Modern Data Platform with DataOps, Kubernetes, and Cloud-Native Ecosystem

Thesis · April 2023

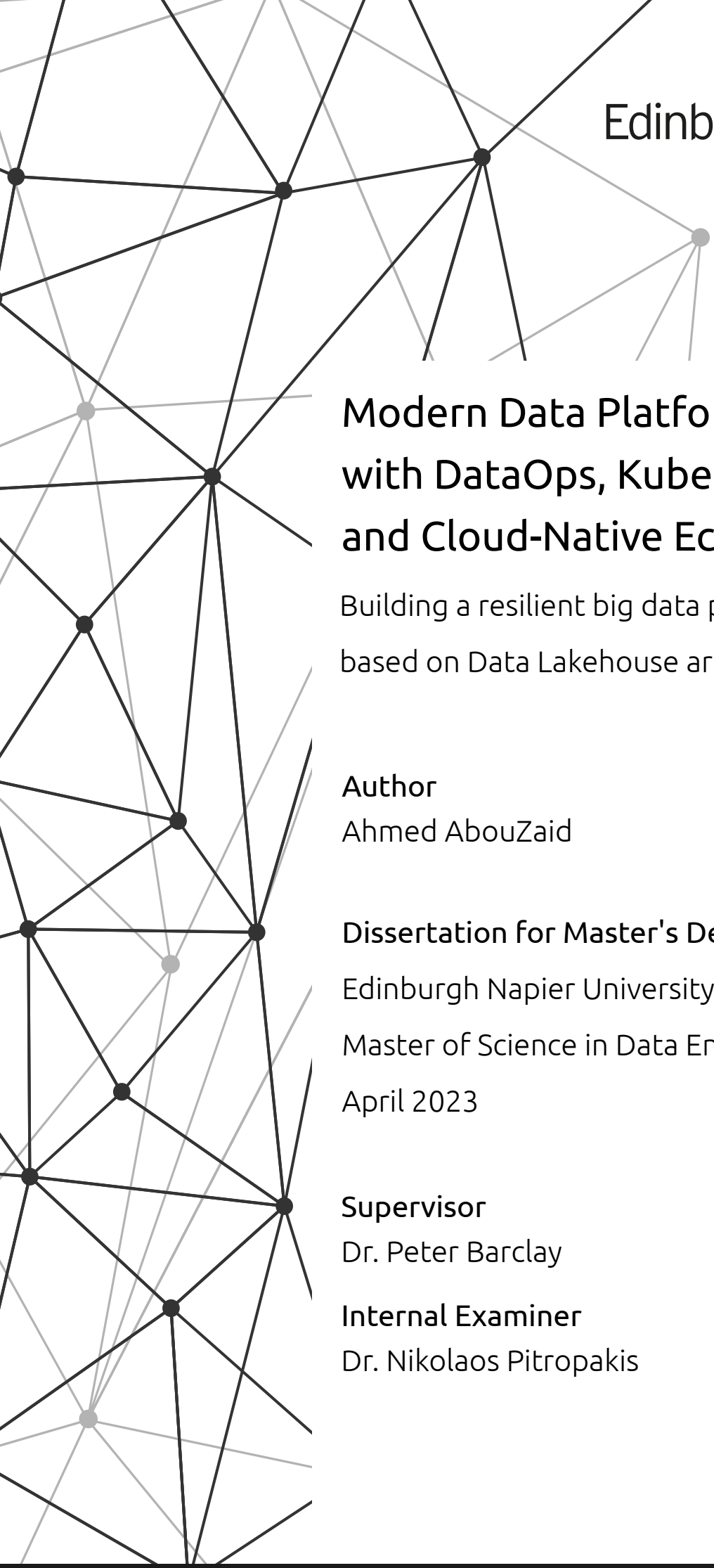
DOI: 10.13140/RG.2.2.15360.71689

1 author:

