TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Devon Nathaniel Gawley
179043IADB

Mikroteenuste andmevoo optimeerimine Katana Technologies OÜ näitel

Bakalaureusetöö

Juhendaja: Margus Sumla
MSc

Tallinn 2024

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: Devon Nathaniel Gawley

04.01.2024

Annotatsioon

Käesolev lõputöö keskendub Katana Technologies OÜ mikroteenuste arhitektuuri põhise veebirakenduse "Katana" andmevoogude optimeerimise analüüsile ja selle tulemusel lahenduste väljatöötamisele. Töö autor kirjeldab olemasolevat süsteemi, analüüsib seda kasutades nii avalikke allikaid kui ka spetsiifilisi meetodeid, ning tuvastab probleemid seoses andmete replikeerimisega mikroteenuste vahel. Need probleemid väljenduvad andmebaaside jõudluse aeglustumises ja andmevahetuse keerukuses, mis põhjustab süsteemi efektiivsuse langust.

Töö käigus selgub, et olemasolev käsitsi andmete replikeerimise protsess on ebaefektiivne ja aeganõudev, eriti arvestades süsteemi kasvavaid nõudmisi ja andmemahu suurenemist. Autor kavandab lahenduse, mis hõlmab üleminekut käsitsi andmete replikeerimisest automaatsele, kasutades selleks KafkaConnecti ja Debeziumi tehnoloogiaid.

Lahenduse juurutamisega kirjeldatakse uusi protsesse ja tehnilist arhitektuuri, mille eesmärgiks on optimeerida andmevoogusid, suurendada süsteemi efektiivsust ning vähendada tööjõu- ja infrastruktuurikulusid. Lõputöö tulemused on analüüsitud püstitatud probleemide ja mõõdikute alusel, peamiselt keskendudes andmevoogude latentsusele, läbilaskevõimele ja kulude kokkuhoiule.

Lõputöö on kirjutatud eesti keeles ja sisaldab teksti 23 leheküljel, 6 peatükki, 13 joonist ja 2 tabelit.

Abstract

Optimizing Microservices Data-pipelines by the Example of Katana Technologies OÜ

The purpose of this thesis was to analyse the microservices-based web application „Katana“ developed by Katana Technologies OÜ and propose optimizations to the services’ data-pipelines. The author analyses the current implemented processes and technologies, and identifies issues related to data replication between microservices. These problems express themselves as bottlenecks in application performance and increases in general developmental complexity.

The analysis reveals inefficiencies in the current manual data replication processes and a solution is proposed the involves replacing the manual replication process in favour of an automatic system by leveraging KafkaConnect and Debezium.

The thesis is written in Estonian and consists of 23 pages of text, 6 chapters, 13 figures, and 2 tables.

Lühendite ja mõistete sõnastik

ACID	*Atomicity Consistency Isolation Durability*, atomaarsus järjepidevus eraldatus püsivus
API	*Application Programming Interface*, reeglid ja vahendid rakendusprogrammi suhtluseks teiste süsteemidega
AWS	*Amazon Web Services*, Amazoni pilveteenuste platvorm
CDC	*Change Data Capture*, andmemuudatuste püüdmine
CI	*Continous Integration*, pidevintegratsioon
DBZ	*Debezium*, Debezium
JS	*JavaScript*, JavaScript
RDBMS	*Relational Database Management System*, relatsioonandmebaasi juhtimissüsteem
RDS	*Relational Database Service*, relatsioonandmebaasi teenus
SaaS	*Software as a Service*, tarkvara teenusena
SKU	*Stock Keeping Unit*, laoseisu ühik
SQL	*Structured Query Language*, struktureeritud päringu keel
UOM	*Unit of Measurment*, mõõtühik

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Mikroteenuste arhitektuur on tänapäeval tarkvara kujundamisel populaarne lähenemisviis. Mikroteenused on tarkvaraarenduses muster, mis põhineb teenuste ning vastutuse eraldamisel, organiseerides rakenduse kui lahtiselt ühendatud teenuste kogumi. See annab võimaluse rakendusi väiksemateks osadeks jagada, mis omakorda aitab kaasa mõistetavusele, arendatavusele, testimise ja vastupidavuse arhitektuuri erosioonile. [1] Mikroteenuste peamisteks eelisteks on skaleeritavus, vastupidavus ja vastutuse eraldamine, mis on olulised elemendid tõhusa ja paindliku tarkvarasüsteemi loomisel. [2]

Katana Technologies OÜ veebirakendus Katana on välja töötatud mikroteenuste arhitektuuri alusel, kus iga teenus omab eraldiseisvat andmebaasi ja vastutab konkreetsete äriprotsesside eest. See struktuur loob vajaduse teenuste vahelise andmete pärimise järele, mis võib omakorda põhjustada andmebaaside jõudluse halvenemist ja mõjutada rakenduse üldist efektiivsust. [3]

Andmete replikeerimine mikroteenuste arhitektuuris on oluline protsess, mis suurendab süsteemi usaldusväärsust, võimaldab tõrgete korral kiiret ümberlülitumist ja parandab jõudlust. [4] Katana veebirakenduses toimub hetkel andmete replikeerimise seadistamine käsitsi, mis võib olla aja- ja ressursimahukas ning suurendab vigade tekkimise riski.

Antud lõputöö eesmärk on asendada käsitsi replikeerimine automaatse lahenduse vastu, kasutades tehnoloogiaid ning protsesse, mis võimaldavad andmete automaatset kirjutamist andmebaasidesse. Üleminek automaatsele replikeerimisele mitte ainult ei paranda andmevoogude efektiivsust, vaid aitab ka vähendada vigade riski ning hallatavat koodi mahtu. Antud lahendus pakub see lähenemine paindlikumat arhitektuuri, mis võimaldab mikroteenustel kasutada optimeeritud andmebaase, suurendades sellega arendusmeeskondade agiilsust. [4]

2 Taust

Käesolevas peatükis tutvustab autor Katana Technologies OÜ-d ja selle ettevõte keskest toodet – Katana veebirakendus. Alustatakse ettevõte tutvustuse ning ülevaatega, käsitledes ettevõte ajalugu, missiooni ning turupositsiooni. Seejärel liigub autor edasi Katana veebirakenduse analüüsile. Autor seletab Katana veebirakenduse peamisi funktsioone ja kuidas need aitavad tootjatel oma tegevusi efektiivsemalt läbi viia ning lõpuks keskendub autor veebirakenduse andmevoo kirjeldusele.

2.1 Ettevõtte kirjeldus

Katana Technologies OÜ on tarkvaraettevõte, mis asutati kuus aastat tagasi, pakkudes kaasaegseid lahendusi tootmise planeerimise ja juhtimise valdkonnas. Ettevõte loodi 2017. aastal.

Katana Technologies OÜ peamine toode, Katana veebirakendus, on välja ehitatud eesmärgiga pakkuda tootjatele mugavaid tööriistu oma tootmisprotsesside haldamiseks reaalajas. Katana veebirakenduse kaudu on paljud ettevõtted suutnud muuta oma tootmist ning planeerimist palju lihtsamaks ja efektiivsemaks. See aitab ettevõtete klientidel oma tootmisprotsesside üle saavutada suuremat läbipaistvust ja kontrolli.

2.2 Katana veebirakenduse kirjeldus

Katana veebirakenduse põhifunktsioonid keskenduvad tootmisprotsesside optimeerimisele, varude haldamisele, tellimuste jälgimisele ja andmeanalüütikale. Need on esmatähtsad vajadused tootjatele oma tootmisprotsesse paremini juhtimiseks ja jälgimiseks. Katanal on avalikult kättesaadavad materjalid, mis kirjeldavad nende lahendust ja selle kasutamist. [5]

Katana veebirakenduse eesrakendus on loodud kasutades ReactJS raamistikku, koos ReduxSaga liidesega, et haldad rakenduse olekut ja tagada sujuva kasutajakogemuse. Tagarakenduse poolel on võetud kasutusele NodeJS koos PostgreSQL andmebaasiga, mis tagab tõhusa andmetöötluse ja halduse. Lisaks on integreeritud Apache Kafka, mis toetab andmevoogude haldamist ja tagab süsteemi skaleeritavuse ja jõudluse.

Pilvehalduse ja pideva integratsiooni valis Katana Heroku, mis pakub paindlikku ja skaleeritavat infrastruktuuri. Heroku võimaldab kiiret rakenduse juurutamist ja haldust, toetades nii arendusprotsessi kiirendamist kui ka süsteemi stabiilsust.

2.3 Katana andmevoo kirjeldus

Antud peatükis kirjeldab autor Katana veebirakenduse andmevoogu puudutavaid tehnoloogiad. Autor toob esile Apache Kafka rolli sündmuste edastamisel ja andmevoo haldamisel. Samuti tutvustab autor *outbox* mustrit, mis on oluline komponent efektiivse ja usaldusväärse andmevoo loomisel, mis annab võimaluse jälgida andmete muutusi ning kindel olla süsteemi töövõimes ärikriitilisi protsesse läbi viies.

