UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Rasmus Bobkov

Design and Implementation of an Incremental ELT Pipeline for a Jira Data Warehouse using Data Vault 2.0 Methodology and HP Vertica

Master's Thesis (15 ECTS)

Supervisor: Feras M. Awaysheh, Phd

Tartu 2023

Design and Implementation of an Incremental ELT Pipeline for a Jira Data Warehouse using Data Vault 2.0 Methodology and HP Vertica

Abstract:

This master's thesis outlines the design and implementation of a containerized ELT pipeline for TEHIK, a company requiring an efficient way to analyze Jira Software data. The pipeline is designed to incrementally load data into a Vertica DWH, constructed following DV 2.0 principles. The containerized architecture enables easy deployment in production environments. Considering the extensive breadth of the subject, the thesis aims to provide an overarching understanding of DE, DV 2.0, Agile methodologies, and implementation. Instead of delving into intricate specifics of each area, it focuses on presenting a broad perspective, offering a more comprehensive view of these fields.

The thesis begins by examining the current system, underlining its limitations, and then introduces the proposed solution, emphasizing its advantages. The Background Knowledge and Related Work section endeavors to provide a solid understanding of the central concepts in DE, DWH'ing, and the DV 2.0 methodology, along with deployment in production environments. This section touches upon key topics such as ingestion, ELT vs ETL architecture, DWH architectures, and the essence and benefits of the DV 2.0 methodology.

While the practical application of Kubernetes, logging, monitoring, and orchestration with Airflow is not included in the thesis due to time restrictions, these aspects are still crucial for a holistic understanding of the project. Hence, a conceptual overview of orchestration using Airflow and a theoretical implementation for logging and monitoring are provided.

The implementation section comprehensively explores the project's process, unveiling the specific steps and methodologies employed, the challenges faced, and their respective solutions. The subsequent 'Results and Analysis' section critically compares the proposed solution and the existing one. It evaluates aspects like reporting capabilities, compliance with SLAs, and an analysis of the pipeline's performance, considering its ability to handle large data volumes and scalability.

In conclusion, this thesis delivers a robust, scalable, and efficient solution comprising an ELT pipeline and a DV 2.0-based DWH tailored for TEHIK's Jira Software data analysis needs. This integrated solution outperforms the existing system, providing a solid foundation for future enhancements and expansions.

Keywords:

ELT pipeline, Data Engineering, DWH, HP Vertica, Jira , TEHIK, DV 2.0, Meltano, shell scripting, vsql, sofware development, Docker, scalability.

CERCS:

P170 Computer science, numerical analysis, systems, control

P175 Informatics, systems theory

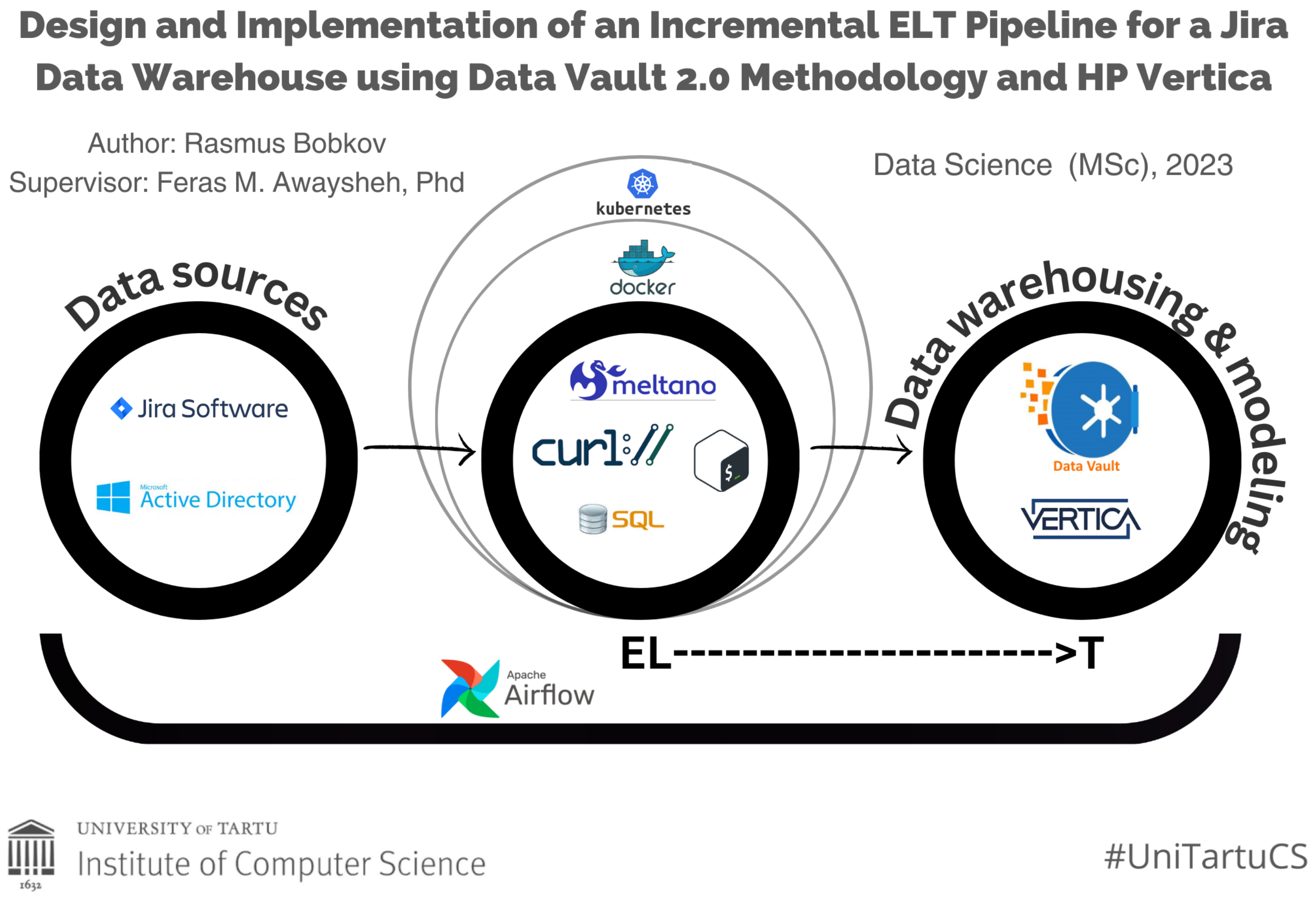

Design and Implementation of an Incremental ELT Pipeline for a Jira Data Warehouse using Data Vault 2.0 Methodology and HP Vertica
Author: Rasmus Bobkov
Supervisor: Feras M. Awaysheh, Phd
Data Science (MSc), 2023
Data sources
Jira Software
Microsoft Active Directory
kubernetes
docker
meltano
curl://
SQL
Data warehousing & modeling
Data Vault
VERTICA
EL---------------------->T
Apache Airflow
UNIVERSITY OF TARTU
Institute of Computer Science
1632
#UniTartuCS

Figure 1. Visual abstract

Inkrementaalselt täiendava ELT töövoo ja Jira andmelao arendus ja juurutamine kasutades Data Vault 2.0 metoodikat ja HP Verticat

Lühikokkuvõte:

Käesolevas magistritöös on esitletud konteineriseeritud ELT töövoo lahenduse väljatöötamist ja rakendamist ettevõttele TEHIK, kes vajab tõhusat võimalust Jira tarkvara andmete analüüsimiseks. Töövoog on mõeldud andmete täiendavaks laadimiseks Vertica andmelattu, mis on ehitatud vastavalt DV 2.0 põhimõtetele. Konteinerkonstruktsioon võimaldab lihtsat kasutuselevõttu tootmiskeskkondades. Arvestades teema ulatuslikkust, on töö eesmärgiks võetud anda pigem kõrgetasemelisi teadmisi andmetehnika, DV 2.0, Agiiilsete metoodikate ja nende rakendamise kohta, otseselt mitte keskendudes süvendatult ühelegi neist.

Lõputöö alustab kehtiva lahenduse uurimisega, selle piirangute välja toomisega, ning seejärel tutvustab pakutavat lahendust, rõhutades selle eeliseid. Taustateadmiste ja sellega seotud töö jaotises püütakse anda lühike ülevaade andmetehnika, andmelaonduse ja DV metoodika kesksetest kontseptsioonidest ning lahenduse juurutamisest tootmiskeskkondades. Selles jaotises käsitletakse põhiteemasid, nagu andmete sisse saamine, ELT vs ETL arhitektuur, andmelao arhitektuurid ning DV 2.0 metoodika olemus ja eelised.

Ajapiirangute ja ka tööle esitatud mahu tõttu ei sisaldu praktilise rakendamise osas: Kubernetese, logimise, monitoorimise ja Airflow'ga orkestreerimise osa. Kuna need aspektid on siiski üliolulised projekti terviklikuks mõistmiseks antakse kontseptuaalne ülevaade Airflow töö põhimõtetest, orkestreerimisest koos logimise ja monitoorimisega.

Rakendamise osas selgitatakse protsessi, tutvustades konkreetseid samme ja kasutatud metoodikaid, erinevaid probleeme ja nende lahendusi. Järgnevas tulemuste ja analüüsi peatükis on kehtivat ja töö käigus väljatöötatud lahendust võrreldud, hinnates sellised apsekte nagu: raporteerimise võimalused ja võimekus, vastavus SLA-dele ja töövoo jõudluse analüüs, võttes arvesse selle võimet käsitleda suuri andmemahtusid ja skaleeritavust.

Kokkuvõtvalt võib öelda, et käesolev lõputöö pakub tugevat, skaleeritavat ja tõhusat lahendust, mis koosneb ELT töövoost ja DV 2.0-põhisest andmelaost, mis on kohandatud TEHIKu Jira tarkvara andmeanalüüsi vajadustele. Pakutud integreeritud lahendus ületab olemasolevat süsteemi, võimaldades tugeva aluse tulevasteks täiendusteks ja laiendusteks.

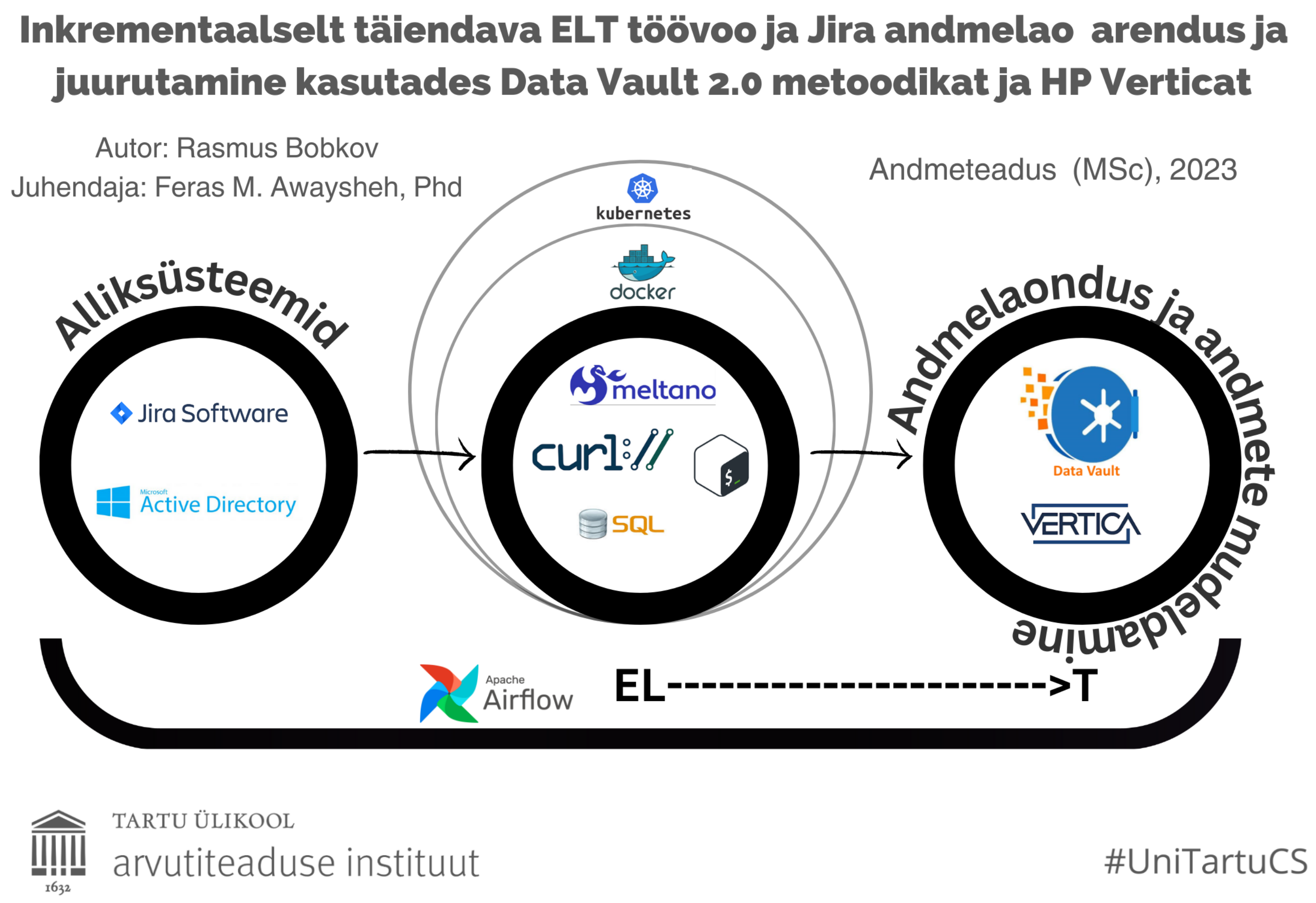
Keywords:

ELT töövoog, Andmetehnika, Andmeladu, HP VErtica, Jira, TEHIK, DV 2.0, Meltano, terminali skriptid, vsql, Tarkvara arendus, Docker, skaleeritavus.

CERCS:

P170 Arvutiteadus, numbriline analüüs, süsteemid, kontroll

P175 Informaatika, süsteemide teooria



Inkrementaalselt täiendava ELT töövoo ja Jira andmelao arendus ja juurutamine kasutades Data Vault 2.0 metoodikat ja HP Verticat
Autor: Rasmus Bobkov
Juhendaja: Feras M. Awaysheh, Phd
Andmeteadus (MSc), 2023
Alliksüsteemid
Jira Software
Microsoft Active Directory
kubernetes
docker
meltano
curl://
SQL
Andmelaondus ja andmete mudeldamine
Data Vault
VERTICA
Apache Airflow
EL------------------------->T
TARTU ÜLIKOOL
arvutiteaduse instituut
1632
#UniTartuCS

Figure 2. Visuaalne abstrakt.

Acknowledgements

I would like to take a moment to express my heartfelt gratitude to my colleague Kristjan for assisting me in selecting the topic for my thesis. In particular, I would like to extend my deepest appreciation to my colleague Karl for patiently guiding me through the challenging journey of navigating the complex world of data engineering. His mentoring has been invaluable in helping me overcome the numerous technicalities and difficulties encountered throughout this project. I would also like to express my sincere thanks to Ida, a friend and classmate, who had a significant impact on the final form of this paper. Her valuable insights and the stimulating discussions we had throughout the intense writing process during the last two weeks made it possible for the paper to appear in its current form.

Furthermore, I would like to express my sincere appreciation to my better half, Katri and the ones close to my heart, whose unwavering love and support have been a source of strength and inspiration throughout this journey. Their magnificent presence in my life has made this achievement possible.

In the process of developing this master's thesis, the assistance of ChatGPT, an AI language model developed by OpenAI, was utilized for several purposes. These include generating ideas, structuring the thesis, and refining sections to enhance readability and effectively convey the author's intended message. It is important to note that the use of ChatGPT in this work was primarily to support the author's own efforts, rather than to create the content itself.

The author was careful to ensure that the information generated by ChatGPT was accurate and reliable, and any references or claims made by the AI were thoroughly checked and verified. In line with ethical guidelines, the use of ChatGPT in this thesis is clearly disclosed, and any assistance received from the AI is acknowledged.

By incorporating ChatGPT's assistance, the author was able to focus on refining the core ideas and arguments presented in the thesis, ensuring a high-quality and well-structured work that meets the academic requirements of the Institute of Computer Science.

Contents

1 Abbreviations used

- **3NF** - Third-Normal-Form
- **AD** - Active Directory
- **API** - Application Programming Interface
- **Bash** - Bourne-Again Shell
- **BD** – Big Data
- **ChatGPT** - Chatbot Generative Pre-trained Transformer
- **CIF** - Corporate Information Factory
- **CLI** - Command Line Interface
- **CPU** - Central Processing Unit
- **CSV** - Comma Separated Values
- **DAG** - Directed Acyclic Graph
- **DB** – Database
- **DDL** - Data Definition Language
- **DE** - Data Engineering
- **DML** - Data Manipulation Language
- **DWH** - Data Warehouse
- **DV** - Data Vault
- **DS** - Data Science
- **E-Health** - Electronic Health
- **EAV** - Entity-Attribute-Value
- **EDW** - Enterprise Data Warehouse
- **ELK** - ElasticSearch, LogStash, Kibana
- **GDPR** - General Data Protection Regulation

- **HIPAA** - Health Insurance Portability and Accountability Act
- **HP** - Hewlett Packard
- **HTTP** - Hypertext Transfer Protocol
- **IT** - Information Technology
- **JSON** - JavaScript Object Notation
- **JSONL** - JavaScript Object Notation Lines
- **JQL** - Jira Query Language
- **KPI** - Key Performance Indicator
- **ODS** - Operational Data Store
- **PIP** - Python Package Installer
- **PIT** - Point in Time
- **RAM** - Random Access Memory
- **REGEX** - Regular Expression
- **REST** - Representational State Transfer
- **SHA** - Secure Hash Algorithm
- **SKAIS** - Social Insurance Board
- **SQL** - Structured Query Language
- **STG** - Staging
- **TCL** – Transaction Control Language
- **TKT** - Technical User Support
- **VM** - Virtual Machine
- **Vsql** - Vertica SQL
- **XML** - eXtensible Markup Language

2 Introduction

2.1 Background on TEHIK

TEHIK is a crucial organization responsible for managing, developing, and coordinating Estonia's health and welfare information systems [teh]. TEHIK was established with the goal of ensuring a high-quality, efficient, and secure digital health and welfare services system for Estonian residents. TEHIK's primary tasks involve the development, management, and integration of health and welfare information systems [teh]. The organization ensures the implementation of digital solutions in healthcare, welfare institutions, and other areas to provide citizens with better access to quality services.

TEHIK's main departments are:

- **Health and Welfare Information Systems Management and Development**: This department focuses on managing, developing, and coordinating health and welfare information systems, ensuring their effectiveness and security [teh].
- **E-Health Solutions Coordination**: This department is responsible for implementing, coordinating, and developing e-health solutions, ensuring easy access to digital health services for healthcare professionals and citizens [teh].
- **Data Analysis and Statistics**: This department focuses on collecting, analyzing, and disseminating health and welfare data to enable better decision-making and policy formulation in the field [teh].
- **IT Services Management and Support Services**: This department ensures the efficient operation, management, and provision of support services for the health and welfare information systems' IT infrastructure [teh].

TEHIK's work is significant not only for Estonian residents and healthcare professionals but also in an international context, contributing to the promotion of digital health and the spread of e-health solutions worldwide [teh]. TEHIK's efforts support Estonia's goal of being an innovator and promoter in e-health and digital healthcare.

The following are the top 10 most critical IT systems managed by TEHIK:

- Old-age pension payments, approximately 310,000 people (SKAIS1), around 180 million EUR per calendar month.
- Family benefits payments, approximately 200,000 families (SKAIS2).
- Disability benefits, approximately 130,000 people.
- digilugu.ee, personal health data, approximately 1 million uses per month [diga].

- digiregistratuur.ee [digb].
- Health Information System, health data for health service providers (including general practitioners, hospitals).
- Emergency medical services mobile workstation.
- Prescription Center [digc].
- SamTrack - the State Agency of Medicines' procedural system [sam].
- Medre - Health care management information system [med].
- TEIS - Employment Information System [too].
- COVID data services [kor].

2.2 What is Jira software and how it's used in TEHIK

TEHIK has implemented Jira as its internal issue-tracking software to manage and address technical issues within its organization effectively[jirb]. Jira, a widely adopted and robust project management tool, enables TEHIK to monitor and manage tasks, incidents, and requests in a centralized and organized manner. This process ensures that all relevant stakeholders within TEHIK are aware of ongoing issues and can efficiently allocate resources to address them.

One key aspect of this thesis focuses on developing a reporting solution based on TKT tickets. These tickets originate from various departments within the Ministry of Social Affairs, the State Agency of Medicines, the Health Board, and other related organizations[sm, rav, ter]. The TKT tickets encompass various technical issues and requests, such as software bugs, system failures, and user support inquiries, requiring prompt and efficient resolution.

By utilizing Jira for managing TKT tickets, TEHIK can benefit from a streamlined process that enables task categorization, prioritization, and assignment to appropriate team members. Furthermore, Jira provides robust reporting capabilities that allow TEHIK to analyze the data associated with TKT tickets, identify trends and recurring issues, and measure the effectiveness of their resolutions. These insights can, in turn, guide TEHIK in making informed decisions regarding resource allocation, process improvements, and strategic planning.

2.3 Overview of the existing solution and its limitation

AS-IS solution: The individual who requests a Tableau report provides the data composition and, if necessary, a JQL query[taba]. In order to export data from Jira to Tableau

Desktop, the Tableau Connector Pro for Jira is utilized, which connects via the Tableau Web Data Connector Jira link[tabb]. Based on the conditions of the JQL query and the selected data fields, Tableau Desktop automatically generates a Tableau-specific file containing the data, known as a *.hyper extract. This extract serves as the initial source for Tableau Prep Builder, where the data is cleaned, enriched, and linked to other data sources if necessary.

The output of the Tableau Prep Builder data stream is a hyper file containing the processed dataset or several files published on Tableau Server. These published data files serve as inputs to Tableau reports. A newly created report in Tableau Desktop is bundled together with the dataset and published on Tableau Server, where it is accessible to users. Extracts and reports are updated on the server.

The initial step to update a report in Tableau Server is to update the extract in Tableau Desktop by refreshing all extracts. Following that, a data flow is initiated in Tableau Prep Builder, which results in the extract with the dataset on Tableau Server being updated. The final step is to update the report published on the server. The visual overview of the As-is solution can be seen in Figure 3.

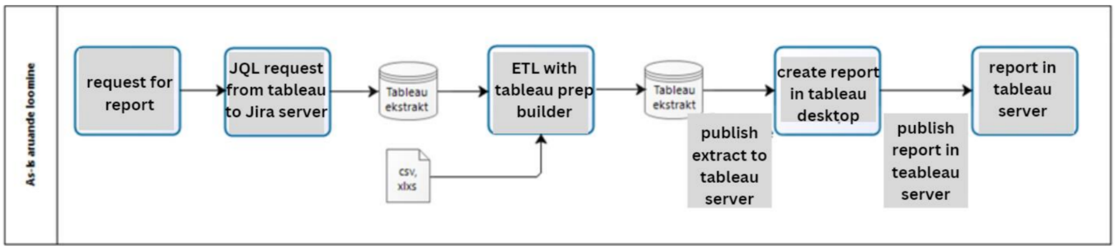

As-is aruande loomine
request for report
JQL request from tableau to Jira server
Tableau ekstrakt
ETL with tableau prep builder
csv, xlxs
Tableau ekstrakt
publish extract to tableau server
create report in tableau desktop
publish report in teableau server
report in tableau server

Figure 3. AS-IS solution

The problems with AS-IS solution

- The process of updating reports (1 or 2 times a month, depending on the report, approx (20 reports) is done manually today and has become a time- and resource-intensive process due to the increase in data volumes. At the initial stage of data update, about 40 extracts are created, some of which take more than an hour to generate. This practically means that this process occupies the entire computer resource. The extract cannot be created/updated at all if there is not enough memory. The problem derives mainly from the fact that the entire set must be queried every time.

- Using Tableau Connector Pro for Jira, it is not possible to perform all requests optimally, so in the conditions of JQL requests where not all Jira data fields are usable (for example, the request cannot be limited to history fields) and therefore, it has turned out to be impossible to create some reports[tabb].

- There is no history of Jira users belonging to entities. For example, if a user moves from one department to another, the department-based report for the same period will be different if viewed at different points in time. A user's membership comes from AD groups, and he can belong to different AD groups at the same time. The latter can cause a situation where the user is in two departments. Those who left the institution are without a department.

2.4 Proposed Solution

Building a DWH for Jira data in an HP Vertica DB requires leveraging existing tools and methods to ensure efficiency and minimize complexity[verb]. The proposed solution uses REST API, Meltano, and custom scripts for data extraction[Mel], while vsql, bash, SQL, and Python facilitate loading the data into the DWH. DV 2.0 serves as the DWH model, providing a flexible and scalable foundation that is adaptable to changes in data structures and requirements.

Also, Vertica is employed in the solution for scalability and fast analytics support, ensuring the DWH can handle growing volumes of data without compromising performance. SQL and custom scripts are used for data transformations, streamlining the process of integrating and analyzing the data from Jira and allowing for more accurate insights and decision-making.

On the other hand, monitoring is handled through custom scripts in the solution, ensuring system health and providing real-time insights into performance. Potential issues can be proactively identified and resolved, reducing downtime and ensuring consistent data quality. Containerization is achieved using Docker, which simplifies deployment, improves consistency across environments, and makes the solution more portable and maintainable.

In the current implementation of the solution, the orchestration of the data pipeline is managed using cron jobs[cro]. At the same time, Kubernetes is employed to handle and manage the deployed containers efficiently[kub]. Looking ahead, the plan is to incorporate Airflow for more advanced and automated orchestration of the data pipelines[airb]. Airflow's capabilities, including dependency management, error handling, and job scheduling, will further streamline the maintenance and monitoring of the data pipeline.

The proposal using these tools and methods will make the solution adaptable, automated, and easy to monitor, ensuring its long-term value to the organization, as demonstrated in our experiments. Prioritizing scalability, simplicity, and effective management will lead to a robust and efficient solution that meets the organization's needs and remains well-equipped to handle future challenges. This approach minimizes reinventing the wheel and maximizes the benefits of existing open-source technologies.

3 Background Knowledge and Related Work

3.1 Data Engineering

3.1.1 Overview of DE

DE is a vital discipline in the big data ecosystem, responsible for designing, constructing, and maintaining data infrastructure that enables efficient storage, processing, and data analysis. As the foundation of modern data analytics, machine learning, and artificial intelligence, DE is pivotal in extracting valuable insights and driving data-driven decision-making across diverse industries.