Ahmed M. Abouzaid

3 PUBLICATIONS **3** CITATIONS

SEE PROFILE



Edinburgh Napier
UNIVERSITY

Modern Data Platform with DataOps, Kubernetes, and Cloud-Native Ecosystem

Building a resilient big data platform based on Data Lakehouse architecture

Author
Ahmed AbouZaid

Dissertation for Master's Degree

Edinburgh Napier University

Master of Science in Data Engineering

April 2023

Supervisor
Dr. Peter Barclay

Internal Examiner
Dr. Nikolaos Pitropakis

Modern Data Platform

with DataOps, Kubernetes, and Cloud-Native Ecosystem

Building a resilient big data platform

based on Data Lakehouse architecture

Ahmed AbouZaid

Submitted in partial fulfilment of the requirements of

Edinburgh Napier University

for the Degree of

Master of Science in Data Engineering

School of Computing

April 2023

MSc Dissertation Checklist

Learning outcome	The markers will assess	Pages[1]	Hours spent
Learning outcome 1 Conduct a literature search using an appropriate range of information sources and produce a critical review of the findings.	* Range of materials; list of references * The literature review/exposition/background information chapter	22-44	180 Hours
Learning outcome 2 Demonstrate professional competence by sound project management and (a) by applying appropriate theoretical and practical computing concepts and techniques to a non-trivial problem, or (b) by undertaking an approved project of equivalent standard.	* Evidence of project management (Gantt chart, diary, etc.) * Depending on the topic: chapters on design, implementation, methods, experiments, results, etc.	16-21 45-103	240 Hours
Learning outcome 3 Show a capacity for self-appraisal by analysing the strengths and weakness of the project outcomes with reference to the initial objectives, and to the work of others.	* Chapter on evaluation (assessing your outcomes against the project aims and objectives) * Discussion of your project's output compared to the work of others.	104-112	110 Hours
Learning outcome 4 Provide evidence of the meeting learning outcomes 1-3 in the form of a dissertation which complies with the requirements of the School of Computing both in style and content.	* Is the dissertation well-written (academic writing style, grammatical), spell-checked, free of typos, neatly formatted. * Does the dissertation contain all relevant chapters, appendices, title and contents pages, etc. * Style and content of the dissertation.		120 Hours
Learning outcome 5 Defend the work orally at a viva voce examination.	* Performance * Confirm authorship		1 hour

Have you previously uploaded your dissertation to Turnitin? Yes/No

Has your supervisor seen a full draft of the dissertation before submission? Yes/No

Has your supervisor said that you are ready to submit the dissertation? Yes/No

[1] Please note the page numbers where evidence of meeting the learning outcome can be found in your dissertation.

Authorship Declaration

I, Ahmed AbouZaid, confirm that this dissertation and the work presented in it are my own achievement.

- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work;
- I have acknowledged all main sources of help;
- If my research follows on from previous work or is part of a larger collaborative research project, I have made clear exactly what was done by others and what I have contributed myself;
- I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work of this dissertation following the School's ethical guidelines

Signed: A. AbouZaid

Date: 04.04.2023

Matriculation no: 40497354

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please write your name below ***one*** of the options below to state your preference.

The University may make this dissertation, with an indicative grade, available to others.

A. AbouZaid

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Preface

This dissertation is the culmination of my Master of Science in Data Engineering at Edinburgh Napier University. It was a unique learning experience by all means.

My intention to enrol in a master's program was not recent; it began in my final college year. In 2010, I heard that a university-wide research competition had started; hence, I formed and led a research group of three members and participated in the competition. Our research achieved third place, with only four marks after the first place.

At that time, I knew that I wanted to have a similar experience again, but I decided first to gain hands-on experience in the technology industry. Later, in 2020, after a decade-long of that research competition, I enrolled in this program at Edinburgh Napier University, where I experienced significant growth in my personal, academic, and professional skills.

In my journey, I faced many personal challenges, such as balancing my job and family responsibilities, the unprecedented impact of the COVID-19 pandemic, being a father for the first time, maintaining growth in my career, and transitioning to a Tech lead position. Although it was a challenging combination, it presented a chance to step up and be a better version of myself. At this particular moment, I am happy to say that I persevered and achieved my goals.

Finally, I would like to conclude with one of my favourite quotes (usually attributed to the writer Mark Twain but I never found a reliable reference):

> "The secret of getting ahead is getting started. The secret of getting started is breaking your complex overwhelming tasks into small manageable tasks and starting on the first one.

Ahmed AbouZaid

Berlin, Germany, April 2023

لا يَحمِلُ الحِقدَ مَن تَعلو بِهِ الرُتَبُ ... وَلا يَنالُ العُلا مَن طَبعُهُ الغَضَبُ

– عنترة بن شداد

A man of high standing should not be malevolent … And who is irascible shall not achieve excellence

– Antara Ibn Shaddad (an ancient Arabic poet)

Acknowledgements

Academically, I would like to express my heartfelt gratitude to my dissertation supervisor Dr. Peter Barclay, for his invaluable guidance and support throughout my research journey. Despite his busy schedule, he always made time for me, providing timely and thoughtful feedback that allowed me to make significant progress in my research. I am also grateful to Dr. Nikolaos Pitropakis for his vast knowledge and detailed technical feedback. His expertise and advice shaped this dissertation and made it the best possible. Both professors have helped me develop a deeper understanding of my research topic, refine my research methodology, and improve my writing skills.

Personally, I am grateful to my parents for sowing the seeds of knowledge in me, to my wife, who has been with me during this dissertation, and to everyone who contributed to my growth to evolve into a better version of myself. I also would like to express my appreciation to the people with whom I had discussions on software or data engineering topics in my journey, namely, Amr Ali, Amr Atya, Ashraf Aaref, Hesham Eid, Moustafa Mansour, and Youssef Mamdouh.