2.3.1 Apache Kafka

Apache Kafka [6] on avatud lähtekoodiga voogedastusplatvorm, mis on välja töötatud andmevoogude tõhusaks käsitlemiseks reaalajas. Apache Kafka sai oma alguse LinkedIn'is, et lahendada probleemi suurte andmekogumite töötlemiseks kiirelt ning usaldusväärselt. Apache Kafka on tänaseks kasutusel paljudes erinevates organisatsioonides. Kafka peamine ülesanne on toimida rakenduse andmevoo keskpunktina, mis võimaldab mugavalt andmeid luua, salvestada, töödelda ja analüüsida.

Apache Kafka arhitektuur koosneb *produceritest,* mis tekitab andmeid, *consumeritest*, mis neid andmeid töötlevad, ning *topicutest*, mis on andmevoogude kanalid ning võimaldavad andmeid eraldada üksteisest. Kafka aitab arendajaid paljude andmetöötlusega seotud väljakutsetega nagu logide kogumine ning reaalajas sündmuste jälgimine.

Apache Kafka võimaldab rakendada ka *event sourcing*'ut, mis aitab andmevoogudes toimunud sündmuste põhjal analüüsida süsteemi seisundit. See tähendab, et igal ajahetkel on võimalik pakkuda andmeid äriprotsesside jälgimiseks ja auditeerimiseks.

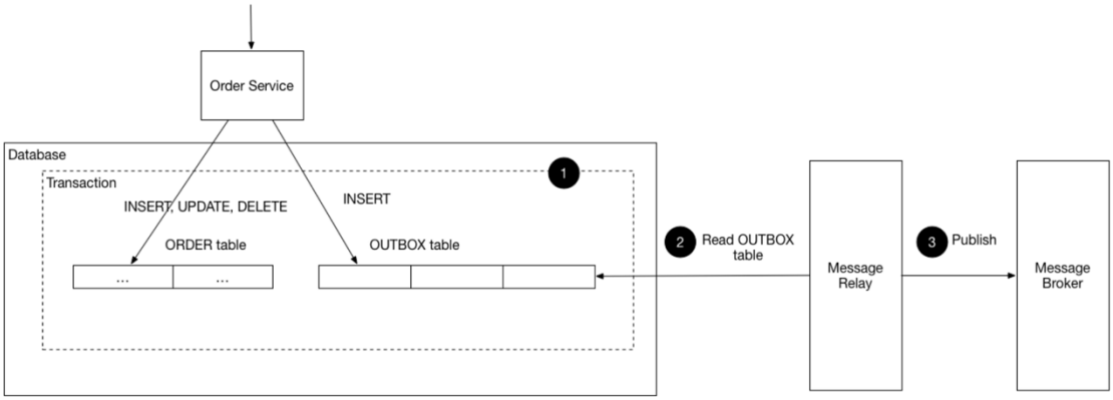
Erilist tähelepanu pöörab Apache Kafka jõudlusele ja skaleeritavusele. See teeb Kafkast suurepärase lahenduse nii väikestele kui ka suurtele veebirakendustele, kes vajavad kiiret ja usaldusväärset andmetöötlust. Kafka lahendab peamiselt suurte andmevoogude haldamise ja analüüsimise probleemi, kuid pakub ka kõrget läbilaskevõimet,

vastupidavust andmekadude vastu ja võimalust andmeid replikeerida vastavalt ettevõtte vajadustele. Katana veebirakenduses kasutatakse Apache Kafkat koos *outbox* mustriga.

2.3.2 Outbox muster

Outboxi muster on mikroteenuste arhitektuuris välja töötatud protsess, mis aitab lahendada keerulisi andmejärjepidevuse ja andmesünkroniseerimise probleeme. Outbox muster on kasutatud paljudes suurtes hajussüsteemides, kus andmete õigsus ning usaldusväärsus on kriitilise tähtsusega. [7] Outboxi mustri põhiolemus seisneb selles, et kõik andmebaasi rea muudatused salvestatakse esmalt ajutisse "outbox" tabelisse enne nende edastamist teistesse süsteemi osadesse või teenustesse.

See lähenemine võimaldab süsteemil jälgida ja hallata andmete liikumist, tagades, et kõik andmemuudatused on usaldusväärselt edastatud ja järjepidevalt sünkroniseeritud. Tänu integratsioonile *Change Data Capture* (CDC) tööriistadega, nagu Debezium, ja andmevoogude platvormidega, nagu Apache Kafka, suudab *outboxi* muster pakkuda tõhusat lahendust andmete voogedastuseks ja järjepidevuseks, mis on hädavajalik kaasaegsetes tagarakendussüsteemides. Selle mustri kasutamine aitab mitte ainult tagada andmete ühtsust ja terviklikkust, vaid pakub ka suurt paindlikkust ja skaleeritavust, tehes sellest hinnatud lahendus paljudes andmekesksetes rakendustes. [8, 7] Joonis 1 kirjeldab visuaalselt outbox mustri töövoogu.



Joonis 1. Outbox mustri skeem.

Katana veebirakenduse kontekstis oleme loonud igasse teenuse andmebaasi, mis kasutab outboxi mutstrit, tabeli *outbox*, mille struktuur koosneb nendest väljadest. Tabel 1. kirjeldab *outbox* tabeli andmebaasi struktuuri.

Tabel 1. Outbox tabeli andmebaasi struktuur.

Veerg	**Andmetüüp**	**Semantika**
id	Integer	Outbox tabeli rea primaarvõti.
key	string	Kafka sõnumi identifikaator.
value	string	Sõnumi väärtus.
topic_name	string	Kafka teema nimetus, kuhu sõnumit saadetakse.
published_date	timestamp	Sõnumi saatmise hetkel lisatud ajaline väärtus tähistamaks, et antud reaga on tegeletud.

Outbox mustri rakendamisel Katana veebirakenduses oleme saavutanud olulisi eeliseid. Tabeli struktuur on kujundatud nii, et see talletab kõik vajalikud sündmused, mida hiljem Kafka sõnumitena edastatakse. See eraldamine võimaldab sisemiste API päringute tulemusi tõhusalt hallata, eraldades need Kafka sõnumite produtseerimisest, mis omakorda aitab vältida sõnumite aegumist ja tagada süsteemi sujuvam töö. Selline lähenemine vähendab oluliselt sõnumite kaotsimineku riski ja parandab süsteemi reageerimisvõimet.

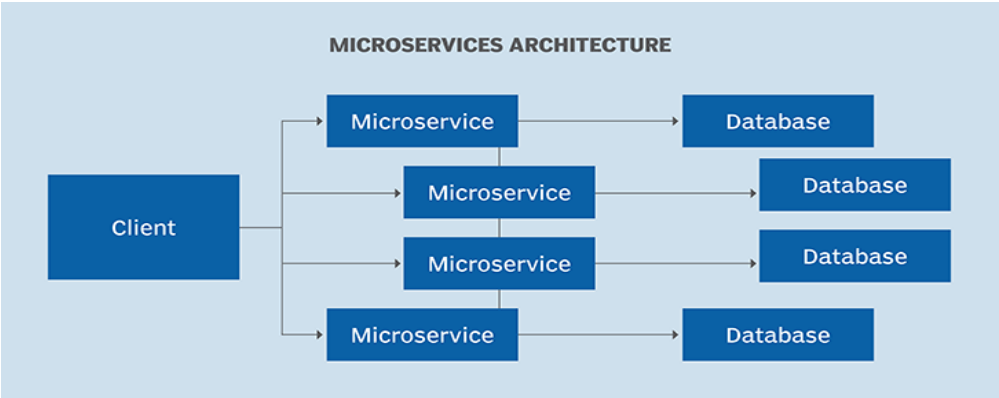
3 Probleemi analüüs

Antud peatükis keskendub autor Katana veebirakenduse olemasolevale andmevoogude haldusele. Autor toob esile kaks kasutuses olevat mikroteenust – "Manufacturing" ja "Items" – ning kirjeldab nende teenuste omavahelisi andmevahetusprotsesse. Autori eesmärk on tuua välja praegused kitsaskohad ja väljakutsed, mis on seotud andmete käsitsi replikeerimisega, ning analüüsida, kuidas võiks "KafkaConnecti" kasutuselevõtt koos "Debeziumiga" neid probleeme lahendada. See analüüs keskendub mõju üldisele äritegevusele ning kuidas tekitatud muudatused aitavad kaasa arendusprotsesside optimeerimisele ja ettevõtte kulude vähendamisele.

3.1 Olemasoleva süsteemi kirjeldus

Katana Technologies OÜ veebirakenduse "Katana" keskmes on mikroteenuste arhitektuur, mis võimaldab rakenduse arendusprotsessi teha võimalikult paindlikuks ning skaleeritavaks. Teenused ning vastutusvaldkonnad on lihtsasti eraldatavad ning sõltumatud üksteisest.

Katana veebirakendus koosneb hetkel ligi 80st mikroteenusest. Iga teenus, nagu näiteks "Manufacturing" või "Items", vastutab kindlate funktsioonide eest. Mikroteenuste arhitektuur annab viisi eraldada teenuste vastutusvaldkondi, kus iga komponent on sõltumatu üksteisest. See võimaldab uuendusi vastu võtta ja skaleeruda sõltumata teiste mikroteenuste kättesaadavusest või reageerimiskiirusest. [9] Joonis 2 pakub visuaalset näidet kuidas mikroteenuste arhitektuur välja näeb.