The DE lifecycle encompasses several essential stages that ensure a seamless process and high-quality outcomes. These stages include:

- **Data ingestion**: Acquiring raw data from various sources such as APIs, databases, or log files, while considering aspects like data formats, schemas, and consistency. [RH22]

- **Data validation and cleansing**: Identifying and rectifying inconsistencies, errors, or missing values in the dataset, often employing techniques like outlier detection, deduplication, or data imputation.[RH22]

- **Data transformation**: Converting and structuring data into a format suitable for storage and analysis, utilizing techniques like normalization, denormalization, encoding, or aggregation, while adhering to specific schema designs (e.g., star schema, snowflake schema) and data modeling principles.[RH22]

- **Data storage and management**: Organizing and storing data in DB's, DWH's, or data lakes, utilizing storage technologies like relational DB's (e.g., PostgreSQL[pos], MySQL[mys]), columnar DB's (e.g., HP Vertica, Apache Cassandra[cas]), or distributed storage systems (e.g., Hadoop Distributed File System[had], Amazon S3[ama])[RH22].

- **Data retrieval and analysis**: Accessing, processing, and analyzing relevant data using query languages (e.g., SQL), programming languages (e.g., Python, R), or data processing frameworks (e.g., Apache Spark[apab], Apache Flink[apaa]) to generate actionable insights and support data-driven applications.

By comprehending and proficiently managing these stages, data engineers can develop robust and efficient data pipelines that enable organizations to unlock the full potential of their data assets. This technical understanding is critical in addressing the ever-evolving challenges and opportunities presented by the rapidly expanding data landscape. A comprehensive overview of the field known as Data Engineering can be observed in Figure 4.

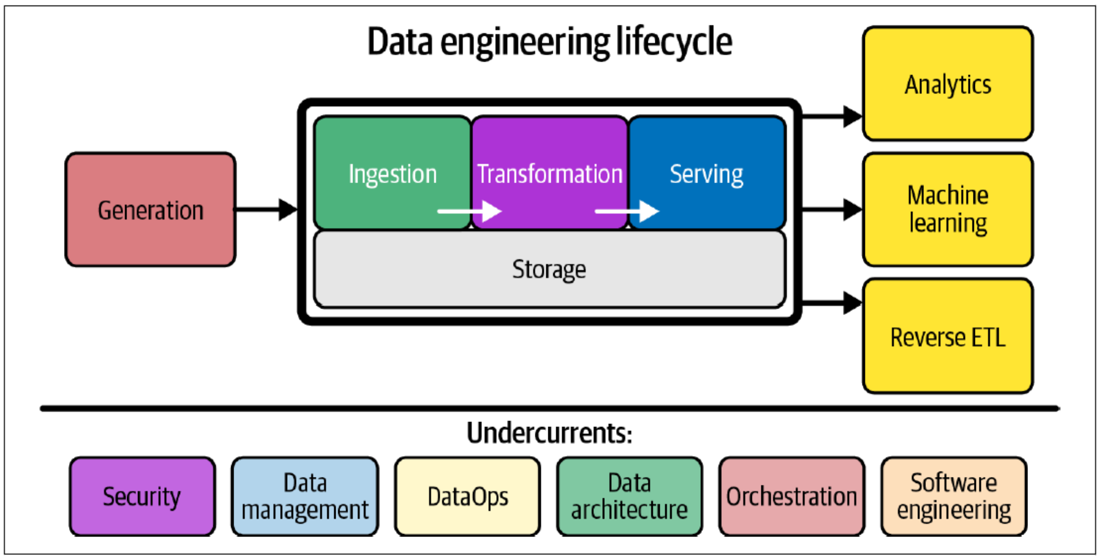


Figure 4. Data Engineering lifecycle [RH22].

3.1.2 Ingestion

Some popular open-source tools for data ingestion encompass Apache NiFi[nif], Logstash[log], and Fluentd[flu], which support various data sources such as APIs, databases, and log files. These tools offer pre-built connectors, customizability, and scalability to accommodate data formats, schemas, and volumes.

Ingestion techniques can be categorized into batch, streaming, and real-time. Batch ingestion involves accumulating data in chunks at regular intervals, processing it collectively, and transferring it into a data storage system. This approach suits non-time-sensitive data or scenarios where low latency is not crucial[RH22].

Conversely, streaming ingestion addresses continuous data flows, processing and storing data as it emerges. Tools like Apache Kafka[kaf] and RabbitMQ[rab] enable efficient streaming ingestion, fostering real-time analytics and immediate insights[RH22].

3.1.3 ETL vs ELT

Data integration is essential in DE, involving unifying data from multiple sources into a cohesive structure. Two primary approaches to data integration are ETL and ELT, each with its advantages and disadvantages.[RH22]

ETL is a traditional approach where data is extracted from various sources, transformed into a desired format or schema, and loaded into a target DB or DWH. ETL processes typically involve data cleansing, enrichment, and aggregation. This approach is suitable for scenarios where data quality and consistency are paramount, as transformations are performed before loading.[RH22]

However, ETL can be resource-intensive and time-consuming, particularly with large data volumes. The advent of modern data storage and processing systems, such as data lakes and distributed computing platforms, has given rise to the alternative ELT approach.

In the ELT approach, raw data is extracted, loaded into the target system, and transformed afterwards. This approach leverages the processing capabilities of modern data storage systems, enabling parallel and distributed transformations for improved performance. ELT is well-suited for scenarios where data processing speed and scalability are essential.[RH22]

However, one disadvantage of ELT is that data quality may be less strictly controlled, as raw data is loaded directly into the target system without prior transformations. This drawback can make it challenging to maintain consistency and quality.

In conclusion, the choice between ETL and ELT depends on the specific use case, requirements, and existing infrastructure. ETL is generally preferable when data quality and consistency are crucial, while ELT is better suited for situations where processing speed and scalability are the primary concerns. [Bar]

3.1.4 Data Transformations and Preparation

The importance of thorough source data preparation and analysis cannot be overstated in the DE lifecycle, as it lays the foundation for deriving reliable and actionable insights. Data transformation and preparation, comprising essential sub-tasks like data cleaning, validation, standardization, normalization, and denormalization, convert raw data into a usable and consistent format for analysis.[RH22]

Data cleaning is critical for identifying and rectifying errors, inconsistencies, and missing values in datasets, utilizing techniques such as outlier detection, deduplication, and data imputation. High-quality data ensures the reliability and accuracy of subsequent analysis and insights.[RH22]

Data validation significantly confirms that data adheres to specified rules or constraints, guaranteeing data accuracy and fitness for its intended purpose. On the other hand, standardization focuses on converting data into a consistent format or schema, allowing seamless integration and analysis across various data sources.

Normalization and denormalization are essential concepts in data preparation, mainly when dealing with relational DBs. Normalization is the process of organizing data into multiple related tables to reduce redundancy and improve data integrity. This technique ensures efficient data storage and facilitates data consistency across the database.[LO15]

Denormalization, in contrast, is the process of combining data from multiple tables into a single table or view, which can improve query performance by reducing the need for complex joins. While denormalization can lead to data redundancy and increased storage requirements, it can significantly enhance analytical processing and reporting capabilities.[KR13]

Handling diverse data formats and parsing them into structured representations is also essential for data transformation and preparation. Standard formats like CSV, JSON, XML, and Parquet each come with their own set of parsing libraries and tools[par]. For instance, Python's pandas library supports reading and writing data in multiple formats, while the lxml library parses XML data, and Apache Arrow manages columnar data formats like Parquet[pan][lxm][arr].

By thoroughly preparing and analyzing source data and mastering data transformation and preparation techniques, data scientists ensure that their data is clean, consistent, and compatible with analytical tools and models. This approach paves the way for more reliable and actionable insights, ultimately empowering data-driven decision-making.

3.1.5 Serving and data modeling

In the broader context of the DE lifecycle, serving and data modeling are critical components contributing to the overall effectiveness of data processing and analytics. Both elements are essential in a data pipeline, regardless of the specific implementation or platform used.

Data serving is the stage in which processed and transformed data is made available to data consumers, such as data scientists, business analysts, and other stakeholders, for analysis and decision-making. A key aspect of data serving is the performance of data retrieval operations, which depends on several technical factors, such as query optimization, indexing, and partitioning. Efficient data serving ensures that the data is readily accessible to consumers while minimizing latency and maximizing throughput. This process often involves caching mechanisms, parallel processing, and load-balancing techniques to scale the serving infrastructure to handle high data requests. [KR13]

Data modeling defines the structure and organization of data to be stored and processed within a DWH or DB. It involves the creation of schemas, tables, relationships, and constraints that accurately represent the underlying business processes and data requirements. Data modeling is essential for ensuring efficient data storage, retrieval, and manipulation, which ultimately impacts the performance of the entire data pipeline. [KR13]

Several aspects of data modeling require technical expertise, including selecting appropriate data types, normalizing or denormalizing data, and establishing primary and foreign key relationships. Additionally, data modeling must account for the specific analytical needs of the system, which may involve the design of star or snowflake schemas, materialized views, or indexing strategies to optimize query performance. [KR13]

Developing robust and scalable DE solutions requires a deep understanding of serving and data modeling components. This knowledge enables the creation of effective data pipelines that can handle the dynamic requirements of data consumers, ensuring that data is organized and delivered optimally for analysis and decision-making purposes.[LO15]

3.2 Orchestration and Workflow Management

The significance of orchestration and workflow management in DE pipelines is underpinned by their capacity to streamline the coordination and execution of intricate data processing tasks. DE pipelines often comprise multiple interrelated steps, including data ETL, necessitating proficient scheduling and resource allocation. Orchestration tools such as Apache Airflow, Apache NiFi, and Luigi facilitate this process by providing a comprehensive framework for delineating, managing, and monitoring data workflows[spo].[RH22]

Apache Airflow, a prominent open-source solution, enables the design and execution of DAGs representing sequences of tasks with explicit dependencies. Through the programmatic definition of tasks and their dependencies, Airflow allows for easy construction, scheduling, and monitoring of complex data workflows. This approach bolsters the maintainability and dependability of data pipelines and enhances overall productivity by automating repetitive tasks and mitigating the risk of manual errors. An example of the Airflow DAG used to load the EDW model with staging can be seen in Figure 5. [airb]

Likewise, Apache NiFi and Luigi offer potent capabilities for managing data workflows, each with distinct features tailored to specific requirements. For example, NiFi emphasizes data flow automation and provides a visual interface for designing and monitoring pipelines, while Luigi concentrates on dependency management and fault tolerance within batch processing environments.[nif, spo]

Employing orchestration and workflow management tools within DE pipelines is indispensable for preserving data integrity and guaranteeing the timely delivery of insights. These tools assist in addressing the growing complexity and scale of data processing tasks while fostering collaboration and standardization across teams. By adopting best practices and leveraging the capabilities of orchestration tools such as Airflow, NiFi, and Luigi, organizations can optimize their DE pipelines and extract valuable insights from their data assets, driving informed decision-making and competitive advantage. [RH22]

3.3 Logging and Monitoring

Logging and monitoring are paramount aspects of DE pipeline management, as they offer insights into data processing tasks' performance, stability, and reliability. Due to the intricate nature of data workflows, which frequently entail multiple steps and dependencies, tracking pipeline execution is crucial for upholding data quality and guaranteeing the timely dissemination of insights. Tools such as Elasticsearch, Logstash, Kibana (ELK Stack), Grafana, and Prometheus have emerged as prominent solutions for

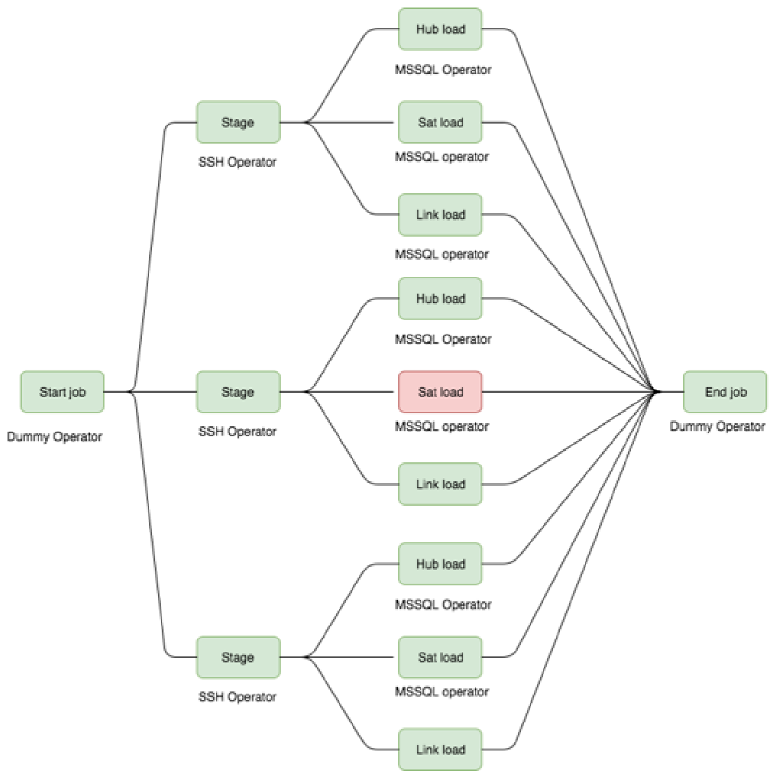


Figure 5. Airflow DAG example [Whe].

logging and monitoring data pipelines, enabling the diagnosis and resolution of issues expeditiously[ela][kib][gra][pro].[airb]

Effective logging encompasses capturing and storing pertinent information regarding the execution of data processing tasks, including progress, errors, and performance metrics. This information allows for pinpointing bottlenecks, optimizing resource allocation, and troubleshooting issues that may transpire during pipeline execution. By implementing comprehensive logging practices, organizations can diminish the likelihood of data inconsistencies, enhance pipeline efficiency, and minimize the impact of errors on downstream analytics and reporting.[gra]

Conversely, monitoring centers on the real-time observation and analysis of data pipelines, offering an all-encompassing view of their operational health. Configuring automated alerts and dashboards exhibiting KPIs allows potential issues to be proactively detected and addressed before they escalate, ensuring the uninterrupted availability of data and insights for decision-making. Furthermore, monitoring tools can assist in identifying trends and patterns in pipeline performance, directing attention toward areas that necessitate optimization or improvement.[gra]

In conclusion, logging and monitoring are indispensable in managing DE pipelines, aiding organizations in maintaining data quality, optimizing performance, and minimizing downtime. By harnessing state-of-the-art tools and best practices for logging and monitoring, enhanced visibility and control over data workflows can be achieved, ensuring the prompt and precise delivery of valuable insights to support informed decision-making and drive business growth.

3.4 Data Warehousing

3.4.1 Overview of DWH'ing and its Importance in Modern Business

DWH'ing is the process of collecting, storing, and managing vast amounts of data from disparate sources within an organization, aiming to facilitate data-driven decision-making. It is crucial in modern businesses, enabling efficient analysis and reporting of complex, heterogeneous data.

A DWH is a centralized repository that stores historical and current data, typically structured to optimize retrieval and analytical performance. The primary function of a DWH is to integrate data from multiple sources, such as transactional databases, log files, and external data sources, and provide a unified view of the organization's data landscape. This data consolidation promotes consistency and reduces redundancy, ensuring decision-makers have access to accurate, high-quality information.[Inm05]

The importance of DWH'ing in modern businesses must be considered. As organizations generate and consume vast quantities of data daily, the need for efficient data management and analysis has grown exponentially. DWH'ing provides a scalable solution that enables businesses to store and analyze large volumes of data, helping them uncover trends, patterns, and correlations that can drive strategic decision-making.[Inm05]

In addition to facilitating analysis and reporting, DWH'ing also plays a vital role in supporting advanced data processing techniques, such as data mining, machine learning, and AI. These technologies rely on large, diverse datasets to build predictive models and uncover hidden insights, DWH'ing an essential component of modern data-driven businesses.[RH22]

Furthermore, DWH'ing is instrumental in promoting data governance and regulatory compliance. By centralizing data storage and enforcing data quality standards, data warehouses help organizations maintain consistency and ensure adherence to industry regulations, such as GDPR and the HIPAA.[RH22]

In conclusion, DWH'ing is a critical component of the data management process, serving as the foundation for effective data analysis, reporting, and advanced data processing techniques. Its role in modern businesses is indispensable, as it enables organizations to harness the power of their data, driving informed decision-making and fostering a competitive edge.

3.5 Three different DWH architectures

In the DWH'ing realm, there are three primary architectural approaches: Inmon's CIF, Kimball's Dimensional Modeling (also known as the Star Schema), and DV 2.0. Each methodology has its own merits, drawbacks, and use cases, but DV 2.0 is considered by many to be a superior approach in specific scenarios. In this comparison, we will explore the technical aspects of these architectures and highlight DV 2.0's advantages. [YL16]

1. **Inmon's Corporate Information Factory** :

   Inmon's CIF approach, named after its creator Bill Inmon, is based on the principle of creating a centralized, normalized data repository known as an EDW. This top-down method emphasizes the importance of data integration and data consistency. Data from various sources is ETL-ed into the EDW, where it is stored in 3NF tables. This normalized structure reduces data redundancy and maintains referential integrity.[Inm05]

   The CIF approach uses data marts to provide subject-specific, denormalized data to end-users. These data marts are created by extracting and transforming data from the EDW and are designed to support specific business processes or analytical needs. In this architecture, data marts serve as the primary data source for reporting and analytics. See figure 6 for the Inmon's architecture overview. [Inm05]

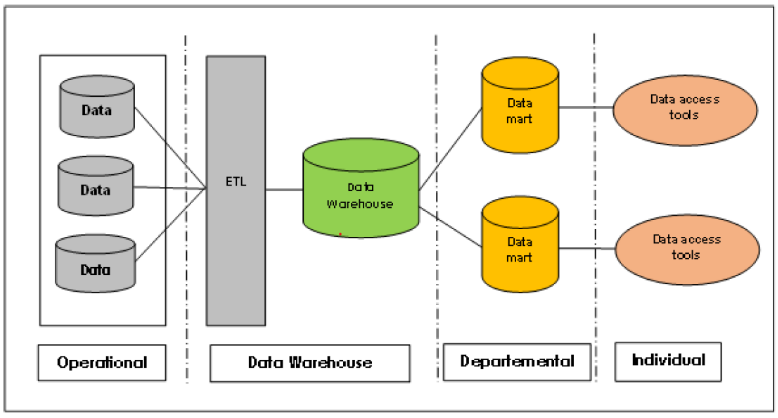


Figure 6. Inmons DWH architecture [YL16].

2. **Kimball's Dimensional Modeling**: Kimball's Dimensional Modeling, or Star Schema, is named after its creator Ralph Kimball. It is a bottom-up approach that focuses on business processes and aims to deliver data directly to end-users. In this methodology, data is organized into fact and dimension tables. Fact tables contain quantitative data (e.g., sales amount), while dimension tables store descriptive data (e.g., customer information). [KR13]

Fact tables are typically denormalized, with keys referencing associated dimension tables. This structure enables efficient querying and reporting by reducing the number of joins required for analysis. Kimball's approach prioritizes ease of use, as the schema closely resembles how business users think about their data. See Figure 7 for the Kimball's architecture overview.[KR13]

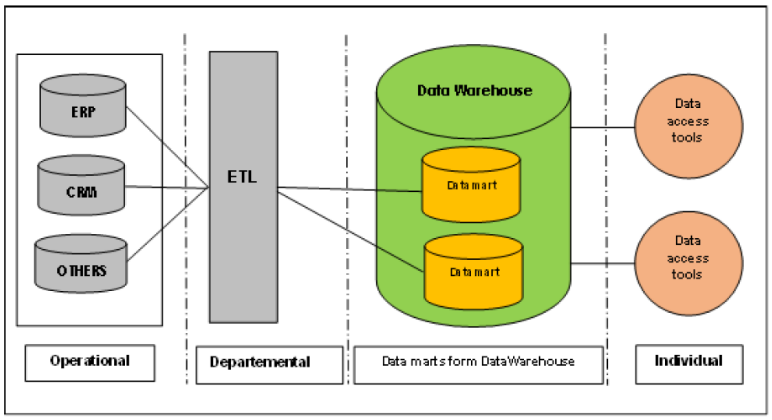


Figure 7. Kimballs DWH architecture [YL16].

3. **Data Vault 2.0**:

   DV 2.0, developed by Dan Linstedt, is a hybrid approach that combines the best features of the CIF and Kimball methodologies. It is designed to handle rapidly changing data sources, enforce referential integrity, and provide high flexibility and scalability. The DV 2.0 architecture stores data in a normalized form, similar to the CIF approach, but maintains a modular structure that simplifies incorporating new data sources.[LO15]

   The key advantages of DV 2.0 include scalability, data lineage and audibility, enhanced data quality, agility, near real-time data loading, efficient handling of historical data, and integration of structured and unstructured data. These benefits make DV 2.0 a compelling choice for organizations dealing with growth, BD, or rapidly changing data landscapes. Its modular design allows for efficient development, while its focus on data lineage and referential integrity ensures high data quality and compliance. See Figure 8 for the DV 2.0's architecture overview.[LO15]

   Advantages of DV 2.0 and Why It's Superior:

   (a) **Scalability**: DV 2.0's parallel load and query capabilities enable it to handle large volumes of data more efficiently than the CIF and Kimball methodologies. This advantage is particularly beneficial for organizations dealing with BD or significant increases in data volume.

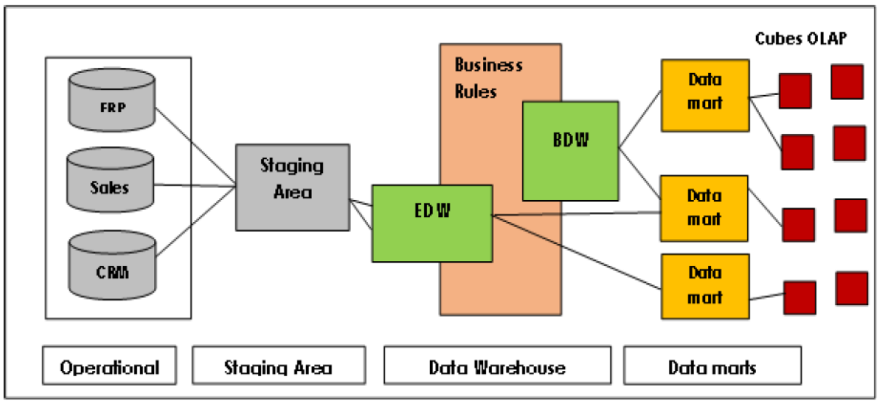


Figure 8. DV 2.0 architecture [YL16].

(b) **Data Lineage and Auditability**: DV 2.0 inherently captures data lineage and history, providing a clear view of how data has changed over time. This advantage is vital in industries where compliance and data governance are essential.

(c) **Enhanced Data Quality**: DV 2.0's architecture enforces referential integrity, ensuring that relationships between entities are maintained. This advantage results in higher data quality and fewer inconsistencies compared to the Kimball approach, which does not enforce referential integrity.

(d) **agility**: DV 2.0 enables a more agile development process by allowing teams to work on different components simultaneously. This is due to its modular design, which separates data into Hubs, Links, and Satellites.

(e) **Near Real-Time Data**: DV 2.0's architecture supports near real-time data loading, making it suitable for organizations that require up-to-date information for decision-making. This advantage is accomplished through incremental loading and parallel processing techniques, which enable fast data ingestion and querying.

(f) **Better Handling of Historical Data**: DV 2.0 can efficiently store and track historical data through its Satellites. This feature facilitates the analysis of data changes and trends over time, which can provide valuable insights for organizations in various industries.

(g) **Integration of Structured and Unstructured Data**: DV 2.0 can accommodate both structured and unstructured data, making it a versatile solution for organizations with diverse data types. This feature allows businesses to analyze various data sources and derives insights that might not be possible with more traditional data warehousing methodologies.[LO15]

In summary, while Inmon's CIF and Kimball's Dimensional Modeling approaches have their merits, DV 2.0 offers a combination of benefits that make it a superior choice in certain situations. Its flexibility, scalability, data lineage, audibility, enhanced data quality, agility, support for near real-time data, efficient handling of historical data, and integration of structured and unstructured data set it apart from the other methodologies.

DV 2.0's unique features enable it to address the challenges of rapidly changing data landscapes, making it an ideal solution for organizations experiencing growth or dealing with BD. Its modular design allows for efficient development, while its focus on data lineage and referential integrity ensures high data quality and compliance. As a result, DV 2.0 is a compelling choice for businesses seeking a versatile and robust DWH'ing solution that can adapt to their evolving needs.