Finally, I want to take this opportunity to express my sincere gratitude to the countless individuals across the world who have contributed their time, skills, and expertise to the development and maintenance of free (as in freedom) and open-source software. Their contributions have had a transformative impact on the world of technology, opening up new possibilities for innovation, collaboration, and progress. To all the free and open-source software contributors, thank you for your invaluable contributions that made the world a better place.

Abstract

In today's Big Data world, organisations can gain a competitive edge by adopting data-driven decision-making. However, that requires a modern data platform that is portable, resilient, and efficient to manage organisations' data and support their growth. Building such a platform requires a deep understanding of the latest data approaches, namely, Data Lakehouse architecture. The newly emerging architecture provides best-of-breed based on Data Warehouse and Data Lake characteristics. Furthermore, the change in the data management architectures was accompanied by changes in storage formats, particularly open standard formats like Apache Hudi, Apache Iceberg, and Delta Lake. This research investigates capabilities provided by Kubernetes and other Cloud-Native software, using DataOps methodologies to build a generic data platform that follows the Data Lakehouse architecture. As a result, the research project defines the data platform specification, architecture, and core components to build a proof of concept for the platform. Moreover, the project implemented the core of the proposed platform, which are infrastructure (Kubernetes), ingestion and transport (Argo Workflows), storage (MinIO), and finally, query and processing (Dremio). Subsequently, after the implementation verification and validation, a performance benchmark was conducted using an industry-standard benchmark suite to assess Dremio's caching capabilities, which demonstrated a 33% median enhancement of query duration. In the end, the project concludes that Data Lakehouse architecture is young and still evolving. Hence, the data management landscape expects a fundamental transformation in the near future, especially with the rapid adoption of the open tables formats and leveraging Cloud-Native software, particularly Kubernetes, as a unified orchestration platform. Henceforth, the boundaries between the Cloud and on-premises could fade over time, offering greater flexibility to handle vast amounts of data for different organisations' sizes and use cases.

Contents

List of Tables

List of Figures

List of Appendix Figures

Chapter One:
Introduction

1.1 Background

Undoubtedly, we live in the “Big Data” era, where data size has multiplied several times in the last twenty years (Cisco Visual Networking Index, 2019, p. 5). Not only have data sizes changed, but also new data types and formats have emerged from different sources. The exponential increase in the data brought many new challenges where the old methods could not cope with that kind of data which is different in quantity and quality. Moreover, what makes handling that data challenging is not just being “big” but all of its properties or what is known as the “4 Vs”, which are Volume, Velocity, Variety, and Veracity (Etzion & Aragon-Correa, 2016, p. 148).

The massive increase in the data brings many new opportunities, yet with more challenges (Etzion & Aragon-Correa, 2016, p. 152). For example, “the challenges multiply when data from multiple sources are combined, particularly where the identifiers used by the underlying systems are not the same. The risks are particularly serious where the data are sensitive” (Clarke, 2015, p. 82). Furthermore, utilising external services like the public Cloud to handle such data could increase the risk of vendor lock-in (Kratzke, 2014, p. 2). Thus, the old methods might not be able to deal with that change, where finding new ways to handle the new challenges of Big Data is mandatory. Especially for data management, where it is the first gate to all other data-driven activities like Artificial Intelligence, Machine Learning, and Business Intelligence. For example, the 2018 Kaggle Machine Learning & Data Science survey showed that data professionals spent most of their time cleaning data (23%) (Hayes, 2019). As a result, investing in building a modern data platform, as well as defining the characteristics of that platform, like the architecture, technologies, and methodologies of building it, are crucial on an equal footing for many organisations.

1.2 Aims and Scope

This project aims to investigate the current Big Data challenges and how building a modern data platform using Kubernetes and Cloud-Native software in harmony with DataOps methodologies could address some of those challenges and provide an efficient solution for data management. To achieve that goal, this project aims to answer the following research questions:

- **RQ1:** What are the main current challenges of managing Big Data?
- **RQ2:** How can DataOps methodologies help to manage Big Data?
- **RQ3:** How can Kubernetes and Cloud-Native software help to build an efficient data platform for Big Data?

The significant change in the data management landscape in recent years, influenced by Big Data, requires an organisational transformation that affects the "Golden Triangle": People, Process, and Technology (Simon, 2019). Nevertheless, this project focuses on two sides of the Golden Triangle, process and technology, to build a blueprint and a proof-of-concept of a modern data platform using DataOps guidelines and Kubernetes and Cloud-Native software. After building the initial version of the platform, it will be verified and validated against the defined focus areas using real-world data for COVID-19 cases between 2020 and 2022. Finally, the platform performance will be assessed using an industry-standard benchmark suite, namely, Transaction Processing Performance Performance Performance Council for Decision Support (TPC-DS).

1.3 Methodology

To achieve the project's objectives and answer the research questions, the Design Science Research Methodology (DSRM) by Peffers et al. (2007) will be used. The DSRM model has six stages:

1. Identify the Problem and Motivate.
2. Define the Objectives of a Solution.
3. Design and Development.
4. Demonstration.
5. Evaluation.
6. Communication.