Joonis 2. Mikroteenuste arhitektuuri visuaalne kujutis [10]

"Manufacturing" mikroteenus keskendub tootmisprotsessidel ning vastutab tootmisega seotud tegevuste ja andmete eest. See sisaldab ka ärikriitilist infot tootmisülesannete, tootmisplaanide ja tootmiste oleku kohta.

"Items" mikroteenus on loodud tooraine ja toodete haldamiseks. See tegeleb tooraine varude, toodete spetsifikatsioonide ja varude taseme jälgimisega.

3.2 Andmevahetus mikroteenuste vahel

Oluline osa Katana Technologies OÜ veebirakendusest on andmevahetus erinevate mikroteenuste vahel. Eriti kriitiline on see "Manufacturing" ja "Items" mikroteenuste puhul, kus tootmisprotsesside sujuvus sõltub otseselt andmete järjepidevast ja täpsest vahetusest. Mikroteenuste arhitektuuris ei ole andmevahetus nii lihtne kui monoliitsetes süsteemides, kuna see nõuab uusi lähenemisi ja mehhanisme, et hoida süsteemi stabiilsena ja funktsionaalsena**.** [11]

Mikroteenuste puhul on iga teenus seadistatud omaette andmebaasiga, mis võimaldab neil sõltumatult skaleeruda ilma, et see põhjustaks probleeme teistes süsteemi osades. Selline lähenemine toob kaasa väljakutsed seoses oleku halduse, väliste andmete ja andmehaldusega mikroteenustes. [12]

Andmebaasi halduse optimeerimine mikroteenuste rakendustes nõuab sobiva rakenduse kujunduse ja andmebaasi tehnoloogia kombinatsiooni, mis peab keskenduma ACID atribuutidele: *atomicity* (atomaarsus), *consistency* (järjepidevus), *isolation* (eraldatus) ja *durability* (püsivus). [12] See tähendab, et andmebaasioperatsioonid ei tohi kunagi jääda poolikuks. Iga tehing peab toimima ilma teisi mõjutamata ning nii püsivad andmete õigsus ja süsteemi vastupidavus.

Lisaks tavalistele RDBMS-idele (Relational Database Management System) võib kasutada ka mittetraditsioonilisi andmebaase nagu NoSQL, mis võib edastada terviklikke andmeobjekte korraga kõigile teenustele. Sellegipoolest ei paku see mudel täielikku lahendust, sest kaotades ühe andmeobjekti, võime kaotada kogu rakenduse tehingud, mis toob kaasa vajaduse arhitektuuriliste mustrite järele, mis käsitlevad spetsiifiliselt mikroteenuste ja andmebaasi suhteid. [13]

3.3 Olemasolev andmete replikatsiooni protsess

Olemasolevas süsteemis kasutatakse andmete replikeerimist, et tagada andmete kättesaadavus ja kooskõla erinevates mikroteenuste vahel. Näiteks, "Items" mikroteenus omab "variants" tabelit, mis sisaldab andmeid tootevariantide kohta. Antud tabeli replikeerimiseks luuakse uus tabel "variants_replicated" "Manufacturing" mikroteenuse andmebaasi.

Antud tabeli replikeerimine on tehtud eesmärgiga vähendada teenuste vahelisi ristpäringuid ja võimaldada kiiret juurdepääsu vajalikele ärikriitiliste andmetele teenusesiseselt. Replikatsiooniprotsess toimub läbi eraldi loodud "worker-replication" rakenduse, mis kasutab Kafka consumerit CDC (Change Data Capture) sõnumite jälgimiseks ja andmete sünkroniseerimiseks.

Sellegipoolest on oluline märkida, et olemasoleva süsteemi replikatsiooniprotsess nõuab iga andmebaasi tabeli eraldi seadistamist. Selle protsessi raames on vajalik luua outbox tabelid ja konfigureerida vastavad topicud Apache Kafka jaoks. Käsitsi andmete järjepidevuse ja tõrgeteta sünkroniseerimine protsess on aeganõudev ning nõuab hoolikat seadistamist ning jälgimist.

Kasutades automatiseeritud lahendusi oleks võimalik sellist käsitsi tööd efektiivsemaks muuta. Kasutades tööriistu, nagu Debezium koos KafkaConnectiga, saab automatiseerida andmete replikatsiooni protsessi. Need tööriistad suudavad dünaamiliselt jälgida andmebaasi muudatusi ja genereerida CDC [3.1] sõnumeid ilma vajaduseta iga tabeli jaoks eraldi seadistusi luua. Automatiseerimine mitte ainult ei vähenda vigade esinemise võimalust, vaid ka vähendab oluliselt aega ja ressursse, mis on vajalikud replikatsiooniprotsessi haldamiseks.

CDC, ehk *Change Data Capture,* on muster, mis jälgib andmete muudatusi ja teavitab teisi süsteeme ning teenuseid, kui antud andme tabeliga on muudatus toimunud. [14] Andmed on tootmisvaldkonnas äritegevuse nurgakiviks ja kuna need on konstantselt muutuvas seisundis, peavad tootjad suutma nende muutustega sammu pidada.

CDC platvormid, nagu Debezium, jälgivad andmebaasi tehingulogi, et edastada spetsiifilised andmebaasiridade muudatused Apache Kafka *topicutesse*. See lähenemine pakub mitmeid eeliseid võrreldes päringupõhise protsessiga, näiteks:

- Kõik muudatused on jäädvustatud: CDC on loodud selleks, andmebaasimuudatuste säilitamiseks. Ilma CDC-ta võivad vahepealsed muudatused ja uued andmed, nagu uuendused ja kustutamised, kahe päringu vahel kaduma minna. [15] Kui tekib andmekadu on hiljem võimalik kõik need jäädvustatud muudatused uuesti läbi käia.

- Madal ülekoormus: CDC ja Apache Kafka kombinatsioon pakub reaalajalähedast andmemuudatuste edastust. Sellega välditakse suurenenud protsessorikoormust tihedast küsitlusest. [16]

- Andmemudelile ei ole mõju: CDC kasutamisel ei ole enam vaja ajatempli veerge viimase andmeuuenduse määramiseks. [16]

3.4 Andmevoogude väljakutsed

Katana veebirakenduse puhul on mikroteenuste arhitektuuri kasutusele võtmine toonud kaasa mitmeid eeliseid ja mugavusi, kuid on ka esile kerkinud mitmeid väljakutseid, kõige rohkem just andmevoogude haldamisel. Need väljakutsed mõjutavad süsteemi kasutajakogemust ning ärilist efektiivsust.

Antud peatükis käsitleb autor Katana veebirakenduse andmevoost tulenenud kitsaskohti ning väljakutseid.

3.4.1 Andmete replikeerimise probleemid

Praegu kasutab "Katana" käsitsi andmete replikeerimise protsessi, et tagada andmete kättesaadavus eri mikroteenustes. Näiteks "Items" mikroteenuse "variants" tabeli replikeerimine "Manufacturing" mikroteenuses tekitab mitmeid probleeme:

- Jõudlusprobleemid: Suure andmemahu korral võib replikeerimisprotsess muutuda aeglasemaks, põhjustades viivitusi andmete uuendamisel ja päringutele vastamisel. [17]

- Koordineerimise keerukus: Käsitsi replikeerimise puhul on keeruline tagada andmete järjepidevust ja kooskõla, eriti arvestades mitme andmebaasi kasutamist. [17]

3.4.2 Ressursside ja kulude küsimused

Käsitsi andmete replikeerimine Katana veebirakenduses mõjutab otseselt ettevõtte kulusid, kuna see nõuab olulist ressursside investeeringut.

- Tööjõukulud: Arendajate ajaressurss on esmatähtis, eriti kui on tegemist ajamahukate ülesannetega nagu andmete replikeerimisega seotud funktsionaalsuste arendamine ning hooldamine. Vabanenud aja saaks efektiivsemalt kasutada suunates uute väärtust loovate funktsioonide arendamisele.

- Infrastruktuurikulud: Käsitsi replikeerimise protsessid nõuavad lisaservereid ja andmebaaside ressursse. Pilveteenuste integreerimise puhul tähendab see otsest kulu suurenemist, kuna hinnastamine põhineb ressursside kasutusel. [18]

- Ebaefektiivne ressursikasutus: Käsitsi replikeerimine võib põhjustada ressursside alakasutamist madala nõudluse perioodidel ja ülekoormust kõrge nõudluse perioodidel, muutes kulud ettearvamatuks. [19]

3.4.3 Süsteemi hooldatavuse ja skaleeritavuse probleemid

Käsitsi loodud replikatsioonisüsteemide hooldamine ja skaleerimine võib olla keeruline olukorras kus süsteem laieneb ja andmemaht suureneb. See nõuab pidevat tähelepanu ja potentsiaalselt keerukaid muudatusi olemasolevas koodibaasis.

