Andmeaida süsteemide korrasoleku kontrolli automatiseerimine

Lõputöö

Juhendaja/õppejõud:	Toomas Lepikult (PhD)
Kaasjuhendaja:	Igor Artemtšuk (PhD)
Üliõpilane	Aleksandra Sepp 176276IDAR
Üliõpilase meiliaadress	aleksandra.sepp@mail.ru
Õppekava nimetus	IDAR17/16 - IT süsteemide administreerimine

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aleksandra Sepp

12.05.2020

Annotatsioon

Käesoleva lõputöö eesmärgiks on kaardistada vaadeldavas andmeaidas erinevate agiilsete meeskondade haldusega tegelevaid spetsialistide sisendeid, sõnastada nende alusel tingimusi valitavale platvormile, vaadelda analoogseid lahendusi, mis olid juurutatud teistes organisatsioonides ning seejärel analüüsida 3 majasisest platvormi, valida parim lahendus ning teostada valitud platvormil tehnilist testimist. Lõpptulemusena valitud ja testitud automatiseerimise platvormi saavad edaspidi arendada, seadistada ja kasutada andmeaida halduse rolli täitvad spetsialistid, et omada tervikpilti, optimeerida enda igapäevaseid manuaalseid toiminguid ning ennetada kriitiliste olukordade tekkimist vaadeldavas andmeaidas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 6 peatükki, 30 joonist, 4 tabelit.

Abstract

Implementation of Automatic System Health Check in Data Warehouse

The aim of this diploma thesis is to collect the inputs from different agile team's maintenance specialists in the observed data warehouse, to formulate conditions for the future monitoring platform, to overview implemented similar solutions in other organizations, to perform analysis of 3 automation platforms, choose the best solution and provide a proof of concept. Chosen tested automation platform will be further developed, configured and used by data warehouse maintenance specialists to have a complete picture, optimize their daily manual operations and prevent critical situations in current data warehouse.

The thesis is in Estonian and contains 45 pages of text, 6 chapters, 30 figures, 4 tables.

Sisukord

Mõisted ja lühendid

DW, EDW (Data Warehouse, Enterprise Data Warehouse) **–** andmeait ehk mitmest erinevast allikast pärinevate integreeritud andmete ladu.

BI (Business Intelligence) – äriteave. Igasugune organisatsiooni ajalugu, jooksvat olukorda ja tulevikuperspektiive puudutav teave.

ETL (Extract-Transform-Load) **-** andmete eraldamine, teisendamine ja laadimine allikast andmeaida.

ELT (Extract-Load-Transform) – andmete eraldamine, laadimine ja teisendamine. Antud süsteem on vaadeldavas andmeaidas kasutusel.

SA (Staging Area) – andmete teisendamiskiht andmeaidas.

FTE (Full-time equivalent) – täistööaja ekvivalent, mis näitab töö tegemiseks kuluvat aega mõne kindla perioodi jooksul.

SLA (Service Level Agreement) – teenustaseme lepe, mis kirjeldab oodatava teenusekvaliteedi lepingu kehtivusaja kestel.

JRE (Java Runtime Environment) – osa Sun'i Java arenduskomplektist (JDK) ning kujutab endast programmeerimisriistu Java rakenduste kirjutamiseks. JRE koosneb Java virtuaalmasinast, tuumklassidest ja tugifailidest [1].

CI/CD (Continuous Integration (CI) and Continuous Deployment or Continuous Delivery)

ODI (Oracle Data Integrator) - platvorm ETL protsessite jaoks.

CPU (Central Processing Unit)

Tivoli (IBM Tivoli Workload Scheduler)

LDAP (Lightweight Directory Access Protocol) – lihtsustatud kataloogisirvimise protokoll, mis võimaldab peaaegu igal rakendusel ja igal arvutil ligipääsu kataloogides asuvale informatsioonile (nt. avalikud võtmed, e-posti aadressid jms.)

OSA (ODI Staging Area) – vaadeldava andmeaida kasutuses olev andmete teisendamiskiht.

JDK (Java Development Kit) - Java tarkvaraarenduskeskkond mida kasutatakse Java rakenduste arendamiseks.

Teradata SQL Assistant – rakendus, mis salvestab, hangib ja töötleb andmeid Teradata andmebaasist.

Testing Tool – käesolevas andmeaidas kasutatav rakendus tehniliste testite käivitamiseks.

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Suures andmeaidas on ülioluline kõikide süsteemide ööpäevaringne katkematu töö. Pidevate muudatuste ja süsteemide kompleksuse tõttu nende peab samuti pidevalt arenema jätkusuutlikkuse regulaarne monitoorimine. Käesoleva lõputöö raames autor otsib lahendust, kuidas optimeerida halduse rolli täitvate spetsialistide igapäevaseid toiminguid ning saada kiirelt ning konsolideeritult ülevaade süsteemi hetkeolukorrast. Lõputöö raames autor kogub sisendeid ja nõudeid tööriistvara kohta, analüüsib erinevaid platvorme, mis võimaldavad jälgida olulisi mõõdikuid vaadeldavas andmeaidas ning testib valitud platvormi sobivust. Autori arvamusel võimaldab ühtse ülevaate loomine tõsta efektiivsust ja tagada süsteemi stabiilsust.

1.1 Taust

Autor töötab tarkvara insenerina andmeaidas, täites igapäevaselt uude arenduste tehnilist testimist ning meeskonnale kuuluvate protsesside ja teenuste haldamist. Tegemist on baltimaade suurima andmeaidaga, kus vaadeldavasse osakonda kuulub rohkem kui 400 inimest.

Hiljutiste agiilsete metoodikate rakendamisest organisatsioonis on tekkinud vajadus juurutada automaatseid mõõtmissüsteeme, mis annaksid regulaarselt ülevaadet süsteemi seisundist ning kvaliteedist.

Enne agiilsete metoodikate juurutamist ööpäevaringne süsteemi monitoorimine ja kõikide ETL – protsesside[1] (ETL–Extract-Tranform-Load) [2] haldus toodangus kuulus ühele meeskonnale. Peale muudatusi protsesside halduse rollid olid jaotatud väiksemate meeskondade vahel vastavalt valdkonnale.

[1] Käesolevas andmeaidas kasutatakse ELT-meetodeid, mis erinevalt ETL omadest võimaldavad laadida kordades rohkem andmeid, toetades andmejärve süsteemi (ingl. *Data Lake*) ning transformeerimisel andmed jäävad andmeaida andmebaasidesse [13].

1.2 Probleem

Lõputöös käsitletava andmeaida agiilsetel meeskondadel puudub tänapäeval ühtne kontrolli mõõdikute automatiseerimist võimaldav platvorm, mis mõõdaks regulaarsel baasil süsteemi jätkusuutlikkust ning hoiataks ette võimalikest muredest.

1.3 Eesmärk

Töö ülesanne on teostatud analüüsi põhjal valida välja ühtne platvorm andmeaida süsteemi korrasoleku kontrolliks, mida saavad kasutada agiilsetes meeskondades halduse rolli täitjad oma igapäevases töös, et tagada süsteemi pidev töö ning mitte tegeleda tagajärgedega.

1.4 Ülevaade töö struktuurist

Lõputöö koosneb kolmest osast. Esimeses osas autor kogub sisendeid osapooltelt, sõnastab nõudeid süsteemi automatiseerimist võimaldavale platvormile ning kirjeldab hetkeolukorda.

Teises osas antakse ülevaade kahele erinevale lähenemisele, mis põhinevad kahel erineval platvormil. Seejärel viiakse läbi analüüs kolmele majasisestele platvormidele vastavalt varem sõnastatud nõuetele ning valitakse parima lahenduse, mida pakutakse halduse rolli täitvate spetsialistide igapäevaste tööülesannete optimeerimiseks ja kogu süsteemi täiustamiseks.

Lõputöö kolmandas osas autor teostab tehnilist testimist, rakendades valitud platvormil 4 tööülesannet, mis tulenevad kaardistatud vajadustest halduse rolli igapäevase töö optimeerimiseks. Seejärel tehakse testimise tulemuste põhjal järeldusi pakutud lahenduse sobivusest.

2 Metoodika

Eesmärgi saavutamiseks autor kogus sisendeid, ehk ärinõudeid, agiilsetelt meeskondadelt uuele platvormile, mis võimaldaksid teostada erinevaid monitoorimise funktsioone, sealhulgas:

- ✓ andmebaaside ja andmete teisendamiskihtide (Staging Area, SA [1]) mahu täitumise jälgimine ja ennetamine;
- ✓ virtuaalserverite logi ketaste mahtude monitoorimine;
- ✓ failiserverite mahtude täituvuse ning vaba ruumi olemasolu automaatkontroll;
- ✓ kõikide andmebaaside koormuse pidev ja konsolideeritud monitoorimine;
- ✓ protsesside jõudluse automaatvalideerimine *(ingl. process performance automated check);*
- ✓ andmete laadimine ning puudujäägid andmetes (kaardistamata vastendite tuvastamine), kus pole uusi andmeid laetud;
- ✓ ebaõnnestunud protsessid ning sagedused, trendid, dünaamika;
- ✓ täitevfunktsioonid (andmefailide arhiveerimine, andmete teisenduskihi puhastus (*ingl. staging area cleanup*) jne.

Seejärel analüüsib lõputöö autor olemasolevaid rakendusi, mis on hetkel vaadeldavas andmeiadas juba kasutusel: IBM Tivoli, Jenkins, Oracle Data Integraator (edaspidi ODI) ning vajadusel võimalikke uusi lahendusi. Tulemuseks saame lahenduse, millel saame automatiseerida tööülesandeid ning mida edaspidi ettevõttes kasutatakse ja arendatakse.

2.1 Ülevaade hetkeolukorrast

Tänapäeval olemasolev süsteem toimib ilusti ning andmeaidas ei esine kriitilisi muresid. IT – monitooringu protsessid jälgivad süsteemi tervikuna ning teavitavad vastutavaid isikuid juhul, kui süsteemis on esinenud kõrvalekalded normist. „Elutsüklid“ pikemas perspektiivis on alati planeeritud ning vajalikud ressursid on tagatud optimaalsel määral. Kasutatakse erinevaid tööriistu, mis kontrollivad ning jaotavad koormust süsteemile.

Tänapäeval erinevad tehnoloogiad on suures plaanis jaotatud erinevate taristute (*ingl. infrastructure operations*) vahel. Sageli monitooritakse nende poolt ainult kriitilisi mõõdikuid ning järgitakse trende pikemas perspektiivis ressursside planeerimiseks. Paraku nendest süsteemi seiretest pole piisav ning vigade ilmnemisel tegeletakse avariide likvideerimisega.

Kiiresti muutuvas keskkonnas, kus agiilne metoodika on rakendatud üle erinevate meeskondade, on tekkinud vajadus suurema pildi nägemiseks haldusrolli täitvatele inimestele. Arendusmeeskondades teenuste haldusega tegelevate spetsialistide vastutuse alla kuuluvad:

- meeskonnale kuuluvate jooksvate protsesside monitoorimine;
- andmete laadimised;
- protsessid, mis vajavad manuaalset sekkumist (nt. uute nõuete lisandumisel andmete ümberarvutamine, uute andmete sisse laadimine suurtes mahtudes);
- arendusmeeskonna uute arenduste testimine ja aktsepteerimine enne tarnet;
- juurutatud protsessite valideerimine peale tarnet;
- erakorralised kampaaniad ning palju teisi tegevusi.

Täites lisaks tarkvara tehnilisele testimisele halduse rolli enda meeskonnas, on lõputöö autor jõudnud seisukohale, et halduse rolli esindajal igas arendusmeeskonnas peaks olema selge ülevaade süsteemi hetkeolukorrast selleks, et ennetada olukordi, kus süsteemi töö võiks olla häiritud. Igal tööviisil on alati omad plussid ja miinused ning miinuseid on võimalik parandada väljatöötatud abistavate protsessidega ning parema läbipaistvusega. Paljud süsteemid on üldkasutatavad erinevate meeskondade poolt, ning siin ülevaade paralleelsetest tegevustest piirneb ja vastutus hägustub. Enam levinumad süsteemi stabiilsust häirivad tegurid võivad tekkida manuaalse tegevuse tagajärjel, nende seas:

- ketaste ruumi äkiline täitumine võib juhtuda prognoosimata tegevustest nagu andmete kopeerimine tagavarasalvestuseks;
- uute andmete laekumine ajaloo ümberarvutamiseks;

- faili laekumine vales formaadis või vale nimega;
- failide kokku surumise süsteemi protsessi ebaõnnestumine;
- faili avamine ning unustamine selle sulgemiseks;
- protsesside manuaalsed taaskäivitused teatud vigade tõttu jne.