Although the model has a nominal sequential order which might look somehow rigid at first glance, in fact, the model's creators do not expect researchers to always work in that manner. In addition, the model uses an iterative method, allowing it to fall back to the previous stages. For these reasons, the model fits this project's use case. Figure 1.1 summarises the sections relevant to each process stage in agreement with the DSRM model.

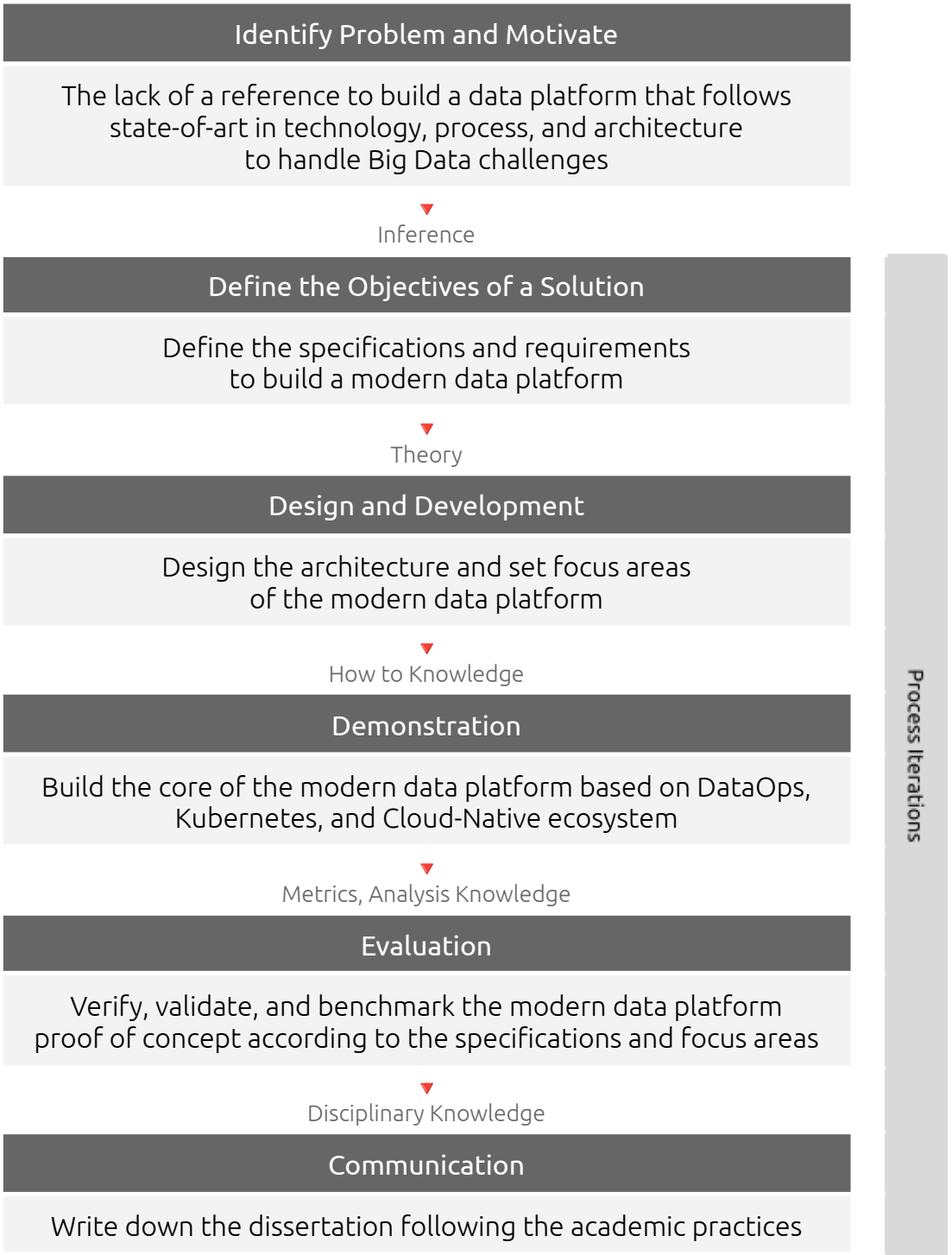


Figure 1.1: The DSRM process model (adapted from Peffers et al., 2007).

1.4 Dissertation Structure

In addition to the introduction chapter, the remainder of this dissertation is organised as follows:

Chapter Two - Literature review

This chapter provides a comprehensive overview of the existing research and literature relevant to the current project topic. It critically evaluates the theories, concepts, and findings of previous studies that have addressed similar research questions and identifies the gaps and limitations that need to be addressed.

Chapter Three - Specifications

This chapter outlines the platform specifications, functional and non-functional requirements, and the project's high-level plans. It also includes the prioritisation methodology to define the focus areas of the platform's initial architecture.

Chapter Four - Architecture

This chapter defines the platform's architecture and design, which will also be developed in this project. It describes the platform's infrastracture, components, functionalities, and how they interact. Further, it will cover the initial design of the architecture which be implemented.