Olemasolev käsitsi andmete replikeerimissüsteem Katana veebirakenduses tekitab mitmeid hooldatavuse ja skaleeritavusega seotud probleeme:

- Hooldatavuse keerukus: Käsitsi loodud replikatsiooniga seotud arenduste hooldamine ja uuendamine on aeganõudev ja keeruline, eriti arvestades vajadust tagada andmete kooskõla ning järjepidevust. [19]

- Skaleeritavuse piirangud: Käsitsi replikeerimise lähenemine ei pruugi olla piisavalt paindlik, et toime tulla suurenevate andmemahude ja kasutajate arvuga. Võib tekkida olukordi kus, süsteemi laiendamine võib nõuda liialt ümberkorraldusi ja ressursse. [19]

- Muudatuste haldamise keerukus: Iga muudatus andmebaasis või rakendusloogikas nõuab replikatsiooniprotsessi põhjalikku ülevaatamist ja kohandamist, suurendades vigade riski ja arendusaega. [19]

3.4.4 Võimalikud andmete kaotsiminekud

Käsitsi replikeerimise puhul on suurenenud risk andmete kaotsiminekuks või vigade tekkimiseks replikatsiooniprotsessis, millega kaasneb suurenenud oht andmete ebatäpsuse vastu.

- Sünkroniseerimise vead: Inimlikud vead replikatsiooniloogika kirjutamisel või muutmisel võivad põhjustada andmete sünkroniseerimise probleeme, nagu andmete dubleerimine või vananenud andmete esinemine. [20]

- Andmete järjepidevuse puudumine: Käsitsi süsteemis võivad tekkida viivitused andmete uuendamisel, mis võivad viia olukorrani, kus kasutajad näevad vananenud või vastuolulist teavet.

- Kaotsiminek kriisiolukordades: Süsteemi tõrgete või katkestuste korral võib käsitsi replikeerimine osutuda ebausaldusväärseks, suurendades andmete kaotsimineku riski.

4 Lahenduse kavandus ning analüüs

Antud peatükis keskendub autor Katana veebirakenduse andmevoogude optimeerimise lahendusele. Autor lähtub töös eelnevalt analüüsitud ning tuvastatud väljakutsetest. Autori eesmärk on juurutada "Manufacturing" ja "Items" mikroteenused juba olemasolevasse KafkaConnecti infrastruktuuri, mille on välja töötanud Katana andmetiim.

Autor tutvustab KafkaConnecti infrastruktuuri põhielemente, analüüsib selle konfiguratsioone ja kirjeldab, kuidas need aitavad kaasa mikroteenuste sujuvamale andmevahetusele. Autor käsitleb ka konkreetseid muudatusi, mida rakendati mikroteenustes, et olemasolevad arendused uude KafkaConnecti infrastruktuuri üle viia. Autor viib ka läbi ka vanade andmete replikeerimise eemaldamine.

Uute tehnoloogiate ja protsesside juurutamisel hindab autor lahenduse eeliseid ja võimalike riske, pakkudes ülevaadet, kuidas KafkaConnecti kasutuselevõtt Katana veebirakenduses toetab autori eelnevalt välja toodud eesmärke andmevoogude optimeerimisel. Lõpetuseks, pakub autor järeldusi ja tulemusi, rõhutades, kuidas see lahendus on kaasa aidanud ettevõtte pikaajalistele püüdlustele ja tehnoloogilisele arengule.

4.1 KafkaConnecti tööpõhimõtted

KafkaConnect on Apache Kafka komponent, mis loodi, et lihtsustada andmete liikumist erinevate süsteemide ja andmeallikate vahel. Selle tööpõhimõtted põhinevad reaalajal toimival andmete voogude integreerimisel, mis võimaldab süsteemil reageerida ja kohaneda muutustega kiiresti ja tõhusalt. [21]

KafkaConnecti infrastruktuuri peamine ülesanne on andmete edastamine mikroteenuste ja nende vastavate andmebaaside vahel võimalikult väikese viivitusega. See tulemust saavutatakse KafkaConnecti võimega jälgida andmebaasides toimuvaid muudatusi reaalajas, kasutades CDC [3.3] mustrit. See tähendab, et iga kord, kui andmebaasis tehakse muudatus (nt rida lisatakse, kustutatakse või uuendatakse), peegeldub see muudatus automaatselt ka replikeeritud andmebaaside tabelites.

KafkaConnect võimaldab andmete integreerimist andmeallikast sihtpunkti (*source to sink*) ja sihtpunktist andmeallikasse (*sink to source*). Katana veebirakenduses kasutatakse peamiselt allikast sihtpunkti suunalist andmevoogu. Andmed liiguvad mikroteenuste andmebaasidest KafkaConnecti integratsiooni kaudu teistesse süsteemi osadesse, kus replikeeritud andmeid äriprotsesside jaoks vaja läheb.

KafkaConnecti üks olulisematest omadusest on CDC (*Change Data Capture)*. CDC võimaldab KafkaConnectil tuvastada ja reageerida muudatustele andmebaasides kohe kui andmebaasis peaks muudatus toimuma. Näiteks kui "Items" mikroteenuse andmebaasis lisatakse uus tooraine või kustutatakse täienisti rida ära, siis see muudatus saadetakse koheselt teistesse süsteemi osadesse, sealhulgas "Manufacturing" mikroteenusesse. Antud muudatust on nüüd võimalik rakendada replikeeritud andmebaasides, säilitades andmete ühtsust seotud andmebaasitabelite vahel.

4.2 Muudatuste rakendamine mikroteenustes

Antud peatükis kirjeldab autor protsessi "Manufacturing" mikroteenuse käsitsi replikeerimine viimist üle automaatsele lahendusele. Autor toob lahendusega kaasa arenduslikku paindlikkust ning jõudluse tõusu. Autor vaatleb, kuidas KafkaConnecti juurutamine on osa püüdlusest vähendada projekti arendajate manuaalset töökoormust ning rakenduse jõudlust täiendada.

Järgnevad alapeatükid on jaotatud vastavalt tööprotsessi loogilistele sammudele, mis on dokumenteeritud JIRA piletites. Igas juurutamise sammus esitab autor ülevaate rakendatud muudatustest, nagu replikeeritud andmete tabeli loomine ning vanade käsitsi replikeerimisprotsesside eemaldamine. Juurutamise protsessis arvestab autor tekkivaid eeliseid ning riske.

4.2.1 Protsessi ülevaade ja JIRA piletite tutvustus

JIRA piletid ei ole ainult tööülesannete loetelu, vaid ka projektijuhtimise vahendid, mis tagavad keerulistes arendusprojektides iga sammu selget skoobi ning vastutuse. JIRA süsteem võimaldab ettevõtetel jälgida ülesannete edenemist, dokumenteerida kitsaskohti või lahendusi ning hinnata sammude mõju kasutajatele. [22] JIRA piletid olid jaotatud selliselt:

- *Replicate Variants to manufacturing db through sink connector*: Esimene samm on autoril seadistada uus andmebaasitabel variants _replicated_dbz ja luua *sink connector*, mis tagab andmete reaalajas sünkroniseerimise. [Joonis 7]

- *Change source of variants in manufacturing*: Pärast edukat andmete sünkroniseerimist muudab autor *variants* mudeli andmeallikat, mis suunab selle uue tekitatud tabeli vastu. [Joonis 8. JIRA pilet – Variants mudeli allika ümbersuunamine]

- *Drop manual variants replication from manufacturing*: Viimase sammuna eemaldab autor vananenud käsitsi replikeerimise koodi, et vähendada süsteemikoormust, andmete dubleerimist ja pilveteenuse kulusid. [Joonis 9. JIRA pilet – Käsitsi replikeerimise protsessi eemaldamine]

4.2.2 Uue *sink connectori* loomine ja sünkroniseerimine

Järgmine samm üleminekus automaatsele replikeerimisele Katana veebirakenduse mikroteenustes on uue *sink connector'i* loomine. See samm on vajalik, et võimaldada andmete reaalajas suundumist mikroteenuste vahel ilma käsitsi sekkumiseta. Uue *sink connector'i* loomine ja selle sünkroniseerimine nõuab täpset ja metoodilist lähenemist, et tagada andmete täpsust ja süsteemi usaldusväärsuse peale muudatuste läbi viimist.

Sink connector'i loomise esimene samm on uue tabeli loomine. See uus tabel on replikeeritud andmete sihtkoht. Autori uue tabeli nimeks valitakse „*variants_replicated_dbz*“. Tabeli nimetus *"variants_replicated_dbz"* peegeldab selle funktsiooni ja vastutust Katana veebirakenduse andmebaasi arhitektuuris. Komponentide kombinatsioon nimetuses annab ülevaate tabeli eesmärgist ja kasutatavatest tehnoloogiatest.

- *variants*: See osa nimetusest viitab tabelile mida replikeetakse.

- *replicated*: Lisamine "replicated" tabeli nimetusele viitab sellele, et antud tabeli andmed on replikeeritud.

- *dbz*: Lühend "dbz" on viide Debeziumile ning annab märku, et see konkreetne tabel kasutab Debeziumi võimalusi andmete sünkroniseerimiseks reaalajas.

Tabeli loomise protsess hõlmas migratsiooniskripti [Joonis 13] kirjutamist ning rakendamist. See on tähtis süsteemi andmebaasistruktuuri muutmiseks ning uute andmevoogude haldamiseks. Replikeeritud andmete tabeli andmestruktuur on välja toodud tabelis 2.