3.6 Data Vault 2.0

3.6.1 DV 2.0 Model

The DV 2.0 model, developed by Dan Linstedt, is a modern data modeling methodology designed to provide a flexible, scalable, and agile solution for DWH'ing. The model is built around three primary components: Hubs, Links, and Satellites, which create a robust, adaptable, and efficient data storage and retrieval system. The DV 2.0 model has gained significant traction for its superiority in facilitating agile development and enabling businesses to adapt to ever-changing data requirements and structures.[LO15]

Hubs are the foundational elements of the DV 2.0 model, representing the core business concepts or entities. Each Hub contains a unique identifier (such as a primary key) and a minimal set of attributes that are unlikely to change over time. Hubs are stable structures that ensure the integrity and consistency of core business data, even as other aspects of the data model evolve.[LO15]

Links serve as the glue that connects Hubs, representing the relationships between different business entities. A Link contains keys from two or more Hubs, effectively establishing many-to-many relationships and enabling the modeling of complex interactions between entities. Links are designed to be flexible and adaptable, allowing for adding or removing relationships as business requirements change. By keeping relationships separate from Hubs, the DV 2.0 model maintains the stability of core business concepts while allowing for changes in how they relate to one another.[LO15]

Satellites, the final component of the DV 2.0's model, store the descriptive attributes associated with Hubs and Links. These attributes are often prone to change and may have various formats, data types, or value ranges. Satellites allow for adding, modifying, or removing attributes without impacting the core business concepts (Hubs) or relationships (Links). They also facilitate the management of historical data, enabling the storage of multiple versions of an attribute as it changes over time.[LO15]

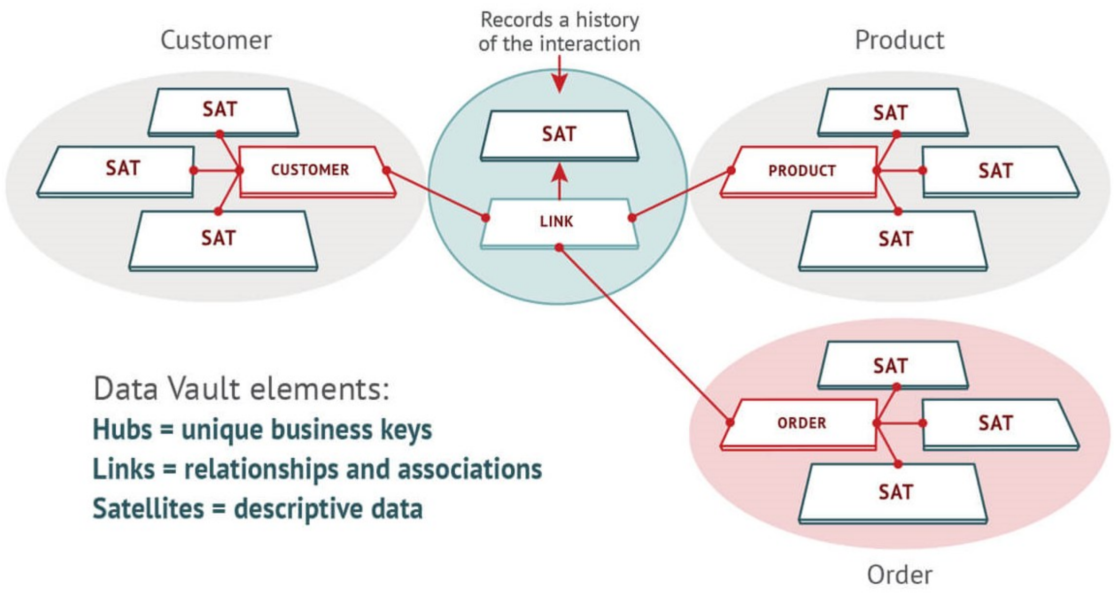

Customer
Records a history of the interaction
Product
SAT
SAT
CUSTOMER
SAT
SAT
LINK
PRODUCT
SAT
SAT
SAT
Data Vault elements:
Hubs = unique business keys
Links = relationships and associations
Satellites = descriptive data
SAT
ORDER
SAT
SAT
Order

Figure 9. DV 2.0's Core Model architecture [Whe].

The DV 2.0 model's superiority for agile development stems from its flexibility, adaptability, and scalability. The separation of core business concepts, relationships, and descriptive attributes into Hubs, Links, and Satellites allows individual components to evolve independently. This modularity enables businesses to respond quickly to changing data requirements and structures while maintaining the integrity and consistency of their DWH. See Figure 9 for an overview of the DV 2.0's model.

Furthermore, the DV 2.0 model supports parallel loading and processing, significantly improving performance and reducing data load times. This capability is particularly beneficial in large-scale DWH'ing projects, where processing vast amounts of data efficiently is critical. Parallel loading and processing are made possible by the model's modular design and the separation of concerns among its components, such as hubs, links, and satellites. This structure enables multiple data loads and transformations to be executed simultaneously without conflicts or dependencies, thus maximizing resource utilization and reducing overall processing time. Additionally, modern DWH'ing systems, such as Vertica, are designed to leverage parallel processing and distributed architectures, further enhancing the DV 2.0 model's performance benefits when used with these technologies.[LO15]

The DV 2.0 model also enables better data lineage and traceability, as it inherently captures the history of changes in data attributes and relationships. This feature allows data scientists and analysts to track the evolution of data over time, providing valuable insights into business trends and patterns.[LO15]

Another advantage of the DV 2.0's model is its adaptability to various data storage technologies, including traditional relational DB's, data lakes, and distributed storage systems. This flexibility allows businesses to leverage the latest data storage and processing technologies to optimize performance, reduce costs, and meet their specific DWH needs.[LO15]

In conclusion, the DV 2.0 model, with its unique structure of Hubs, Links, and Satellites, offers a superior solution for agile development in DWH'ing. Its flexibility, adaptability, and scalability allow businesses to respond quickly to changing data requirements and structures while maintaining data integrity and consistency. The model's support for parallel processing, data lineage, and compatibility with various data storage technologies further enhances its value for businesses seeking a robust, efficient, and future-proof DWH'ing solution.

3.6.2 The use of hashing in DV2.0

Hashing plays a crucial role in the DV2.0 methodology, providing several advantages over traditional methods in DWH'ing. The primary purpose of hashing in DV2.0 is to create unique and consistent keys for the data entities, allowing for efficient and reliable data integration, retrieval, and maintenance.[LO15]

In DV2.0, hashing is used to generate surrogate keys for hubs, links, and satellites. These surrogate keys are created by applying a deterministic hash function to the business keys or a combination of keys, resulting in a fixed-length, unique, and consistent value. By using hashing to generate surrogate keys, DV2.0 addresses several challenges associated with traditional methods:[LO15]

- **Consistency:** Hashing ensures consistent surrogate keys across different data sources, even when the underlying data structures or formats vary. This consistency simplifies data integration and reduces the risk of data integrity issues when merging data from multiple sources.
- **Scalability:** Hashed surrogate keys allow for efficient partitioning, indexing, and data retrieval, as they are uniformly distributed across the key space. This uniform distribution reduces the likelihood of performance bottlenecks or hotspots, enabling the data warehouse to scale more effectively as data volumes grow.
- **Data Privacy:** Using hashing to generate surrogate keys, sensitive business keys can be replaced with non-descriptive, hashed values. This process helps protect sensitive information and maintain data privacy while allowing for efficient data integration and retrieval.
- **Collision Resistance:** High-quality hash functions exhibit a low probability of generating duplicate keys, known as collisions. This characteristic ensures that surrogate keys generated using hashing are unique, minimizing the risk of data integrity issues due to key collisions.

- **Concurrency:** Surrogate keys generated through hashing do not require centralized sequence generators or other coordination mechanisms, thus enabling concurrent data processing and reducing the risk of contention or performance issues during data loading.[LO15]

Incorporating hashing as a core component of the DV2.0 methodology offers superior performance, scalability, consistency, and data privacy compared to traditional DWH'ing techniques. Hashing in DV2.0 ensures that data entities are uniquely and consistently identified, facilitating effective data integration, retrieval, and maintenance in modern, large-scale DWH environments.

3.7 Presentational layer (DWH) in DV2.0

In the context of DV2.0's architecture, DWH is the presentation layer, acting as an interface between the underlying data storage and the end-users. A key aspect of the presentation layer is the concept of Information Marts, which are designed to provide tailored views of the data to meet the specific requirements of various business units and end-users. DV2.0 implements a dimensional model in the presentation layer, making it more suitable for business users who understand business terms but may need to be better versed in technical jargon.[LO15]

The dimensional model focuses on presenting only the required data, rather than exposing the entire range of data available in the EDW layer. This approach ensures that end-users' information is relevant, concise, and easily understandable. Dimensional models often consist of fact tables and dimension tables, which help users navigate and analyze the data using familiar business concepts and terminology.[LO15]

3.8 Deployment in Production Environment

A well-deployed data pipeline exhibits several key characteristics that ensure its successful integration, maintainability, and adaptability within a production environment. Such a pipeline is a prime example of best practices in DE and DS projects.

Firstly, a well-deployed pipeline is easily transferable between different machines and environments. Using containerization technologies like Docker allows for smooth deployment without extensive setup or configuration efforts. It encapsulates dependencies and configurations, ensuring the pipeline can be seamlessly transferred and executed on various platforms with minimal adjustments.[RH22]

Incorporating orchestration platforms like Kubernetes further enhances the deployment process by managing containerized applications at scale. Kubernetes automates containerized applications' deployment, scaling, and management, ensuring optimal resource utilization and resilience in production environments.[kub]

Secondly, maintainability and extensibility are essential features of a well-deployed pipeline. The codebase should be structured and modular, enabling easy updates and modifications. Adherence to established coding standards and comprehensive version control systems, such as Git, facilitate collaboration, tracking of changes, and management of updates. This standard allows for continuous improvement and adaptation of the pipeline to evolving requirements and data sources. Version control ensures the pipeline remains reliable and up-to-date, providing a robust foundation for data-driven insights and decision-making processes. [RH22]

Thorough documentation is crucial for successfully deploying and adopting a data pipeline. It should include clear instructions for setting up, configuring, and executing the pipeline, as well as details about input data, expected outputs, and potential error scenarios. Comprehensive documentation enables team members, including developers, data scientists, and business analysts, to understand the pipeline's inner workings, troubleshoot issues, and make informed decisions about future enhancements.

In summary, a well-deployed data pipeline is characterized by its ease of transferability, maintainability, extensibility, thorough documentation, and incorporation of robust technologies such as containerization and orchestration platforms. These features ensure that the pipeline remains an effective, reliable, and valuable tool in the rapidly evolving landscape of data science and reporting.

4 Implementation

This section discusses the proposed solutions implementation. The use case, architecture design, essential aspects of implementation, different layers of implementation, and the tools and software are thoroughly discussed and presented. This section aims to give the reader insights into the solution environment and how to regenerate it.

4.1 Use Cases and User Stories

DV 2.0 embraces Agile and Scrum methodologies as integral components of its implementation process. Agile is a project management approach that prioritizes iterative development, cross-functional collaboration, and customer feedback to deliver a high-quality product that meets evolving requirements. Scrum is a specific Agile framework that organizes work into smaller, manageable units called "user stories," each representing a specific feature or functionality. These stories are the basis for prioritizing tasks, allowing teams to adapt quickly to changing needs and ensuring continuous improvement.[LO15]

This study gathered the initial requirements from an analyst responsible for creating AS-IS reports. To identify more potential users who could benefit from the TO-BE Jira data warehouse, the author reached out to other stakeholders in the organization.

The Testing department manager at TEHIK was one such stakeholder who expressed a need for improved reporting capabilities. The department faced significant challenges in obtaining valuable insights and reports from their data using Tableau. They had to rely on the limited capabilities of Jira Dashboard for some form of reporting. This situation hindered the manager's ability to make informed decisions and effectively manage the testing processes within the organization, as many reports could not be generated using Jira dashboards alone.

The manager expressed interest in having additional Tableau reports, which include:

- **1. An overview of testing for different projects/services, providing information such as**:
  - Duration of a ticket in a specific status
  - Tester, testing results, and other relevant data for evaluating the quality of testing documents
  - All other fields that need to be filled out to evaluate the quality of testing documents
- **2. A separate report for each project/service, showcasing the product quality with information like**:
  - List of defects
  - Open defects
  - Resolved defects
- **3. An overview of security testing partner contracts**:
  - Importing the list of contracts from JIRA dashboard into Tableau
  - Creating a table and visual graph that displays the framework agreement, including expected volume, volume of accepted contracts, and volume of signed contracts

These requirements demonstrate the real-world application of Agile principles, where user stories drive the development of a data warehouse solution tailored to the specific needs of various stakeholders.

4.2 Constraints set for the Architecture by TEHIK

TEHIK had certain constraints that influenced the choice of tools and technologies used in the architecture. The organization already used Jira as its project management platform, limiting the available data extraction options. Jira supports several data extraction tools, including its REST API, webhooks, and third-party connectors. The data retrieved from Jira is typically in JSON format, a widely adopted data interchange format. Jira follows an EAV data model, which allows for flexibility in storing and retrieving diverse data types. However, it can be more challenging compared to traditional relational models.[jira]

The company also specified using Vertica as the DB for their DWH. Vertica is a high-performance, columnar, analytics-oriented DB management system. It is designed to handle large volumes of data and perform complex analytical queries quickly and efficiently. Vertica's strengths lie in its ability to provide high concurrency, real-time analytics, and horizontal scalability. However, due to its specialized architecture and resource requirements, Vertica might not be ideal for very small-scale projects. [vera]

TEHIK preferred experimenting with quickly deployable, open-source tools with minimal overhead. The company aimed to assess its viability for other extract and load components in the future, including the potential integration of data orchestration tools such as Apache Airflow. Open-source tools offer the advantage of cost-effectiveness, community-driven development, and easy adaptability. However, they may need more robustness, support, and features commercial solutions offer.

Considering these constraints and future plans, the architecture needed to integrate Jira's data extraction capabilities with Vertica's analytical prowess. It also incorporates open-source tools for a seamless, scalable, cost-effective solution. As a result, the focus was on finding a balance between addressing the organization's specific needs and working within the imposed limitations while keeping the door open for future enhancements and integrations with data orchestration tools like Apache Airflow.

The second set of requirements for the application stems from the need to ensure compatibility with the State Cloud, a government-mandated cloud infrastructure currently being developed in Estonia. The country has established strict policies prohibiting private cloud service providers for government services. It has built its cloud infrastructure to meet its unique needs and ensure data sovereignty.

In order to develop a compatible application, it is essential to adhere to the guidelines and requirements laid out by the Estonian government, which includes using specific common components within the application architecture. These components are Docker, Kubernetes, Git, and Jenkins, each serving a crucial role in developing, deploying, and managing applications within the State Cloud[riia].

Docker is a platform for containerization, enabling the packaging of applications and their dependencies into lightweight, portable containers that can run consistently across various computing environments. This platform ensures the application can be seamlessly deployed and scaled within the State Cloud[doc, riia].

Kubernetes is a container orchestration platform that automates containerized applications' deployment, scaling, and management [kub]. By utilizing Kubernetes, the application can leverage the State Cloud's inherent scalability, resilience, and self-healing capabilities, ensuring high availability and efficient resource utilization.

Git is a widely-used distributed version control system that allows developers to collaborate on code, track changes, and manage the application's source code effectively [git]. Integrating Git into the application development process makes it easier to adhere to the State Cloud's development guidelines and maintain a consistent and transparent development workflow.

Jenkins is an open-source automation server that facilitates the implementation of continuous integration and continuous delivery (CI/CD) pipelines. Through Jenkins, the application can automate its build, testing, and deployment processes, ensuring that the application is continuously updated, tested, and delivered in compliance with the State Cloud's requirements.[jen]

The application's architecture must be designed with these components in mind, integrating them effectively to create a solution that is compatible with the State Cloud and leverages its capabilities to the fullest extent. By doing so, the application can deliver the required functionality while adhering to the strict policies and guidelines the Estonian government sets, ultimately fulfilling the project's objectives and contributing to the country's digital transformation journey. Figure 10 provides a comprehensive overview of the architecture.

4.3 Utilizing Testing and Live Environments for Development

TEHIK has established testing and live environments for Jira and Vertica, ensuring a structured and secure development process. The testing environment serves as a controlled space where new features, improvements, and bug fixes can be developed, tested, and validated without affecting the live environment. This approach minimizes the risk of introducing errors, data corruption, or performance issues in the live system, which could negatively impact business operations.

Developing in the testing environment provides several benefits:

- **Isolation**: The testing environment is isolated from the live environment, allowing developers to work on new features or improvements without the risk of impacting the live system's stability or performance.

- **Iterative development**: Developing in the testing environment enables iterative development, where changes can be made, tested, and refined multiple times before being deployed to the live environment.

- **Debugging and troubleshooting**: Issues can be identified and resolved in the testing environment, reducing the risk of introducing errors or performance issues in the live system.

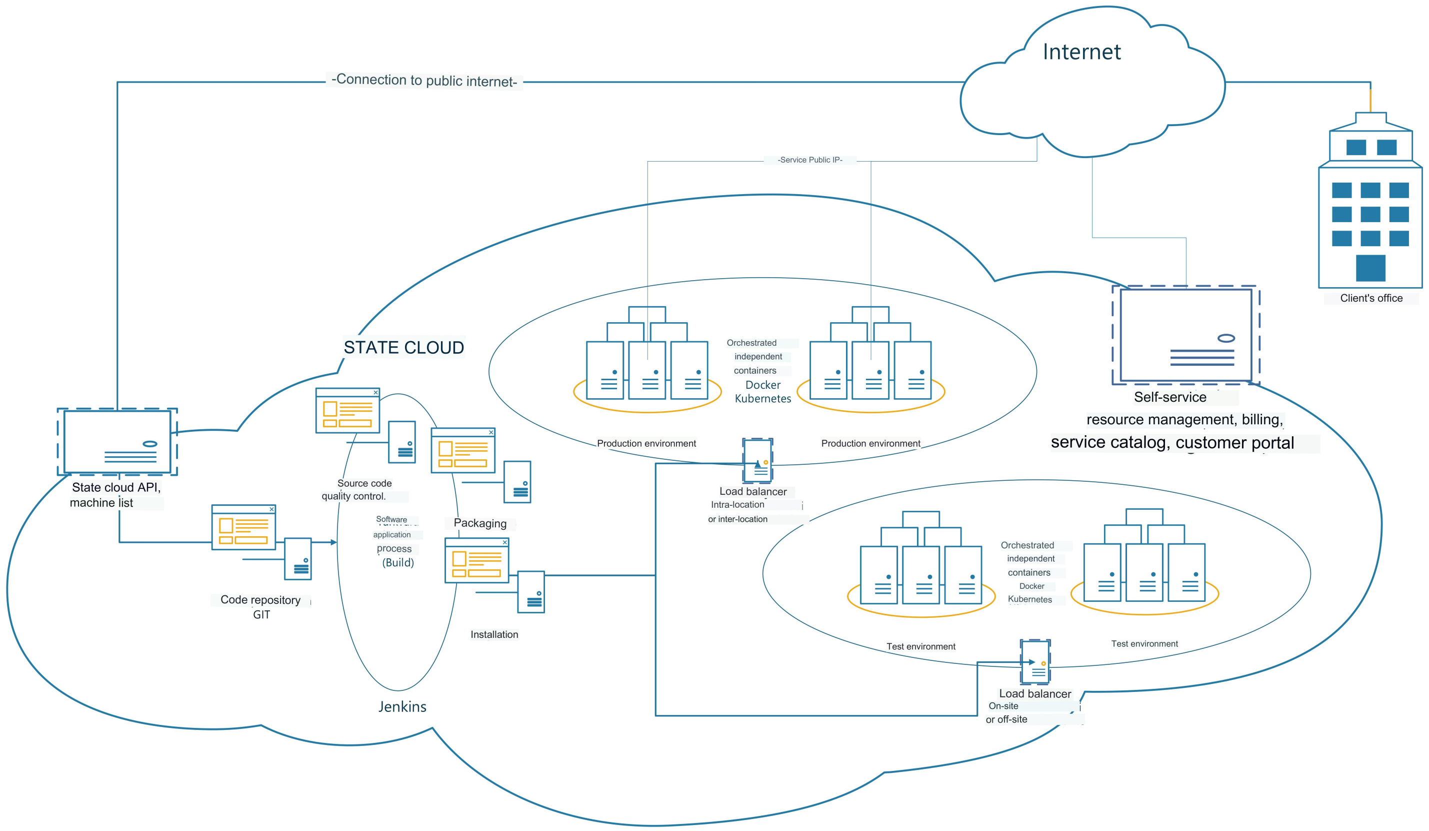


Figure 10. Cloud-enabled application development process[riib]

- **Validation and verification**: The testing environment allows developers to validate and verify the functionality and performance of the proposed ELT pipeline before deploying it to the live environment, ensuring that it meets the requirements and expectations.

Developing in the live environment is not recommended, as it can lead to unforeseen consequences, such as data corruption, system instability, and performance issues. By separating the development process into distinct testing and live environments, TEHIK ensures a secure, controlled, and efficient development process for the proposed ELT pipeline.

Utilizing both testing and live environments for Jira and Vertica is critical to the development process. It allows TEHIK to maintain the stability and performance of the live system while enabling the development of the proposed ELT pipeline in a controlled and secure manner.

While separate testing and live environments are crucial for a controlled and secure development process, there can be potential drawbacks when the two environments differ. In the case of TEHIK, specific differences existed between the testing and live environments, particularly in Jira and Vertica configurations. This section highlights the challenges arising from non-identical testing and live environments and their impact on the development process.

- **Missing features in Jira test environment**: A non-identical testing environment may lack certain features or configurations in the live environment, making replicating and testing specific scenarios difficult. In TEHIK's case, the Jira test environment had some missing features compared to the live environment, which could limit the ability to test and validate the proposed ELT pipeline thoroughly.
- **Version differences in Vertica**: At one point, the Vertica testing and live environments had different software versions. Version differences can lead to discrepancies in functionality and performance, making it challenging to ensure that the pipeline developed and tested in the testing environment will work seamlessly in the live environment.
- **Cluster configuration differences**: The Vertica live environment had a four-node cluster, while the testing environment had only one node. This discrepancy in cluster configurations can impact the pipeline's performance, as the testing environment may not accurately reflect the live environment's performance characteristics. In turn, this can result in unexpected performance issues when deploying the pipeline to the live environment.

4.4 On the Importance of Structure and Naming Conventions

A consistent file tree structure and naming conventions are crucial for efficient and automated data processing. Consistent organization and naming allow for smoother automation of processes, making it easier to develop and maintain scripts and generative SQL for creating DDL and DML statements. Moreover, a well-defined structure and naming system enhance readability, understandability, and maintainability, ensuring that current and future team members can effectively manage and develop the DWH. In essence, adhering to consistent structure and naming conventions is a key aspect of streamlining and optimizing DE processes, ultimately leading to better overall performance and maintainability of the DWH.

In this project, adhering to established naming conventions within the DV 2.0 methodology was crucial to maintain consistency and clarity[Dat]. The chosen conventions were as follows:

For Hubs, the format was **<Entity_name>_HUB**. For Satellites, the naming convention was **<Entity_name>_<Source_Shortname>_SAT**, while for multi-value Satellites (in cases of arrays), it was **<Entity_name>_<Source_Shortname>_MSAT**.

In addition to these conventions, the primary keys (PK), foreign keys (FK), and unique keys (UK) followed a specific naming pattern as well. The primary keys were named as **pk_<table_name>**, foreign keys as **fk_<table_name>_<referenced_table>**, and unique keys as **uk_<table_name>**.

Adhering to these naming conventions throughout the project ensured that the data model was easily interpretable and maintainable, allowing for streamlined development and a simplified understanding of the relationships between various components. see Appendix I.

4.5 Setting up Git repositories and on The Importance of Access Control in DWH's and Databases

At the beginning of any new development process, setting up a Git code repository for effective version control is essential. By doing so, developers can track changes in their code, collaborate more efficiently, and avoid losing progress due to unforeseen issues or mistakes. The development of the DWH is organized into three separate repositories, each serving a specific purpose and layer within the architecture:

- **DW_JIRA_ODS** - This repository is dedicated to the staging and ODS layers, where data is initially ingested and transformed.
- **EDW_JIRA**- The operational layer, consisting of hubs, links, and satellites, is managed within this repository.

- **DWH_JIRA** - The presentational layer, designed for end-users such as analysts, is maintained in this repository, providing them with easy access to the processed data.

Each repository contains comprehensive instructions on setting up and executing the associated application layer. Furthermore, these repositories adhere to established naming conventions and ideally provide a clear data model representation. This structure ensures that every aspect of the DWH development process is well-documented, organized, and easily accessible to all team members.

In today's data-driven world, building DWH's and databases with robust access control mechanisms is crucial. This stage ensures that sensitive data remains protected and access to different components is managed appropriately. There are several reasons why implementing granular access control is essential in any data management system.

- **Data ownership changes**: Over time, the ownership of specific data may change due to various reasons, such as organizational restructuring or acquiring new data sources. When this occurs, it is essential to have a system that allows for the quick and efficient updating of access permissions. This permission ensures that the new data owner can easily access and manage the data while previous owners are restricted from accessing it.

- **Employee turnover**: When data engineers or other team members leave a company, it is crucial to revoke their access to sensitive data and systems. A well-designed access control system makes managing these changes easy and ensures that only authorized personnel have access to the DWH or DB.

In TEHIK, the DWH is built with modularity in mind, adopting a microservices architecture style derived from Data Vault 2.0 principles[LO15]. This approach enables agile development, allowing for the rapid and efficient adaptation of the system as requirements evolve.

One of this modular architecture's key components is using service accounts to manage access control. Each data warehouse layer has its dedicated service account designed for a specific purpose. This service ensures that access to each layer is restricted to the appropriate services, minimizing the risk of unauthorized access.

By implementing a robust and granular access control system, TEHIK ensures that its DWH and databases are secure, adaptable, and easy to manage. This approach protects sensitive data and allows the organization to adapt quickly to changes in data ownership and personnel, maintaining the integrity and security of the system. Below is a commented TCL about how the DW_JIRA_ODS was set up access rights wise.

```
-- Create schema
CREATE SCHEMA DW_JIRA_ODS DEFAULT INCLUDE PRIVILEGES;
-- Create service account - change the password to something
    reasonable
CREATE USER DW_JIRA_ODS IDENTIFIED BY '123456';
-- Apply authentication method to the user or role
GRANT AUTHENTICATION pass_auth TO DW_JIRA_ODS;
-- Service account access to schema
GRANT ALL PRIVILEGES ON SCHEMA DW_JIRA_ODS to DW_JIRA_ODS WITH GRANT
    OPTION;
-- Grant all privileges on all created schema objects to the service
    account
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA DW_JIRA_ODS to
    DW_JIRA_ODS WITH GRANT OPTION;
-- Create a viewer role for the schema
CREATE ROLE DW_JIRA_ODS_VIEWER;
-- Grant the viewer role the right to view the schema
GRANT
SELECT
  ,
  USAGE ON SCHEMA DW_JIRA_ODS TO DW_JIRA_ODS_VIEWER;
-- Attach role to the user
GRANT DW_JIRA_ODS_VIEWER TO user_name -- Change default role for the
    user
ALTER USER DW_JIRA_ODS DEFAULT ROLE DW_JIRA_ODS_VIEWER;
```

4.6 Ingestion

This section discusses extracting data from Jira and the various methods explored for this task. Initially, the Jira REST API was considered as a possible solution. The REST API is an architectural style that enables communication between systems over HTTP. However, using the REST API with custom Python scripts would have been time-consuming to develop, and since REST is not a standard, each endpoint would need to be manually entered. If the source changed, the entire process would have to be reworked.

After researching further, Singer.io, a data integration tool that facilitates data extraction and loading through taps and targets, was discovered. It provides a Jira tap and Vertica target, which seemed promising. However, the search for a more capable solution continued, leading to the discovery of Meltano and Airbyte. Both tools offer a tap developer kit, integration with Airflow, and, in the case of Airbyte, integration with Dagster [Mel].