Chapter Five - Implementation

This chapter provides details about implementing the platform developed for this project. It discusses the development approach, the software used to build the platform, and the workflow of the implemented components.

Chapter Six - Technical Evaluation

This chapter presents the evaluation of the platform's initial design. It uses standard evaluation practices, precisely verification and validation, to analyse and compare the implementation with the defined specifications and architecture.

Chapter Seven - Benchmarking

This chapter further extends the evaluation of the platform's initial design since the performance assessment is an essential part of the evaluation. Therefore, it conducts a performance benchmark focusing on caching capabilities using an industry-standard benchmark suite.

Chapter Eight - Results and Discussion

This chapter summarises the project's key findings and highlights the research project's contributions. Then it discusses the implications and suggests areas for future work.

Chapter Two:
Literature Review

2.1 Overview

This chapter aims to discover why building a modern data platform based on the DataOps, Kubernetes, and Cloud-Native ecosystem could be vital to deal with Big Data challenges.

2.2 Big Data

A variety of data types has rapid growth due to the increase of data sources such as social networks, semantic webs, Internet of Things sensors, and location-based services. As previously stated in the introduction, the “Big Data” characteristic is not only about being “big” but the “4 Vs”, which are Volume, Velocity, Variety, and Veracity (Etzion & Aragon-Correa, 2016, p. 148). The increase in the data provided many opportunities, such as (Travica, 2017, p. 4):

- **Improve decision-making:** Analysing large amounts of data, organisations can gain insights and make more informed decisions.
- **Personalisation:** Big Data can be used to personalise products, services, and experiences for individual customers or users.
- **Fraud detection:** Large datasets can be analysed to identify patterns of fraudulent activity and help organisations prevent losses.
- **Risk assessment:** Big Data can be used to assess and mitigate risks in a variety of industries, including finance, insurance, and healthcare.
- **Supply chain optimisation:** Big Data can be used to optimise the efficiency of supply chain operations and reduce costs.
- **Predictive maintenance:** Analysing vast amount of data from sensors and other sources, organisations can predict when equipment will fail to take preventive action.
- **Improved customer experience:** By analysing customer data, organisations can improve the customer experience and increase customer loyalty.
- **New product development:** Big Data can identify new product or service opportunities to meet customer needs and preferences.

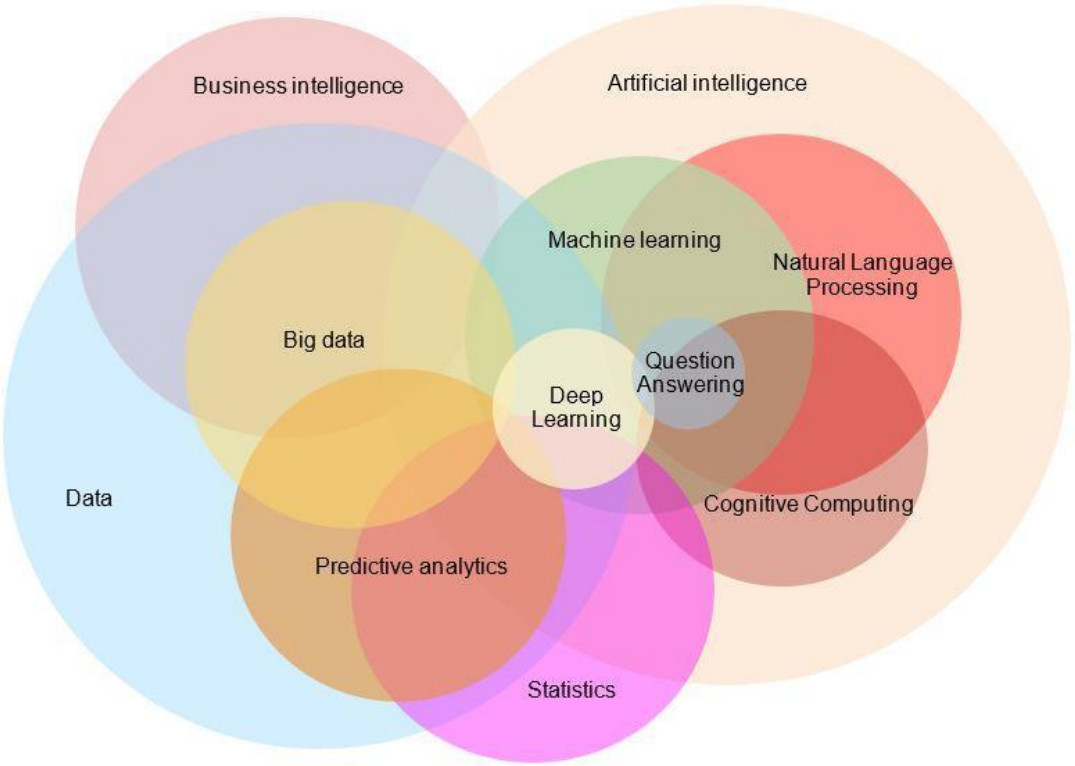


Figure 2.1: A Venn diagram illustrates how Big Data overlaps directly or indirectly with different fields (Data Fields Overlap, 2020).