Kõik need näited ei tähenda seda, et puuduvad vahendid või võimalused teatud näidikuid jälgida – need on olemas. Põhiline mure on mugavus ning infole kiire ligipääsetavus.

Selleks, et halduse ülesandeid täitev inimene saaks kiire ülevaate süsteemi seisundist, peaks ta kontrollima umbes 5 erinevat tööriista. Teradata puhul – Teradata Viewpoint, ODI osas – ODI Studio, kettaruumi mahtu kontrollimiseks tuleb vaadata infot virtuaalserverist, faili olemasolu, arhiveerimist ning pakkimist – eraldi kataloogides, jõudluse infot (ingl. CPU) pärima andmebaasist skriptidega jne.

Võimalikke jälgitavaid mõõdikuid reaalajas on palju, mis annaksid parema ülevaate süsteemist ning võimaldaksid muuta ennetamisprotsessi sujuvamaks. Kõik need aspektid teevad süsteemi hoomamise ning haldamise kohati komplekseks ning autori arvates võiksid need olla optimeeritud.

2.2 Vajaduste analüüs

Suurel andmeaidal on kasvav vajadus jälgida ja hallata keerulisi arvutusalgoritme ning kontrollida, et kõik süsteemid töötavad laitmatult. Selle eesmärgi saavutamiseks on ülioluline, et administraatorid saaksid hõlpsalt kontrollida kasutajate põhitoiminguid ja monitoorida kogu süsteemi jõudlust, saades kiiresti aru, kui konfiguratsioon ei ole optimaalne. [3]

Põhjalikult kaardistades vajadusi ning kirjeldades hetkeolukorda, oleme jõudnud põhitingimustele, millele peaks vastama andmeaida korrasoleku kontrolli automatiseerimissüsteem:

1. Võimeline salvestama statistikat analüüsi tarbeks ning jälgimaks trende.
2. Võimeline käivitama erinevaid skripte, erinevate keskkondade suhtes.

3. Lihtsasti hallatav.
4. Selge elutsükliga (*ingl. life cycle*).
5. Omama liidest.
6. Võimeline edastama teavitusi.
7. Võimaldaks kavandada tööülesandeid *(ingl. tasks).*
8. Võimalus manuaalselt käivitada tööülesanne uuesti.

3 Ülevaade analoogsetest lahendustest

Täiendava kogemuse ning ülevaate saamiseks autor tutvus mitmete lahendustega, mis olid analüüsitud ja väljatöötatud erinevate riikide IT-spetsialistidena ja IT-teadlastega. Käesoleva lõputöö raames soovib autor täpsemini vaadelda kahte tööd, kus lõpptulemuse saavutamiseks kasutati 2 erinevat automatvalideerimist toetavat IT-platvormi: Jenkins ja Microsoft Power BI.

3.1 Jenkins platvormil põhinev lahendus

Esimesena autor tutvustab lahendust, mis oli väljatöötatud ning ettekantud 4 autori poolt 2017.aastal iga-aastasel konverentsil HPCSYSPROS17 Denveris (*High Performance Computing (HPC) Systems Professionals Workshop*).

Tegemist on *Texas Advanced Computing Center (TACC)* üksusega, kus juba eelnevalt oli väljatöötatud mitu lahendust, mis aitasid kontrollida süsteemi terviklikkust ja potentsiaalseid probleeme enne, kui need muutuksid kasutajatele häirivaks [3]. Kuid kuna sarnaselt käesolevas lõputöös vaadeldavale osakonna olukorrale, paljud neist testidest viidi läbi sihtotstarbeliselt, sõltumatult ja mitte jälgides ajaloolist statistikat, tekkis vajadus luua ühtset monitoorimise tööriista kogu süsteemile. Uue rakenduse eelistusteks pidid olema ka andmete visualiseerimine ja võimalus intuitiivselt tuvastada potentsiaalseid probleeme. Samuti uus lahendus pidi võimaldama jälgida nii kogu süsteemi hetkeseisundit, kui ka jõudluse infot möödunud perioodide kohta [3].

Põhilisteks kriteeriumideks said seatud lühikesed tööprotsesside käivitamise ajad ning süsteemi kommunikatsiooni- ja arvutusjõudluse jälgimise võimalus. Teised olulised aspektid olid süsteemi jätkusuutlikkust monitooritavad tsüklid, mis jälgiksid süsteemi konfiguratsiooni olekut ja kontrolliksid, et mõned põhilised kasutajate töötegevused toimivad korralikult [3]. Samuti uue platvormi üliloluliseks omaduseks pidi olema võime töötada koos ressursihalduse rakendusega nagu näiteks Slurm (*Simple Linux Utility for Resource Management*). Slurm on Kuutõrvaja keskkonna autorite poolt kirjeldatud järgnevalt: “Slurm on arvutiklastri jaoks mõeldud rakendus, mille ülesandeks on jagada etteantud töid mööda arvutiressursse laiali, monitoorida töötavaid protsesse ja pidada tööde üle järjekorda [4]“.

Peale mitme võimaliku automatiseerimist toetava platvormi hindamist nad sisuliselt ehitasid enda lahendust Jenkins platvormi põhjal, kuna käesoleva lahenduse autorid on leidnud, et see on populaarne ja hästi toetatud testimise automatiseerimise raamistik, mida kasutatakse süsteemi jõudluse ja regressiooni testimiseks. Jenkins ja Slurm'i vaheline ühendus sai projekti raames teostatud Python programmeerimiskeele skripti vahendusel [3].

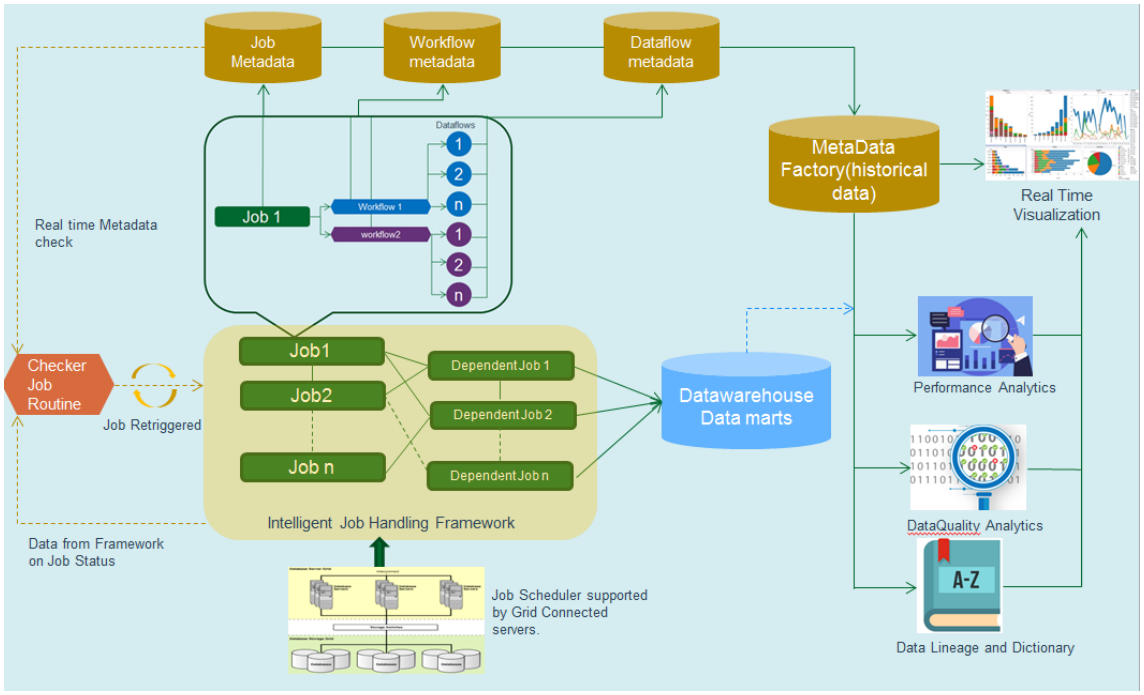
Lõpptulemusena oli ehitatud uus paindlik ja edasiarendatav süsteem, mis autorite arvamusel võimaldab süsteemiadministraatoritele ja töö protsessite halduril varakult märgata süsteemis toimuvaid muudatusi tänu uue lahenduse visualiseerimise funktsionaalsusele ning ennetada tõsiste tõrgete tekkimist [3].

Lõputöö autori arvates on vaadeldud töö väga põhjalik ning väljatöötatud lahendus omab suurt potentsiaali ka tulevikus, mida saab kohandada ja täiustada vastavalt nõuetele või muudatustele.

3.2 Microsoft Power BI platvormil põhinev lahendus

Teise lahenduse tutvustamist valis lõputöö autor põhjusel, et see oli väljatöötatud finantsasutuse andmeaida jaoks. Microsofti poolt loodud Power BI platvorm ei ole hetkel lõputöös vaadeldavas andmeaidas kasutusel, kuid tutvudes alloleva lahendusega ja selle tulemustega autor arvab, et Power BI platvorm oleks mõistlik alternatiiv tulevikuks juhul, kui järgneva analüüsi tulemusena olemasolevad platvormid ei osutu piisavalt sobilikuks püstitatud ülesande jaoks.

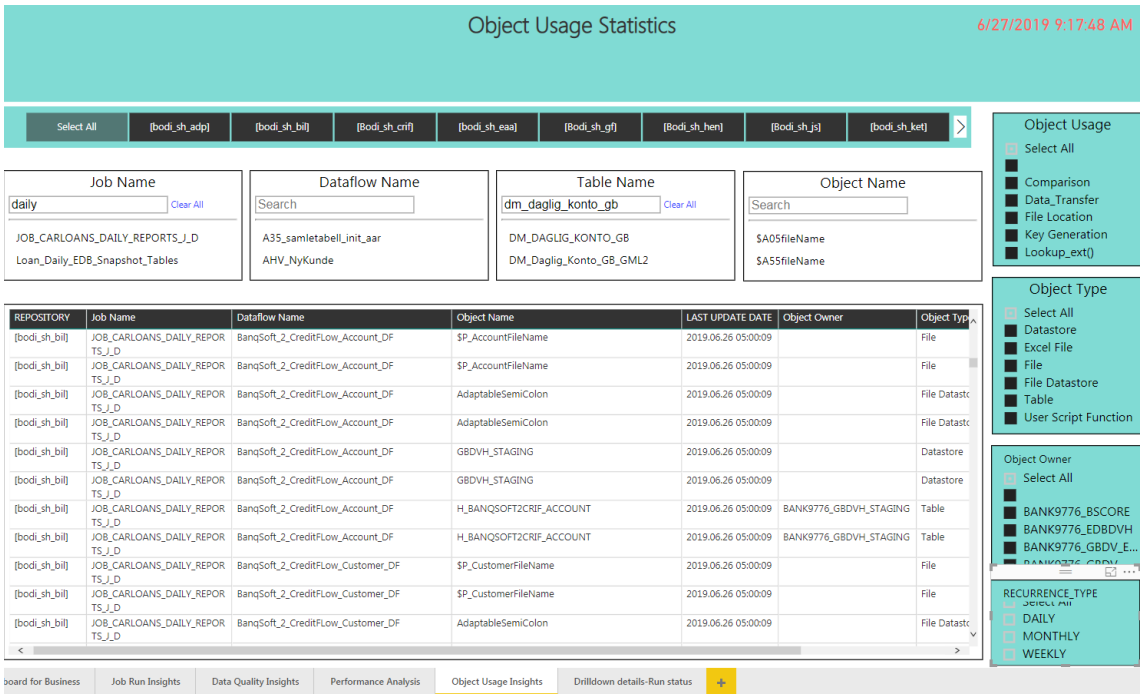
Vaadeldava lahenduse juurutamise autoriks on Sebastin Stephen Natarajan, kes töötab andmeaida insenerina Norra pangas – Gjensidige. Natarajan esitab automatiseeritud andmeaida süsteemi järgnevalt (vt. Joonis 1):



Joonis 1. Automatiseeritud andmeaia arhitektuur, kavandamine ja monitoorimine [5]

Samuti Stephen oma töös tõi välja 3 olulist etapi andmeaida halduse automatiseerimisel:

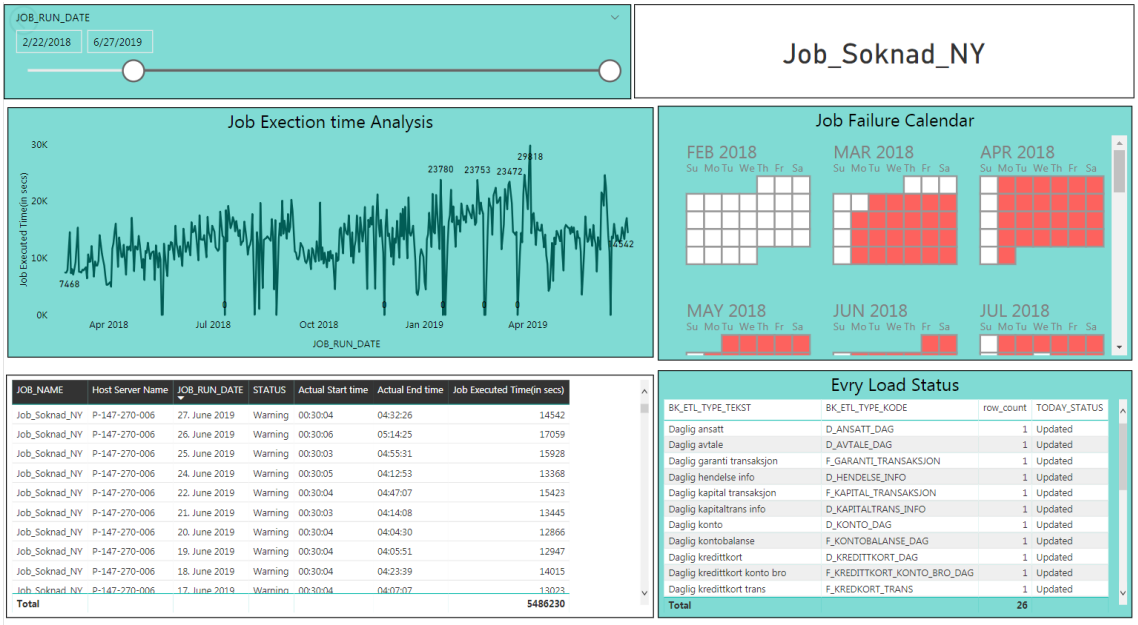
1. Andmeaida regulaarsete protsessite automatiseerimine intelligentse, programmeeritava platvormi abil (vt. Joonis 2**Error! Reference source not found.**):



Joonis 2. Objektide jõudluse monitoorimise näide Power BI rakenduse abil [5]

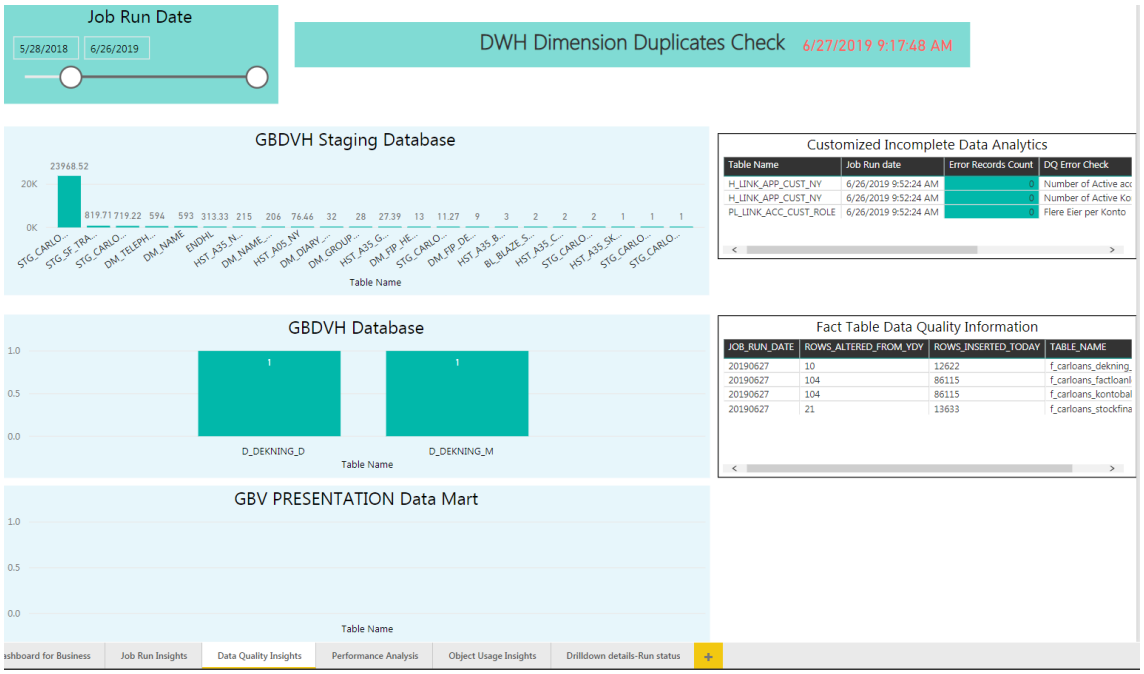
Joonisel 2 on kujutatud Power BI rakenduse juhtpaneel, mis kuvab jõudlustrendi analüüsi andmevoo tasemel. Kasutades erinevaid filtreid on võimalik täpsemini jälgida nii konkreetset tööprotsessi, kui ka uurida kogu andmevoogu, lihtsasti tuvastades suurest andmevoost kehvasti toimivaid protsesse.

Samuti käesolev juhtpaneel näitab kõiki andmevooge koos trendidega, mis erinevad selle tavapärasest jõudlusest nii positiivse kui ka negatiivse poole pealt (vt. Joonis 3) [5].



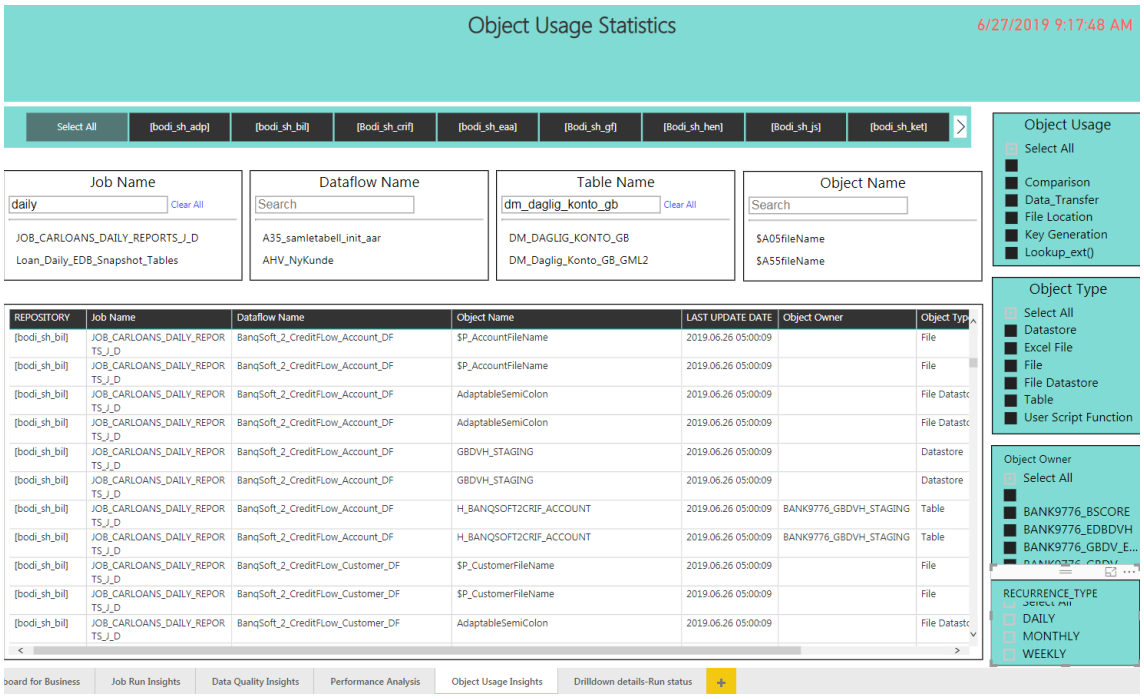
Joonis 3. Power BI juhtpaneeli ekraanipilt: Statistilised näitajad, ajaloolised andmed [5]

2. Andmebaaside seire ja andmete kvaliteedi automatiseerimine Power BI visualiseerimislahenduse abil (Joonis 4).



Joonis 4. Power BI juhtpaneeli ekraanipilt: andmete kvaliteedi juhtimine [5]

3. Metaandmete abil automatiseeritud objektide vahelised seosed (Joonis 5).



Joonis 5. Power BI juhtpaneeli ekraanipilt: integreeritud metaandmed (protsessite sõltuvused) [5]

Töö lõpuosas Sebastin Natarajan toob välja mõned eelised, mida Gjensidige pangas saavutati Power BI automatiseerimisprojekti raames:

- ✓ Andmeaida haldusosakonna ressurss vähenes 4-lt töötajalt 2-ni (*ingl. FTE*), kuna tänu Power BI rakendusele käsitsi sekkumiste vajadus oluliselt vähenes. Osakonna jaoks see tähendas olulist kulude kokkuhoidu.

- ✓ Keskmine andmeaida protsesside valmimisaeg enne rakenduse töölevõtmist oli 10 tundi, mis nüüd on lühendatud 7-le tunnile. Selline muudatus sai võimalikuks tänu süsteemi jõudluse täiustamiseks teostatud ülesannetele, mis olid lahendatud rakenduse juhtpaneeli (*ingl. dashboard*) kaudu.

- ✓ Teenustaseme leppele (*ingl. SLA*) vastavus enne automatiseerimist moodustas keskmiselt 80%, mis tähendas seda, et ligi 6 päeva kuus teenuste kvaliteet ei vastanud oodatule ning nõudis manuaalseid toiminguid, et taaskäivitada vajalikke tööprotsesse. Kolme kuu möödudes muutus SLA'le vastavuse tase 95%-ni.

- ✓ Paranenud andmete kvaliteet tekitas suuremat usaldust andmete raporteerimisel põhineva juhtimis- ja turundusosakonna poolt [5].

Põhjalikult uurides antud finantsettevõte näitel uuenduslikku lähenemist ja selle tulemusi, lõputöö autor leiab, et on toimunud suur areng nii andmete kvaliteedi, jõudluse, süsteemi ülevaate ja efektiivsuse osas.

4 Potentsiaalsete tööriistade analüüs

Selleks et leida sobiv platvorm, mis vastaks kõikidele meie üksuse ülaltoodud vajadustele, analüüsime kõigepealt majasisesed tööriistad, mis vaadeldavas ettevõttes on juba kasutusel ning kui sobivat ei leidu, siis uurime ka väliseid.

Käesolevas peatükis autor vaatleb ja analüüsib kasutusel olevate rakenduste sobilikkust püstitatud ülesannetega. Võrreldavateks lahendusteks said valitud: IBM Tivoli Workload Scheduler, Oracle Data Integrator ning Jenkins. Analüüsimisvahendina kasutame lõputöö autori poolt koostatud mõõdikute kaarte (ingl. *Score Cards*), mis olid loodud autori poolt erinevate vastaspoolte käest kogutud ja kaardistatud vajaduste põhjal. Lahenduste sobivust on hinnatud 3-skaalaga, kus võimalikud variandid on: „+“ – täiesti sobib, „+/-“ – funktsionaalsus olemas, kuid on vajalik lisaarendus ning „-“ – mis tähendab, et selline funktsionaalsus puudub. Autori hinnangul selline analüüsiformaat on piisavalt sisukas, kergesti jälgitav ning võimaldab lihtsasti võrrelda analüüsitavate platvormide funktsionaalsust ning selle alusel teha järeldusi.

4.1 IBM Tivoli Scheduler

Käesolevas alampeatükis vaatleme IBM Tivoli Workload Scheduler (edaspidi Tivoli) tööriista.

Tivoli on tarkvara, mis kavandab, käivitab ja jälgib töid mitmel platvormil ja erinevates keskkondades. Rakenduse põhiline eesmärk on hallata arvutikogude arvutusressursse selleks, et võimaldada erinevatel kasutajatel neid ressursse tõhusalt jagada paralleelsete protsessite samaaegseks käivitamiseks [7].