Comparing Meltano and Airbyte, the main strength of Meltano is its CLI approach, which makes state tracking easier to control. The CLI-based approach offers higher value as the code can be version-controlled, tested, deployed, and orchestrated more efficiently.[Mel]

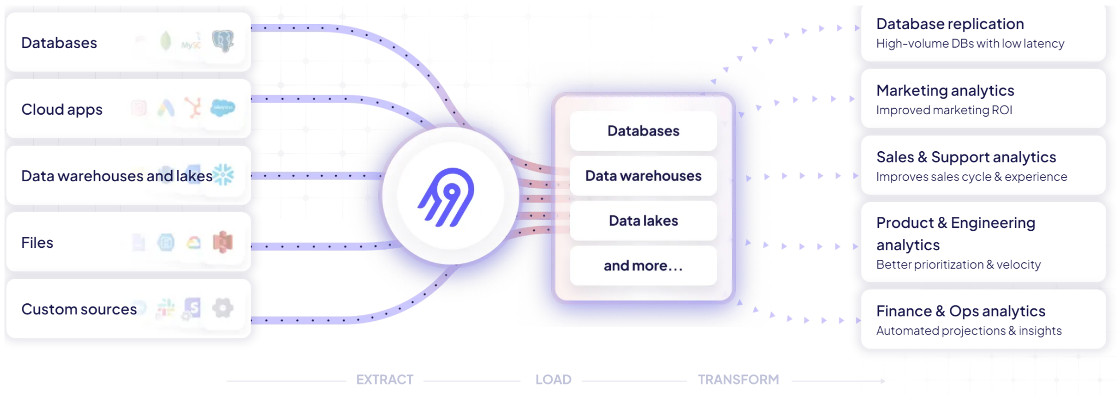

Databases
Cloud apps
Data warehouses and lakes
Files
Custom sources
Databases
Data warehouses
Data lakes
and more...
Database replication
High-volume DBs with low latency
Marketing analytics
Improved marketing ROI
Sales & Support analytics
Improves sales cycle & experience
Product & Engineering analytics
Better prioritization & velocity
Finance & Ops analytics
Automated projections & insights
EXTRACT
LOAD
TRANSFORM

Figure 11. Overview of Data integration tool [Aira]

Meltano is a powerful open-source data integration platform developed using Python. It simplifies the process of ETL-ing data from various sources into a target data store. Meltano achieves this through Singer.io's taps and targets, which serve as pre-built connectors to facilitate seamless data integration[Sin]. For easier understanding of what a data integration tool is, see Figure 11.

Taps, developed by Singer.io, are pre-built connectors that package a collection of REST API endpoints. They function as data extractors, enabling Meltano to connect with various data sources and fetch the required data. By leveraging these taps, Meltano can significantly reduce the time and effort required to build custom connectors and maintain compatibility with various data sources.[Mel][Sin]

On the other hand, Targets are responsible for loading the extracted data into a desired destination, such as a DB or DWH. These targets work in conjunction with taps to ensure smooth data flow from the source to the destination, simplifying the ETL process and promoting efficient data integration.[Mel]

One of Meltano's key features is its ability to manage multiple environments, such as testing, staging, and production. This flexibility allows for seamless transitions between different stages of the development process and ensures the integrity and reliability of the data pipeline.[Mel]

Meltano also provides robust state management capabilities, allowing for efficient tracking of pipeline progress. It achieves this through its internal Meltano.db database or by exporting the state to various common backends such as AWS S3, Azure Blob Storage, Google Cloud Storage, or a local filesystem. This feature ensures that the pipeline can resume from its last known state in case of interruptions or failures, thus minimizing data loss and redundancy.[Mel]

Another notable aspect of Meltano is its integration with dbt (Data Build Tool) and Apache Airflow. Dbt is a popular open-source transformation tool that enables data engineers to define, test, and document data transformations using SQL. Meltano's integration with dbt simplifies the transformation process and improves the overall efficiency of the data pipeline.[Mel]

Airflow, a widely used open-source platform for orchestrating complex data workflows, is also supported by Meltano. This integration is particularly beneficial for organizations like TEHIK, as it allows for better management and scheduling of data pipeline tasks, ensuring timely and accurate data processing.

Lastly, Meltano's internal database, Meltano.db, is an effective logging mechanism for tracking pipeline runs. This feature enables users to monitor the progress and performance of their pipelines, identify bottlenecks, and troubleshoot any issues that may arise during the data integration process.[Mel]

In summary, Meltano is a versatile data integration platform that leverages the power of Singer.io's taps and targets to streamline the ETL process. Its support for multiple environments, robust state management, integration with dbt and Apache Airflow, and efficient logging capabilities make it an ideal choice for organizations seeking to simplify and enhance their data integration workflows. With a strong focus on efficiency, flexibility, and scalability, Meltano is well-suited to meet the demands of modern data-driven organizations.

Setting up Meltano locally is quite straightforward. Initially, a local installation was utilized for experimentation purposes. Below are steps to get one's local Meltano project up and running with a minimal setup.

```
--install meltano
pipx install "meltano"
```

```
--set up new meltano project
meltano init DW_JIRA_ODS
```

```
--Add extractor
meltano add extractor tap-jira
```

```
--Configure extractor
  meltano config tap-jira set --interactive
```

```
--add loader
meltano add loader target-jsonl
```

```
--configure loader
  meltano config target-jsonl set --interactive
```

```
--after tap and target are configured
  meltano run tap-jira target-jsonl
```

The file structure created by initializing a new Meltano project is illustrated in Appendix III. From this figure, the hidden .meltano directory is visible and responsible for managing the meltano.db backend and other essential directories for their respective purposes.

Tap and target configurations are primarily stored in the meltano.yml file. However, sensitive information such as passwords is saved separately in the .env file to ensure security. A fully configured meltano.yml file is depicted in Appendix IV. This file also specifies the API endpoints to be requested during the data extraction process.

An extensive list of possible endpoints for the tap-jira component of Meltano can be found in Appendix II. By leveraging these endpoints, the Meltano tool can efficiently extract data from various sources and integrate it into the data warehouse for further analysis and utilization[Mel]. The modular nature of Meltano's architecture enables seamless data extraction, transformation, and loading, streamlining the entire data pipeline process for a more effective workflow.

4.6.1 Output from Meltano

Meltano extracts data from various endpoints and stores the resulting information in JSONL files. JSONL, or JSON Lines, is a file format that differs from the standard JSON format in terms of size limitations. While a JSON file cannot exceed 2 GB, a JSONL file can be larger, as long as no single line within the file is more than 2 GB. This flexibility makes JSONL a suitable choice for large-scale data extraction and storage.[jso]

The extracted data from Jira includes several JSONL files, such as:

- **issues.jsonl** - contains the bulk of data about an issue, including common fields like summary, description, and status.
- **issue_comments.jsonl** - stores comments made on issues.
- **issue_transitions.jsonl** - records the changes in an issue's status, such as transitioning from "open" to "in progress" or "resolved."
- **changelogs.jsonl** - captures the history of changes made to an issue, including updates to fields, attachments, and other modifications.

It is worth mentioning that Meltano's default configuration does not encompass the vital **fields.json** endpoint. This dataset encompasses key-value mappings for all custom fields and the schema for every custom field. The cURL command-line tool was employed to address this constraint and retrieve the necessary data from the fields.json endpoint, ensuring its integration into the data extraction process[cur].

4.7 First layer - DW_JIRA_ODS

4.7.1 Staging layer

During the development process, a singer.io target for Vertica was discovered. Unfortunately, it was unavailable in the pypi[pyp][tarb]. Despite the challenges, the package was eventually installed locally after several attempts. However, due to its outdated nature, the package exhibited significant shortcomings, making it unsuitable for use as a loader component.

This situation led to incorporating an intermediate conversion step to JSONL format between the extraction and loading processes using meltanos target-jsonl[tara]. Adopting this additional step allowed for a more flexible and manageable data pipeline, ensuring that data could be successfully integrated into the staging layer. As a result, the pipeline remained functional and adaptable to changing requirements, even in the face of limitations presented by the outdated Vertica target[tarb].

Vertica is a high-performance columnar database management system designed to handle both structured and unstructured data efficiently. Its capabilities make it an ideal choice for scalable solutions. It supports horizontal scaling by adding more nodes to the cluster, enabling organizations to adapt to increasing data volumes and performance requirements easily.[verb]

Vertica's architecture is built upon a distributed, shared-nothing design, where each node operates independently and shares no memory or disk storage with other nodes. This approach ensures high availability and fault tolerance, making it suitable for large-scale data warehousing and analytics use cases.[vera]

One of the notable features of Vertica is its support for Flex Tables, which provide a flexible and efficient means of storing and querying unstructured or semi-structured data. Flex Tables offer several advantages in an ELT architecture context. First, they allow for a schema-less design, enabling the ingestion of data without the need for predefined structures. This flexibility simplifies incorporating new data sources or making changes to existing ones.[vera]

Moreover, Flex Tables also support advanced analytics capabilities, such as full-text search and pattern matching, which can be particularly useful when working with unstructured data[vera]. However, there are some trade-offs to consider when using Flex Tables. Due to their schema-less nature, they can be less efficient than traditionally structured tables for certain queries, particularly those requiring complex joins or aggregations. Nonetheless, the benefits they offer in terms of flexibility and ease of use often outweigh these drawbacks, particularly in cases where the data schema needs to be more well-defined and subject to frequent changes.

Vertica's Flex Tables also support bulk loading, an efficient method for ingesting large volumes of data into the DB[vera]. With bulk loading, data is loaded in large batches rather than row by row, resulting in significant performance improvements. Flex Tables further enhance the bulk loading process by allowing users to define default operations for columns being copied in through DDL statements. This feature simplifies the data-loading process and reduces the likelihood of errors caused by missing or mismatched columns.

In addition to its robust DB capabilities, Vertica offers a CLI DB client called vsql. A DB client is a software application that enables communication between a user or another application and the DB server. vsql allows users to interact with the Vertica DB, execute SQL queries, and manage DB objects from the command line. This makes it an invaluable tool for developers and administrators working with Vertica. Furthermore, vsql is particularly effective when utilizing containerization due to its minimal overhead. In this context, manual loadings and other DB management tasks were primarily carried out using a SQL executor such as DBeaver, highlighting the versatility and flexibility of the available tools for working with Vertica.[vera][dbe]

In the context of the DV 2.0 staging layer, the following SQL code demonstrates how the Vertica DB can be utilized to create and modify a flex table for managing Jira issue data. The flex table, named **DW_JIRA_ODS.stg_issues**, is designed according to Data Vault methodology, incorporating specific fields that serve essential purposes in the DWH'ing process.

```sql
DROP TABLE IF EXISTS DW_JIRA_ODS.stg_issues;
CREATE FLEX TABLE DW_JIRA_ODS.stg_issues();
ALTER TABLE DW_JIRA_ODS.stg_issues ADD COLUMN CHANGE_DTTM TIMESTAMP
    DEFAULT CURRENT_TIMESTAMP;
ALTER TABLE DW_JIRA_ODS.stg_issues ADD COLUMN HASH_DIFF INT DEFAULT
    HASH(__raw__['key'], __raw__['fields.updated']);
ALTER TABLE DW_JIRA_ODS.stg_issues ADD COLUMN ISSUES_HASH_KEY INT
    DEFAULT HASH(__raw__['key']);
```

The **CHANGE_DTTM** column, which stores the timestamp of the data change, is added to the flex table. It helps track the changes made to the data and ensures that the most recent information is always available.

Next, the **HASH_DIFF** and **ISSUES_HASH_KEY** columns are created to store hash values. These values are calculated using Vertica's **HASH** function, a 64-bit function that returns an integer value. Given the number of records held, this hash function is sufficient to mitigate the risk of hash collisions. The **HASH** function can be executed during the local **COPY** command, ensuring efficient computation. The **HASH_DIFF** column is based on the unique combination of the Jira issue key and the updated field. In contrast, the **ISSUES_HASH_KEY** column is derived solely from the Jira issue key for possible faster querying for joins if necessary.

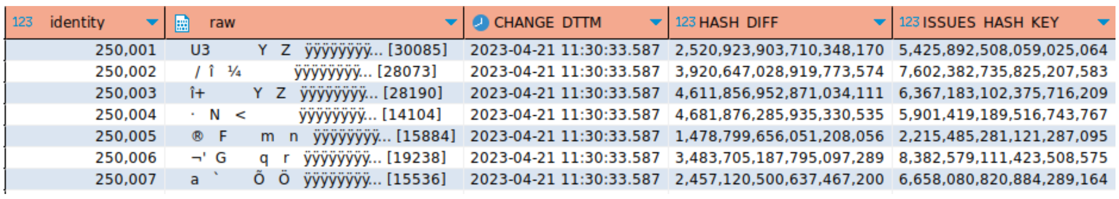
Additionally, the SQL code utilizes Vertica's built-in JSON parser to extract relevant information from the raw Jira issue data[vera]. By parsing the JSON data, the DB can effectively identify and process the necessary attributes, such as the Jira issue key and the updated field. This approach ensures that the data warehousing solution can effectively manage and process the Jira data according to the Data Vault methodology.

The SQL code provided below demonstrates the simplicity and efficiency of the data loading process due to the carefully designed DDL for the **DW_JIRA_ODS.stg_issues** flex table.

First, the **stg_issues** table is truncated, ensuring it is empty and ready to receive new data. Then, the **COPY** command loads data directly from the local JSONL file located in the container's volume into the **stg_issues** table. This process is achieved using Vertica's built-in **fjsonparser()** function, which parses the JSONL data and inserts it into the corresponding flex table columns.

```
TRUNCATE TABLE DW_JIRA_ODS.stg_issues;
COPY DW_JIRA_ODS.stg_issues("__raw__")
FROM LOCAL '/project/output/issues.jsonl' PARSER fjsonparser();
```

The remaining entities in the staging layer were managed using a comparable approach. The overview of staging table can be seen in Figure 12.



identity	raw	CHANGE_DTTM	HASH_DIFF	ISSUES_HASH_KEY
250,001	U3 Y Z ÿÿÿÿÿÿÿÿ... [30085]	2023-04-21 11:30:33.587	2,520,923,903,710,348,170	5,425,892,508,059,025,064
250,002	/ î ¼ ÿÿÿÿÿÿÿÿ... [28073]	2023-04-21 11:30:33.587	3,920,647,028,919,773,574	7,602,382,735,825,207,583
250,003	î+ Y Z ÿÿÿÿÿÿÿÿ... [28190]	2023-04-21 11:30:33.587	4,611,856,952,871,034,111	6,367,183,102,375,716,209
250,004	· N < ÿÿÿÿÿÿÿÿ... [14104]	2023-04-21 11:30:33.587	4,681,876,285,935,330,535	5,901,419,189,516,743,767
250,005	® F m n ÿÿÿÿÿÿÿÿ... [15884]	2023-04-21 11:30:33.587	1,478,799,656,051,208,056	2,215,485,281,121,287,095
250,006	¬' G q r ÿÿÿÿÿÿÿÿ... [19238]	2023-04-21 11:30:33.587	3,483,705,187,795,097,289	8,382,579,111,423,508,575
250,007	a ` Õ Ö ÿÿÿÿÿÿÿÿ... [15536]	2023-04-21 11:30:33.587	2,457,120,500,637,467,200	6,658,080,820,884,289,164

Figure 12. stg_issues screenshot from Dbeaver CE

4.7.2 ODS layer

The ODS layer serves as an intermediate storage area for data, functioning as a bridge between the staging layer and the final DWH. The ODS layer provides a clean and integrated view of the data while preserving the low-level details needed for operational reporting and decision-making. In this context, a Vertica table had to be created for the ODS layer, as reading from the flex table was not feasible due to certain technical limitations.

The following SQL demonstrates the creation of the ODS layer table, named **DW_JIRA_ODS.ods_issues**:

```
CREATE TABLE IF NOT EXISTS DW_JIRA_ODS.ODS_ISSUES (ISSUES_JSON LONG
    VARBINARY(32000000), CHANGE_DTTM TIMESTAMP, HASH_DIFF INT,
    ISSUES_HASH_KEY INT) UNSEGMENTED ALL NODES;
```

In this SQL code, the **ods_issues** table is created with four columns: **issues_json**, **CHANGE_DTTM**, **HASH_DIFF**, and **ISSUES_HASH_KEY**. The **issues_json** column is of type *long varbinary* and is designed to store the JSON data related to Jira issues. The other three columns serve the same purposes as described in the staging layer section.

The **UNSEGMENTED ALL NODES** clause at the end of the SQL code defines how the table data is distributed across the nodes in a Vertica cluster. By specifying **UNSEGMENTED ALL NODES**, the table data is stored in an unsegmented manner, meaning that each node in the cluster receives a full copy of the table. This approach ensures that the data is equally distributed and accessible across all nodes, improving query performance and fault tolerance by minimizing the need for data redistribution during query execution.

The following pseudoSQL query demonstrates the process of populating the **DW_JIRA_ODS.ods_issues** table with data from the **DW_JIRA_ODS.stg_issues** table.

```
INSERT INTO ods_issues(<columns>)
SELECT stg_<columns>
FROM stg_issues LEFT JOIN ods_issues ON ods_issues.HASH_DIFF =
    stg_issues.HASH_DIFF
WHERE ods_issues.HASH_DIFF is null;
COMMIT;
```

The SQL query effectively utilizes hash values in the **DW_JIRA_ODS.ods_issues** table for efficient comparisons and joins. By using the **HASH_DIFF** column as the join condition between the **stg_issues** and **ods_issues** tables, the query can quickly identify new or changed records without the need for complex and lengthy comparisons.

This approach, which leverages the precomputed hash values, substantially shortens the query length compared to alternative methods that might involve multiple nested **CASE WHEN** statements. The use of hash values not only simplifies the query but also improves its performance, making it a suitable choice for handling large-scale DWH'ing tasks.

The remaining entities in the ODS layer were managed using a similar approach. The resulting layer structure overview can be seen in Figure 13.

4.8 EDW layer

Within the EDW layer, the Raw Vault is an essential subcomponent that stores the data extracted from multiple source systems. The Raw Vault follows the DV 2.0 methodology, which emphasizes separating data into distinct entities such as Hubs, Links, and Satellites.

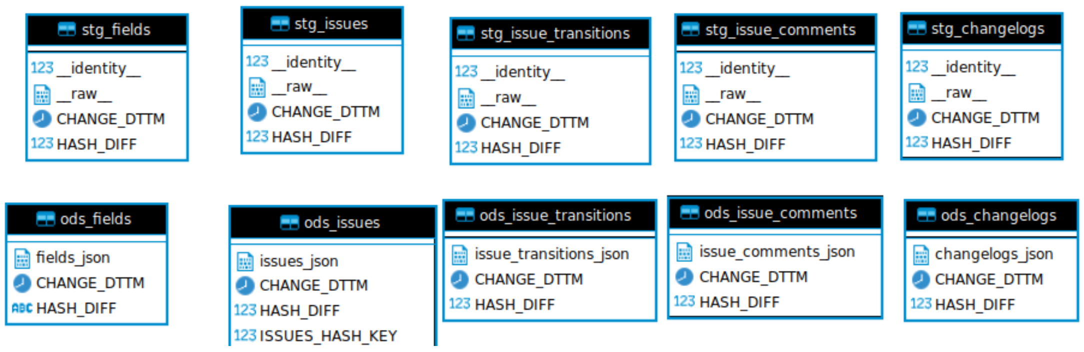


Figure 13. DW_JIRA_ODS ERD screenshot from Dbeaver CE

This approach enables the seamless integration of new data sources, provides a high degree of flexibility, and ensures data integrity and traceability. [LO15]

Before diving into the modeling process, it is essential to conduct thorough preparation with the source data and engage in discussions with analysts. This preparation consists of several crucial steps that ensure the creation of an accurate and efficient data model. The following steps outline the process in detail:

- **a) Identify business keys**: The first step is to pinpoint the business keys, which are the unique identifiers for various data entities. These keys play a vital role in data integration and retrieval, providing a reliable method of connecting and associating data across different sources.

- **b) Discover relationships between business keys**: Once the business keys have been identified, it is crucial to determine their relationships. Understanding the connections among various data entities will aid in creating a comprehensive and accurate data model that accurately reflects the underlying business processes and requirements.

- **c) Identify attributes of business keys**: After determining the relationships between business keys, the next step is to identify the attributes associated with each key. Attributes provide additional information and context for the data entities, enabling a more detailed understanding of the data and facilitating more accurate and insightful analysis.

- **d) Draw a logical model for each entity**: With the business keys, relationships, and attributes identified, the next step is to create a logical model for each issue. This model serves as a visual representation of the data entities, their attributes, and the relationships between them. It helps in understanding the overall structure of the data and provides a solid foundation for the subsequent steps in the modeling process.

- **f) Implement the entity**: Finally, after completing the logical model, the last step is implementing the issue. This step involves designing and building the physical data model, considering the insights gathered from the logical model, the relationships between business keys, and the attributes of the data entities.

Based on the detailed preparation and understanding of the source data, business keys, relationships, and attributes, sprints were planned and executed accordingly. This approach ensured that the development process was in line with the Agile methodology, focusing on delivering value incrementally and responding effectively to changing requirements.

Due to the limitations set on the length of this thesis, an in-depth discussion of the planning and execution of the sprints is not included in the work.

4.8.1 Jira Issues Structure Mapping and Collaborating with Analysts for Data Verification and Validation

The general structure of a Jira issue comprises several standard fields, such as status, name, priority, and more. These fields are essential for organizing and managing issues within the platform. In addition to the standard fields, Jira allows users to define custom fields tailored to their specific needs. These custom fields can be created based on a combination of Jira's supported data types and object types.[jira]

A full json tree about one Jira issue can be seen on Appenix V. The extensive vertical tower represents all available custom fields Jira's backend provides. Those without connections between values, indicating that they are not mapped, can be customized to suit specific needs.

Data Types: Jira provides various data types for custom fields, which determine the kind of values the REST API accepts and returns for the field, the field's behavior in JQL, and its default rendering behavior. The available data types include:

- **String**: Plain strings with autocomplete and exact comparison in JQL.
- **Number**: Double-precision 64-bit IEEE 754 floating-point numbers.
- **User**: Users identified by Atlassian account IDs, with behavior similar to other user fields in Jira.
- **Group**: Groups identified by names, with behavior similar to other group fields in Jira.
- **Object**: Arbitrary JSON objects (see Object Type below for more details).
- **Datetime**: Strings representing dates with time and timezone, behaving like other datetime fields in Jira.[jira]

Object Types: In addition to the data types mentioned earlier, Jira also provides a variety of object types for custom fields, which can be added to request types and made visible on the customer portal. These object types allow users further to customize their issue-tracking and project-management experience. Three out of a total of the supported object types include:

- **Checkbox**: Enables users to select multiple options from a predefined list.
- **Radio Button**: Provides a set of options where users can select only one.
- **Single choice and Multiple choice**: Allow users to select one or multiple options, respectively, from a predefined list.[jira]

4.8.2 Preparation with the Analyst

Before delving into the modeling process, a significant amount of groundwork was undertaken in collaboration with the analyst to ensure the accuracy and integrity of the data. This initial preparation stage involved a thorough examination of all fields utilized in the reports, as well as an assessment of their data types, structures, categories, and relationships between attributes. Furthermore, priorities were set for the Agile development process, emphasizing the rapid delivery of simple and easily achievable results before proceeding sprint by sprint, aligning with the DV 2.0 methodology.

In total, 110 fields were meticulously examined and discussed with the analyst to guarantee their correctness and validity. This process often involved modifying certain fields to ensure their suitability for modeling. The emphasis during these discussions was on understanding the underlying data structure and relationships, which facilitated the prioritization of tasks for Agile development. The collaboration with the analyst proved invaluable in identifying potential issues early in the process, allowing for necessary adjustments to be made before proceeding with the modeling phase.

This comprehensive analysis of the fields laid the foundation for effective modeling, ensuring that the data was accurately represented and suitable for the intended purpose. In addition to thoroughly examining each field, the priorities set for Agile development enabled a streamlined approach, focusing on delivering results quickly and efficiently. By tackling more straightforward tasks first, the solution gained momentum and built upon previous successes, setting the stage for a more effective sprint-by-sprint progression in line with the Data Vault 2.0 framework.

The snapshot of an excel table used to identify unique combinations of data types and entities to be modeled, can be seen in Appendix VI.

The complete list of entities to be modeled can be seen in Appendix VII.

4.8.3 First iteration - hubs

In the first iteration of the EDW, the business keys are identified as composite keys, consisting of the project key (the string component) and the issue's queue number, for example TKT-123456. For the sake of simplicity, it was decided to use the entire string as the issue's business key and designate the project string as another separate business key. Based on that, two hubs: **ISSUES_HUB** and **PROJECT_HUB** were created.

The next logical step was establishing the relationship between these two hubs. This step is achieved through the creation of the **ISSUES_PROJECT_LINK**, which serves to connect the **ISSUES_HUB** and **PROJECT_HUB** effectively.

The DDL without defined datatypes for the sake of length for **ISSUES_HUB** is as follows:

```sql
/*DROP TABLE IF EXISTS EDW_JIRA.ISSUES_HUB CASCADE;*/
CREATE TABLE IF NOT EXISTS EDW_JIRA.ISSUES_HUB (
  ISSUES_ID, ISSUES_BK, REC_SOURCE, LOAD_DTS,
  CONSTRAINT pk_ISSUES_HUB PRIMARY KEY (ISSUES_ID)
)
ORDER BY ISSUES_ID SEGMENTED BY HASH (ISSUES_ID) ALL NODES;
COMMENT ON TABLE EDW_JIRA.ISSUES_HUB IS 'Jira issues business keys';
GRANT SELECT ON EDW_JIRA.ISSUES_HUB TO EDW_JIRA_VIEWER WITH GRANT
    OPTION;
GRANT SELECT ON EDW_JIRA.ISSUES_HUB TO DWH_JIRA WITH GRANT OPTION;
```

DML for hubs, the following SQL query, is also relatively straightforward. Only the distinct keys are selected, simplifying the process.

```sql
INSERT INTO EDW_JIRA.ISSUES_HUB
SELECT ods_issues.ISSUES_ID, ods_issues.ISSUES_BK, 'JIRA',
    CURRENT_TIMESTAMP
FROM (
  SELECT DISTINCT sha1(issues_json['key']::VARCHAR(64)) AS "ISSUES_ID
      ", issues_json['key']::VARCHAR(64) AS "ISSUES_BK"
  FROM DW_JIRA_ODS.ods_issues
) ods_issues
LEFT JOIN EDW_JIRA.ISSUES_HUB ON EDW_JIRA.ISSUES_HUB.ISSUES_ID =
    ods_issues.ISSUES_ID
WHERE EDW_JIRA.ISSUES_HUB.ISSUES_ID IS NULL;
COMMIT;
```

DDL and DML for the**PROJECT_HUB** was done in a comprehensible manner.