Figure 2.1 shows the interactions between Big Data and other data fields. Data management is crucial because it is the first gate to all other data-driven activities like Artificial Intelligence, Machine Learning, and Business Intelligence. While "data is the new gold" is a common phrase, every opportunity comes with its own set of challenges, and traditional methods often fall short of addressing them. Hence, to get that gold, there are obstacles to reaching that goal, specifically finding an efficient way to manage it (Yugal, 2022, p. 237). That is why data management platforms have evolved over time to deal with the changes qualitatively and quantitatively. For example, the data quality could severely affect the opportunities of Big Data (Clarke, 2015, pp. 78–80), and high quality requires a strong process to handle it. That led to emerging "DataOps" as a result.

2.3 DataOps

The term “DataOps” was coined in 2014 for the first time by Lenny Liebmann (3 Reasons Why DataOps Is Essential for Big Data Success | the Big Data Hub, 2014), notwithstanding the practices it encompasses have been around for much longer. Based on the changes in the nature of data in the last two decades, it is clear that “data management approaches have changed drastically in the past few years due to improved data availability and increasing interest in data analysis (e.g., artificial intelligence). Data volume, velocity, and variety require novel and automated ways to ‘operate’ this data. In accordance with software development, where DevOps is the de-facto standard to operate code, DataOps is an emerging approach advocated by practitioners to tackle data management challenges for analytics” (Mainali et al., 2021, p. 61).

DataOps provides a process-oriented approach on top of normal data pipelines, which is a crucial element in handling Big Data (Munappy et al., 2020, p. 182). Following the Agile path, DataOps Manifesto was released in 2017 to provide a framework for data professionals to work together more effectively and efficiently. That Manifesto has eighteen principles as a guide to applying DataOps methodologies which are: (1) Continually satisfy your customer, (2) Value working analytics, (3) Embrace change, (4) It is a team sport, (5) Daily interactions, (6) Self-organize, (7) Reduce heroism, (8) Reflect, (9) Analytics is code, (10) Orchestrate, (11) Make it reproducible, (12) Disposable environments, (13) Simplicity, (14) Analytics is manufacturing, (15) Quality is paramount, (16) Monitor quality and performance, (17) Reuse, (18) Improve cycle times (DataOps Manifesto, 2017).

Furthermore, DataOps combines the Agile, DevOps, and Lean Manufacturing principles with the integration of people, processes, and technology to create a more efficient and effective data processing pipeline. It is often seen as a natural evolution of DevOps, a set of practices that aims to improve collaboration, communication, and integration between software developers and IT professionals (Strod, 2019b). “It is the combination of proven methodologies that helped to grow other industries: DevOps and Agile methodology from the

software industry and Lean Manufacturing from the automotive/manufacturing industry" (Mainali et al., 2021, p. 63). Figure 2.2 illustrates the roots of DataOps.

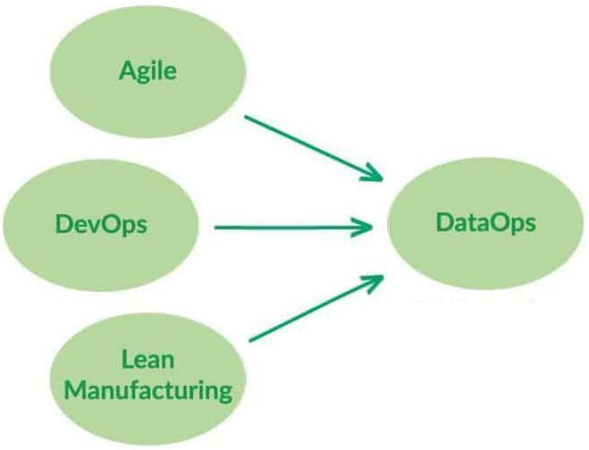


Figure 2.2: The three roots of DataOps: Agile, DevOps, and Lean Manufacturing (Strod, 2019b).

Moreover, DataOps builds on those principles and applies them to data processing and analytics, where it utilises orchestration, workflow management, and automation tools to enable flexible and customised data transformations as needed (Mainali et al., 2021, p. 65). It is often seen as a way to bridge the gap between data engineers, who are responsible for building and maintaining the infrastructure and pipelines that handle data, and data scientists, who are responsible for analysing and interpreting the data to generate insights and predictions (Mainali et al., 2021, pp. 64-65). Figure 2.3 summarises the data-driven aspirations in the manner of the DataOps model.