Tabel 2. Variants_replicated_dbz tabeli struktuur

Veerg	**Andmetüüp**	**Semantika**
id	integer	Variants_replicated_dbz tabeli primaarvõti.
sku	string	Stock keeping unit – toote variandi unikaalne indentifikaator.
sales_price	string	Toote variandi hind.
purchase_price	string	Toote variandi sisseostmise hind.
item_id	integer	Items tabeli primaarvõti.
factory_id	integer	Factories tabeli primaarvõti.
name	string	Toote variandi nimetus.
name_without_sku	string	Toote nimetus ilma SKU-ta.
uom	string	Toote variandi ühik
is_producible	boolean	Kas toodet luuakse majasiseselt.
is_purchasable	boolean	Kas toodet ostetakse sisse.
item_name	string	Toote nimi.

Peale tabeli loomise migratsiooniskripti käivitamist ning uue tabeli loomise valideerimist on järgmine samm KafkaConnecti *sink connector*'i seadistamine. *Sink connector*'i seadastamine oma olemuselt on kindlate konfiguratsiooniparameetrite ette valmistamine vastavad teenuse nõuetele, nagu näiteks andmebaasi ühenduse andmed, tabeli nimi ning replikeeritud andmeväljad.

Sobilike KafkaConnecti konfiguratsioonide kirjutamine on Katana veebirakenduse andmevoogude optimeerimise keskmes. Järgnevalt analüüsib autor kahte konkreetset KafkaConnecti konfiguratsiooni: *cdc-debezium.manufacturing.json* [Joonis 11] ja *debezium-outbox.manufacturing.json* [Joonis 10]. Nende konfiguratsioonide kaudu on

võimalik täpselt seadistada, kuidas andmeid töödeldakse ja edastatakse "Manufacturing" mikroteenuses, kasutades Debeziumi ja KafkaConnecti.

See konfiguratsioon [Joonis 10] rakendab Debeziumi outbox [2.3.2] mustrit, mida saame mugavalt ära kasutada andmete edastamise jaoks mikroteenuste vahel. *Transforms.outbox.type* viitab *EventRouter* kasutamisele, mis on keskne osa outbox mustris. [23]

Outbox transformatsioonid, nagu *route.by.field* ja *route.topic.replacement*, on seadistatud sündmuste suunamiseks. See tagab, et sündmused suunatakse õigesse Kafka teemasse, võttes aluseks *field*s väljas määratud väärtused.

Mõlemad konfiguratsioonid sisaldavad *heartbeat* seadeid, mis on olulised süsteemi tervise ja järjepidevuse jälgimiseks. Need seadistused tagavad, et andmeühendused püsivad aktiivsed ja toimivad. Joonis 11 kirjeldab kasutatud konfiguratsiooni “Manufacturing“ teenuse CDC [3.3] sõnumi tabeli jaoks.

Kasutades *io.debezium.connector.postgresql.PostgresConnector*, ühendub see konfiguratsioon *PostgreSQL* andmebaasiga, mida "Manufacturing" mikroteenus kasutab. Andmebaasi parameetrid, nagu hostname, port, kasutajanimi ja parool, on dünaamiliselt viidatud, mis tagab paindlikkuse ja turvalisuse.

Konfiguratsioon sisaldab *table.include.list*, mis määrab kindlaks, milliseid tabeleid jälgida. See võimaldab keskenduda ainult asjakohastele andmetele, parandades jõudlust ja vähendades tarbetut andmevoogu.

Transformatsioonid, nagu *extractKey* ja *dropSource*, on olulised andmete struktureerimisel ja puhastamisel, eemaldades mittevajalikke välju ja säilitades olulise informatsiooni.

Mõlemad konfiguratsioonid sisaldavad heartbeat.interval.ms ja heartbeat.action.query parameetreid, mis on olulised süsteemi tervisliku seisundi jälgimiseks. Need seadistused tagavad, et andmeühendused püsivad aktiivsed ja süsteem on järjepidev.

4.2.3 Andmeallika ümbersuunamine

Antud peatükis käsitleb autor "Manufacturing" mikroteenuse *variants_replicated* tabeli andmeallika ümbersuunamist. Pärast KafkaConnecti konfiguratsioonide loomist ning uue andmebaasitabeli ettevalmistamist on tähtis samm mudeli ümber suunamine.

Muudatus toimub mudeli annotatsiooni tasandil. Originaalselt oli mudel defineeritud järgnevalt:

```
@model({settings: {strict: true}, name: 'variants_replicated'})
```

Joonis 3. Variants_replicated mudeli algne annotatsioon.

Selle annotatsiooni kaudu oli mudel seotud algse tabeliga. Nüüd, et suunata andmevoog uuele tabelile, 'variants_replicated_dbz', muudame annotatsiooni järgmiselt:

```
@model({settings: {strict: true}, name: 'variants_replicated_dbz'})
```

Joonis 4. Variants_replicated mudeli annotatsioon peale muudatuste rakendamist.

Pärast muudatuse tegemist on hädavajalik teostada põhjalik testimine. Testimise käigus kontrollib autor, kas uus mudel toimib oodatud viisil ja kas andmete liikumine uude tabelisse on korrektne. Samuti jälgib autor süsteemi jõudlust peale muudatust, et tagada, et eesmärgiks seatud optimeerimine on saavutatud.

Peale muudatuste lõplikku rakendamist on tähtis jälgida veebirakenduse toiminguid tervikuna. Olgugi, et mikroteenused on eraldiseisvad, teatud olukordades võivad muudatused ühes teenuses mõjutada teiste teenuste tööd.

4.2.4 Käsitsi replikeerimise protsessi eemaldamine

Esimene samm käsitsi replikeerimise protsessi eemaldamisel on vana tabeli *variants_replicated* kustutamine. Selleks kirjutas ja rakendas autor migratsiooniskripti. Antud skript tagas tabeli eemaldamist andmebaasist turvaliselt, ilma andmete või süsteemi terviklikkust ohustamata. Joonis 12 [Joonis 12] kirjeldab kasutatud migratsiooniskripti.

Järgmise sammuna eemaldas autor "Manufacturing" mikroteenusest "worker-replication" rakenduse. See rakendus oli varem vastutav andmete replikeerimise eest. Selle

eemaldamine tähendas ka sellega seotud mudelite, repositooriumite ja teenuste kustutamist. See samm oli kriitiline, et vähendada süsteemi keerukust ja halduskoormust.

"Worker-replication" rakenduse ja sellega seotud elementide eemaldamisega saavutasime olulisi eeliseid. Esiteks, vähendasime oluliselt koodi hulka, mille eest Katana veebirakenduse arendajad vastutavad, mis omakorda tähendab vähem hooldustöid ja madalamaid riske vigade tekkeks.

Teiseks, kuna sai eemaldatud pilveteenuse keskkonnas liigse rakenduse, sai autor vähendada Katan pilveteenuste kulutusi. See on näide sellest, kuidas tehnoloogilised optimeerimised võivad kaasa tuua ka majanduslike sääste.

5 Tulemuste analüüs

Järgnevas peatükis keskendub autor rakendatud lahenduse põhjalikule hindamisele Katana veebirakenduses. Pärast KafkaConnecti ja Debeziumi tehnoloogiate edukat juurutamist ning oluliste muudatuste läbiviimist mikroteenustes, on hädavajalik mõõta ja analüüsida nende muutuste mõju süsteemi üldisele toimimisele. Selline analüüs on oluline, et mõista paremini, kuidas tehnoloogilised uuendused on aidanud kaasa süsteemi tõhususele, ning et identifitseerida edasised arendusvõimalused ja potentsiaalsed valdkonnad, kus saab veelgi parendusi teha.

Antud peatüki raames analüüsib töö autor näitajaid nagu andmete läbilaskevõime, andmete latentsus, süsteemi stabiilsust ning kulude optimeerimine. Antud näitajad on olulised, et hinnata tehnoloogiliste uuenduste mõju rakendusele tervikuna ja kindel olla nende väärtuses ettevõttele.

Andmevoogude jõudluse analüüsimisel keskendub autor sellele, kuidas uued tehnoloogiad on aidanud vähendada andmete edastamiseviivitusi ja suurendada süsteemi läbilaskevõimet, mis on eriti oluline reaalajas töötlevates süsteemides.

Kulude optimeerimisel analüüsib autor kuidas tehnoloogilised uuendused on aidanud kaasa ettevõtte püüdmisi kulusid kokku hoida. Need võivad mõjutada tootetiimi või infrastruktuuri kulisid ning inimressurssi efektiivsust. Lõpuks, skaleeritavuse analüüs käsitleb süsteemi võimet kohaneda kasvavate andmemahtude ja kasutajate nõudmistega, mis on tähtsad dünaamilises ning konkurentsitihedas keskkonnas.

Selle tervikliku analüüsi abil püüab autor jõuda järeldustele, mis ei anna mitte ainult tagasisidet rakendatud lahenduste edukuse kohta, vaid pakub ka väärtuslikku sisendit edasiste tehnoloogiastrateegiate kujundamiseks Katana Technologies OÜ-s.