Tabel 1. Tivoli platvormi analüüs vastu nõudmisi

Tingimus	Selgitus	Sobivus
Statistika salvestamine	Tööriistal on ettenähtud ainult enda käivituste põhise info salvestamise funktsionaalsus. Päringute statistika eeldab lisaarendust (ingl. *custom code*).	+/-

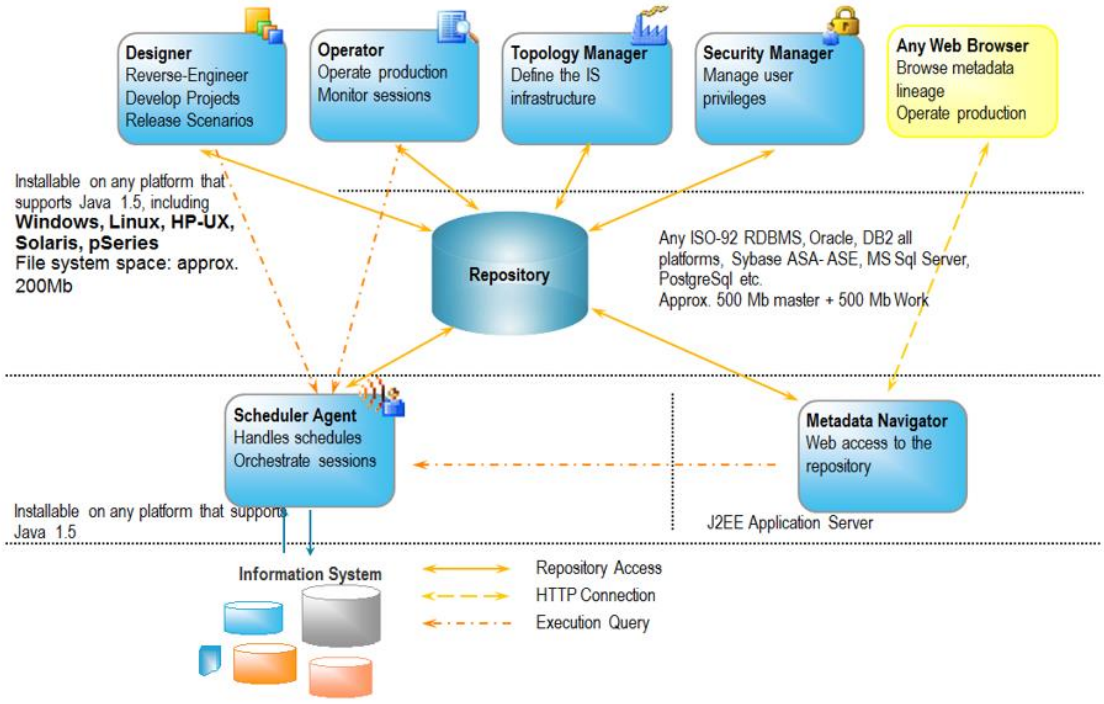
Skriptide käivitus	Funktsionaalsus eksisteerib. Paraku näeb ette suuremat sorti ettevalmistust ning haldust.	+/-
Haldamise lihtsus	Tivoli haldamine eeldab teatud kompetentsi omamist, kuna rakendus sisaldab suurel hulgal erinevaid parameetreid, mille alusel on võimalik juhtida töökäske.	+
Selge „elutsükkel“	Toetatud IBM poolt.	+
Liidese olemasolu	Tivoli liidese funktsionaalsus on suhteliselt piiratud, kuvades kavandatud töökäsu põhilist informatsiooni: kas töökäsu käivitus õnnestus, töökäsu staatust ning kas töökäsk on katkenud või edukalt lõppenud. Logide informatsioon asub serveri masinas ning ei ole lihtsasti kättesaadav.	-
Teavituste edastamine	Rakendus omab eraldi seadistatava teavituste mehhanismi (ingl. *alert engine*), mis vajadusel saab edastada teateid vastavatele e-mailidele.	+
Kavandamine	Tivoli on mitmekülgne kavandamise süsteem, mis võimaldab üles ehitada komplitseeritud protsessite sõltuvusi ja seoseid.	+
Ülesande manuaalne taas käivitamine	Funktsionaalsus eksisteerib. Paraku näeb ette suuremat sorti ettevalmistust ning haldust.	+/-

IBM Tivoli lahendus on võimas tööriist erinevate töökäskude täitmiseks, mis omab väga laialdast funktsionaalsust just kavandamisel ja töökäskude planeerimisel ning haldamisel (vt. Tabel 2). Autori arvamusel, kasutades antud platvormi on võimalik luua keerukat, kuid hästi struktureeritud automatiseeritud tööprotsesse koos sõltuvustega ning päästikprotsessidega (ingl. *triggers*) jne.

4.2 Oracle Data Integrator (ODI)

ODI on laiaulatuslik andmete integreerimise platvorm, mis hõlmab kõiki andmete integreerimise nõudeid, milleks on: suuremahulised ja suure jõudlusega või hoopis sündmuspõhiste andmete laadimine, keerukate andmetega integreerimisprotsessid jne. [8].

ODI arhitektuuri vaadeldavas andmeaidas kirjeldab allolev Joonis 7:



Joonis 6 ODI arhitektuur ettevõtte andmeaidas [8] [9]

ODI nõuetele vastavuse analüüs on esitatud alljärgnevas Tabelis 3:

Tabel 2. ODI platvormi analüüs vastu nõudmisi

Tingimus	Selgitus	Sobivus
Statistika salvestamine	Täiendava statistika salvestamist on võimalik teostada eraldi arendusega – ODI tarkvaraliidesega, mis salvestab tagastatavaid andmeid andmebaasi.	+/-
Skriptide käivitus	Skriptide käivitamise funktsionaalsus eksisteerib.	+

Haldamise lihtsus	Iga vaadeldav juhtum tähendab uue/muudetud komponendi eriarendust ning selle tarnet (ingl. *release*) järgides igakord tarne protsesse. See teeb antud lihtsa ülesande keerulisemaks.	-
Selge „elutsükkel“	ODI on rakendus selge süsteemiga, kuid keerulise ülesehitusega, millel on rida nõudeid programmi versioonide uuenduseks.	+
Liidese olemasolu	Laialdasema funktsiooniga tarkvaraliides, mis salvestab protsessi iga sammukohast informatsiooni, lihtsasti loetav ning ligipääsetav. Paraku antud liides ei ole ettenähtud tagastatud tulemuste kuvamiseks.	-
Teavituste edastamine	Vajab eraldi arendust. Üldjuhul on tegemist alamkäsuga, mida kutsutakse välja teatud tingimustel.	+/-
Kavandamine	Rakendus võimaldab protsesside kavandamise teostust.	+
Ülesande manuaalne taas käivitamine	Võimalus eksisteerib. Protsesse saab vajadusel manuaalselt uuesti käivitada.	+

Teostatud vaatluse järelduseks võib öelda, et ODI on kõigepealt ETL - protsesside tööriist, mis on ettenähtud andmete liigutamiseks, kuigi platvorm on laia funktsionaalsusega, mis võimaldab üles ehitada erineva otstarbega protsesse.

4.3 Jenkins

Jenkins on iseseisev, avatud lähtekoodiga platvorm, mida saab kasutada erinevate tarkvara ehitamise, testimise ning tarnimise või juurutamisega seotud ülesannete automatiseerimiseks.

Tabel 3. Jenkins platvormi analüüs vastu nõudmisi

Tingimus	Selgitus	Sobivus
Statistika salvestamine	Statistika kogumiseks on kaks võimalust: kas eraldi andmebaas (trendide jaoks) või andmete kogumine logidesse.	+/-
Skriptide käivitus	Jenkins omab mitmekesist funktsionaalsust, mis võimaldab kasutada erinevaid tarkvaramooduleid skriptide käivitamiseks.	+
Haldamise lihtsus	Töökäske on võimalik iseseisvalt muuta, täiendada ning hallata.	+
Selge „elutsükkel“	Jenkins rakendusel on tugev kogukond, mis toetab vabavaralist tarkvara. Teiselt poolt sellega kaasneb teatud risk elutsükli näol, kuna tegemist on entusiastliku algatusega.	+/-
Liidese olemasolu	Kõik töökäsud on võimalik grupeerida ühe vaate alla ning kuvada tagastatavat staatust.	+
Teavituste edastamine	On olemas teavituste edastamise võimalus.	+
Kavandamine	Eksisteerib kavandamise funktsioon.	+
Ülesande manuaalne taas käivitamine	Igat protsessi on võimalik uuesti sobival ajal välja kutsuda.	+

Jenkins on multifunktsionaalne platvorm, mis on mõeldud CI/CD (ingl. *Continuous Integration (CI) and Continuous Deployment or Continuous Delivery (CD)*) tarbeks, et automatiseerida arendussamme, testimise samme ning lihtsustada integreerimist [11] ning autori arvamusel Jenkins on üks kindel kandidaat edaspidiseks vaatlemiseks.

5 Analüüsi tulemused

Käesolevas peatükis autor võtab kokku läbivaadatud platvormide analüüsitulemusi, teeb täiendava võrdluse kahe parima lahenduse vahel ning selle põhjal valib kõige sobilikuma automatiseerimise ja monitoorimise platvormi, mida tulevikus hakatakse arendama ja kasutama andmeaida halduse rollide tarbeks.

ODI mittevalimise peamiseks põhjuseks on see, et iga muudatus protsessis eeldab tarneprotsessi läbimist vastavalt kehtivatele ettevõtte sisemistele reeglitele ning sellega kaasneb kohustus vastava dokumentatsiooni loomiseks, muudatuste kirjeldamiseks, testimiseks jne.

Sobivateks lahendusteks on kujunenud Tivoli ning Jenkins. Mõlemad tarkvaraplatvormid on oma iseloomult loodud tööprotsesside automatiseerimiseks ning on võimelised hõlpsasti täitma eelpool kirjeldatud nõudeid.

Võrdleme mõlema protsessi puhul tingimusi, et valida kahest potentsiaalsest lahendustest parim. Töö alguses kogutud tingimustele vastavuse kokkuvõte Tivoli ja Jenkins kohta on esitatud allolevas autori poolt koostatud tabelis (Tabel 4).

Skaala selgitus:

1 – Vajalik funktsionaalsus puudub.
2 – Rakendusel võimekus olemas, kuid näeb ette suuremahulist lisaarendust.
3 – Vajab lisaarendust või on tegu piiratud funktsionaalsusega, et täita etteantud ülesannet.
4 – Rakendus on kasutatav, kuid vajab kergemat kohandamist ja seadistamist.
5 – Rakendus omab vajalikku funktsionaalsust, et täita püstitatud ülesande täielikult.

Tabel 4. Tivoli ja Jenkins platvormide kokkuvõttev analüüs

Tingimus	Tivoli sobivus	Jenkins sobivus	Selgitus
Statistika	3	4	Tivoli: Piiratud logid ning ülevaade töökäskude täitumisest. Täiendava statistika kuvamine ja kogumine nõuab täiendavaid arendusi. Jenkins: Põhjalikud, detailsed logid. Hea ülevaade.