4.8.4 First iteration - links

DDl for the **ISSUES_PROJECT_LINK**:

```
CREATE TABLE IF NOT EXISTS EDW_JIRA.PROJECT_ISSUES_LINK (PROJECT_ID
    CHAR(40), ISSUES_ID CHAR(40), REC_SOURCE VARCHAR(20), LOAD_DTS
    TIMESTAMP, CONSTRAINT pk_PROJECT_ISSUES_LINK PRIMARY KEY (
    PROJECT_ID, ISSUES_ID), CONSTRAINT
    fk_PROJECT_ISSUES_LINK_PROJECT_HUB FOREIGN KEY (PROJECT_ID)
    REFERENCES EDW_JIRA.PROJECT_HUB (PROJECT_ID), CONSTRAINT
    fk_PROJECT_ISSUES_LINK_ISSUES_HUB FOREIGN KEY (ISSUES_ID)
    REFERENCES EDW_JIRA.ISSUES_HUB (ISSUES_ID)) ORDER BY PROJECT_ID,
    ISSUES_ID SEGMENTED BY HASH(PROJECT_ID) ALL NODES;

CREATE PROJECTION IF NOT EXISTS EDW_JIRA.
    PROJECT_ISSUES_LINK_ISSUES_ID AS SELECT PROJECT_ID, ISSUES_ID FROM
     EDW_JIRA.PROJECT_ISSUES_LINK ORDER BY ISSUES_ID, PROJECT_ID
    SEGMENTED BY HASH(ISSUES_ID) ALL NODES;

GRANT SELECT ON EDW_JIRA.PROJECT_ISSUES_LINK TO EDW_JIRA_VIEWER,
    DWH_JIRA WITH GRANT OPTION;
```

The part of the query where the projection is created serves a critical purpose in the Vertica DB. Projections are a crucial feature in Vertica that optimizes query performance by pre-sorting, pre-joining, and pre-aggregating data.[vera]

In this query, a projection named **EDW_JIRA.PROJECT_ISSUES_LINK_ISSUES_ID** is created using the **CREATE PROJECTION IF NOT EXISTS** statement. The projection is based on the **EDW_JIRA.PROJECT_ISSUES_LINK** table and contains the **PROJECT_ID** and **ISSUES_ID** columns.

DML for the link:

```
INSERT INTO EDW_JIRA.PROJECT_ISSUES_LINK (PROJECT_ID, ISSUES_ID,
    REC_SOURCE, LOAD_DTS)
SELECT DISTINCT sha1(issues_json['fields.project.key']::VARCHAR(64))
    AS PROJECT_ID, sha1(issues_json['key']::VARCHAR(64)) AS ISSUES_ID,
     'JIRA', CURRENT_TIMESTAMP
FROM DW_JIRA_ODS.ods_issues ods_issues
LEFT JOIN EDW_JIRA.PROJECT_ISSUES_LINK link ON link.PROJECT_ID =
    ods_issues.PROJECT_ID AND link.ISSUES_ID = ods_issues.ISSUES_ID
WHERE link.ISSUES_ID IS NULL;
COMMIT;
```

The **ORDER BY** clause in the projection definition specifies the sorting order of the data, first by **ISSUES_ID**, and then by **PROJECT_ID**. The **SEGMENTED BY**

HASH(ISSUES_ID) clause is used to distribute the data across all nodes in the Vertica cluster based on the hash value of the **ISSUES_ID** column. In summary, the creation of this projection helps improve query performance by pre-sorting the data based on specified columns and distributing it efficiently across the Vertica cluster using a hash-based segmentation strategy.

4.8.5 First Iteration - Satellites

In the first iteration, the focus was on incorporating purely string or option-type attributes, which could be easily integrated into the underlying structure created by the DDL. This approach ensured that the initial implementation could handle the simplest data types and set the foundation for more complex attributes in subsequent iterations.

To determine the appropriate data lengths for these attributes, a length measurement technique was employed to obtain a rough estimate of the required sizes. This measurement served as a guideline for creating the DDL statements and ensuring efficient data storage. Using a window function, the following SQL query maps all keys from the **issues_json** to their maximum length. It filters out keys with a specific **type_oid** (199) and keys that contain the strings 'customfield' and 'project'. The result is then grouped by **keys** and **type_oid**, and ordered in descending order based on the first column. **type_oid** (199) means array data type in Vertica DB.[vera]

```
SELECT keys, type_oid, MAX("length") max_length
FROM (SELECT MAPKEYSINFO(issues_json) OVER(PARTITION BEST) FROM
    DW_JIRA_ODS.ods_issues) AS a
WHERE type_oid NOT IN (199) AND keys NOT LIKE '%customfield%' AND
    keys NOT LIKE '%project%'
GROUP BY keys, type_oid
ORDER BY 1 DESC;
```

Additionally, generative SQL was utilized to create both DDL and DML statements dynamically. This approach streamlined the process of defining and manipulating the data structure, enabling a smoother transition into more complex data types in future iterations.

This following SQL query selects distinct keys, removes the prefix **'fields.project.'** using a regular expression, and assigns an alias for each key. The query is based on a subquery that maps keys from the **issues_json** using a window function.

```
SELECT DISTINCT keys, REGEXP_REPLACE(keys, '^fields\.project\.', '')
    AS alias, 'VARCHAR(1000)' AS datatype
FROM (SELECT MAPKEYSINFO(issues_json) OVER(PARTITION BEST) FROM
    DW_JIRA_ODS.ods_issues) AS a
WHERE type_oid NOT IN (199) AND keys NOT LIKE '%customfield%' AND
    keys NOT LIKE '%project%'
```

```
GROUP BY keys, type_oid
ORDER BY keys DESC;
```

In order to optimize the data integration process and minimize the overhead caused by redundant comparisons against the ODS layer, staging satellites (STG satellites) are employed. These STG satellites act as intermediary storage, facilitating efficient data handling from the source systems before loading it into the target ODS.

The logic behind the STG satellites loading process is as follows:

1. Initially, the staging layer is purged to ensure a clean and updated environment for data processing. Following the purge, data is extracted from the ODS layer and loaded into the STG satellites. The selection criteria are based on the **MAX(LOAD_DTS)** from the EDW satellite, which provides the starting point for querying records up to the most recent entry in the STG satellite.

2. Subsequently, a comparison occurs between the data residing in the STG satellites and the EDW layer. This layer is accomplished by utilizing the **HASH_DIFF** value, which is derived from hashing a lengthy concatenated string of column values from the satellite. The primary objective of this step is to detect any newly added, modified, or removed records that require synchronization with the ODS layer.

3. Once the differences between the STG satellites and EDW layer satellites have been identified, the necessary data updates and insertions are performed in the EDW layer. This process ensures that the EDW layer remains up-to-date and consistent with the source systems' data.

Staging satellites make the data integration process more efficient, as the redundant complete set comparison against the ODS layer is avoided. This approach leads to improved performance, reduced resource consumption, and streamlined data processing.

The loading of the satellite, see the following SQL query, is similar to STG to ODS transition.

```
INSERT INTO EDW_JIRA.ISSUES_JIRA_HDR_SAT (<list of columns>)
SELECT STG.<list of columns>
FROM EDW_JIRA.STG_ISSUES_JIRA_HDR_SAT AS STG
LEFT JOIN EDW_JIRA.ISSUES_JIRA_HDR_SAT SAT
ON STG.ISSUES_ID = SAT.ISSUES_ID
AND STG.HASH_DIFF = SAT.HASH_DIFF
WHERE
SAT.ISSUES_ID IS NULL;
COMMIT;
```

4.8.6 Following Iterations

As the project progressed, more complex elements were introduced in subsequent iterations/sprints. One of these additions was the implementation of SLAs, which are contracts that define the level of service expected by a customer from a service provider. To handle SLAs, they were grouped into two types of satellites:
ISSUES_JIRA_SLA_STR_SAT and **ISSUES_JIRA_SLA_NS_ARR_SAT**. The first variant, **ISSUES_JIRA_SLA_STR_SAT**, deals with SLAs with string data types, while the second variant, **ISSUES_JIRA_SLA_NS_ARR_SAT**, handles SLAs with nested array data types.

Another critical aspect of the project was managing data related to projects handled by the testing department. To address this need, all data associated with these projects were grouped under two testing satellites: **ISSUES_JIRA_TESTING_SAT** and **ISSUES_JIRA_TESTING_MSAT**. The **ISSUES_JIRA_TESTING_MSAT** is a multi-active satellite, meaning one key can have multiple values.

One crucial aspect of working with complex data structures is parsing JSON arrays. JSON arrays are ordered lists of values comprising various data types, including strings, numbers, objects, and even other arrays. In order to effectively query and manipulate this type of data, it is necessary to perform denormalization.

Denormalization transforms a hierarchical or nested data structure, such as a JSON array, into a flattened, tabular format. This process enables easier querying and analysis of the data since it removes the need to navigate through the nested structure. Denormalizing the data makes it more accessible and can be efficiently used in various analytical processes.[RH22]

The importance of denormalization lies in its ability to facilitate data querying and manipulation. With denormalization, it can be easier to work with nested data structures, as they are not easily interpretable by traditional querying methods. Users can effectively query and extract the necessary information from the data by converting these structures into a flattened format.

Below is a SQL that was used to select the fields going into ISSUES_ISSUES_LINK. The attributes for the ISSUES_ISSUES_LINK_SAT can be received similarly.

```sql
SELECT
SHA1(src."key") AS ISSUES_ID,
SHA1(src."values") AS RELATED_ISSUES_ID,
 src."key" AS ISSUES_BK,
 src."values" AS RELATED_ISSUES_BK

FROM (
SELECT sub.key,
sub.values,
REGEXP_REPLACE(keys, '\d+.', '') keys_path,
REGEXP_REPLACE(REGEXP_REPLACE(keys,'[^0-9.]','' ),'.+','.') keys_ident
FROM (
SELECT
issues_json['key']::VARCHAR(64) "key",
MAPITEMS(MAPJSONEXTRACTOR(MAPTOSTRING(issues_json['fields.
    customfield_19203']))) OVER ( PARTITION BY
issues_json['key']::VARCHAR(64))
FROM DW_JIRA_ODS.ods_issues
) sub
WHERE keys_path = 'key'
) src
```

In this query, we start by **flattening the JSON data** using the **MAPJSONEXTRACTOR** function in the first subquery. This function helps us extract relevant data from the nested JSON structure. Next, we use the **mapitems** function to map the extracted keys and values.

After mapping the keys and values, we apply some **REGEX** to process and clean up the keys. Specifically, we remove digits and periods from the keys to obtain the **keys_path**, and we strip out all non-numeric characters and consecutive periods from the keys to get the **keys_ident**.

Once we have processed the keys, we filter the results only to include rows where the **keys_path** is equal to 'key.' This key helps us focus on the data we are interested in.

Finally, we select the desired columns from the filtered data, such as the original issue keys and their related issue keys, as well as their hashed representations using the **sha1** function.

The Data Vault model for the EDW_JIRA, although not complete, can be seen in Appendix VIII. Further enhancements and features were developed in the following iterations, but they are not covered in this paper due to length constraints.

4.9 DWH layer

Within the scope of this thesis, the presentation layer was designed to provide a unique combination of states, specifically, the latest states from the satellites. This approach ensures end-users can access the most recent and relevant information for their decision-making needs. However, the DV2.0 methodology allows for more flexibility and control over the combinations of data states using PIT tables.

PIT tables are an essential component of the DV2.0 architecture, enabling users to access historical snapshots of data at various points in time. By providing a time-based reference for the data stored in the Raw Vault, PIT tables allow users to track changes and analyze trends over time. This functionality is precious for organizations that need to understand the historical context of their data or analyze the impact of business decisions and events over time.[LO15]

In the context of this project, as it currently stands, three entities have been implemented within the DWH layer. **D_ISSUES** - issues dimension, **D_PROJECT** - project dimension, **B_PROJECT_ISSUES** - bridge object between dimensions.

The following SQL query illustrates the logic used to construct the **D_ISSUES** dimension by combining data from 3 satellite tables.

```
CREATE OR REPLACE VIEW DWH_JIRA.D_ISSUES AS
SELECT
    IH.ISSUES_BK AS D_ISSUES_ID, IJHS.LOAD_DTS,
    CASE WHEN IJHS_LAST.ROW_NUM = 1 THEN 1 ELSE 0 END AS
        LAST_VALUE_IND,
    IJHS.<rest of columns of sat1>
FROM EDW_JIRA.ISSUES_HUB IH
JOIN EDW_JIRA.ISSUES_JIRA_HDR_SAT IJHS ON IH.ISSUES_ID = IJHS.
    ISSUES_ID
JOIN (SELECT ISSUES_ID, LOAD_DTS, ROW_NUMBER() OVER (PARTITION BY
     ISSUES_ID ORDER BY LOAD_DTS DESC) AS ROW_NUM FROM EDW_JIRA.
    ISSUES_JIRA_HDR_SAT) IJHS_LAST ON IJHS_LAST.ISSUES_ID = IH.
    ISSUES_ID AND IJHS_LAST.LOAD_DTS = IJHS.LOAD_DTS
LEFT JOIN (SELECT <columns of sat2>, ROW_NUMBER() OVER (PARTITION
     BY ISSUES_ID ORDER BY LOAD_DTS DESC) AS ROW_NUM FROM EDW_JIRA
    .ISSUES_JIRA_CF_STR_OPT_SAT) CF_STR_SAT ON CF_STR_SAT.
    ISSUES_ID = IH.ISSUES_ID AND CF_STR_SAT.ROW_NUM = 1
LEFT JOIN (SELECT <columns of sat3>, ROW_NUMBER() OVER (PARTITION
     BY ISSUES_ID ORDER BY LOAD_DTS DESC) AS ROW_NUM FROM EDW_JIRA
    .ISSUES_JIRA_SLA_STR_SAT) SLA_STR_SAT ON SLA_STR_SAT.ISSUES_ID
     = IH.ISSUES_ID AND SLA_STR_SAT.ROW_NUM = 1;

GRANT SELECT ON DWH_JIRA.D_ISSUES TO DWH_JIRA_VIEWER WITH GRANT
    OPTION;
```

The unique combination of last states is achieved by using **ROW_NUMBER() OVER (PARTITION BY ISSUES_ID ORDER BY LOAD_DTS DESC) AS ROW_NUM**. This calculation is performed separately for each satellite. The **PARTITION BY** clause groups the data by **ISSUES_ID**, ensuring that the numbering restarts for each unique issue. The **ORDER BY** clause sorts the data in descending order based on the **LOAD_DTS** column, which represents the load timestamp. As a result, the most recent record for each issue will have a **ROW_NUM** value of **1**, indicating the last state.

After calculating the row numbers for each satellite, the subqueries are then left-joined to the main query using the **ISSUES_ID** column and the condition **ROW_NUM = 1**. This ensures that only the most recent records from each satellite, representing the unique last states, are included in the final view. By utilizing the **ROW_NUMBER** window function in this manner, the DWH view combines the latest information from all satellites, providing a comprehensive and up-to-date picture of the entities.

4.10 Containerization with Docker

TEHIK uses Docker to containerize its applications, enabling the organization to package applications and dependencies into a single unit. TEHIK employs a private Docker repository to store and manage all container images for security purposes. This approach ensures that only authorized personnel access these images, minimizing the risk of unauthorized usage or tampering.

Docker is a platform that simplifies the deployment and management of applications by automating the process of creating, deploying, and running application containers. A container is a lightweight, portable unit that includes an application and all its dependencies, such as libraries, runtime, and system tools. Containers provide a consistent environment for applications, regardless of the underlying infrastructure.[doc]

Docker utilizes Dockerfiles, text files containing instructions for building Docker images. Docker images are created by following these instructions in a step-by-step manner. The process starts with a base image, a minimalistic and secure operating system layer. The base image is then extended by adding application-specific components and dependencies according to the Dockerfile instructions. The resulting Docker image contains everything needed to run the application, ensuring consistency and reliability across different environments.[doc]

When a Docker image is executed, it creates a runtime environment called a Docker container. Containers isolate the application from the host system, ensuring that the application runs consistently across various platforms. This isolation also provides security benefits by limiting the potential impact of vulnerabilities in the application or its dependencies.[doc]

Docker supports mounting external storage volumes, which allow containers to store and retrieve data outside the container's filesystem. This feature helps retain files and data across container restarts or updates, enabling persistent storage and seamless data sharing between containers.[doc]

The first step was to containerize vsql. The algorithm to achieve this is as follows:

- 1. Specify the base image, such as a lightweight Linux distribution (e.g., Alpine Linux).
- 2. Install any required dependencies, like the Vertica client libraries and other system tools.
- 3. Copy the VSQL binary or install the VSQL package from the official repository.
- 4. Set any necessary environment variables, such as DB connection information, credentials, or default settings.

The code for actual vsqls dockerfile can be seen in Appendix X. Based on the same conceptual algorithm, Meltano was also containerized.

All necessary authorization information, such as DB addresses, ports, usernames, and passwords, is stored in a .env file. Additionally, the .env file contains an environment variable called **MELTANO_ROOT**, which holds the root folder path for Meltano's local directory.

In the implementation process, script wrapping is employed to manage the execution of multiple scripts sequentially. Script wrapping refers to using one script to execute other scripts or commands, acting as an intermediary between the user and the target scripts or commands. This approach helps in structuring the workflow, handling dependencies, and maintaining the readability of the overall code.

Two series of scripts are utilized to extract data from Jira and load it into Vertica using Docker containers. The first script series focuses on starting the Meltano container and setting up the required environment. The **meltano_docker_executable.sh** script starts the Meltano container and subsequently executes the **entrypoint.sh** script within the container. The **entrypoint.sh** script serves as the entry point, initializing the environment and ensuring that all necessary dependencies and configurations are in place.

The second series of scripts handle the actual data extraction from Jira and loading it into Vertica. The first sequence is visualized in Figure 14 and the second in Figure 15.

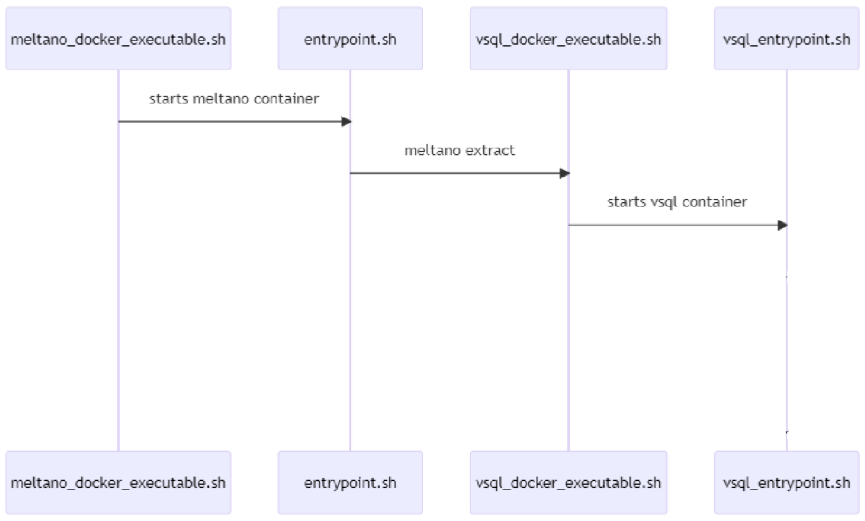


Figure 14. Sequence of scripts to get Data from Jira to local machine with Meltano container

The rest of the steps can be run analogously with the vsql container. The implementation of the pipeline is quite uncomplicated:

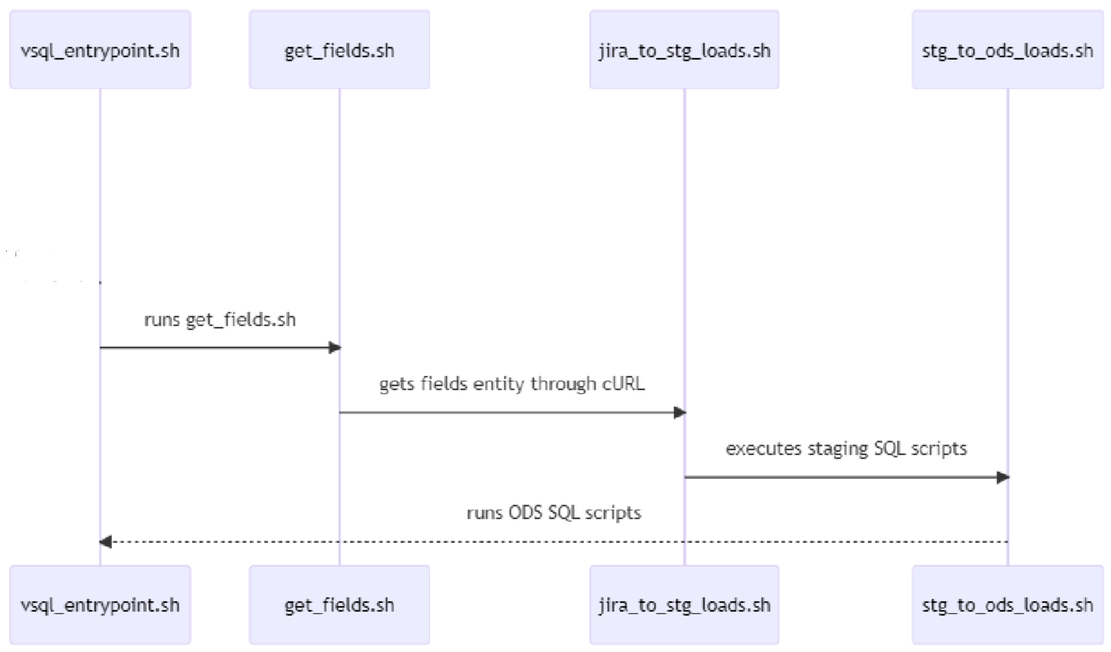


Figure 15. Sequence of scripts to get Data from local machine to Vertica ODS layer with vsql container

- 1. Pull the prebuilt Meltano and vsql images from the TEHIK container repository
- 2. Create the .env file and add all necessary credentials to it
- 3. Change the MELTANO_ROOT=directory where the .env file is located
- 4. Run Extract:

```
source /\$(pwd)/.env && docker run -it --rm --mount type=bind,
    source="$MELTANO_ROOT/.env",target=/project/.env --entrypoint
    =bash -v "$MELTANO\_ROOT:/project/" docker\_meltano -c "/
    project/scripts/entrypoint.sh"
```

- 5. Run load:

```
source /\$(pwd)/.env && docker run -it --rm --mount type=bind,
    source="$MELTANO_ROOT/.env",target=/project/.env --entrypoint
    =bash -v "$MELTANO_ROOT/output:/project/output" docker_vsql -
    c "/project/vsql_entrypoint.sh"
```

Some of the bash scripts used, can be seen in Appendix XI.

4.11 Implementation environment

In order to validate the developed data pipeline components, this section outlines the environment setup for developing and running the example data pipelines. The required steps for performing this evaluation are listed below.

- A local machine with Windows 10 operating system and appropriate hardware resources was configured (16GB of RAM, 8 CPU cores)
- Oracle VirtualBox was installed on the local machine to host an Ubuntu 20.04 VM with allocated VM resources (50% of host resources)
- Zsh shell and Bash shell were installed, with their respective versions
- Vim and Visual Studio Code were set up as the text editors for the project
- Python programming language, versions 3.8 through 3.11, were utilized for the data pipeline development
- Jira software, v8.22.4, was employed for project management and task tracking. The same version was also used for extracting the data from
- Git was used for version control and as a code repository
- Vertica SQL was used
- Meltano, with its corresponding versions from v2.9.0 to v2.17.1, was implemented for the extract part. Last runs were executed with v2.17.1
- Docker Engine v23.0.3 was utilized for containerization and deployment
- Vertica DB version 12.0.3 was used. The Vertica cluster hardware specifications cannot be published here due to confidentiality concerns, but it can be said that the live cluster has 4 nodes and the test cluster currently has 1 node, which has been upgraded to 3 by the time anyone reads this thesis
- vsql version 12 command line tool was employed for querying the Vertica DB
- DBeaver Community Edition version 23.0.3 was set up and used as the DB management tool

5 Results and Analysis

This section presents the results of the newly implemented incremental ELT pipeline for Jira data using DV 2.0 methodology and HP Vertica. The results are organized according to the key aspects of the proposed solution, highlighting the improvements over the existing AS-IS solution and the benefits gained by TEHIK.

5.1 Comparison with Existing Solution

The proposed solution was compared with the existing AS-IS solution regarding reporting capabilities, adherence to SLA conditions, and other aspects relevant to the case. The following subsections discuss the advantages of the new solution and its impact on TEHIK's operations.

5.1.1 Reduction in Manual Work

The new solution significantly reduces the manual work required to update reports by automating the data pipeline process. In the AS-IS solution, updating reports was time-consuming and resource-intensive due to the manual generation of extracts and the need to query the entire dataset each time. With the proposed solution, the incremental ELT pipeline ensures that only the necessary data is updated, reducing the time and resources required for report generation.

Automating the data pipeline also reduces the potential for human error, leading to more accurate and reliable reports. Staff can now focus on more critical tasks and decision-making, increasing productivity and efficiency within the organization.

5.1.2 Increased Flexibility

The new solution offers greater flexibility in handling various data requirements and adapting to changes in the data structure. The DV 2.0 methodology allows for adding, removing, or modifying data fields without affecting the existing warehouse structure. This flexibility enables TEHIK to adapt to evolving business requirements and ensure the DWH remains relevant and valuable.

Additionally, the proposed solution overcomes the limitations of the Tableau Connector Pro for Jira, which restricted specific JQL queries and prevented the creation of some reports. By leveraging custom scripts and Vertica SQL, the new solution can accommodate a broader range of query requirements and generate comprehensive reports that better serve the organization's needs.

5.1.3 Query Tuning and Performance Optimization

The proposed solution improves query performance by utilizing Vertica's capabilities in optimizing nodes, projections, and join strategies. Vertica's columnar storage and advanced compression techniques allow faster query execution and better performance than traditional row-based storage systems. This performance leads to more efficient data processing and reduced latency in report generation.

Furthermore, the custom scripts and SQL transformations streamline the integrating and analyzing data from Jira, enabling faster insights and decision-making. The improved query performance contributes to a more efficient and agile DWH that supports TEHIK's growing data needs.

5.1.4 Up-to-date Reports

The new solution ensures that reports are always current and accurate by automating the data pipeline process and incrementally updating the necessary data. Hence, it eliminates the need for manual updates and reduces the risk of outdated or inaccurate information being used for decision-making.

By providing up-to-date reports, TEHIK can make more informed decisions and respond quickly to emerging trends and changes in the organization's environment. The increased accuracy and timeliness of the reports contribute to improved decision-making and a more agile organization.