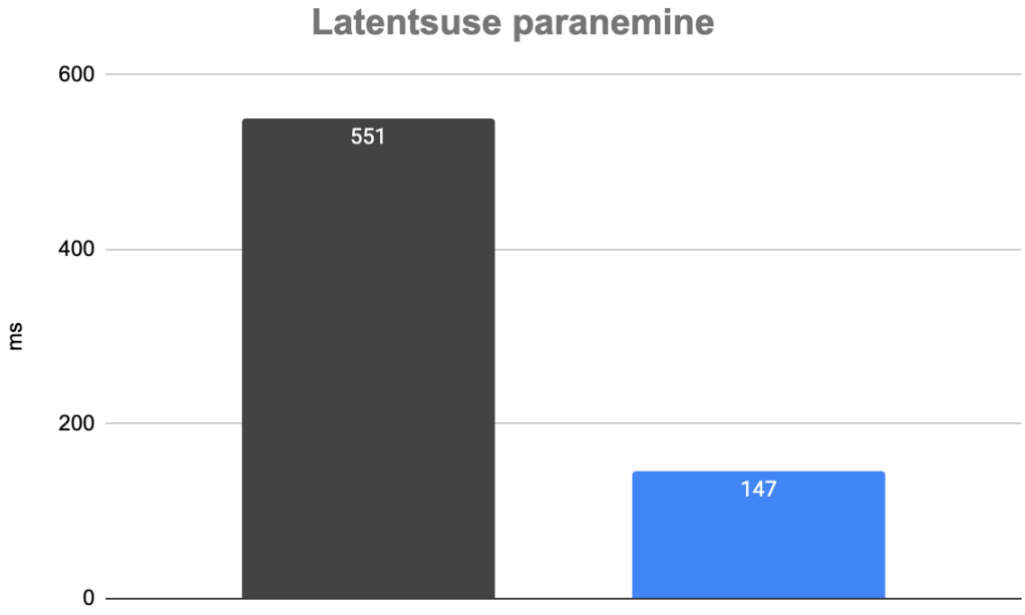
5.1 Andmevoogude jõudluse parandamine

Jõudluse mõõtmiseks Katana veebirakenduses kasutab autor mitmeid tehnikaid ja vahendeid. Latentsuse mõõtmiseks jälgib autor ajatemplid igal sammul andmevoos, alates andmete sisestamisest kuni nende kättesaadavuseni sihtpunktis. Läbilaskevõime hindamiseks kasutab autor stressitestimise tööriistu, mis simuleerivad kõrget andmevoo koormust, et mõõta, kui palju päringuid süsteem suudab töödelda teatud ajas.

5.1.1 Latentsuse analüüs

Enne muudatusi oli keskmine latentsus andmebaasist sõnumitoojale Katana veebirakenduses 551 ms. Pärast KafkaConnecti ja Debeziumi kasutuselevõttu langes see 147 ms-ni, mis on 70% paranemine. Selline oluline paranemine on kriitilise tähtsusega reaalajas andmetöötluse ja reageerimise jaoks, tagades andmete kiirema ja tõhusama kättesaadavuse. Latentsuse vähendamine on eriti oluline rakendustes, kus andmete värskus ja reageerimiskiirus on äärmiselt olulised.

Muudatuste analüüsimisel kasutas autor Datadogi, mis on jõudluse jälgimise ja analüüsi platvorm. [24] Datadog võimaldab koguda, visualiseerida ja analüüsida reaalajas andmeid süsteemi jõudluse kohta. See tööriist oli kasulik latentsuse mõõtmisel ja analüüsimisel, pakkudes detailset ülevaadet süsteemi erinevate komponentide reageerimisajast.



Joonis 5. Latentsue paranemise visuaaline kujutus millisekundites

5.1.2 Läbilaskevõime analüüs

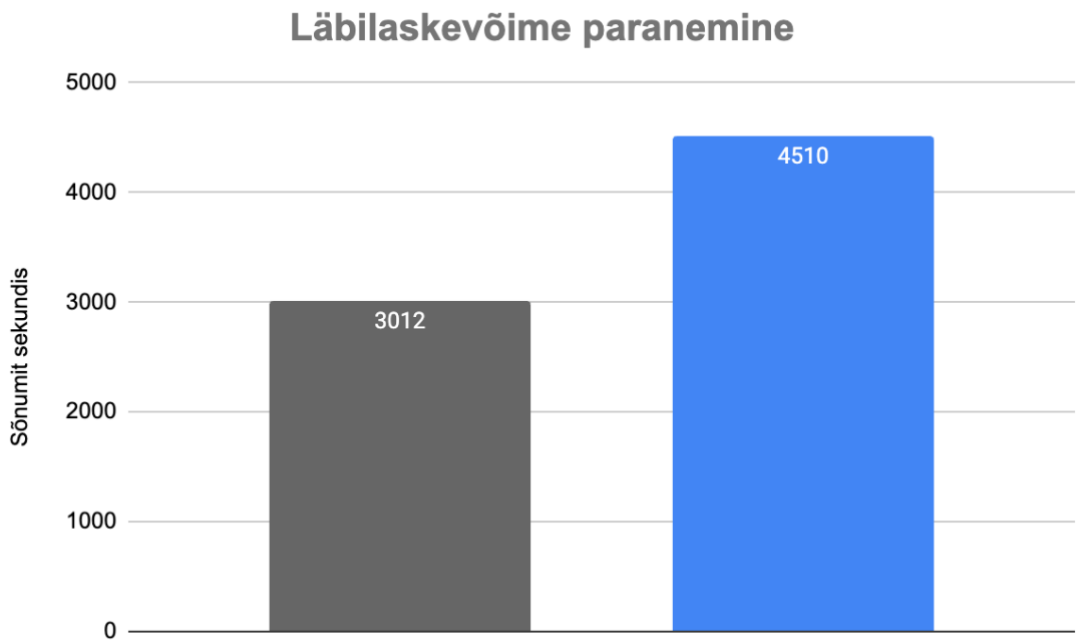
Stressitestimine näitas, et süsteemi läbilaskevõime suurenes enne muudatusi 3012 päringult sekundis kuni 4510 päringuni sekundis pärast muudatusi, mis on ligi 50% tõus. Läbilaskevõime suurendamine on oluline mitte ainult kasvava kasutajabaasi teenindamiseks, vaid ka süsteemi võimekuse suurendamiseks toime tulla andmemahtude

ja keerukate päringutega, mis on tänapäeva äriprotsesside kasvavate andmete hulkadega üha olulisem.

Läbilaskevõime mõõtmise protsessi keskmes oli Apache JMeter, mida kasutati laialdaselt stressitestimiseks ja süsteemi läbilaskevõime hindamiseks. JMeter on avatud lähtekoodiga tarkvara, mis on mõeldud koormustestimiseks ja jõudluse mõõtmiseks. [25] Selle tööriista abil simuleeris autor suure hulka päringuid, et mõõta süsteemi võimet erinevates koormustingimustes.

Autor koostas JMeteris testiplaani, mis määratles, kuidas ja millist tüüpi päringuid süsteemile esitatakse. Testiplaan hõlmas päringute parameetreid, kasutajate arvu ja testi kestvust. JMeter genereeris määratletud testiplaani alusel süsteemile suure koormuse, simuleerides sadu või isegi tuhandeid kasutajaid, kes esitavad päringuid samaaegselt.

JMeter kogus andmeid ka süsteemi vastuse kohta igale päringule, sealhulgas vastuse ajad ja süsteemi läbilaskevõime erinevatel koormustasemetel. Kogutud andmete põhjal analüüsis autor läbilaskevõime muutusi ja tuvastas süsteemi jõudluse pudelikaelu. Läbilaskevõime paranemise visuaalne kujutis, on esitatud joonisel 6.



Joonis 6. Läbilaskevõime paranemise visuaalne kujutis

5.2 Kulude vähendamine

Antud peatükis keskendub autor kulude vähendamise analüüsile, mis on tulenenud Katana veebirakenduse andmevoogude optimeerimisest. Autor vaatleb, kuidas automaatsete lahenduste rakendamine, nagu KafkaConnect ja Debezium, ning käsitsi replikeerimise protsessi eemaldamine on mõjutanud ettevõtte kulustruktuuri.

5.2.1 Tööjõukulude vähendamine

Automaatsüsteemide, nagu KafkaConnect ja Debezium, rakendamine on oluliselt vähendanud arendajate aega, mis kulus varem käsitsi replikatsiooniga seotud ülesannetele. Selle asemel, et veeta aega andmesünkroniseerimise ja -haldusega, saavad arendajad nüüd keskenduda teistele projektidele. See mitte ainult ei suurenda meeskonna üldist produktiivsust, vaid parandab ka töötajate rahulolu ja motivatsiooni, kuna neil on rohkem aega keskenduda loovamatele ja põnevamatele väljakutsetele.

5.2.2 Halduskulude optimeerimine

Andmevoogude automaatse haldamise rakendamine vähendas oluliselt igapäevaseid halduskulusid. Automaatlahendus vähendab vajadust pideva seire ja sekkumise järele, mis tähendab väiksemat koormust tootemeeskonnale. Maasolekuajast tulenevaid riske ning ootamatute probleemide lahendamiseks kulutatud ressursse aitab see samuti vähendada.

5.2.3 Infrastruktuurikulude optimeerimine

Süsteemi efektiivsuse suurenemine on kaasa toonud ka infrastruktuurikulude languse, kuna enam ei pea maksma replikeerimise eest vastutavate rakenduste eest pilveteenustes. See aitab vähendada ettevõtte igakuiseid kulutusi infrastruktuurile.