			Täiendava statistika kuvamine ja kogumine lahendatav eraldi integreeritud skriptidena. **Kokkuvõte**: Jenkins rakendusel on suurem läbipaistvus logidel ning lihtsamini hallatav statistika.
Skriptide käivitus	4	5	Tivoli: Skriptide käivitamise funktsionaalsus eksisteerib Jenkins: Mitmekesised integreeritavad tarkvara moodulid võimaldavad lihtsasti käivitada erinevaid skripte. **Kokkuvõte**: Mõlemad platvormid on võimelised käivitama erinevaid skripte, ehk mõlemad on antud tingimuse juures suhteliselt võrdväärsed.
Haldamise lihtsus	3	4	Tivoli: Kõik vajalik on olemas, kuid iga uuendus või muudatus näeb ette keerulist ning aeganõudvat protsessi. Jenkins: Töökaske on vajadusel võimalik iseseisvalt luua, muuta, parandada. **Kokkuvõte**: Jenkins rakendusel on suurem läbipaistvus logidel ning lihtsamini hallatav statistika.
Selge „elutsükkel“	5	4	Tivoli: Tivoli omab IBM’i poolset tuge ning uuendusi mille arendusega tegeleb IBM’i ettevõtte. Jenkins: Jenkins on tasuta tarkvara, mida arendab Jenkins kogukond. **Kokkuvõte**: Kuna uuendused ja nende riski kontroll sageli näeb ette suuremat testimist ja tähelepanu, siis autori arvamusel Tivoli omab selget eelist võrreldes Jenkinsiga.
Graafilise liidese olemasolu	4	5	Tivoli: Graafiline liides on hästi arendatud, kuid suurte projektide puhul vajab palju arvuti ressursi. Jenkins: Liides on lihtne ning kiire. **Kokkuvõte**: Antud tingimusele vastavad mõlemad platvormid võrdselt.
Teavituste edastamise võimalus	5	5	Tivoli: Olemas läbimõeldud sündmuste juhtimise süsteem (ingl. *event management system*), mis omab väga laialdast

			funktsionaalsust reageerida erinevatele sündmustele ning edastada teateid. Jenkins: olemas paindlik lahendust, mis võimaldab erinevatel etteantud tingimustel edastada teateid valitud saajatele. **Kokkuvõte**: Mõlemal teenusel on olemas teavituste edastamise funktsionaalsus. Siinkohal mõlemad teenused täidavad tingimusi täielikult.
Kavandamine *(ingl. scheduling)*	5	5	Tivoli: Mitmekülgne protsesside kavandamise funktsionaalsus (arvestades tööde omavahelisi seoseid ja sõltuvusi). Jenkins: Lai valik võimalusi planeerida protsesse ja skriptide automaatset käivitamist erineva regulaarsusega. **Kokkuvõte**: Mõlemal platvormil on kavandamise funktsionaalsus esindatud.
Ülesande manuaalne taas käivitamine	5	5	Tivoli: Töökäskude ja protsesside manuaalne taaskäivitamise võimalus eksisteerib. Jenkins: Vajalikku tööülesande on võimalik uuesti käivitada kasutaja valitud ajal. **Kokkuvõte**: Mõlemad platvormid võimaldavad tööprotsesside taaskäivitamist.
Kokku	34	37	Tivoli on väga hea tööriist erinevate tööülesannete kavandamiseks ning keerukate sõltuvustega protsesside ülesehitamiseks, kuid omab teatud täiendavat keerukust ning puudusi etteantud ülesande täitmiseks. Jenkins omab mõnevõrra lihtsamat, paindlikumat ning läbipaistvamat lahendust püstitatud ülesande täitmiseks.

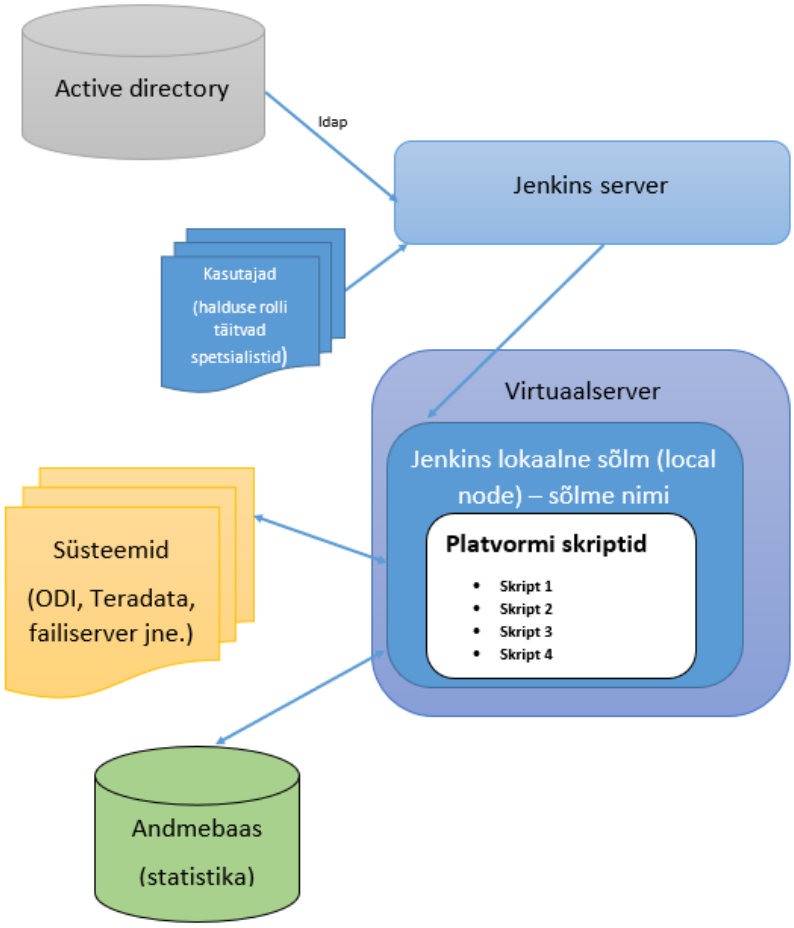
Üheks oluliseks aspektiks valiku tegemisel on hinna kujunemise teema, mida autor sooviks enne lõpliku järeldust esile tuua. IBM Tivoli Workload Automation platvorm omab töökäsu ja agendi põhist hinnastamise loogikat, mille täpsem ülevaatamine paraku pole antud lõputöö raames võimalik, kuna tegemist on ärisaladusega. Jenkinsi puhul – vastupidi, tegemist on vaba tarkvaraga, mille kasutamine ei too endaga täiendavat kulu ettevõttele ning see on ka väga suur eelis antud lahenduse rakendamiseks.

Põhjalikult analüüsides kõiki tegureid, käesoleva lõputöö autor jõudis järeldusele, et lõputöös püstitatud ülesande täitmiseks sobib kõige paremini Jenkins. Järgmiseks sammuks on tehniline teostus pakutud lahendusel vaadeldavas andmeaidas platvormi sobilikkuse testimiseks.

6 Tehniline teostus (*Proof of concept*)

Jenkinsi paigaldamiseks on mitu võimalust: süsteemipakettide abil igas konkreetses ettevõttes, Docker'i[2] kaudu või eraldi käivitada mis tahes masinas, kuhu on eelnevalt paigaldatud Java Runtime Environment (JRE). [10]

Suures pildis rakenduslik osa hõlmab endas Active Directory ühendamist Jenkinsiga kasutajate ligipääsu juhtimiseks, Jenkinsi sõlme loomist virtuaalserveril, tehniliste kasutajate loomist, kes omavad ligipääsu monitooritavatele süsteemidele ning vajalikke skriptide loomine, testimine ja kavandamine käesolevas töös väljatoodud süsteemi jätkusuutlikkuse mõõdikute jälgimiseks (vt. Joonis 8).



Joonis 7 Uue platvormi mudel (ülesehituse üldvaade)

[2] „Docker on avatud lähtekoodiga tarkvara, mille eesmärk on konteinerite abil lihtsustada rakenduste loomist, juurutamist ja käivitamist. Arendajad kasutavad konteinerid rakenduse kokku pakkimiseks kõikide vajalikete komponentidega – sõltuvuste ja andmete kogudega ning juurutatakse ühe paketina [12].“

Selleks, et testida pakutud platvormi võimekust ning saada aru, kas oleme teinud õiget valikut, autor viis läbi erinevaid teste, mis tulenevad esimeses osas süsteemile seatud nõuetest. Neli põhilist teemat, mida autor soovis testida:

1. Andmete teisendamiskihi mahu täituvuse kontroll ja automaatne etteteavitamine.
2. Tuvastada protsessid, mis kõige rohkem kasutavad CPU.
3. Töökäsu regulaarne automaatkäivitamine, mis kontrolliks protsesside või mõne teatud serveri hetkeseisu (teavitamine ebaõnnestumisel).
4. Kaardistamata vastenduste tuvastamine ning rapordi edastamine andmespetsialistidele.

Valitud kontrollide automatiseerimiseks Jenkins baasil, oleme kasutanud põhjana paralleelset projekti. Antud pilootprojektis, mille autor nimetas „GDW Maintenance Optimization“ (Andmeaida Halduse Optimeerimine) kasutatakse GIT – repositooriumi, JDK tarkvara (Java Development Kit), Maven projektihalduse tööriista [12] [13], automatiseerimise mooduleid Serenity BDD [14] ja JUnit [15] ning Teradata ja Oracle standardseid draivereid andmebaasidega ühenduste loomiseks.

Autor tutvustab põhjalikumat ülevaadet esimese teema näitel järgmises alampeatükis.

6.1 Näide 1: andmekihi “OSA” mahu täituvuse kontroll

Selleks, et kontrollida ühendust Teradata andmebaasiga ja platvormi võimalusi, valisime lihtsa testjuhtumi teisendamiskihi andmemahu kontrollimiseks.

ODI Staging Area (edaspidi OSA) on üks vaadeldavas andmeiadas kasutatavaid andmete teisendamiskihtidest, kus hoitakse sessioonipõhiseid andmeid tabelite näol. Alates Teradata 16.20 versioonist puudub toetus Teradata Administrator rakendusele, mis varem võimaldas halduse spetsialistidel jälgida andmebaaside mahu täituvust reaalajas. Iga agiilne meeskond jälgib seda iseseisvalt, samuti IT-monitooringul on seatud vastavad trigerid 90% peale, mille täitumisel saadetakse välja teavitused. Paraku andmeaida pideva kasvamisega tekib olukordi, kus samaaegse suuremahuliste protsessite käivitamise tõttu täitub andmete teisendamiskihtide maht ülikiiresti ning tulemusena protsessid ebaõnnestuvad.

OSA mahu kontrollimiseks on võimalik luua mitu erinevaid skripte. Allolevatel joonistel on SQL-skript ja selle väljund Teradata SQL Assistent rakendusel, mida oleme automatiseerinud Jenkins platvormil (Joonis 9 ja Joonis 10).

```
SELECT
DATABASENAME,
CAST(SUM(MAXPERM)/(1024*1024*1024) AS DECIMAL(7,2)) MAX_PERM,
CAST(SUM(CURRENTPERM)/(1024*1024*1024) AS DECIMAL(7,2)) CURRENT_PERM,
(CAST(SUM(CURRENTPERM)/(1024*1024*1024) AS DECIMAL(7,2)) /CAST(SUM(MAXPERM)/(1024*1024*1024) AS DECIMAL(7,2)))*100  CURRENT_PERM_PERCENT
FROM
DBC.DISKSPACE
WHERE DATABASENAME IN ('OSA')
GROUP BY
DATABASENAME ORDER BY MAX_PERM DESC;
```

Joonis 8 SQL skript OSA andmekihi mahu täituvuse kontrollimiseks

Skripti käivitamise tulemus Teradata SQL Assistant'is:

	Database Name	MAX_PERM	CURRENT_PERM	CURRENT_PERM_PERCENT
1	OSA	3,993.74	1,929.82	48.00

Joonis 9 SQL skripti väljund OSA hetkeseisu kohta

Töökäigust lähemalt:

1. SQL skript on lisatud GIT'i, seejärel oleme loonud uue sammu Maven projektis JUnit'i koodiraamistiku alusel [15], mille poole hiljem pöördume Jenkins töökäsus (Joonised 11, 12 ja 13):

```
public class GdwDataCollection {

    private GdwCommon common = GdwCommon.getInstance();
    private String PERM_PERCENT_LIMIT = System.getProperty("osa.amount.limit");

    @Steps
    GdwTrfTeradataSteps gdwSteps;
```

Joonis 10 Uue klassi isendi loomine (instance)

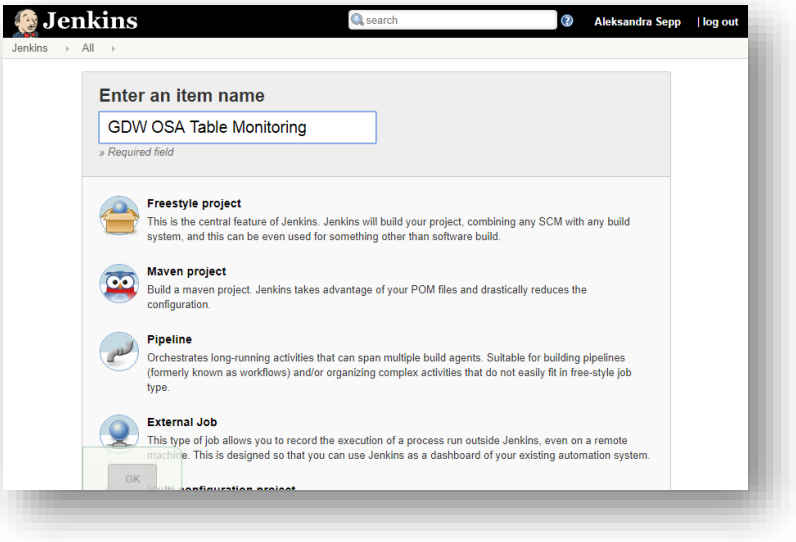
```
@Test
@WithTag("suite:osa.check")
public void osaSpaceCheck()throws SQLException {
    gdwSteps.checkOsaTableIsNotFull(PERM_PERCENT_LIMIT,
        common.convertFileToString( filePath: "src/test/resources/gdw.monitoring.sql/osaSpaceCheck.sql"),
        columnLabel: "CURRENT_PERM_PERCENT");
```