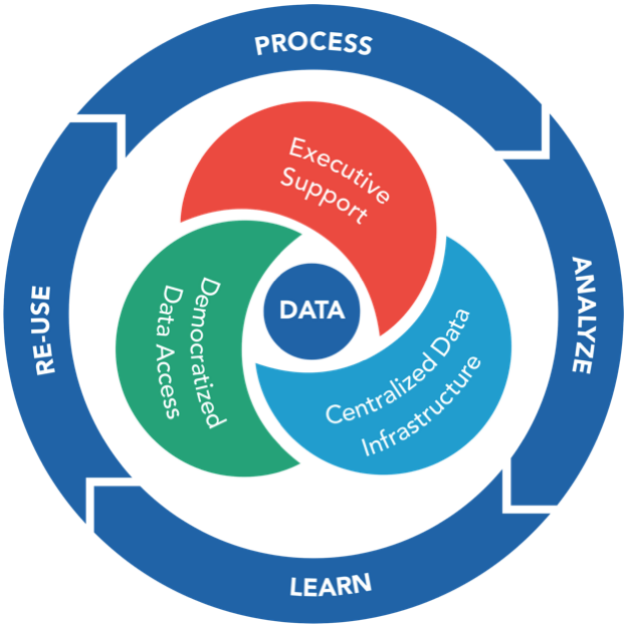


Figure 2.3: The aspirations for a data-driven enterprise (Thusoo & Sen Sarma, 2017, figs. 1.6).

In terms of DataOps adoption, DataOps is considered a highly intuitive approach to building a data environment. Consequently, newer companies are leaning toward adopting DataOps with ease and greater speed than established ones, which requires a significant overhaul of their existing data practices and mindset (Thusoo & Sen Sarma, 2017, Chapter 1). That is clear in one of the surveys about DataOps (Gür, 2021), which showed that the top challenges are:

- Establishing formal processes (55%).
- Orchestrating code and data across tools (%53).
- Staff capacity (50%).
- Monitoring the end-to-end environment (50%).

That proves, in the real world, pure technical challenges like working with code/data across tools and having full visibility of the environment also influence adopting process-oriented practices like DataOps. Therefore, to adopt agile processes like DataOps to deal with Big Data challenges, a change in the data management platforms on the technical level is inevitable.

2.4 Data Management Architecture

A Data Management Platform (DMP) is a central place to manage data from one or more sources. The definition of the DMP varies, depending on the domain (Boch et al., 2022, pp. 16–17). Generally speaking, DMPs could be specialised like DMP to manage research data as well as could be generic to manage different kinds of data (typically, data associated with the business, regardless of its type). The current research focuses on the generic DMP, which ideally should work with structured, semi-structured, and unstructured data. As the data grows, the DMPs architecture evolved over time to provide added value to businesses, namely Data Warehouse, Data Lake, and finally, Data Lakehouse.

2.4.1 Data Warehouse

Data Warehouse (DWH) started in the 1980s and has been used as a suitable location to store analytical data in a predefined format (Inmon, 2005, p. 2). As specified by Ralph Kimball, which has made significant contributions to defining and polishing the DWH foundation, a data warehouse is “a system that extracts, cleans, conforms, and delivers source data into a dimensional data store and then supports and implements querying and analysis for decision making” (Kimball & Caserta, 2004, p. 22).

DWHs have been used for decades, so they are battle-tested and have had the best optimisations for a long time. Hence, they provide many advantages working as a source of truth, high-quality data, effective query, and paramount supporting businesses to embrace data-driven decision-making (Watson et al., 2002, p. 492). On the other hand, DWH also has downsides. Firstly, it works only with structured data, which is insufficient to handle the increase in data types and sizes. Also, operational-wise, DWHs are very expensive; in addition to property software licensing, the data preparation and clean-up processes require a lot of time and resources. These resources are historically used to run on on-premise hardware, adding many operational and maintenance overheads. For these reasons, many enterprises moved to the next generation to deal with these disadvantages, which is Data Lake architecture.

2.4.2 Data Lake

Data Lake (DL) started around 2010; the term was used for the first time by James Dixo, who described it as a "large amounts of heterogeneous data are added from a single source, and users can access them for a variety of analytical use cases" (Giebler et al., 2019, p. 180). Furthermore, Fang (2015, p. 820) defined the term as "a methodology enabled by a massive data repository based on low-cost technologies that improve the capture, refinement, archival, and exploration of raw data within an enterprise. A data lake contains the mess of raw unstructured or multi-structured data that for the most part have unrecognised value for the firm".

With the rise of Cloud Computing and cheap commodity storage solutions, DL was able to handle the new Big Data cases (where data comes in different types and sizes); it was also able to store structured and semi-structured formats, and at the same time, put the cost under control. Furthermore, business-wise, another advantage of DL is that it can store more data without knowing case use. At the same time, it provides an easy way to mine the data to find new business cases since it is in a raw format. Nevertheless, when talking about DL disadvantages, and because of its ability to store different types of data, it is easily turned from a "lake" to a "swamp" where the DL is more or less used as a

place to dump data. Moreover, due to the variety of stored data and the lack of a unified way to create a metadata layer, it is hard to govern and provide fine-grained access to DL resources. Finally, the DL performance is not unified because the underneath data vary in size, type, and format (Giebler et al., 2019, p. 180). For these reasons, a new generation emerged to overcome the DL disadvantages, which is Data Lakehouse architecture.