6 Kokkuvõte

Käesoleva töö raames võttis autor endale eesmärgiks Katana Technologies OÜ mikroteenuste andmevoogude põhjaliku analüüsi ning optimeerimise, keskendudes KafkaConnectile, CDC tehnoloogiatele, outboxi mustrile ning käsitsi replikeerimise asendamisele automaatsete protsessidega. Töö eesmärgiks oli välja töötada lahendused, mis suurendavad andmevoogude tõhusust ja parandavad süsteemi toimimist, tagades samal ajal andmete ühtsuse ja terviklikkuse säilimise.

Praktilises osas rakendatud lahenduste tulemusel saavutati andmebaasi päringute kiirem täitmine, vähendades latentsusaegu eriti suure andmemahtude korral. Samuti saavutati kulude oluline vähendamine – eemaldades käsitsi replikeerimisega seotud tööjõukulud ning vähendades pilveteenuste tasusid, kuna süsteem muutus ressursitõhusamaks. Skaleeritavuse poolest võimaldasid tehtud muudatused süsteemil sujuvalt kohaneda kasutaja koormuse tõusude ja mõõnadega ilma jõudlust kaotamata.

Töö käigus omandatud kogemused ja teadmised andmevoogude optimeerimisest mikroteenuste arhitektuuris on autorile väärtuslikud. Need teadmised aitavad kaasa tulevaste süsteemide projekteerimisel, andes selge suuna, kuidas parendada jõudlust, vähendada kulusid ja suurendada süsteemi paindlikkust. Töö tulemusel loodud uued protsessid ja protseduurid mitte ainult ei paranda Katana Technologies OÜ hetkeolukorda, vaid loovad ka tugeva aluse edaspidiseks tehnoloogiliseks arenguks ja innovatsiooniks.

Kasutatud allikad

[1] C. Lianping, „Microservices: Architecting for Continuous Delivery and DevOps,“ %1 *IEEE International Conference on Software Architecture*, 2018.

[2] C. Richardson, „Pattern: Microservice Architecture,“ [Võrgumaterjal]. Available: https://microservices.io/patterns/microservices.html. [Kasutatud 30 11 2023].

[3] J. Freeman, Medium, [Võrgumaterjal]. Available: https://medium.com/@john_freeman/querying-data-across-microservices-8d7a4667668a. [Kasutatud 19 11 2023].

[4] D. Schmitz, „Dealing with data in microservice architectures,“ [Võrgumaterjal]. Available: https://dev.to/koenighotze/dealing-with-data-in-microservice-architectures-part-3-replication. [Kasutatud 30 11 2023].

[5] „Katana Cloud Inventory,“ [Võrgumaterjal]. Available: https://katanamrp.com/inventory-management-guide/. [Kasutatud 15 12 2023].

[6] „Apache Kafka,“ [Võrgumaterjal]. Available: https://kafka.apache.org/documentation/. [Kasutatud 11 11 2023].

[7] C. Richardson, „Pattern: Transactional outbox,“ Microservices.io, [Võrgumaterjal]. Available: https://microservices.io/patterns/data/transactional-outbox.html. [Kasutatud 5 12 2023].

[8] M. Kleppmann, Designing Data-Intesive Application, Sebastopol: O'Reilly, 2017.

[9] K. Marko, „Deploy microservices in production,“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchitoperations/tip/Follow-these-6-steps-to-deploy-microservices-in-production. [Kasutatud 05 12 2023].

[10] Medium, [Võrgumaterjal]. Available: https://medium.com/@varunchakrabarti7374/whats-microservices-in-devops-411522628eaa. [Kasutatud 19 11 2023].

[11] C. Harris, „Atlassian Software Development,“ [Võrgumaterjal]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. [Kasutatud 06 12 2023].

[12] T. Nolle, „App Architecture,“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchapparchitecture/tip/How-to-master-microservices-data-architecture-design. [Kasutatud 07 12 2023].

[13] T. LoginRadius. [Võrgumaterjal]. Available: https://www.loginradius.com/blog/engineering/relational-database-management-system-rdbms-vs-nosql/. [Kasutatud 15 12 2023].

[14] R. Hat, „Red Hat,“ [Võrgumaterjal]. Available: https://www.redhat.com/en/topics/integration/what-is-change-data-capture. [Kasutatud 07 12 2023].

[15] „What is Change Data Capture,“ Confluent, [Võrgumaterjal]. Available: https://www.confluent.io/learn/change-data-capture/. [Kasutatud 12 12 2023].

[16] B. Buuck, „StreamSets,“ [Võrgumaterjal]. Available: https://streamsets.com/blog/slowly-changing-dimensions-vs-change-data-capture/. [Kasutatud 11 12 2023].

[17] „Syniti,“ [Võrgumaterjal]. Available: https://blog.syniti.com/benefits-challenges-data-replication. [Kasutatud 13 12 2023].

[18] T. Barrera, „Airbyte,“ [Võrgumaterjal]. Available: https://airbyte.com/blog/what-is-data-replication. [Kasutatud 13 12 2023].

[19] „What is data replication?,“ IBM, [Võrgumaterjal]. Available: https://www.ibm.com/topics/data-replication. [Kasutatud 13 12 2023].

[20] C. Custer, „Synchronous and asynchronous database replication explained,“ Cockroach Labs, [Võrgumaterjal]. Available: https://www.cockroachlabs.com/blog/data-loss-prevention-during-outages-you-might-be-losing-data-without-knowing-it/. [Kasutatud 14 12 2023].

[21] „Kafka Connect,“ Confluent, [Võrgumaterjal]. Available: https://docs.confluent.io/platform/current/connect/index.html. [Kasutatud 16 12 2023].

[22] Atlassian, „Jira Documentation,“ [Võrgumaterjal]. Available: https://confluence.atlassian.com/jira. [Kasutatud 16 12 2023].

[23] „EventRouter (Debezium),“ [Võrgumaterjal]. Available: https://docs.confluent.io/platform/current/connect/transforms/eventrouter.html. [Kasutatud 17 12 2023].

[24] „Metrics,“ Datadog, [Võrgumaterjal]. Available: https://docs.datadoghq.com/metrics/. [Kasutatud 16 12 2023].

[25] „JMeter,“ Apache, [Võrgumaterjal]. Available: https://jmeter.apache.org/usermanual/index.html. [Kasutatud 17 12 2023].

[26] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[27] B. Foster, „Launch Crafter,“ [Võrgumaterjal]. Available: https://launchcrafted.com/google-cloud-pub-sub-vs-kafka/. [Kasutatud 11 12 2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

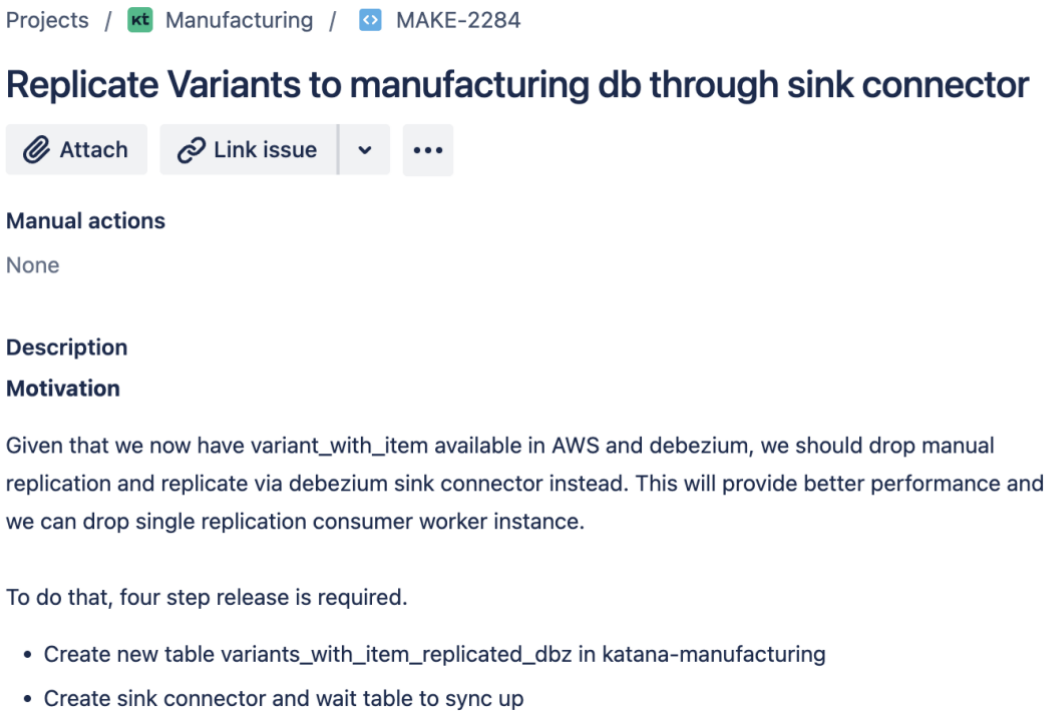
Mina, Devon Nathaniel Gawley

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Mikroteenuste andmevoo optimeerimine Katana Technologies OÜ näitel“, mille juhendaja on Margus Sumla.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.01.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – JIRA pilet – Variants tabeli replikeerimine läbi *sink connectori*