5.1.5 Historical Data and Trend Analysis

The proposed solution maintains a history of Jira data, enabling better trend analysis and decision-making. While the history of Jira users belonging to entities is currently limited due to data access rights concerns, the rest of the historicization is available. It works perfectly, including the history of changes to the issues.

In the existing AS-IS solution, there was no history of Jira users belonging to entities, which led to inconsistencies in department-based reports viewed at different points in time. Despite the current limitations with the history of Jira users belonging to entities, the new solution allows TEHIK to analyze trends and changes over time in other aspects of the Jira data, providing valuable insights for decision-making and strategic planning. This capability enhances the organization's ability to understand and respond to evolving trends and challenges. Once the data access rights concerns are resolved, the solution will offer a more comprehensive historical analysis.

5.2 Containerization and Deployment

The use of Docker for containerization and deployment of the data pipeline offers several benefits to TEHIK, including improved consistency across environments, simplified deployment, and enhanced maintainability. Containers ensure that the VSQL client and other solution components run consistently across different platforms, reducing the risk of environment-related issues. Additionally, containers can be easily scaled up or down, enabling the pipeline to handle fluctuating workloads and resource requirements.

Isolation is another advantage of using containers, as running the pipeline in separate containers isolates it from other processes. This isolation improves security and simplifies troubleshooting, making it easier for TEHIK to maintain and manage the data pipeline.

5.3 Monitoring and Maintenance

Compared to the existing AS-IS solution, the proposed solution significantly simplifies the monitoring and maintenance of TEHIK's data pipeline. The reduction in the number of moving parts, such as eliminating many previously required extracts, has made the system more manageable and less prone to issues.

Custom scripts are employed in the proposed solution for monitoring system health and providing real-time insights into performance. This enables TEHIK to proactively identify and resolve potential issues, reducing downtime and ensuring consistent data quality.

The current implementation of the data pipeline uses cron jobs for orchestration, while Kubernetes manages the deployed containers efficiently. These technologies contribute to a more streamlined and automated process, which reduces the complexity of monitoring and maintaining the pipeline. As a result, the solution has become more reliable and efficient compared to the previous setup.

In the future, incorporating Airflow for more advanced and automated orchestration of the data pipelines will further improve the solution. Airflow's capabilities will allow for compelling data pipeline maintenance and monitoring, including dependency management, error handling, and job scheduling. This continuous improvement will ensure that the solution remains robust and efficient, meeting TEHIK's evolving needs and easily handling future challenges.

5.4 Scalability and Performance

The proposed solution leverages Vertica's scalability and fast analytics support capabilities, ensuring the DWH can handle growing volumes of data without compromising performance. Vertica's columnar storage, advanced compression techniques, and query optimization features enable the DWH to process large datasets and deliver timely insights efficiently.

The DV 2.0 methodology provides a flexible and scalable foundation adaptable to changes in data structures and requirements. Its methodology ensures that the DWH can evolve with TEHIK's needs and continue to deliver value over time. The modular design of the DV 2.0 model makes it easy to add new entities or expand the scope of existing reports, simplifying the process of addressing changing business needs and requirements.

The VD 2.0 methodology makes incorporating additional data sources or adapting to new reporting requirements a breeze. This adaptability future-proofs the solution and reduces the time and effort needed for system updates and modifications, ensuring that TEHIK's DWH remains agile and responsive to the organization's evolving needs.

5.5 Performance Analysis

In the performance assessment, our concentration dwells on four pivotal KPIs and metrics: the duration of data extraction, the response time for queries, the timeframe for data loading, and the period required for data transformation. The subsequent sections delineate the outcomes under each of these KPIs, delivering a comprehensive evaluation of the performance exhibited by the proposed solution.

5.5.1 Extract Time

The extract time refers to the duration required to retrieve data from the source system, in this case, TEHIK's Jira test environment. The extraction process involves retrieving data from several sources, **including issues.jsonl**, **issue_comments.jsonl**, **issue_transitions.jsonl**, **changelogs.jsonl**, and **fields.jsonl** files. Table 1 summarizes the extract time for all files combined, totaling at 8 hours.

Table 1. The current extract time for all files combined is 8 hours.

File	file size	output AA
issues.jsonl	4,5GB	contains 200 000 Jira issues
issue_comments.jsonl	0.35GB	contains 400 000 records
issue_transitions.jsonl	0.75GB	contains 1 000 000 records
changelogs.jsonl	1,1 GB	contains 5 000 000 records
fields.jsonl	0,006 GB	contains 1000 records

However, this is not considered a problem, as when the pipeline runs daily, the extract time is reduced to a maximum of 30 seconds. If necessary, extract time can be further reduced by running multiple instances of Meltano, which can divide the extract time interval accordingly. This optimization is outside the scope of this thesis. Fields entity is not touched upon in the benchmarks since its negligible size makes it irrelevant.

5.5.2 Data Load Time

Data load time refers to the time required to load the extracted data into the data warehouse. The load times for different staging tables in the proposed solution can be seen in Figure 16.

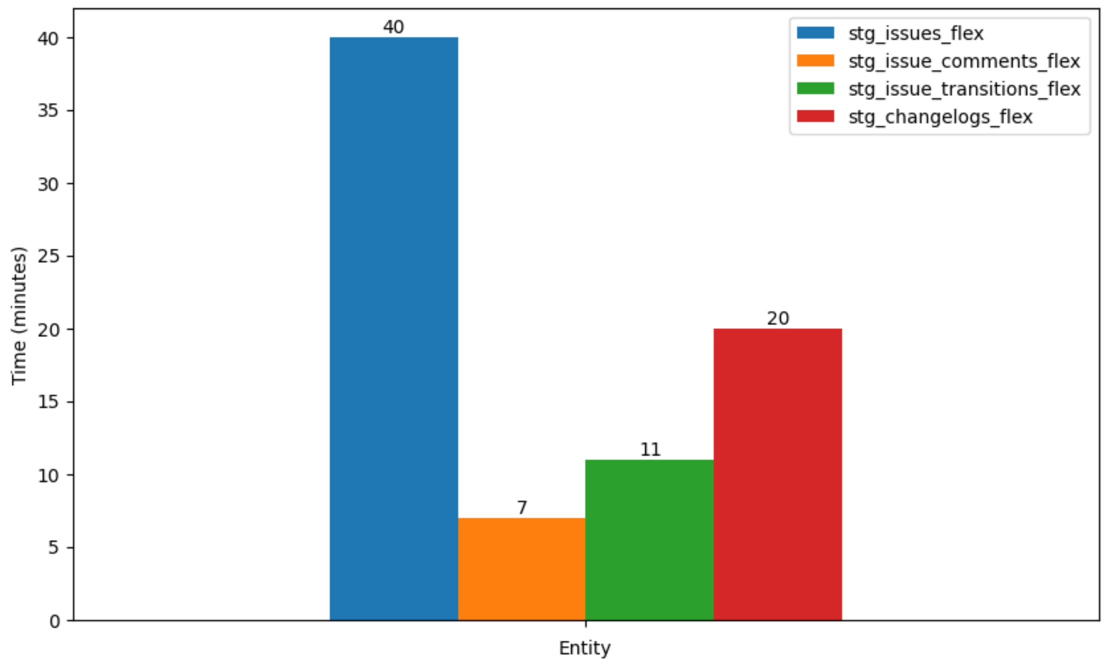


Figure 16. stg to ods load times

These load times indicate the efficiency of the proposed solution in ingesting data into the DWH.

5.5.3 Data Transformation Time

The data transformation time measures the duration required to perform data transformations within the DWH. This section is further divided into subsections for the ODS and EDW layers.

1. **Operational Data Store (ODS)** The transformation time for the ODS layer in the proposed solution can be seen in Figure 17.

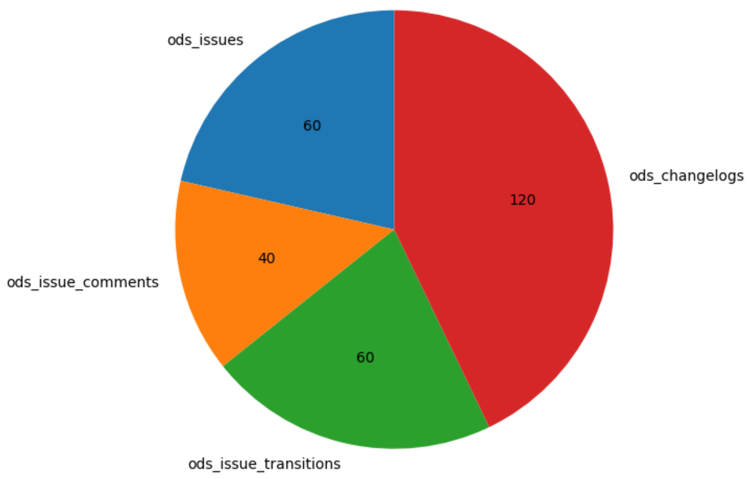


Figure 17. ods layer loading times

2. **Enterprise Data Warehouse (EDW)** The EDW layer consists of hubs, links, and satellites. The loading times of links and hubs are negligible (10...15 seconds) and are not considered. The loading times for the rest of EDW entities can be seen in Figure 18.

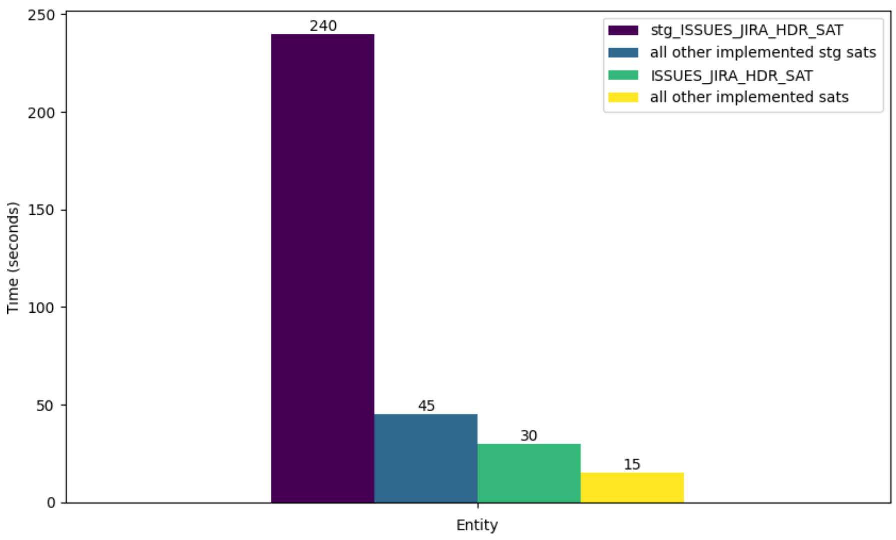


Figure 18. EDW layer loading times

5.5.4 Query Response Time

The query response time quantifies the duration required for executing a specific query against the DWH. This KPI directly influences the DWH's capacity to provide prompt insights to the organization, especially when the entire underlying pipeline is automated. The query times are negligible for the volume of records under consideration, remaining within single-digit seconds.

5.5.5 ELT on everyday basis

It is important to note that the 8 hours is the initial full extract from the Jira test environment. Once the pipeline is operational and regularly runs, the extract times become negligible, and all hubs, links, and satellites can be loaded in parallel. This analysis means that the slowest element in each layer determines the pipeline throughput. In the current scenario, this is likely to be **ISSUES_JIRA_SLA_NS_ARR_SAT**, which requires loading for nine attributes nested arrays. The author provides an estimated timeline for an entire potential run under a daily scenario, assuming all loads are executed in parallel and considering this satellite as the limiting factor. The times mentioned include the necessary container startups and shutdowns. The evaluated times for the entire ELT run on a daily basis can be seen in Figure 19.

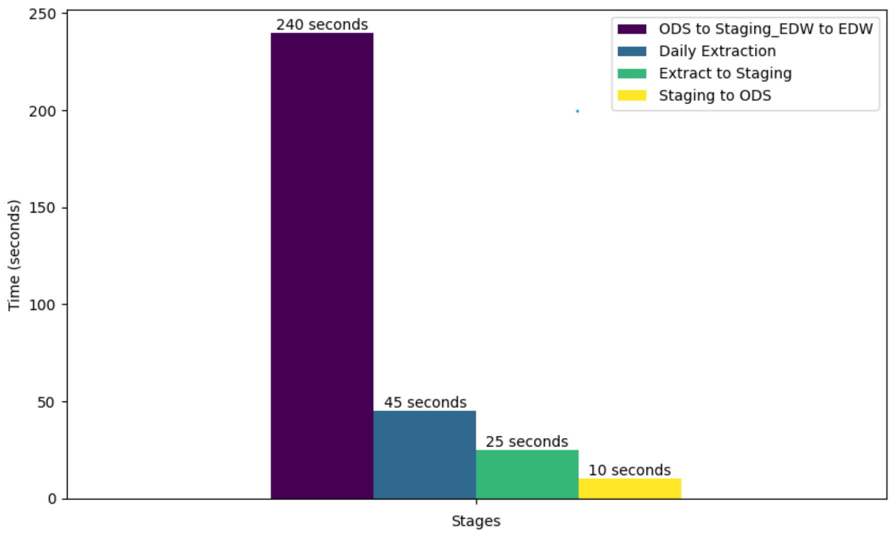


Figure 19. Evaluated full ELT run on a daily basis

Overall, the performance analysis of the proposed solution, based on extract time, query response time, data load time, and data transformation time, highlights substantial enhancements in operational efficiency and the ability to provide timely insights. Under everyday conditions, the pipeline is expected to run at most 5 to 6 minutes, ensuring the data is efficiently processed and readily available for analysis. The proposed solution is well-equipped to address TEHIK's DWH'ing needs and adapt to future requirements.

6 Discussion

6.1 Summary of Key Findings

Throughout this thesis, several key findings, challenges, and surprises emerged. This section reflects on these experiences, the learning process, and the importance of various aspects of the project.

1. **Learning Process and Overcoming Challenges** Embarking on this project required extensive learning in various areas, such as architecture, bash, SQL, and numerous tools. The initial phase was overwhelming, as it seemed like an insurmountable task with many complex concepts and tools to grasp. However, as the project progressed, the different pieces of the puzzle started to come together, and the various concepts began to make sense. This experience highlights the importance of perseverance and continuous learning in overcoming challenges and achieving project success.

2. **Generative SQL and the Value of Metadata** During the project, the discovery of generative SQL proved to be a valuable asset. Generative SQL allows for more dynamic and adaptable queries, which can significantly enhance the efficiency and flexibility of data processing. Furthermore, this project shed light on the importance of metadata and its role in enabling more automated and streamlined solutions. If the field entity had been discovered earlier and the author's competence in DWH'ing and SQL had been higher, a different, more automated approach could have been taken. This experience emphasizes the value of metadata and its potential to drive more efficient and effective solutions when leveraged correctly.

3. **Time-Consuming Testing and the Importance of Cooperation** Testing proved to be a time-consuming aspect of the project, as it involved not only the testing of the implemented solution but also the validation of data. This experience underscores the importance of thorough testing and the need for close cooperation with analysts from the very beginning of the project. Moreover, the project highlighted the importance of understanding the technical and business contexts. A solid grasp of the company's operations, requirements, and goals is essential in delivering a solution that meets their needs and provides valuable insights.

4. **Data Modeling and Making Sense of Complex Data** Throughout the project, the importance of data modeling became increasingly clear. In order to make sense of the complex and often messy data from Jira and to fulfill the reporting requirements, effective data modeling was crucial. This process involved understanding the relationships between different data elements, their relevance to the organization, and how they could be structured and transformed to deliver valuable insights.

5. **Challenges with Meltano Containerization** One of the more frustrating aspects of the project was the struggle with Meltano containerization. The solution sometimes worked seamlessly, while at other times, it failed without any apparent reason. This led to extensive debugging and attempts to identify and resolve the underlying issues. These challenges highlight the importance of persistence, problem-solving, and adaptability in facing obstacles.

6. **Implementation in the Live Environment** The project also revealed the need for considering the live environment when designing and implementing the solution. While the proposed solution demonstrated promising results in the test environment, its performance and adaptability in the live environment must also be assessed. This assessment involves considering factors such as data volume, complexity, and organizational requirements.

6.2 Future Work

There is still much to be accomplished in the context of the project. This section outlines several areas of future work that could further enhance the solution's capabilities and overall effectiveness.

1. **Completing Modelling and Implementation of Missing Satellites** The current data model still needs to incorporate all the necessary satellites. These missing satellites should be modeled and implemented in the future to ensure a more comprehensive data representation. This will allow for a more complete and accurate analysis of the Jira data, leading to better insights and decision-making.

2. **Developing a Logging Model** A logging model can be developed to improve the automation and monitoring of the data pipeline. One possible approach is to store logs in the Vertica DB and use them to drive the pipeline execution. This process would enable the pipeline to be more adaptive and responsive to changes in the data, further enhancing its automation capabilities. A well-designed logging model makes troubleshooting issues easier, monitoring performance, and ensuring consistent data quality.

3. **Implementing Airflow Orchestration** Airflow orchestration has yet to be implemented, and its integration would provide more advanced and automated control over the data pipeline. Airflow offers powerful features like dependency management, error handling, and job scheduling, which can streamline the maintenance and monitoring of the data pipeline. Implementing Airflow would contribute to a more efficient and reliable solution, better equipped to handle the organization's growing data needs.

4. **Incorporating Additional Hubs** Additional hubs could be incorporated into the data model to provide more granular insights into the Jira data. For instance, hubs could be created for epics, tasks, and subtasks, which are hierarchical structures in Jira that represent different levels of work breakdown. Incorporating these hubs may require parsing the issue, epic, and link trees to model their relationships accurately. This hub would enable more detailed reporting and analysis, allowing the organization to make more informed decisions.

5. **Deployment in Live Environment and Modelling of Missing Entities** In the future, the solution should be deployed in a live environment. The missing entities and their attributes should be modeled to provide a more comprehensive data view. This deployment will ensure that the DWH is well-equipped to handle the organization's reporting and analysis needs and adapt to any future changes in data structures or requirements.

6. **Building Predictive Models** Once the data warehouse is complete and fully operational, it may be possible to build predictive models using the data. These models could forecast trends, identify potential issues, and inform strategic planning. For example, a predictive model could be developed to estimate the likelihood of an issue being resolved within a given timeframe based on factors such as issue complexity, team workload, and historical performance. This would allow the organization to allocate resources better and prioritize work.

7. **Implementing AD Integration** AD integration is still a planned feature, and its implementation would allow for more accurate tracking of user memberships and department affiliations. This would improve the quality of department-based reports, making them more consistent and reliable. AD integration would also enable the solution to maintain a history of user affiliations, allowing for more accurate trend analysis and decision-making.In conclusion, while the current solution provides a strong foundation for Jira DWH'ing, there is still much room for improvement and expansion. The solution can be further refined and enhanced by addressing the areas outlined in this section, ensuring its long-term value to the organization and its ability to adapt to future challenges.

7 Conclusion

This thesis unveils a comprehensive DWH solution for TEHIK, focusing on assimilating and scrutinizing Jira data. It employs the flexible DV 2.0 methodology and HP Vertica, which optimizes extensive dataset processing with its columnar storage and advanced compression. Meltano enhances the solution for data extraction, VSQL for DB interaction, Docker for environment isolation, and Bash scripts for automation.

This thesis addressed multiple challenges, including creating the architecture, mastering DV 2.0, bash, python, and SQL, and gaining proficiency in various tools and technologies. The uncovering of generative SQL and the significance of metadata were pivotal moments in the solution's development. Moreover, the emphasis on comprehensive testing, close analyst collaboration, and understanding the business and company underscored the importance of technical expertise and business acumen.

The performance analysis of the proposed solution demonstrated significant improvements in operational efficiency and the ability to deliver timely insights to the organization. The measurements conducted on a Vertica Test cluster using data from TEHIK's Jira test environment indicated that the pipeline would run at most 5 to 6 minutes under everyday conditions, ensuring efficient processing and data availability for analysis.

The Discussion section reflected on key findings, surprises, challenges, and best practices. The journey of developing the DWH solution was a valuable learning experience, uncovering the importance of data modeling, the need for close collaboration with analysts, and the potential of generative SQL for automation. Additionally, containerizing Meltano and deploying the solution in a test environment presented unique challenges and opportunities for growth.

There are numerous opportunities for future work and enhancements to the proposed solution. These include finishing the modeling and implementing missing satellites, developing a logging model for better pipeline automation, implementing Airflow orchestration, and expanding the data model to include more hubs and relationships. Further down the line, it may be possible to build predictive models using the data stored in the warehouse, adding another layer of value to the organization.

In summary, this thesis has successfully developed a DWH solution that addresses TEHIK's needs while providing a solid foundation for future growth and evolution. The lessons learned and challenges overcome along the way have contributed to a richer understanding of the fields of DE, DWH'ing, and the importance of collaboration, testing, and continual learning. With the proposed solution in place, TEHIK is well-positioned to harness the power of its data and make more informed decisions, driving the organization toward a successful future.

References

[Aira] https://airbyte.com. `https://airbyte.com`. Accessed: 2023-04-30.

[airb] https://airflow.apache.org/docs/. Accessed: 2023-05-05.

[ama] https://aws.amazon.com/s3/. Accessed: 2023-05-05.

[apaa] https://flink.apache.org/. Accessed: 2023-05-05.

[apab] https://spark.apache.org/. Accessed: 2023-05-05.

[arr] https://arrow.apache.org/. Accessed: 2023-05-05.

[Bar] Kevin Bartley. Etl vs elt: What's the difference?

[cas] https://cassandra.apache.org/_/index.html. Accessed: 2023-05-05.

[cro] https://www.hivelocity.net/kb/what-is-cron-job/. Accessed: 2023-05-05.

[cur] https://curl.se/. Accessed: 2023-05-05.

[Dat] https://datavaultalliance.com/news/dv/dv-standards/data-vault-2-0-suggested-object-naming-conventions/. `https://datavaultalliance.com/news/dv/dv-standards/data-vault-2-0-suggested-object-naming-conventions/`. Accessed: 2023-04-24.

[dbe] https://dbeaver.io/. Accessed: 2023-05-05.

[diga] https://www.digilugu.ee/login. Accessed: 2023-05-05.

[digb] https://digiregistratuur.ee/login. Accessed: 2023-05-05.

[digc] https://www.tervisekassa.ee/inimesele/ravimid/digiretsept. Accessed: 2023-05-05.

[doc] https://docs.docker.com/. Accessed: 2023-05-05.

[ela] https://www.elastic.co/. Accessed: 2023-05-05.

[flu] https://www.fluentd.org/. Accessed: 2023-05-05.

[git] https://git-scm.com/. Accessed: 2023-05-05.

[gra] https://grafana.com/. Accessed: 2023-05-05.

[had] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Accessed: 2023-05-05.

[Inm05] W. H. Inmon. *Building the Data Warehouse*. Wiley Publishing, 4th edition, 2005.

[jen] https://www.jenkins.io/. Accessed: 2023-05-05.

[jira] https://confluence.atlassian.com/jira. Accessed: 2023-05-05.

[jirb] https://www.atlassian.com/software/jira. Accessed: 2023-05-05.

[jso] https://www.atatus.com/glossary/jsonl/. Accessed: 2023-05-05.

[kaf] https://kafka.apache.org/. Accessed: 2023-05-05.

[kib] https://www.elastic.co/kibana/. Accessed: 2023-05-05.

[kor] https://www.terviseamet.ee/et/koroonaviirus/koroonaviiruse-andmestik. Accessed: 2023-05-05.

[KR13] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley, 3rd edition, 2013.

[kub] https://kubernetes.io/. Accessed: 2023-05-05.

[LO15] Dan Linstedt and Michael Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0*. Morgan Kaufmann, 2015.

[log] https://www.elastic.co/logstash/. Accessed: 2023-05-05.

[lxm] https://lxml.de/. Accessed: 2023-05-05.

[med] https://medre.tehik.ee/home. Accessed: 2023-05-05.

[Mel] https://docs.meltano.com/. `https://docs.meltano.com/`. Accessed: 2023-04-30.

[mys] https://www.mysql.com/. Accessed: 2023-05-05.

[nif] https://nifi.apache.org/. Accessed: 2023-05-05.

[pan] https://pandas.pydata.org/docs/user_guide/index.html. Accessed: 2023-05-05.

[par] https://parquet.apache.org/. Accessed: 2023-05-05.

[pos] https://www.postgresql.org/. Accessed: 2023-05-05.

[pro] https://prometheus.io/. Accessed: 2023-05-05.

[pyp] https://pypi.org/project/pip/. Accessed: 2023-05-05.

[rab] https://www.rabbitmq.com/. Accessed: 2023-05-05.

[rav] https://www.ravimiamet.ee/. Accessed: 2023-05-05.

[RH22] J. Reis and M. Housley. *Fundamentals of Data Engineering*. O'Reilly Media, 2022.

[riia] https://riigipilv.ee/et. Accessed: 2023-05-05.

[riib] https://pilv.ee/pilvest/pilve-tehniline-lahendus/pilvevoimeka-rakenduse-arendus. `https://pilv.ee/pilvest/riigipilve-tehniline-lahendus/pilvevoimeka-rakenduse-arendus`. Accessed: 2023-04-30.

[sam] https://www.tehik.ee/ravimitega-seotud-menetlusprotsesside-infosusteem-samtrack. Accessed: 2023-05-05.

[Sin] https://www.singer.io/. `https://www.singer.io/`. Accessed: 2023-04-30.

[sm] https://www.sm.ee/. Accessed: 2023-05-05.

[spo] https://github.com/spotify/luigi. Accessed: 2023-05-05.

[taba] https://www.tableau.com/. Accessed: 2023-05-05.

[tabb] https://appfire.com/solutions/tableau-connector-pro-for-jira/. Accessed: 2023-05-05.

[tara] https://hub.meltano.com/loaders/target-jsonl/. Accessed: 2023-05-05.

[tarb] https://github.com/full360/pipelinewise-target-vertica. `https://github.com/full360/pipelinewise-target-vertica`. Accessed: 2023-04-30.

[teh] https://tehik.ee/. Accessed: 2023-05-05.

[ter] https://www.terviseamet.ee/et. Accessed: 2023-05-05.

[too] https://www.tooelu.ee/et/73/tooelu-infosusteem-teis. Accessed: 2023-05-05.

[vera] https://docs.vertica.com/12.0.x. Accessed: 2023-05-05.

[verb] https://www.vertica.com/. Accessed: 2023-05-05.

[Whe] https://www.wherescape.com/solutions/project-types/data-vault-automation/. (accessed: 22.04.2023).

[YL16] Lamia Yessad and Aissa Labiod. Comparative study of data warehouses modeling approaches: Inmon, kimball and data vault. In *Proceedings of the Conference on [Your Conference Name Here]*, pages 95–99, 11 2016.