Joonis 11 Uue testi „osa.check" lisamine

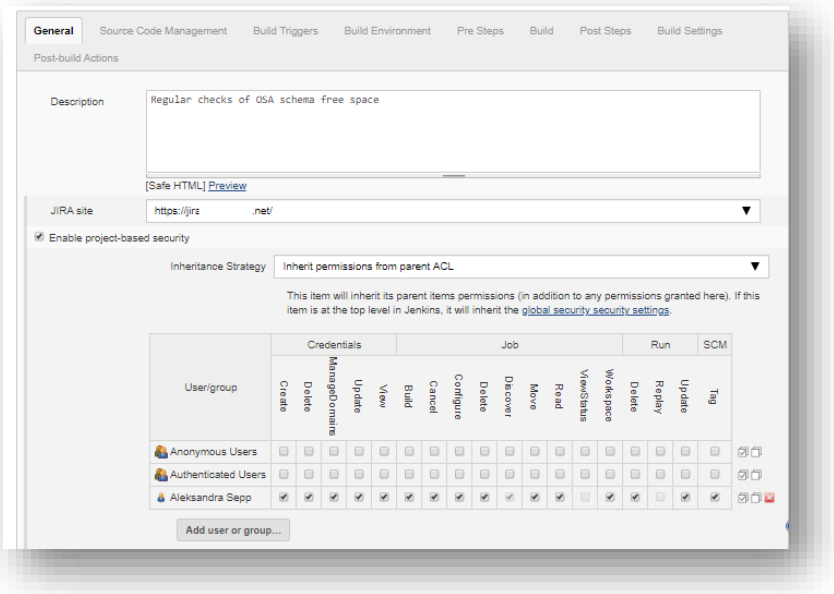
```
@Step("Check amount of data in OSA table against limit: {0}")
public void checkOsaTableIsNotFull(String limit,
                        String sql,
                        String columnLabel ) throws SQLException
{
    actualResult = common.loadDataFromDbAndCheckResult(sql, columnLabel);
    log.info(resultIs, actualResult);
   if( Double.parseDouble(actualResult) < Double.parseDouble(limit)) actualResult="OK";

     assertEquals ( expected: "OK", actualResult);

    common.connectionTeradata.close();
}
```

Joonis 12 Mahu kontrollimine vastu seatud limiiti

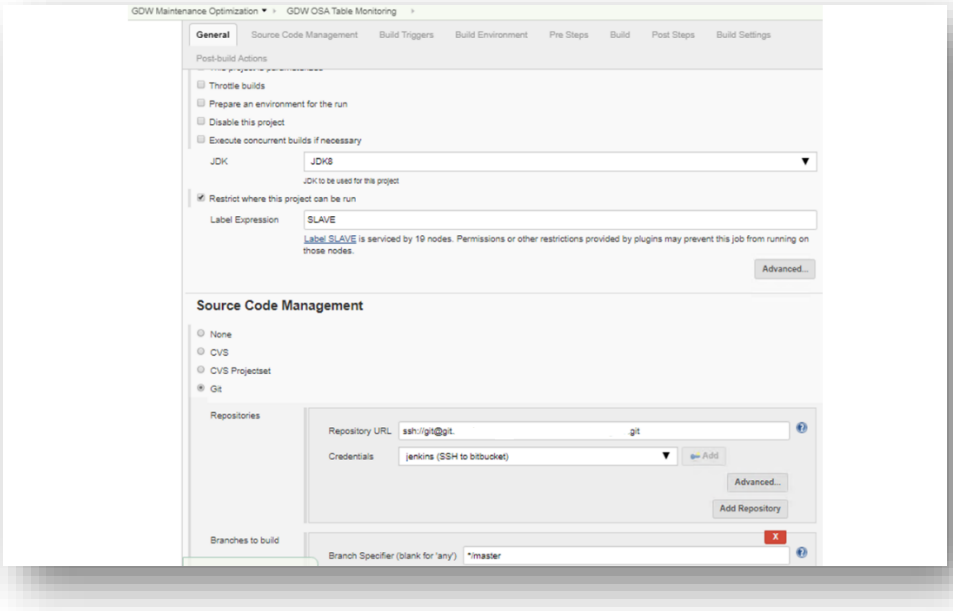
2. Uue töökäsu (ingl. *built*) loomine Jenkinsis Maven projektina (Joonised 14-16).



Joonis 13 OSA monitoorimise töökäsu loomine Jenkinsis

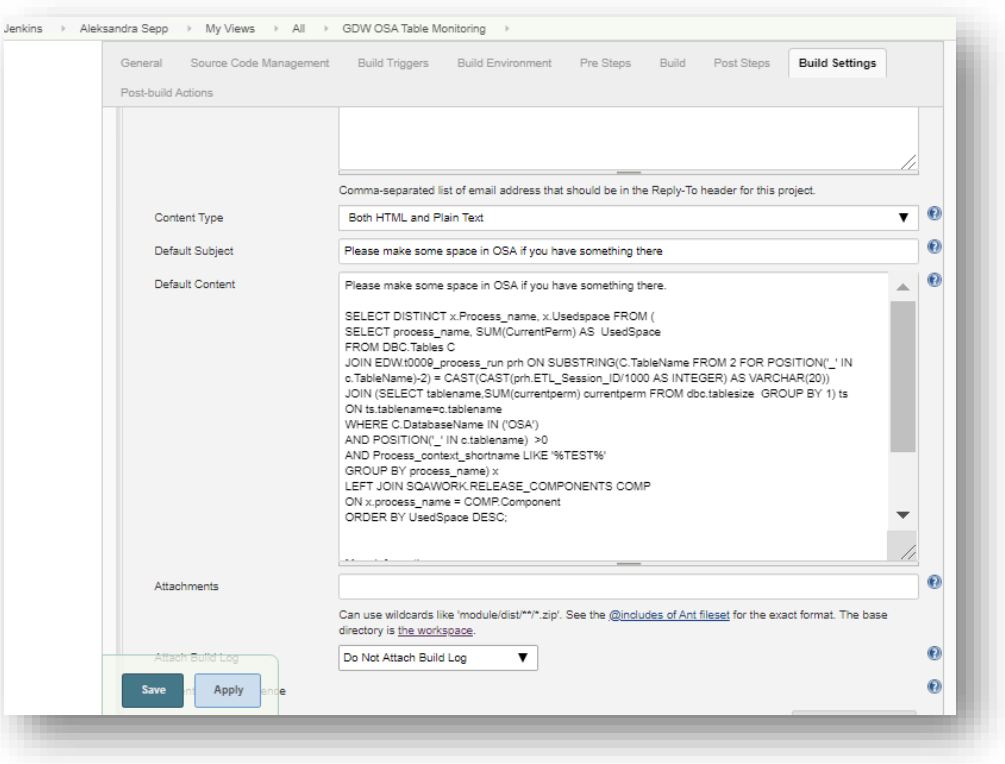


Joonis 14 Õiguste lisamine/piiramine uues Jenkins töökäsus

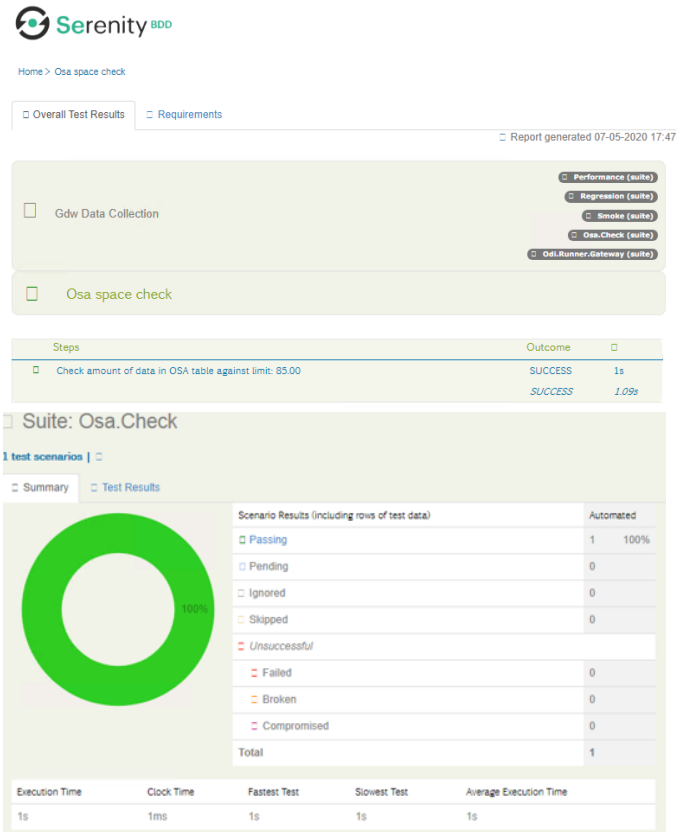


Joonis 15 Git repositooriumi teekonna määramine

3. Määrame automaatsete teavituste regulaarsust ja tingimusi. Teavituse automaatsel edastamisel saame lisada mitu saajat, soovi korral lisada sõnumit, töökäsu tulemust, graafilise rapordi (Joonised 17-18).

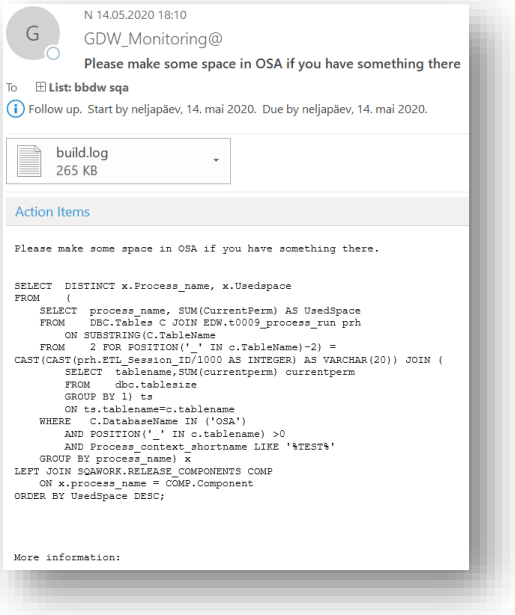


Joonis 16 Teavituse sisu määramine



Joonis 17 Ekraanipilt: OSA andmemahu kontrolli graafiline raport

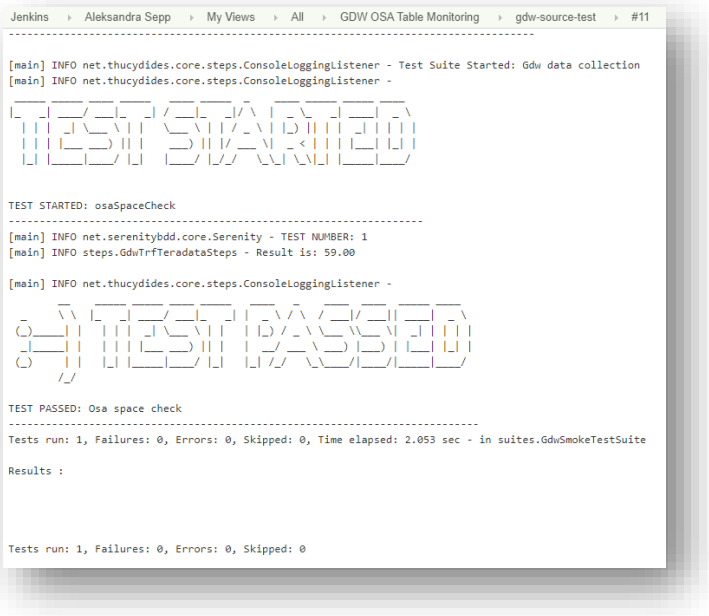
4. Uue monitoorimise töökäsu testimiseks oleme eelnevalt kontrollinud OSA hetkeseisu, mis oli 59%. Enne esimest käivitust oleme ajutiselt vähendanud maksimaalset lubatud limiiti 40%-ni. Tulemusena saime teavitust e-maili teel (Joonis 18), et antud monitoorimise test on ebaõnnestunud ning teavitusesse oli lisatud logi fail, mis on samuti saadaval Jenkinsis (Joonis 19):



Joonis 18 Automaatne e-mail OSA määratud mahulimiidi ületamisel

Joonis 19 Ebaõnnestunud monitoorimise testi logi näide

Seejärel muutsime lubatud mahu limiidi tagasi 85%-ks ning mailile enam teavitust ei edastatud, kuna seadistasime teavituse saatmist ainult mahu ületamisel, kuid Jenkinsi logides oli ilusti näha, et töökäsk toimib ning kõik on normi piires (Joonis 20):



Joonis 20 Edukalt lõppenud monitoorimise testi logi näide

Eespool kirjeldatu põhjal saame järeldada, et oleme edukalt saanud rakendada ja testida andmekihi automaatse kontrolli töökäsu Jenkins platvormil, saavutades oodatud tulemuse. Samuti on õnnestunud katsetada automaatsete teavituste funktsionaalsust, mida on võimalik kohandada vastavalt vajadustele. Graafiline kokkuvõte, kus kajastub töökäsu tulemus, täiesti eksisteerib (Joonis 17), kuid vajab suuremat tähelepanu seadistamisel tulevikus.

