmtree(cfg_basedir)

            os.mkdir(cfg_basedir)
            if not os.path.exists(cfg_datadir):
                os.mkdir(cfg_datadir)
            if not os.path.exists(cfg_innodb_datadir):
                os.mkdir(cfg_innodb_datadir)
            if not os.path.exists(cfg_innodb_groupdir):
                os.mkdir(cfg_innodb_groupdir)

            print "Created new test directory tree under: %s" % cfg_basedir

            # Install the default database
            print "Launching the mysql_install_db program"
            if os.system("mysql_install_db") != 0:
                raise Error("The mysql_install_db command returned error, exiting")

            # Change the ownership of the directory to mysql daemon
            try:
                pwd.getpwnam(cfg_user)
                grp.getgrnam(cfg_group)
            except KeyError, err:
                raise Error("No user/group found matching %s:%s, check config.cfg file"
% (cfg_user, cfg_group))

            if os.system("chown -R %s:%s %s" % (cfg_user, cfg_group, cfg_basedir)) != 0:
                raise Error("Failed to change the ownership of directory tree %s" %
cfg_basedir)

            print "Changed to ownership of the %s to %s:%s" % (cfg_basedir, cfg_user,
cfg_group)

            # Start the mysql daemon
            print "Launching the mysql daemon using %s start"
            if os.system("/etc/init.d/mysql start") != 0:
                raise Error("Launching the mysql daemon failed, exiting")

            print "Running mysql_upgrade"
            if os.system("mysql_upgrade") != 0:
                raise Error("The mysql_upgrade program returned an error, exiting")

            print "Restarting the mysql daemon for the last time"
            if os.system("/etc/init.d/mysql restart") != 0:
                raise Error("Restarting the mysql daemon failed, exiting")

            print "Adding the mybackup test user %s" % cfg_test_user

            if os.system("echo \"GRANT ALL ON *.* TO '" + cfg_test_user + "'@'localhost'
IDENTIFIED BY '" + cfg_test_password + "'; FLUSH PRIVILEGES;\" | mysql -u root") != 0:
                raise Error("Failed to add user %s" % cfg_test_user)

            print ""
            print "You have succesfully created a new test database"
            print ""

        except ConfigParser.Error, msg:
            raise Usage(msg)

    except Usage, err:
        print >>sys.stderr, "ERROR: %s" % err.msg
        print ""
        helpmsg()
```

```
        return 2

    except Error, err:
        print >>sys.stderr, err.msg
        return 1
```

mybackup\tests\ test_populate_database.py

```
import os
import sys
import getopt
import ConfigParser
import MySQLdb
import random
from mybackup.common.utility import *

################################################################################
#
################################################################################
def create_and_use_database(connection, cursor, dbname):
    cursor.execute("CREATE DATABASE %s" % dbname)
    cursor.execute("USE %s" % dbname)
    connection.commit()

################################################################################
#
################################################################################
def create_table(connection, cursor, tablename, engine):
    cursor.execute("""
        CREATE TABLE %s (
        id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
        data LONGTEXT
        ) ENGINE=%s""" % (tablename, engine))
    connection.commit()

################################################################################
#
################################################################################
def insert_row(connection, cursor, tablename, data):
    cursor.execute("INSERT INTO " + tablename + " (data) VALUES(%s)", (data,))
    connection.commit()

################################################################################
#
################################################################################
def convert_to_bytes(value):
    multiplier = 1;

    if len(value) <= 0:
        return "empty"

    if value[-1] == "k" or value[-1] == "K":
        value = value[:-1]
        multiplier = 1024
    elif value[-1] == "M" or value[-1] == "M":
        value = value[:-1]
        multiplier = 1024*1024
    elif value[-1] == "g" or value[-1] == "G":
        value = value[:-1]
        multiplier = 1024*1024*1024

    if value.isdigit() == False:
        return value
```

```
    return int(value) * int(multiplier)

#############################################################################
#
#############################################################################
def generate_data(size, words):
    data = ""
    while len(data) < size:
        data += random.choice(words) + " "
    return data

#############################################################################
#
#############################################################################
def helpmsg():
    usage()
    print ""
    tmp  = "This script creates and populates the mybackuptest with generated data. "
    tmp += "it first drops existing database if it exists and then populates it with new
data. "
    tmp += "The size of the resulting database can be estimated by the following
formula: "
    tmp += "(engines * tables * rows * size)*x, where x is a multiplier taking into
account "
    tmp += "space required for indexes etc."
    print tmp