Appendix

I Tree structure and naming conventions example based on EDW

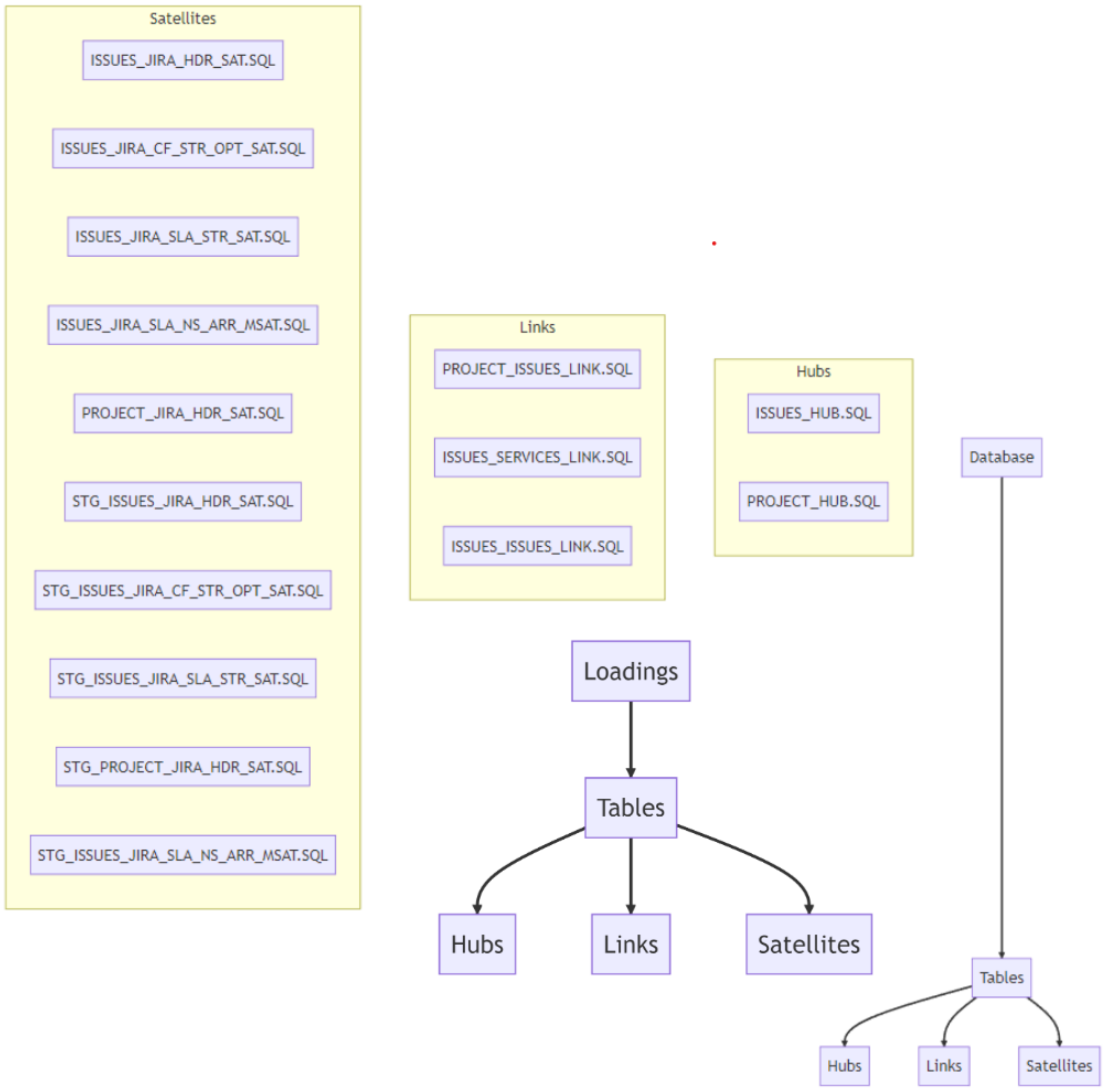
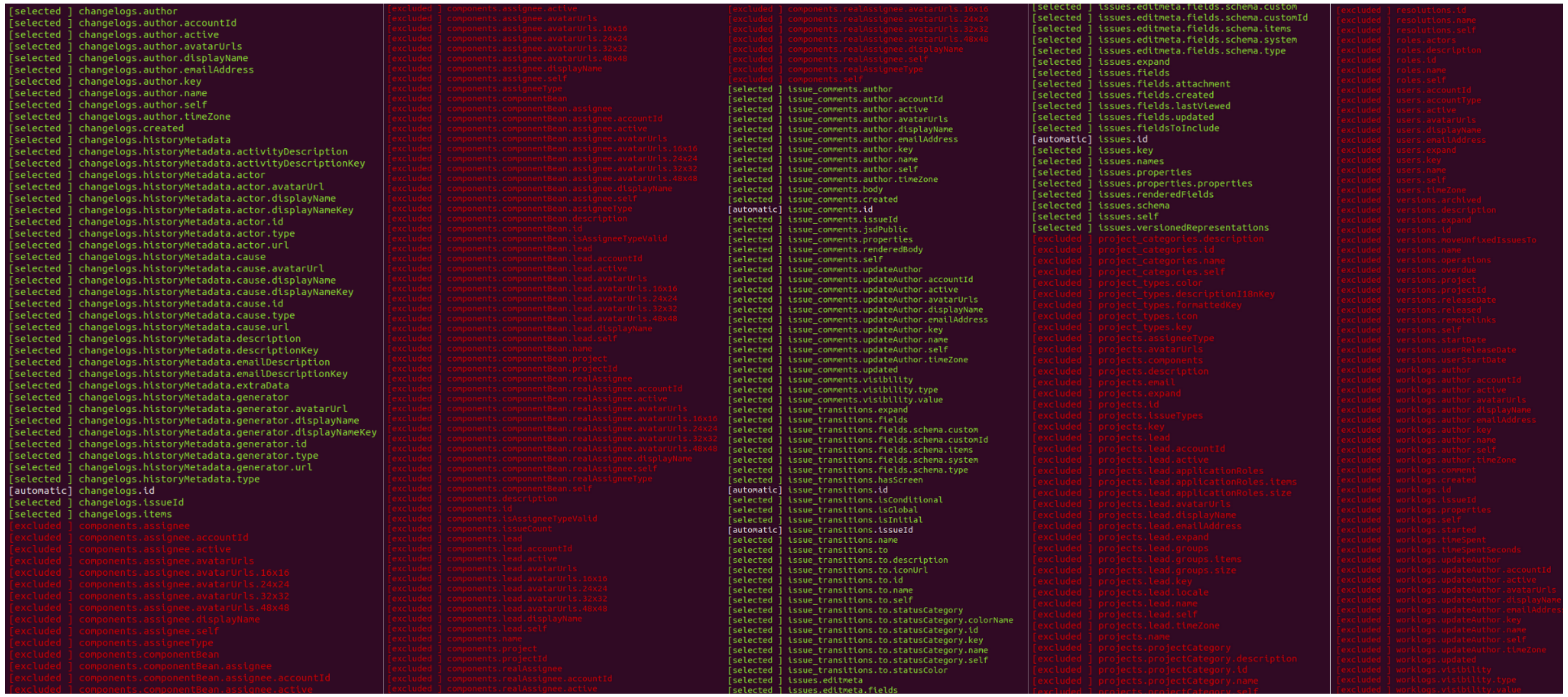

Satellites
ISSUES_JIRA_HDR_SAT.SQL
ISSUES_JIRA_CF_STR_OPT_SAT.SQL
ISSUES_JIRA_SLA_STR_SAT.SQL
ISSUES_JIRA_SLA_NS_ARR_MSAT.SQL
PROJECT_JIRA_HDR_SAT.SQL
STG_ISSUES_JIRA_HDR_SAT.SQL
STG_ISSUES_JIRA_CF_STR_OPT_SAT.SQL
STG_ISSUES_JIRA_SLA_STR_SAT.SQL
STG_PROJECT_JIRA_HDR_SAT.SQL
STG_ISSUES_JIRA_SLA_NS_ARR_MSAT.SQL
Links
PROJECT_ISSUES_LINK.SQL
ISSUES_SERVICES_LINK.SQL
ISSUES_ISSUES_LINK.SQL
Hubs
ISSUES_HUB.SQL
PROJECT_HUB.SQL
Database
Loadings
Tables
Hubs
Links
Satellites
Tables
Hubs
Links
Satellites

Figure 20. Tree structure and naming conventions example based on EDW.

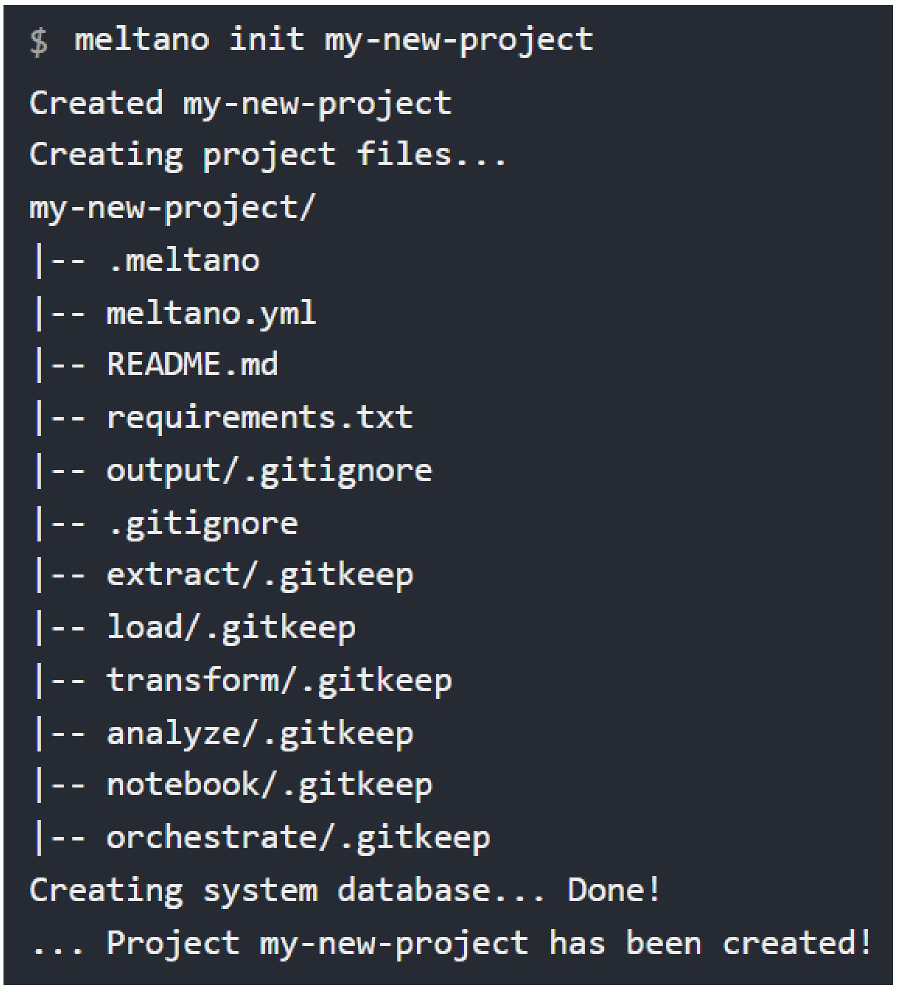
II Meltano tap-jira possible endpoints



```
[selected ] changelogs.author
[selected ] changelogs.author.accountId
[selected ] changelogs.author.active
[selected ] changelogs.author.avatarUrls
[selected ] changelogs.author.displayName
[selected ] changelogs.author.emailAddress
[selected ] changelogs.author.key
[selected ] changelogs.author.name
[selected ] changelogs.author.self
[selected ] changelogs.author.timeZone
[selected ] changelogs.created
[selected ] changelogs.historyMetadata
[selected ] changelogs.historyMetadata.activityDescription
[selected ] changelogs.historyMetadata.activityDescriptionKey
[selected ] changelogs.historyMetadata.actor
[selected ] changelogs.historyMetadata.actor.avatarUrl
[selected ] changelogs.historyMetadata.actor.displayName
[selected ] changelogs.historyMetadata.actor.displayNameKey
[selected ] changelogs.historyMetadata.actor.id
[selected ] changelogs.historyMetadata.actor.type
[selected ] changelogs.historyMetadata.actor.url
[selected ] changelogs.historyMetadata.cause
[selected ] changelogs.historyMetadata.cause.avatarUrl
[selected ] changelogs.historyMetadata.cause.displayName
[selected ] changelogs.historyMetadata.cause.displayNameKey
[selected ] changelogs.historyMetadata.cause.id
[selected ] changelogs.historyMetadata.cause.type
[selected ] changelogs.historyMetadata.cause.url
[selected ] changelogs.historyMetadata.description
[selected ] changelogs.historyMetadata.descriptionKey
[selected ] changelogs.historyMetadata.emailDescription
[selected ] changelogs.historyMetadata.emailDescriptionKey
[selected ] changelogs.historyMetadata.extraData
[selected ] changelogs.historyMetadata.generator
[selected ] changelogs.historyMetadata.generator.avatarUrl
[selected ] changelogs.historyMetadata.generator.displayName
[selected ] changelogs.historyMetadata.generator.displayNameKey
[selected ] changelogs.historyMetadata.generator.id
[selected ] changelogs.historyMetadata.generator.type
[selected ] changelogs.historyMetadata.generator.url
[selected ] changelogs.historyMetadata.type
[automatic] changelogs.id
[selected ] changelogs.issueId
[selected ] changelogs.items
[excluded ] components.assignee
[excluded ] components.assignee.accountId
[excluded ] components.assignee.active
[excluded ] components.assignee.avatarUrls
[excluded ] components.assignee.avatarUrls.16x16
[excluded ] components.assignee.avatarUrls.24x24
[excluded ] components.assignee.avatarUrls.32x32
[excluded ] components.assignee.avatarUrls.48x48
[excluded ] components.assignee.displayName
[excluded ] components.assignee.self
[excluded ] components.assigneeType
[excluded ] components.componentBean
[excluded ] components.componentBean.assignee
[excluded ] components.componentBean.assignee.accountId
[excluded ] components.componentBean.assignee.active

[excluded ] components.assignee.active
[excluded ] components.assignee.avatarUrls
[excluded ] components.assignee.avatarUrls.16x16
[excluded ] components.assignee.avatarUrls.24x24
[excluded ] components.assignee.avatarUrls.32x32
[excluded ] components.assignee.avatarUrls.48x48
[excluded ] components.assignee.displayName
[excluded ] components.assignee.self
[excluded ] components.assigneeType
[excluded ] components.componentBean
[excluded ] components.componentBean.assignee
[excluded ] components.componentBean.assignee.accountId
[excluded ] components.componentBean.assignee.active
[excluded ] components.componentBean.assignee.avatarUrls
[excluded ] components.componentBean.assignee.avatarUrls.16x16
[excluded ] components.componentBean.assignee.avatarUrls.24x24
[excluded ] components.componentBean.assignee.avatarUrls.32x32
[excluded ] components.componentBean.assignee.avatarUrls.48x48
[excluded ] components.componentBean.assignee.displayName
[excluded ] components.componentBean.assignee.self
[excluded ] components.componentBean.assigneeType
[excluded ] components.componentBean.description
[excluded ] components.componentBean.id
[excluded ] components.componentBean.isAssigneeTypeValid
[excluded ] components.componentBean.lead
[excluded ] components.componentBean.lead.accountId
[excluded ] components.componentBean.lead.active
[excluded ] components.componentBean.lead.avatarUrls
[excluded ] components.componentBean.lead.avatarUrls.16x16
[excluded ] components.componentBean.lead.avatarUrls.24x24
[excluded ] components.componentBean.lead.avatarUrls.32x32
[excluded ] components.componentBean.lead.avatarUrls.48x48
[excluded ] components.componentBean.lead.displayName
[excluded ] components.componentBean.lead.self
[excluded ] components.componentBean.name
[excluded ] components.componentBean.project
[excluded ] components.componentBean.projectId
[excluded ] components.componentBean.realAssignee
[excluded ] components.componentBean.realAssignee.accountId
[excluded ] components.componentBean.realAssignee.active
[excluded ] components.componentBean.realAssignee.avatarUrls
[excluded ] components.componentBean.realAssignee.avatarUrls.16x16
[excluded ] components.componentBean.realAssignee.avatarUrls.24x24
[excluded ] components.componentBean.realAssignee.avatarUrls.32x32
[excluded ] components.componentBean.realAssignee.avatarUrls.48x48
[excluded ] components.componentBean.realAssignee.displayName
[excluded ] components.componentBean.realAssignee.self
[excluded ] components.componentBean.realAssigneeType
[excluded ] components.componentBean.self
[excluded ] components.description
[excluded ] components.id
[excluded ] components.isAssigneeTypeValid
[excluded ] components.issueCount
[excluded ] components.lead
[excluded ] components.lead.accountId
[excluded ] components.lead.active
[excluded ] components.lead.avatarUrls
[excluded ] components.lead.avatarUrls.16x16
[excluded ] components.lead.avatarUrls.24x24
[excluded ] components.lead.avatarUrls.32x32
[excluded ] components.lead.avatarUrls.48x48
[excluded ] components.lead.displayName
[excluded ] components.lead.self
[excluded ] components.name
[excluded ] components.project
[excluded ] components.projectId
[excluded ] components.realAssignee
[excluded ] components.realAssignee.accountId
[excluded ] components.realAssignee.active

[excluded ] components.realAssignee.avatarUrls.16x16
[excluded ] components.realAssignee.avatarUrls.24x24
[excluded ] components.realAssignee.avatarUrls.32x32
[excluded ] components.realAssignee.avatarUrls.48x48
[excluded ] components.realAssignee.displayName
[excluded ] components.realAssignee.self
[excluded ] components.realAssigneeType
[excluded ] components.self
[selected ] issue_comments.author
[selected ] issue_comments.author.accountId
[selected ] issue_comments.author.active
[selected ] issue_comments.author.avatarUrls
[selected ] issue_comments.author.displayName
[selected ] issue_comments.author.emailAddress
[selected ] issue_comments.author.key
[selected ] issue_comments.author.name
[selected ] issue_comments.author.self
[selected ] issue_comments.author.timeZone
[selected ] issue_comments.body
[selected ] issue_comments.created
[automatic] issue_comments.id
[selected ] issue_comments.issueId
[selected ] issue_comments.jsdPublic
[selected ] issue_comments.properties
[selected ] issue_comments.renderedBody
[selected ] issue_comments.self
[selected ] issue_comments.updateAuthor
[selected ] issue_comments.updateAuthor.accountId
[selected ] issue_comments.updateAuthor.active
[selected ] issue_comments.updateAuthor.avatarUrls
[selected ] issue_comments.updateAuthor.displayName
[selected ] issue_comments.updateAuthor.emailAddress
[selected ] issue_comments.updateAuthor.key
[selected ] issue_comments.updateAuthor.name
[selected ] issue_comments.updateAuthor.self
[selected ] issue_comments.updateAuthor.timeZone
[selected ] issue_comments.updated
[selected ] issue_comments.visibility
[selected ] issue_comments.visibility.type
[selected ] issue_comments.visibility.value
[selected ] issue_transitions.expand
[selected ] issue_transitions.fields
[selected ] issue_transitions.fields.schema.custom
[selected ] issue_transitions.fields.schema.customId
[selected ] issue_transitions.fields.schema.items
[selected ] issue_transitions.fields.schema.system
[selected ] issue_transitions.fields.schema.type
[selected ] issue_transitions.hasScreen
[automatic] issue_transitions.id
[selected ] issue_transitions.isConditional
[selected ] issue_transitions.isGlobal
[selected ] issue_transitions.isInitial
[automatic] issue_transitions.issueId
[selected ] issue_transitions.name
[selected ] issue_transitions.to
[selected ] issue_transitions.to.description
[selected ] issue_transitions.to.iconUrl
[selected ] issue_transitions.to.id
[selected ] issue_transitions.to.name
[selected ] issue_transitions.to.self
[selected ] issue_transitions.to.statusCategory
[selected ] issue_transitions.to.statusCategory.colorName
[selected ] issue_transitions.to.statusCategory.id
[selected ] issue_transitions.to.statusCategory.key
[selected ] issue_transitions.to.statusCategory.name
[selected ] issue_transitions.to.statusCategory.self
[selected ] issue_transitions.to.statusColor
[selected ] issues.editmeta
[selected ] issues.editmeta.fields

[selected ] issues.editmeta.fields.schema.custom
[selected ] issues.editmeta.fields.schema.customId
[selected ] issues.editmeta.fields.schema.items
[selected ] issues.editmeta.fields.schema.system
[selected ] issues.editmeta.fields.schema.type
[selected ] issues.expand
[selected ] issues.fields
[selected ] issues.fields.attachment
[selected ] issues.fields.created
[selected ] issues.fields.lastViewed
[selected ] issues.fields.updated
[selected ] issues.fieldsToInclude
[automatic] issues.id
[selected ] issues.key
[selected ] issues.names
[selected ] issues.properties
[selected ] issues.properties.properties
[selected ] issues.renderedFields
[selected ] issues.schema
[selected ] issues.self
[selected ] issues.versionedRepresentations
[excluded ] project_categories.description
[excluded ] project_categories.id
[excluded ] project_categories.name
[excluded ] project_categories.self
[excluded ] project_types.color
[excluded ] project_types.descriptionI18nKey
[excluded ] project_types.formattedKey
[excluded ] project_types.icon
[excluded ] project_types.key
[excluded ] projects.assigneeType
[excluded ] projects.avatarUrls
[excluded ] projects.components
[excluded ] projects.description
[excluded ] projects.email
[excluded ] projects.expand
[excluded ] projects.id
[excluded ] projects.issueTypes
[excluded ] projects.key
[excluded ] projects.lead
[excluded ] projects.lead.accountId
[excluded ] projects.lead.active
[excluded ] projects.lead.applicationRoles
[excluded ] projects.lead.applicationRoles.items
[excluded ] projects.lead.applicationRoles.size
[excluded ] projects.lead.avatarUrls
[excluded ] projects.lead.displayName
[excluded ] projects.lead.emailAddress
[excluded ] projects.lead.expand
[excluded ] projects.lead.groups
[excluded ] projects.lead.groups.items
[excluded ] projects.lead.groups.size
[excluded ] projects.lead.key
[excluded ] projects.lead.locale
[excluded ] projects.lead.name
[excluded ] projects.lead.self
[excluded ] projects.lead.timeZone
[excluded ] projects.name
[excluded ] projects.projectCategory
[excluded ] projects.projectCategory.description
[excluded ] projects.projectCategory.id
[excluded ] projects.projectCategory.name
[excluded ] projects.projectCategory.self

[excluded ] resolutions.id
[excluded ] resolutions.name
[excluded ] resolutions.self
[excluded ] roles.actors
[excluded ] roles.description
[excluded ] roles.id
[excluded ] roles.name
[excluded ] roles.self
[excluded ] users.accountId
[excluded ] users.accountType
[excluded ] users.active
[excluded ] users.avatarUrls
[excluded ] users.displayName
[excluded ] users.emailAddress
[excluded ] users.expand
[excluded ] users.key
[excluded ] users.name
[excluded ] users.self
[excluded ] users.timeZone
[excluded ] versions.archived
[excluded ] versions.description
[excluded ] versions.expand
[excluded ] versions.id
[excluded ] versions.moveUnfixedIssuesTo
[excluded ] versions.name
[excluded ] versions.operations
[excluded ] versions.overdue
[excluded ] versions.project
[excluded ] versions.projectId
[excluded ] versions.releaseDate
[excluded ] versions.released
[excluded ] versions.remotelinks
[excluded ] versions.self
[excluded ] versions.startDate
[excluded ] versions.userReleaseDate
[excluded ] versions.userStartDate
[excluded ] worklogs.author
[excluded ] worklogs.author.accountId
[excluded ] worklogs.author.active
[excluded ] worklogs.author.avatarUrls
[excluded ] worklogs.author.displayName
[excluded ] worklogs.author.emailAddress
[excluded ] worklogs.author.key
[excluded ] worklogs.author.name
[excluded ] worklogs.author.self
[excluded ] worklogs.author.timeZone
[excluded ] worklogs.comment
[excluded ] worklogs.created
[excluded ] worklogs.id
[excluded ] worklogs.issueId
[excluded ] worklogs.properties
[excluded ] worklogs.self
[excluded ] worklogs.started
[excluded ] worklogs.timeSpent
[excluded ] worklogs.timeSpentSeconds
[excluded ] worklogs.updateAuthor
[excluded ] worklogs.updateAuthor.accountId
[excluded ] worklogs.updateAuthor.active
[excluded ] worklogs.updateAuthor.avatarUrls
[excluded ] worklogs.updateAuthor.displayName
[excluded ] worklogs.updateAuthor.emailAddress
[excluded ] worklogs.updateAuthor.key
[excluded ] worklogs.updateAuthor.name
[excluded ] worklogs.updateAuthor.self
[excluded ] worklogs.updateAuthor.timeZone
[excluded ] worklogs.updated
[excluded ] worklogs.visibility
[excluded ] worklogs.visibility.type
[excluded ] worklogs.visibility.value
```

Figure 21. Meltano tap-jira possible endpoints.