6.2 Näide 2: 10 protsessi, mis kasutavad kõige rohkem CPU ressurssi

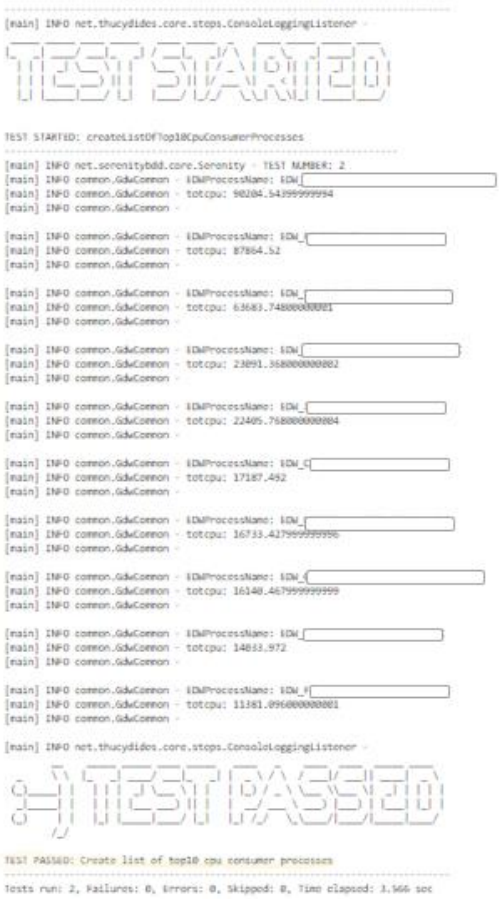
Teise juhtumina oleme automatiseerinud teavitust, kus kajastuvad 10 protsessi, mis tarbivad kõige rohkem CPU ressurssi. Antud info viitab sellele, kas peale uute andmete lisamist on tarvis täiendavat statistika kogumist objektides, päringu optimeerimist, vanade andmete arhiveerimist või muud tegevust.

Oleme toiminud sarnaselt eelnevale näitele ning rakendanud skripti, mis väljastab reaalajas top 10 protsessi, mis kasutavad kõige rohkem ressurssi (Joonis 21).

```
WITH CPU_Usage_full_list AS (
SELECT
edwPROCESSNAME
,SUM ( TOTCPU) as totcpu
FROM sys_info.PerfOutEtl
WHERE edwPROCESSNAME <> '?'  --Process name not unknwon
grouP by edwPROCESSNAME
)
SEL *
FROM CPU_Usage_full_list
QUALIFY ROW_NUMBER() OVER (ORDER BY totcpu DESC) <=10;
```

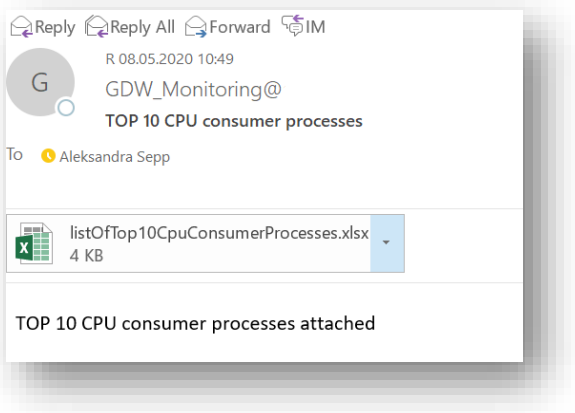
Joonis 21 SQL-skript 10 kõige rohkem ressurssi tarbivate protsessidega

Oleme lisanud teavituste automaatse edastamise igapäevaselt kell 9:00 hommikul, määratud saajate listile. Jenkins logifail näeb välja selliselt (Joonis 22):

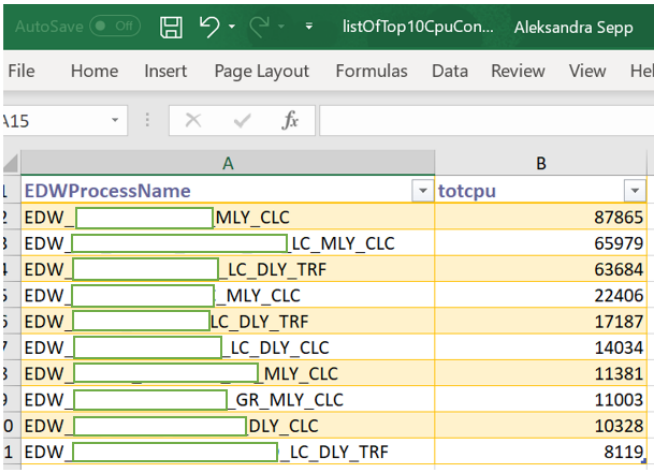


Joonis 22 Ekraanipilt: TOP 10 kõige rohkem ressurssi tarbivate protsessidega (Jenkins logi)

Seekord autor otsustas katsetada teavituste saatmist, kuhu oleks lisatud mitte logi ega graafiline kokkuvõte, vaid Excel vormingus tabel protsessidega ning nende kasutatud CPU mahuga (Joonis 23 ja Joonis 24):



Joonis 23 Automaatteavituse näide



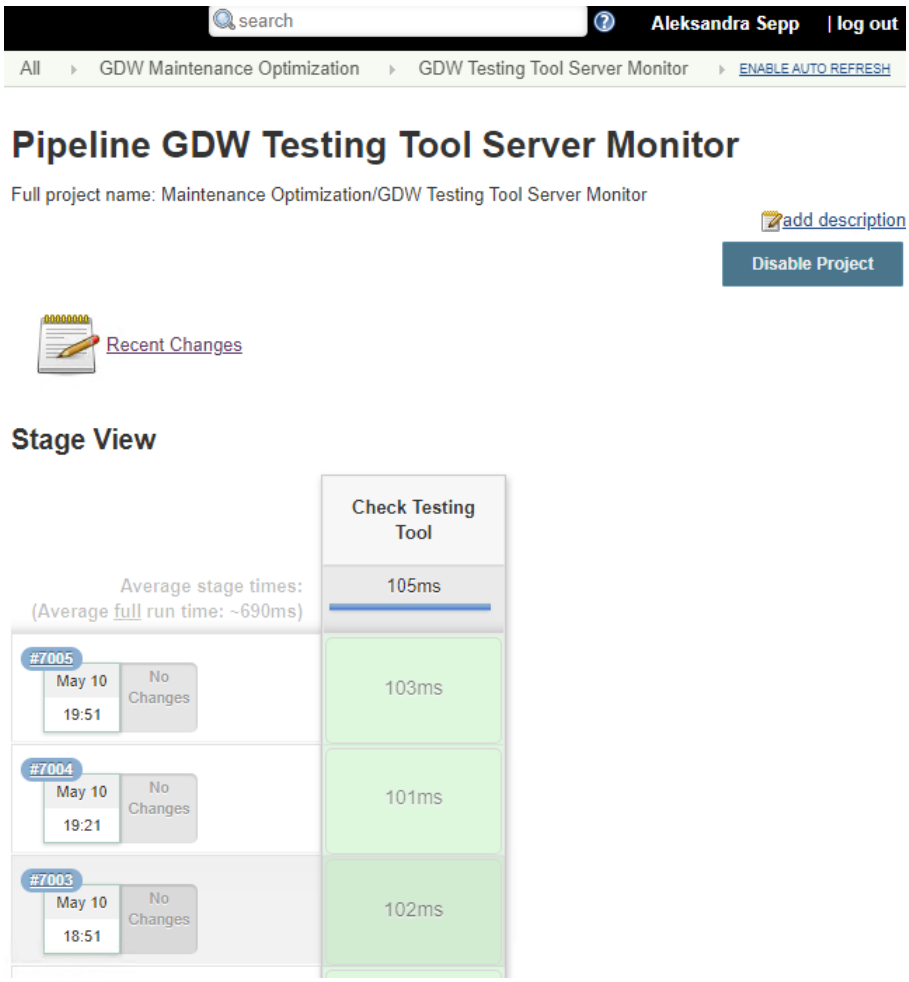
Joonis 24 CPU Top 10 protsessid, Excel faili sisu

Käesoleva juhtumi testimise kokkuvõtmiseks saame öelda, et oleme automatiseerinud kontrolli, mis oli halduse rolli täitvate spetsialistide jaoks manuaalselt teostatav igapäevaselt. Nüüd aga tööle tulles, kõik halduse spetsialistid leiavad enda postkastides automatiseeritud värskete andmetega rapordi, millega saavad kiirelt ja mugavalt tutvuda. Tulevaste arenduste käigus autor soovib rakendada ja testida kogutud statistika visualiseerimist.

6.3 Näide 3: Töökäsk Testing Tool serveri hetkeseisu kontrollimiseks

Testing Tool on vaadeldavas andmeaidas majasisene rakendus tehniliste testide läbiviimiseks. Kuna antud tööriista korrasolek on äärmiselt oluline, siis otsustasime luua automaatkontrolli serveri monitoorimiseks Jenkinsi platvormil (Joonis 25).

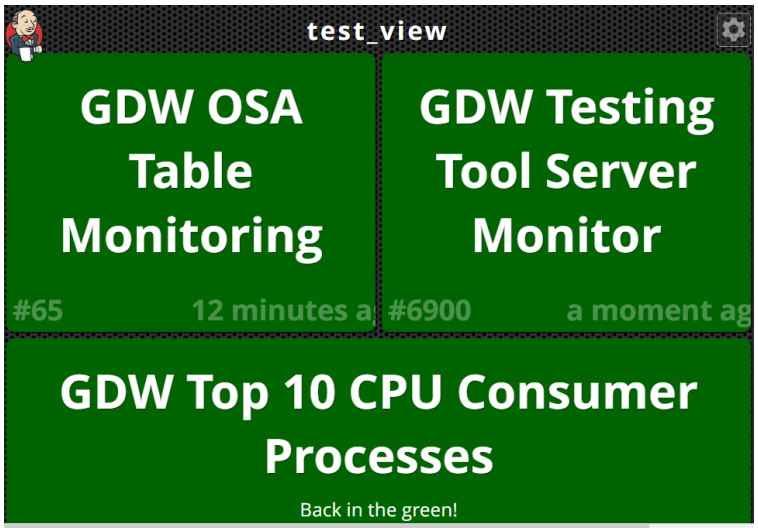
Seekord kasutasime lihtsamat Jenkins moodulit – Pipeline, mida on võimalik kasutada nii lihtsamate kui ka keerulisemate ülesannete juurutamiseks [10].



Joonis 25 Testing Tool rakenduse serveri monitoorimine (Pipeline projekt)

Meie skript on kavandatud käivituma iga 30 minuti tagant, kontrollimaks serveri ühendust. Ühenduse katkemisel tuleb automaatne teavitus määratud e-maili listile. Antud funktsionaalsus aitab säästa aega, automatiseerides manuaalset kontrolli ning võimaldab kiiresti reageerida probleemide tekkimisel ilma kolmandate poolte kaasamist.