#############################################################################
#
#############################################################################
def usage():
    print "Usage: reset_test_database [OPTIONS]..."
    print ""
    print "Use the following commandline options to override settings from configuration
file (config/test_config.cfg)"
    print ""
    print "-t   --tables=   Number of tables to create"
    print "-r   --rows=     Number of rows to generate"
    print "-s   --size=     Size of the data field in bytes: use k, M, G or none for
size multipliers (1024, 1k, 20M etc.)"
    print "-d   --db=       Name of the database to create"
    print "-e   --engine    Engine types to create test tables: myisam, innodb, both"
    print "-h   --help      display this help and exit"
    print ""

#############################################################################
#
#############################################################################
def run():
    argv = sys.argv
    try:
        try:
            # Load the configuration file
            configfile = os.path.dirname(__file__) + "/../../config/test_config.cfg"
            config = ConfigParser.ConfigParser()
            config.read(configfile)

            # Initialize defaults from the config file
            host        = config.get("db", "host")
            user        = config.get("db", "user")
            password    = config.get("db", "password")
            database    = config.get("testdb", "database");
            tables      = config.get("testdb", "tables")
            rows        = config.get("testdb", "rows")
            size        = config.get("testdb", "size")
            engine      = config.get("testdb", "engine").lower()
```

```
            # Parse command line options
            try:
                opts, args = getopt.getopt(argv[1:], "t:r:s:e:h", ["help", "tables=",
"rows=", "size=", "engine="])
            except getopt.error, msg:
                raise Usage(msg)

            for opt, arg in opts:
                if opt in ("-t", "--tables"):
                    tables = arg
                elif opt in ("-r", "--rows"):
                    rows = arg
                elif opt in ("-s", "--size"):
                    size = arg
                elif opt in ("-d", "--db"):
                    database = arg
                    return 0
                elif opt in ("-e", "--engine"):
                    engine = arg.lower()
                    return 0
                elif opt in ("-h", "--help"):
                    helpmsg()
                    return 0
                else:
                    raise Usage("Unknown option %s" % opt)

            # Validate parameters and convert to correct types
            if tables.isdigit() == False:
                raise Usage("The tables parameter (%s) was not correct" % tables)
            tables = int(tables)

            if rows.isdigit() == False:
                raise Usage("The rows parameter (%s) was not correct" % rows)
            rows = int(rows)

            size = convert_to_bytes(size)
            if str(size).isdigit() == False:
                raise Usage("The size parameter (%s) was not correct" % size)

            if engine != "myisam" and engine != "innodb" and engine != "both":
                raise Usage("The engine parameter (%s) was not correct, allowed values
are myisam, innodb or both" % engine)

            # Load the words.txt file into a set
            words = list(open(os.path.dirname(__file__) +
"/../../data/words.txt").readlines())
            words = [word.strip() for word in words]
            print len(words)

            # Open a connection to localhost
            conn = MySQLdb.connect(host, user, password)
            cursor = conn.cursor()

            # Initialize the test database
            create_and_use_database(conn, cursor, database)

            for table in range(0, tables):
                # Create and insert data into myisam tables
                if engine == "myisam" or engine == "both":
                    tablename = "tbl_myisam_" + str(table)
                    create_table(conn, cursor, tablename, "MyISAM")
                    data = generate_data(size, words)
                    for row in range(0, rows):
                        insert_row(conn, cursor, tablename, data)

                # Create and insert data into innodb tables
                if engine == "innodb" or engine == "both":
```

```
                    tablename = "tbl_innodb_" + str(table)
                    create_table(conn, cursor, tablename, "InnoDB")
                    data = generate_data(size, words)
                    for row in range(0, rows):
                        insert_row(conn, cursor, tablename, data)

            # Close  database conncetion
            cursor.close()
            conn.close()
        except ConfigParser.Error, msg:
            raise Usage(msg)

    except Usage, err:
        print >>sys.stderr, "ERROR: %s" % err.msg
        print ""
        usage()
        return 2
    except Error, err:
        print >>sys.stderr, err.msg
        return 1

def dummyfunc():
    pass

if __name__ == "__main__":
    sys.exit(main())
```