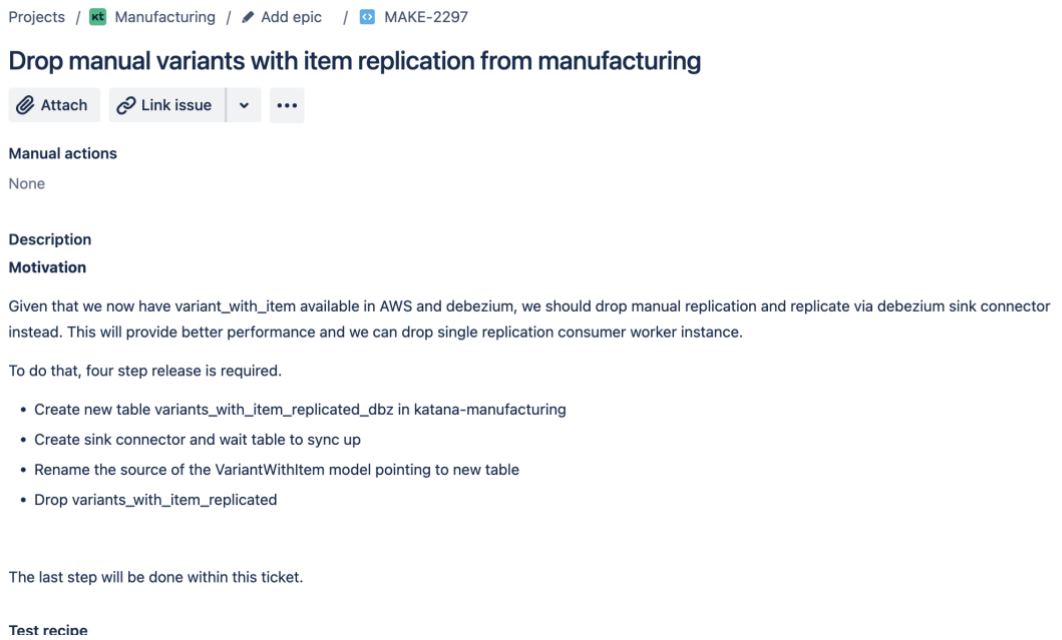


Joonis 7. JIRA pilet – Variants tabeli replikeerimine läbi „sink connectori“

Lisa 3 – JIRA pilet – Variants mudeli allika ümbersuunamine

Joonis 8. JIRA pilet – Variants mudeli allika ümbersuunamine

Lisa 4 – JIRA pilet – Käsitsi replikeerimise protsessi eemaldamine



Joonis 9. JIRA pilet – Käsitsi replikeerimise protsessi eemaldamine

Lisa 5 – Debezium-outbox.manufacturing.json konfiguratsioon

```
{

    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "slot.name" : "dbz_outbox_manufacturing",
    "plugin.name": "pgoutput",
    "database.hostname": "$${global-env-var?var=DB_HOST}",
    "database.port": "$${global-env-var?var=DB_PORT}",
    "database.user": "$${global-env-var?var=DB_USER}",
    "database.password": "$${global-env-var?var=DB_PASS}",
    "database.dbname": "manufacturing",
    "database.server.name": "dbz-outbox-manufacturing-source",
    "topic.prefix": "dbz-outbox-manufacturing-source",
    "schema.name.adjustment.mode": "avro",
    "schema.include.list": "public",
    "table.include.list": "public.dbz_outbox",
    "heartbeat.interval.ms": "10000",
    "heartbeat.action.query": "update debezium_heartbeat set
last_heartbeat_ts = now()",
    "tombstones.on.delete" : "false",
    "transforms" : "outbox",
    "transforms.outbox.type" : "io.debezium.transforms.outbox.EventRouter",
    "transforms.outbox.table.field.event.key": "key",
    "transforms.outbox.table.field.event.payload": "value",
    "transforms.outbox.route.by.field": "topic",
    "transforms.outbox.route.topic.replacement" : "${routedByValue}",
    "transforms.outbox.predicate": "isNotHeartbeatTopic",
    "transforms.outbox.negate": "true",
    "value.converter": "io.debezium.converters.BinaryDataConverter",
    "value.converter.delegate.converter.type":
"org.apache.kafka.connect.json.JsonConverter",
    "value.converter.delegate.converter.type.schemas.enable": "false",
    "predicates": "isNotHeartbeatTopic",
    "predicates.isNotHeartbeatTopic.type":
"org.apache.kafka.connect.transforms.predicates.TopicNameMatches",
    "predicates.isNotHeartbeatTopic.pattern": "^__debezium-heartbeat[.](.*)"
}
```

Joonis 10. Debezium-outbox.manufacturing.json konfiguratsioon.

Lisa 6 – Cdc-debezium.manufacturing.json konfiguratsioon

```
{
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "slot.name" : "debezium_manufacturing_v2",
    "plugin.name": "pgoutput",
    "database.hostname": "$${global-env-var?var=DB_HOST}",
    "database.port": "$${global-env-var?var=DB_PORT}",
    "database.user": "$${global-env-var?var=DB_USER}",
    "database.password": "$${global-env-var?var=DB_PASS}",
    "database.dbname": "manufacturing",
    "database.server.name": "manufacturing_source_v2",
    "topic.prefix": "manufacturing_source_v2",
    "schema.name.adjustment.mode": "avro",
    "decimal.handling.mode": "string",
    "table.include.list":
"public.manufacturing_order_operations,public.manufacturing_order_recipe_rows
,public.manufacturing_orders,public.productions,public.production_ingredients
",
    "heartbeat.interval.ms": "10000",
    "heartbeat.action.query": "update debezium_heartbeat set
last_heartbeat_ts = now()",
    "topic.creation.default.partitions": "5",
    "topic.creation.default.cleanup.policy": "compact",
    "topic.creation.default.compression.type": "gzip",
    "topic.creation.default.replication.factor": "1",
    "transforms": "extractKey, dropSource, changeTopic",
    "transforms.extractKey.type":
"org.apache.kafka.connect.transforms.ExtractField$Key",
    "transforms.extractKey.field": "id",
    "transforms.extractKey.predicate": "isNotIdKeyTopic",
    "transforms.extractKey.negate": "true",
    "transforms.dropSource.type":
"org.apache.kafka.connect.transforms.ReplaceField$Value",
    "transforms.dropSource.blacklist": "source, transaction",
    "transforms.changeTopic.type":
"org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.changeTopic.regex": "([^.]+)\\.([^.]+)\\.([^.]+)",
    "transforms.changeTopic.replacement": "cdc-debezium.manufacturing.$3.v2",
    "predicates": "isNotIdKeyTopic",
    "predicates.isNotIdKeyTopic.type":
"org.apache.kafka.connect.transforms.predicates.TopicNameMatches",
    "predicates.isNotIdKeyTopic.pattern": "^__debezium-heartbeat[.](.*)"
}
```

Joonis 11. Cdc-debezium.manufacturing.json konfiguratsioon.

Lisa 7 – Variants_replicated andmebaasitabeli kustutamise migratsiooniskript

```
'use strict';

var dbm;
var type;
var seed;
exports.setup = function (options, seedLink) {
  dbm = options.dbmigrate;
  type = dbm.dataType;
  seed = seedLink;
};

exports.up = async function (db) {
  await db.runSql(`
    DROP TABLE IF EXISTS variants_replicated;
  `);
};

exports.down = function () {
  throw new Error('Irreversible');
};

exports._meta = {
  version: 1,
};
```

Joonis 12. Variants_replicated andmebaasitabeli kustutamise migratsiooniskript

Lisa 8 – Replikeeritud tabeli variants_replicated_dbz loomise migratsiooniskript

```
'use strict';

var dbm;
var type;
var seed;

/**
 * We receive the dbmigrate dependency from dbmigrate initially.
 * This enables us to not have to rely on NODE_PATH.
 */
exports.setup = function (options, seedLink) {
  dbm = options.dbmigrate;
  type = dbm.dataType;
  seed = seedLink;
};
const tableName = 'variants_replicated_dbz';

exports.up = async function (db) {
  await db.createTable(tableName, {
    id: {type: 'int', notNull: true},
    sku: {type: 'text', notNull: false},
    sales_price: {type: 'numeric', notNull: false},
    purchase_price: {type: 'numeric', notNull: false},
    item_id: {type: 'int', notNull: true},
    factory_id: {type: 'int', notNull: true},
    name: {type: 'text', notNull: true},
    name_without_sku: {type: 'text', notNull: true},
    uom: {type: 'text', notNull: true},
    is_producible: {type: 'bool', notNull: true},
    is_purchasable: {type: 'bool', notNull: true},
    item_name: {type: 'text', notNull: true},
  });

  await db.addIndex(tableName, `${tableName}_id_idx`, ['id'], true);
  await db.addIndex(tableName, `${tableName}_fid_id_idx`, ['factory_id',
'id']);
};
exports.down = function () {
  throw new Error('Irreversible');
};

exports._meta = {
  version: 1,
}
```

Joonis 13. Replikeeritud tabeli variants_replicated_dbz loomise migratsiooniskript