Kuna käesoleva töökäsu testimise käigus katkenud ühendusega olukordi ei esinenud, siis teavituse näidis ei ole lisatud. Kuid antud testi raames autoril tekkis võimalus testida Jenkins moodulit Monitor Vaade (ingl. *Monitor View*), mis kuvab valitud töökäsu hetkeseisu või selle viimast käivituse staatust. See on mugav lahendus suure kuvari peal kasutamiseks kogu meeskonna tarbeks (Joonis 26):



Joonis 26 Ekraanipilt: monitor vaade

6.4 Näide 4: Kaardistamata vastete tuvastamine (ingl. unresolved mappings)

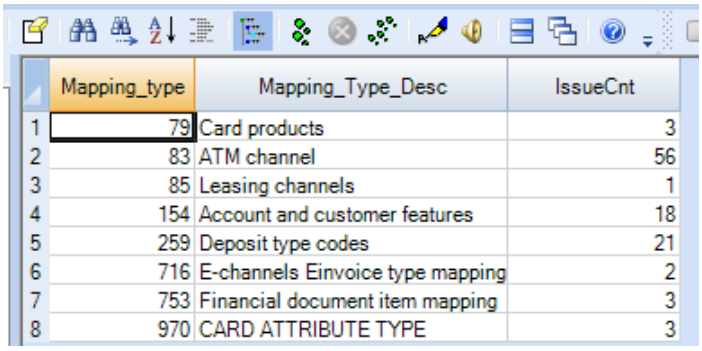
Üldjuhul protsessid on võimelised ise kaardistama vasteid allika andmete ja andmeaida andmete vahel ning määrama allika andmetele andmeaidas reeglipäraseid tüüpe ning selgitusi. Paraku aegajalt tekkivad erandid, mis vajavad manuaalset sekkumist ning defineerimist. Antud näite puhul autor soovib testida raportite loomise funktsionaalsust ning võimalust automatiseerida andmeaida spetsiifilisi seired.

Ülalöeldu teostamiseks autor koostas SQL – skripti, mis pärib Teradatas kaardistamata vasteid, mida on võimalik edastada Excel formaadis erinevate valdkondade andmespetsialistidele andmeaidas (Joonis 27).

```
SELECT * FROM (
SELECT UM.MTYPE AS MTYPE, ST.MAPPING_TYPE_DESC as MAPPING_TYPE_DESC, COUNT(UM.MTYPE) As COUNTER
FROM EDW.T0002_MAPPING_TYPE as ST RIGHT JOIN (
SELECT MAPPING_TYPE_CODE AS MTYPE FROM DWH.T0001_MAPPING T0001
WHERE
T0001.Mapping_Comment LIKE '%UNRESOLVED%'
OR T0001.change_user LIKE '%unresol%'
UNION ALL
--20040302 Kr Lisame ka tundmatute kanalite kontrolli
SELECT MAPPING_TYPE_CODE AS MTYPE
FROM DWH.T0001_MAPPING T0001
WHERE
T0001.Mapping_Type_Code = 54
AND T0001.ID = 0
AND T0001.Mapping_Host_Id=10
--ORDER BY 7, 5, 1, 2, 8
) AS UM
ON ST.MAPPING_TYPE_CODE = UM.MTYPE
GROUP BY UM.MTYPE, ST.MAPPING_TYPE_DESC) MC
WHERE MC.MTYPE not in (17,85,235) --mapping_type_end_date is filled
```

Joonis 27 SQL skript kaardistamata vastenduste tuvastamiseks

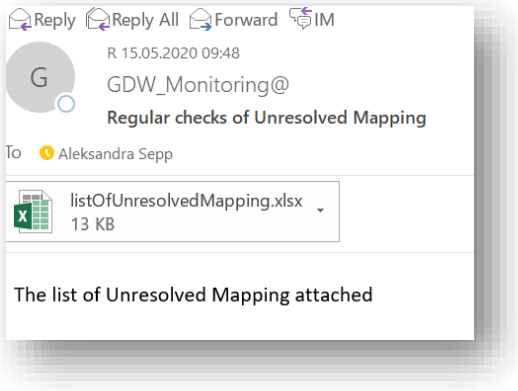
Päringu tulemus Teradata SQL Assistent rakenduses näeb välja selliselt (Joonis 28):



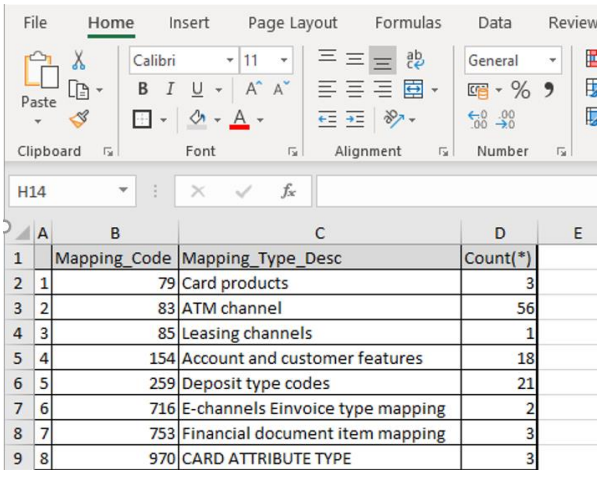
	Mapping_type	Mapping_Type_Desc	IssueCnt
1	79	Card products	3
2	83	ATM channel	56
3	85	Leasing channels	1
4	154	Account and customer features	18
5	259	Deposit type codes	21
6	716	E-channels Einvoice type mapping	2
7	753	Financial document item mapping	3
8	970	CARD ATTRIBUTE TYPE	3

Joonis 28 SQL-päringu tulemus Teradata Assistant'is (kaardistamata vastendused)

Autor moodustas Jenkins platvormil uue töökäsu, mis pärib Teradata andmebaasist kaardistamata vastendusi ning edastab neid Excel formaadis mailile (Joonis 29 ja Joonis 30).



Joonis 29 Kaardistamata vastete raport edastatud meilile



	A	Mapping_Code	Mapping_Type_Desc	Count(*)
1				
2	1	79	Card products	3
3	2	83	ATM channel	56
4	3	85	Leasing channels	1
5	4	154	Account and customer features	18
6	5	259	Deposit type codes	21
7	6	716	E-channels Einvoice type mapping	2
8	7	753	Financial document item mapping	3
9	8	970	CARD ATTRIBUTE TYPE	3

Joonis 30 Kaardistamata vastete automaatteavitus, Excel faili sisu

Analoogselt testitud töökäsule saame tulevikus teostada erinevaid spetsiifilisi andmete seireid ning edastada raporteid vastavatele spetsialistidele andmeaidas. Autor soovib rõhutada, et käesolev lahendus ei enneta kaardistamata vastenduste tekkimise probleemi, kuid tunduvalt säästab andmete spetsialistide tööaega, edastades värsket informatsiooni päevasel baasil. Hilisema tagasiside põhjal on võimalik kohandada rapordi formaati vastavalt vajadustele.

Kokkuvõte

Käesoleva lõputöö eesmärgiks oli koguda analüüsi jaoks sisendeid, teostatud analüüsi põhjal välja valida ühtne platvorm andmeaida süsteemi korrasoleku kontrolliks, mis võimaldaks optimeerida halduse rolli täitvate spetsialistide igapäevaseid tööülesandeid ning tagada andmeaida süsteemide pidev tööd. Samuti teostada tehnilist testimist pakutud platvormile (ingl. *proof of concern*), mille alusel teha järeldusi valiku õigsuse kohta.

Esimese etapi käigus töö autor kohtus erinevate osakondade esindajatega ning kogus sisendeid nii halduse rolli optimeerimise vaatenurgast kui ka manuaalsete toimingute automatiseerimise perspektiivist. Informatsioonikogumine kestis ligi pool aastat ning kaasatud oli üle 7 erineva meeskonna. Kaardistatud sisendite alusel autoril oli vaja sõnastada nõudeid analüüsitavate platvormide sobilikkuse osas ning seejärel uurida, millised lahendused on vaadeldavas üksuses kasutusel. Peale dokumentatsiooniga tutvumist oli tarvis teostada kasutusel olevatele platvormidele analüüs loodud mõõdikute kaartide abil. Järgmisena oli teostatud võrdlus kahe parima platvormi vahel mille tulemusena valiti välja ühtne platvorm halduse rolli täitvate spetsialistide manuaalse töö automatiseerimiseks. Valituks osutus vabavaraline platvorm – Jenkins, mis vastas kõikidele püstitatud nõuetele.

Jenkins platvormil põhineva lahenduse üldvaade on lõputöö autori poolt koostatud ja esitatud töö eelviimases peatükis. Samuti, käesoleva lõputöö raames autoril õnnestus rakendada valitud platvormil töökäske süsteemi olulisemate parameetrite automaatseks kontrolliks ning erinevate teavituste väljasaatmiseks. Viimases peatükis kirjeldatud ja testitud töökäsud on tõestanud platvormi sobivust vastu sõnastatud nõudeid ning autor jätkab käesoleva projekti arendamist ka peale lõputöö kaitsmist.

Kasutatud kirjandus

[1] H. Vallaste, „E-teatmik - ingliskeelsete info- ja sidetehnoloogia terminite seletav sõnaraamat.,“ [Võrgumaterjal]. Available: http://www.vallaste.ee/. [Kasutatud Aprill 2020].

[2] S. Zhao, „What is ETL - Extract-Transform-Load,“ The Experian Data Quality story, 20 Oktoober 2017. [Võrgumaterjal]. Available: https://www.edq.com/blog/what-is-etl-extract-transform-load/. [Kasutatud Märts 2020].

[3] J. A. G. W. C. P. R. T. E. Joseph Voss, „Automated System Health and Performance Benchmarking Platform: High Performance Computing Test Harness with Jenkins,“ ACM Digital Library, Denver, Colorado, 2017.

[4] Kuutõrvaja keskkond - EENet eestikeelsed materjalid tarkvara (eelkõige Linuxi ja muu vaba tarkvara) ning Interneti kohta., „Slurm ressursihaldur,“ [Võrgumaterjal]. Available: https://kuutorvaja.eenet.ee/wiki/Slurm_resursihaldur. [Kasutatud Aprill 2020].

[5] S. S. Natarajan, „Automated Datawarehouse Scheduling and Monitoring,“ Juuni 2019. [Võrgumaterjal]. Available: https://www.researchgate.net/publication/334056597_Automated_Datawarehouse_Scheduling_and_Monitoring. [Kasutatud 10 Aprill 2020].

[6] IBM webpage, „Tivoli Workload Scheduler: User's Guide and Reference,“ [Võrgumaterjal]. Available: https://www.ibm.com/support/knowledgecenter/SSRULV_9.2.0/com.ibm.tivoli.itws.doc_9.2/awsrgmst.pdf. [Kasutatud Aprill 2020].

[7] Oracle Corporation, „Oracle Data Integrator,“ [Võrgumaterjal]. Available: https://www.oracle.com/middleware/technologies/data-integrator.html. [Kasutatud Aprill 2020].

[8] Swedbank AB, „ETL Maintenance - User Guide,“ Tallinn, 2020.

[9] I. Azeri, „What is CI/CD?,“ 4 November 2017. [Võrgumaterjal]. Available: https://dzone.com/articles/what-is-cicd. [Kasutatud 2020 Märts 2020].

[10] Jenkins, „Jenkins User Documentation,“ 2020. [Võrgumaterjal]. Available: https://www.jenkins.io/doc/. [Kasutatud Märts 2020].

[11] „Apache Maven Project,“ The Apache Software Foundation, 2020. [Võrgumaterjal]. Available: https://maven.apache.org/. [Kasutatud 7 Mai 2020].

[12] Jenkins, „Build a Java app with Maven,“ 2020. [Võrgumaterjal]. Available: https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/. [Kasutatud 7 Mai 2020].

[13] J. F. Smart, „The Serenity Reference Manual,“ 15 Juuli 2016. [Võrgumaterjal]. Available: http://thucydides.info/docs/serenity-staging/. [Kasutatud 7 Mai 2020].

[14] JUnit - Open Source Software, „JUnit 4,“ 1 Jaanuar 2020. [Võrgumaterjal]. Available: https://junit.org/junit4/. [Kasutatud 7 Mai 2020].

[15] Opensource.com (Red Hat Inc.), „What is Docker?,“ 2020. [Võrgumaterjal]. Available: https://opensource.com/resources/what-docker. [Kasutatud Aprill 2020].

[16] Guru99 - IT koolituste andmebaas, „ETL vs ELT: Must Know Differences,“ 2020. [Võrgumaterjal]. Available: https://www.guru99.com/etl-vs-elt.html. [Kasutatud 15 Aprill 2020].