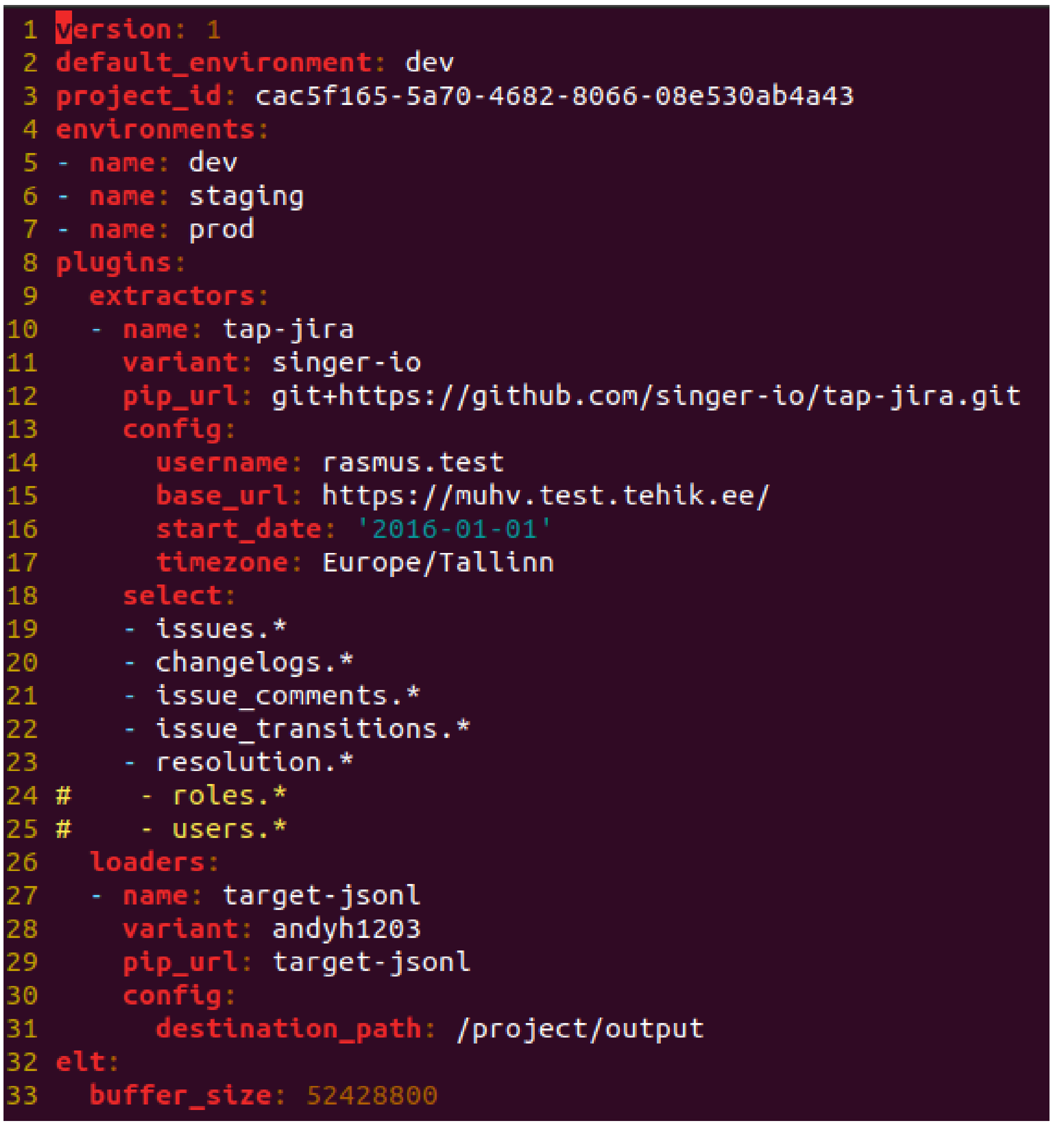
III Meltano init my-new-project



```
$ meltano init my-new-project
Created my-new-project
Creating project files...
my-new-project/
|-- .meltano
|-- meltano.yml
|-- README.md
|-- requirements.txt
|-- output/.gitignore
|-- .gitignore
|-- extract/.gitkeep
|-- load/.gitkeep
|-- transform/.gitkeep
|-- analyze/.gitkeep
|-- notebook/.gitkeep
|-- orchestrate/.gitkeep
Creating system database... Done!
... Project my-new-project has been created!
```

Figure 22. File structure created by blank meltano project.

IV Meltano init promt



```yaml
version: 1
default_environment: dev
project_id: cac5f165-5a70-4682-8066-08e530ab4a43
environments:
- name: dev
- name: staging
- name: prod
plugins:
  extractors:
  - name: tap-jira
    variant: singer-io
    pip_url: git+https://github.com/singer-io/tap-jira.git
    config:
      username: rasmus.test
      base_url: https://muhv.test.tehik.ee/
      start_date: '2016-01-01'
      timezone: Europe/Tallinn
    select:
    - issues.*
    - changelogs.*
    - issue_comments.*
    - issue_transitions.*
    - resolution.*
#     - roles.*
#     - users.*
  loaders:
  - name: target-jsonl
    variant: andyh1203
    pip_url: target-jsonl
    config:
      destination_path: /project/output
elt:
  buffer_size: 52428800
```

Figure 23. Tree structure and naming conventions example based on EDW.

V Jira issue tree

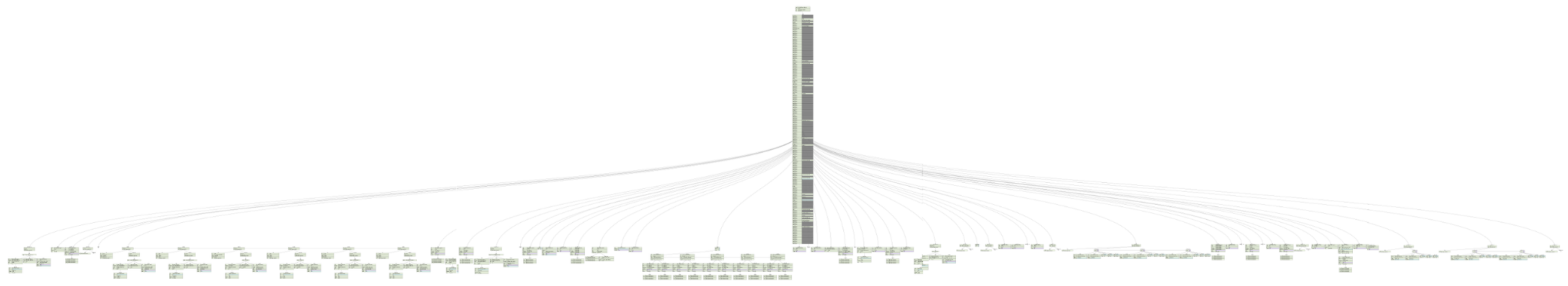
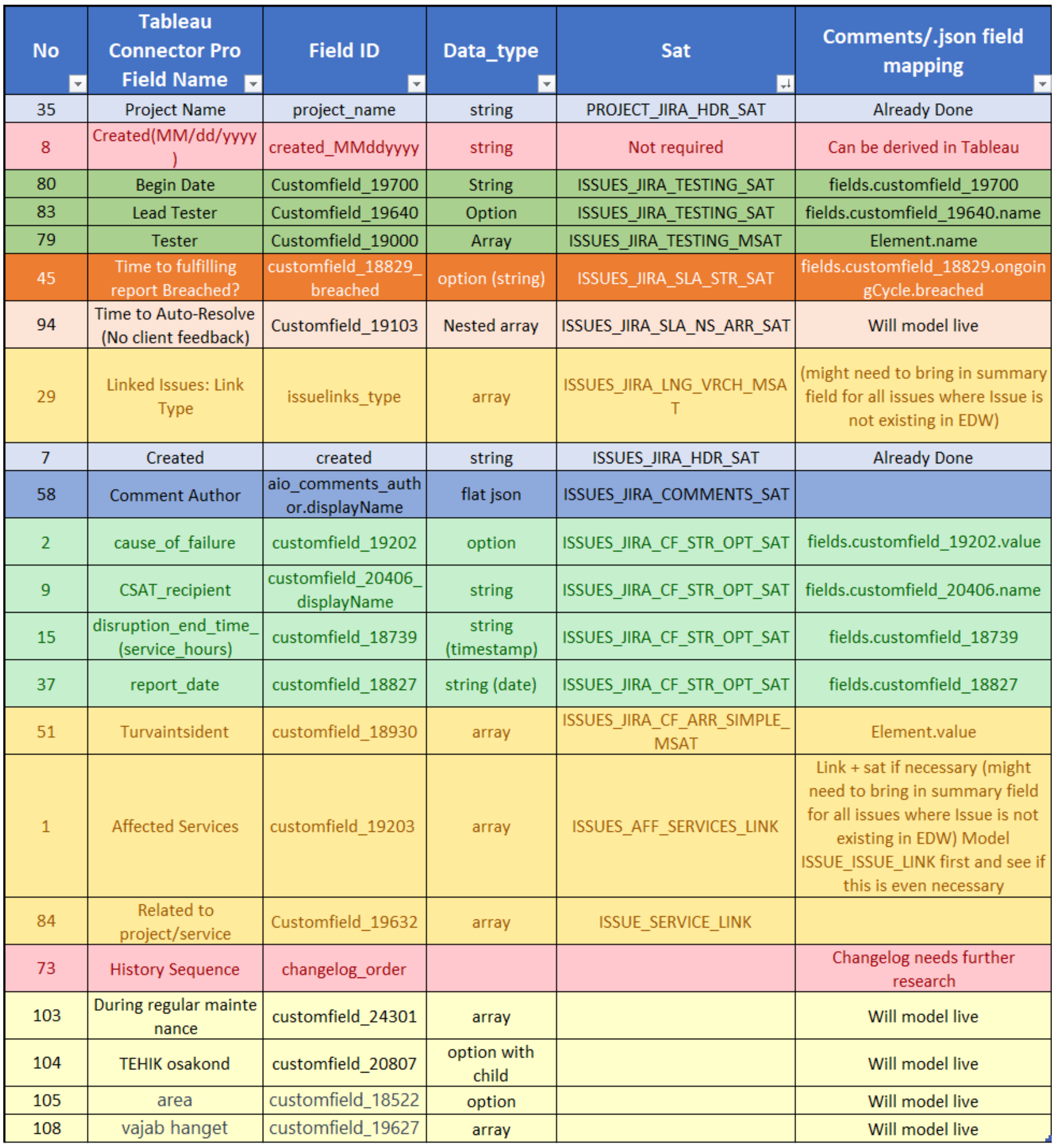


Figure 24. Jira issue tree.

VI Requirements mapping



No	Tableau Connector Pro Field Name	Field ID	Data_type	Sat	Comments/.json field mapping
35	Project Name	project_name	string	PROJECT_JIRA_HDR_SAT	Already Done
8	Created(MM/dd/yyyy)	created_MMddyyyy	string	Not required	Can be derived in Tableau
80	Begin Date	Customfield_19700	String	ISSUES_JIRA_TESTING_SAT	fields.customfield_19700
83	Lead Tester	Customfield_19640	Option	ISSUES_JIRA_TESTING_SAT	fields.customfield_19640.name
79	Tester	Customfield_19000	Array	ISSUES_JIRA_TESTING_MSAT	Element.name
45	Time to fulfilling report Breached?	customfield_18829_breached	option (string)	ISSUES_JIRA_SLA_STR_SAT	fields.customfield_18829.ongoingCycle.breached
94	Time to Auto-Resolve (No client feedback)	Customfield_19103	Nested array	ISSUES_JIRA_SLA_NS_ARR_SAT	Will model live
29	Linked Issues: Link Type	issuelinks_type	array	ISSUES_JIRA_LNG_VRCH_MSAT	(might need to bring in summary field for all issues where Issue is not existing in EDW)
7	Created	created	string	ISSUES_JIRA_HDR_SAT	Already Done
58	Comment Author	aio_comments_author.displayName	flat json	ISSUES_JIRA_COMMENTS_SAT	
2	cause_of_failure	customfield_19202	option	ISSUES_JIRA_CF_STR_OPT_SAT	fields.customfield_19202.value
9	CSAT_recipient	customfield_20406_displayName	string	ISSUES_JIRA_CF_STR_OPT_SAT	fields.customfield_20406.name
15	disruption_end_time_(service_hours)	customfield_18739	string (timestamp)	ISSUES_JIRA_CF_STR_OPT_SAT	fields.customfield_18739
37	report_date	customfield_18827	string (date)	ISSUES_JIRA_CF_STR_OPT_SAT	fields.customfield_18827
51	Turvaintsident	customfield_18930	array	ISSUES_JIRA_CF_ARR_SIMPLE_MSAT	Element.value
1	Affected Services	customfield_19203	array	ISSUES_AFF_SERVICES_LINK	Link + sat if necessary (might need to bring in summary field for all issues where Issue is not existing in EDW) Model ISSUE_ISSUE_LINK first and see if this is even necessary
84	Related to project/service	Customfield_19632	array	ISSUE_SERVICE_LINK	
73	History Sequence	changelog_order			Changelog needs further research
103	During regular maintenance	customfield_24301	array		Will model live
104	TEHIK osakond	customfield_20807	option with child		Will model live
105	area	customfield_18522	option		Will model live
108	vajab hanget	customfield_19627	array		Will model live

Figure 25. Requirements mapping

VII Entities to be modeled in EDW

- **ISSUES_HUB**: Represents the central hub for issues.
- **PROJECT_HUB**: Represents the central hub for projects.
- **ISSUES_PROJECT_LINK**: Establishes the relationship between issues and projects.
- **ISSUES_ISSUES_LINK**: Connects related issues.
- **ISSUES_ISSUES_LINK_SAT**: Contains issue-related string data.
- **ISSUES_JIRA_TESTING_SAT**: Consists of 10 strings and 2 options for Jira testing data.
- **ISSUES_JIRA_TESTING_MSAT**: Holds a single array for Jira testing data.
- **ISSUES_JIRA_HDR_SAT**: Contains 12 strings related to Jira header data.
- **PROJECT_JIRA_HDR_SAT**: Holds 2 strings related to project header data in Jira.
- **NOT REQUIRED**: Consists of 2 strings that can be computed by Tableau and are not needed for modeling.
- **ISSUES_JIRA_SLA_STR_SAT**: Contains 6 options for Jira Service Level Agreement (SLA) data.
- **ISSUES_JIRA_SLA_NS_ARR_SAT**: Consists of 9 nested arrays for Jira SLA data (to be modeled in a live environment).
- **ISSUES_JIRA_COMMENTS_SAT**: Holds 3 arrays and 1 nested array for Jira comments data.
- **ISSUES_JIRA_CF_STR_OPT_SAT**: Contains 9 strings and 8 options for Jira custom field data.

VIII EDW_JIRA ERD

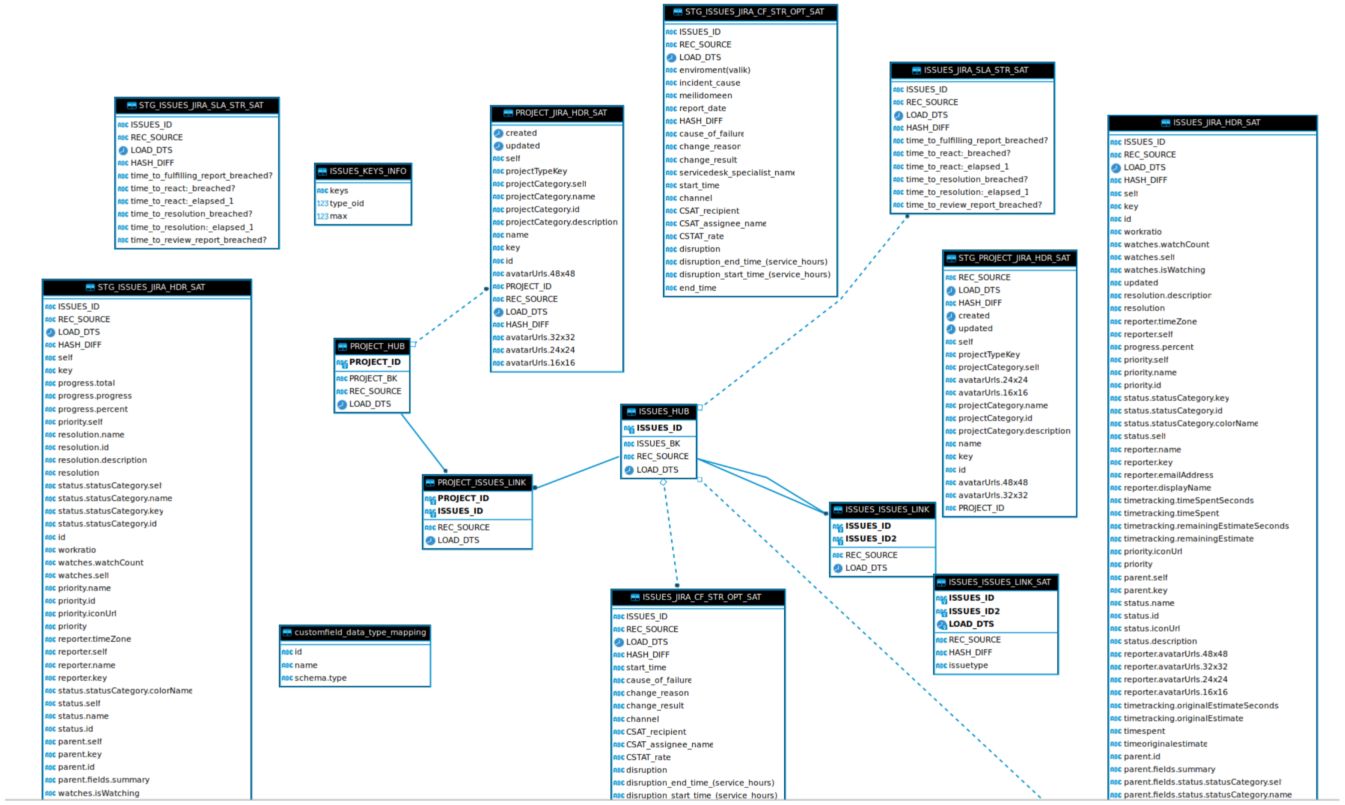


Figure 26. EDW_JIRA ERD.

IX DWH_JIRA ERD

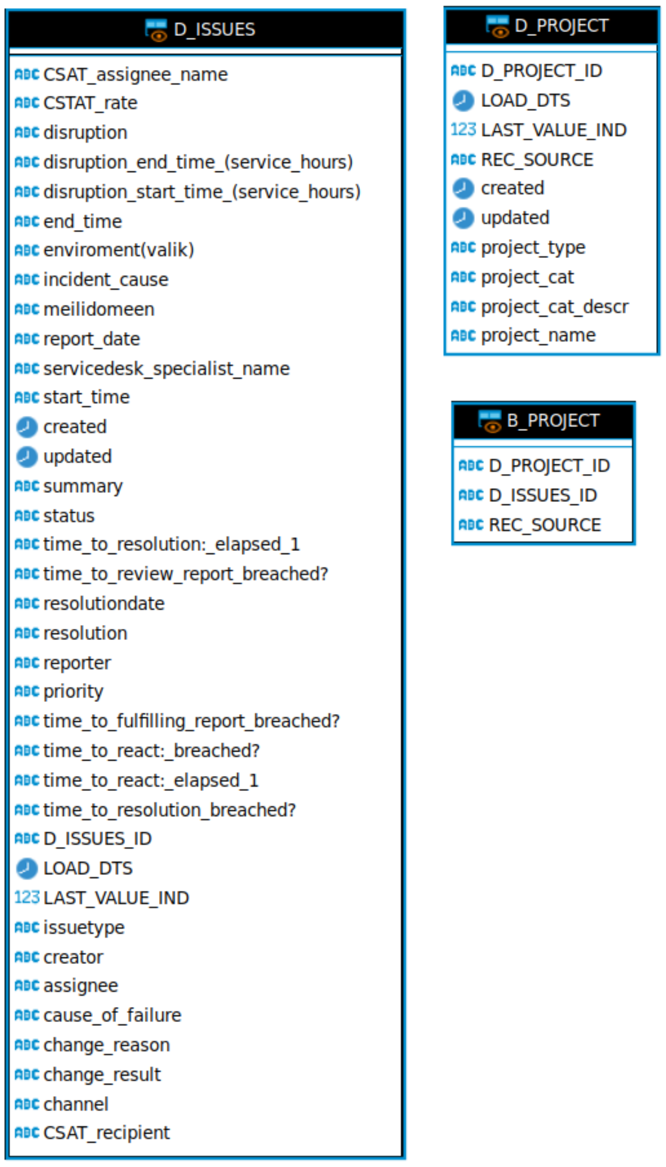
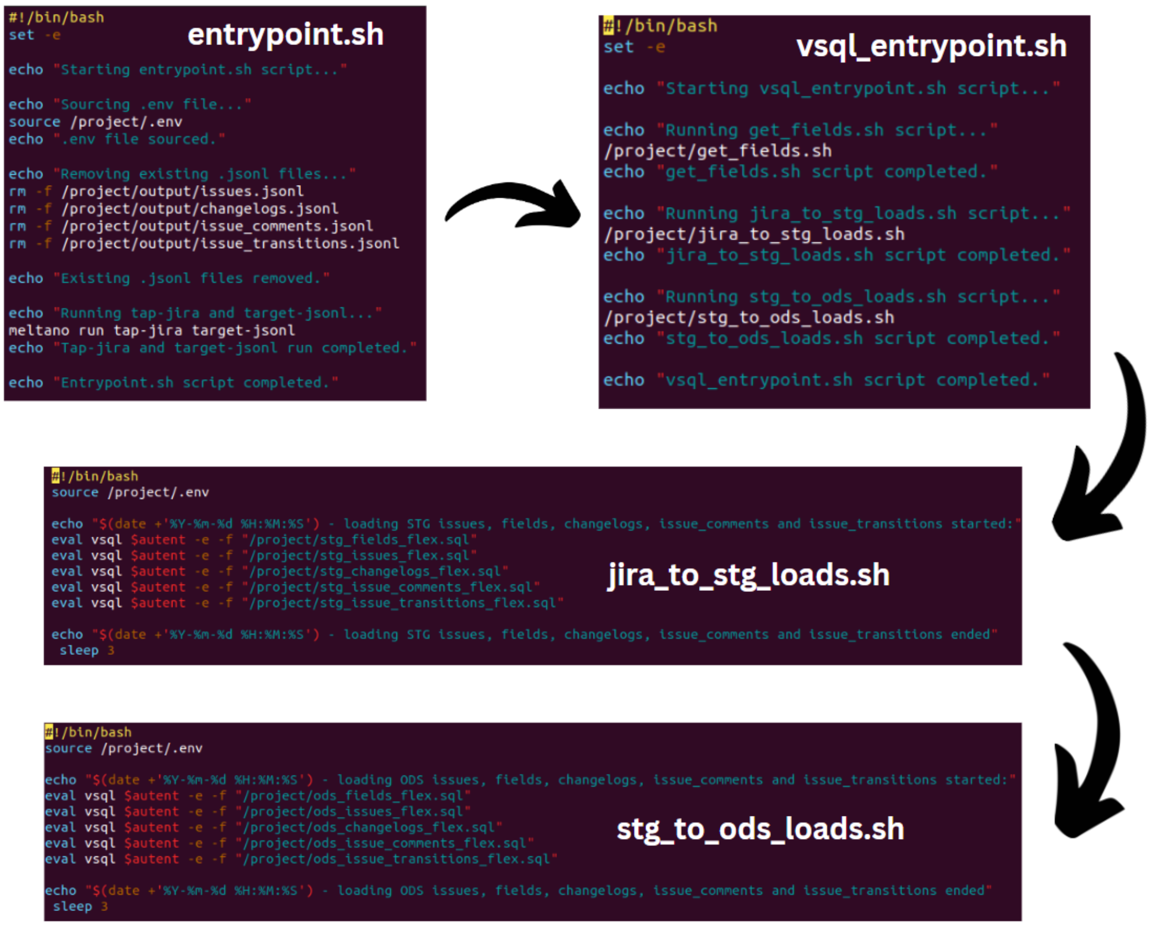


Figure 27. DWH_JIRA_ERD.

X Dockerfile_vsql

```
FROM ubuntu:20.04

RUN apt-get update
RUN apt-get install bash
RUN apt-get -y install curl

\# Create project directory
RUN mkdir -p /project

COPY vertica-client-12.0.3-0.x86\_64.tar.gz /project
COPY ./scripts /project
COPY ./loadings /project

RUN mkdir -p /opt/vertica/
RUN mv /project/vertica-client-12.0.3-0.x86\_64.tar.gz /opt/vertica/
RUN tar vzxf  /opt/vertica/vertica-client-12.0.3-0.x86\_64.tar.gz
WORKDIR /project

RUN find . -maxdepth 1 -type f -exec chmod +x {} \;

ENV PATH=\$PATH:/opt/vertica/bin:/usr/bin
RUN chmod ugo+x /opt/vertica/bin/vsql
ENV  LC\_ALL C.UTF-8
```

XI Bash scripts used for executing containerized pipeline


entrypoint.sh

```
#!/bin/bash
set -e

echo "Starting entrypoint.sh script..."

echo "Sourcing .env file..."
source /project/.env
echo ".env file sourced."

echo "Removing existing .jsonl files..."
rm -f /project/output/issues.jsonl
rm -f /project/output/changelogs.jsonl
rm -f /project/output/issue_comments.jsonl
rm -f /project/output/issue_transitions.jsonl

echo "Existing .jsonl files removed."

echo "Running tap-jira and target-jsonl..."
meltano run tap-jira target-jsonl
echo "Tap-jira and target-jsonl run completed."

echo "Entrypoint.sh script completed."
```

vsql_entrypoint.sh

```
#!/bin/bash
set -e

echo "Starting vsql_entrypoint.sh script..."

echo "Running get_fields.sh script..."
/project/get_fields.sh
echo "get_fields.sh script completed."

echo "Running jira_to_stg_loads.sh script..."
/project/jira_to_stg_loads.sh
echo "jira_to_stg_loads.sh script completed."

echo "Running stg_to_ods_loads.sh script..."
/project/stg_to_ods_loads.sh
echo "stg_to_ods_loads.sh script completed."

echo "vsql_entrypoint.sh script completed."
```

jira_to_stg_loads.sh

```
#!/bin/bash
source /project/.env

echo "$(date +'%Y-%m-%d %H:%M:%S') - loading STG issues, fields, changelogs, issue_comments and issue_transitions started:"
eval vsql $autent -e -f "/project/stg_fields_flex.sql"
eval vsql $autent -e -f "/project/stg_issues_flex.sql"
eval vsql $autent -e -f "/project/stg_changelogs_flex.sql"
eval vsql $autent -e -f "/project/stg_issue_comments_flex.sql"
eval vsql $autent -e -f "/project/stg_issue_transitions_flex.sql"

echo "$(date +'%Y-%m-%d %H:%M:%S') - loading STG issues, fields, changelogs, issue_comments and issue_transitions ended"
 sleep 3
```

stg_to_ods_loads.sh

```
#!/bin/bash
source /project/.env

echo "$(date +'%Y-%m-%d %H:%M:%S') - loading ODS issues, fields, changelogs, issue_comments and issue_transitions started:"
eval vsql $autent -e -f "/project/ods_fields_flex.sql"
eval vsql $autent -e -f "/project/ods_issues_flex.sql"
eval vsql $autent -e -f "/project/ods_changelogs_flex.sql"
eval vsql $autent -e -f "/project/ods_issue_comments_flex.sql"
eval vsql $autent -e -f "/project/ods_issue_transitions_flex.sql"

echo "$(date +'%Y-%m-%d %H:%M:%S') - loading ODS issues, fields, changelogs, issue_comments and issue_transitions ended"
 sleep 3
```

Figure 28. bash scripts used for executing containerized pipeline

XII. Licence

Non-exclusive licence to reproduce thesis and make thesis public