2.4.3 Data Lakehouse

Data Lakehouse (DLH) is a relatively new hybrid architecture shown for the first time in 2020 which combines the capabilities of a DL and a DLH simultaneously. It is designed to handle both structured and unstructured data in a scalable, flexible, and cost-effective way. The research by Armbrust et al. defines DLH as "a data management system based on low-cost and directly-accessible storage that also provides traditional analytical DBMS management and performance features such as ACID transactions, data versioning, auditing, indexing, caching, and query optimisation. Lakehouses thus combine the key benefits of data lakes and data warehouses: low-cost storage in an open format accessible by various systems from the former, and powerful management and optimisation features from the latter" (2021, p. 3). By inspecting that definition, it shows how DLH architecture promises to fix DL weakness while providing all DWH strengths as described in the previous sections.

Additionally, the change in the data management architectures was accompanied by changes in storage systems since the traditional Big Data storage formats like Parquet and ORC are optimised for read-heavy workloads and lack support for updates and deletes. Thus, there was a need for a layer on top of them to maintain data quality and consistency. Hence, the new DLH architectures "center around open storage formats such as Delta Lake, Apache Hudi and Apache Iceberg that implement transactions, indexing and other DBMS functionality on top of low-cost data lake storage (e.g., Amazon S3) and are directly readable from any processing engine. Lakehouse systems are quickly replacing traditional data lakes" (Jain et al., 2023, p. 1). Figure 2.4 illustrates a high-level structure of the Apache Iceberg table as an example of an open table format (Iceberg Table's Architecture, 2022).

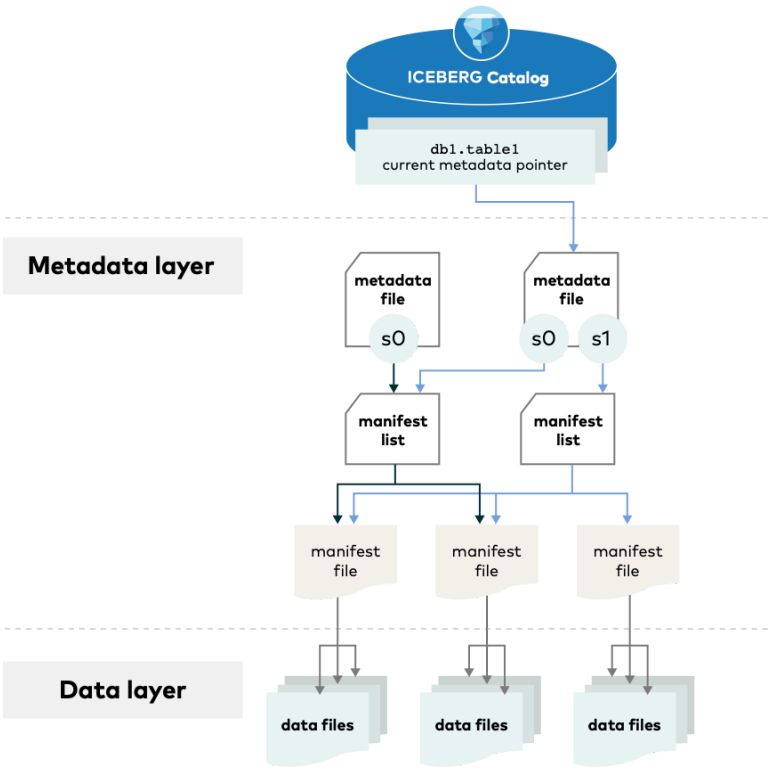


Figure 2.4: The structure of the Apache Iceberg table (Iceberg Table's Architecture, 2022).

Open table formats like Apache Hudi, Apache Iceberg, and Delta Lake are designed to improve data management for Big Data workloads where these formats provide a separation between computing and data. Overall, these open table formats provide organisations with advanced data management capabilities that help improve data quality and consistency, increase productivity, and reduce costs. To deal with Big Data workloads, the new formats provide fundamental advantages like transactional processing, schema evolution, data versioning, and time travel (Armbrust et al., 2020; Späti, 2022; Jain et al., 2023). As of early 2023, all three open table formats (Apache Hudi, Apache Iceberg, and Delta Lake) have seen significant usage in production environments and adoption across the majority of vendors like AWS, Google, Azure, Snowflake, Databricks, Dremio, and Cloudera, which is a promising indicator to widely support the new open formats (Clark, 2023). However, it is worth pointing out that even though the open table formats evolved and provided approximately the same features, one of the latest researches about DLH storage systems by Jain et al. (2023) showed that open table formats vary simultaneously with different factors.

Hence, it is essential to pay attention to the properties of each format and how suitable they are to the nature of the project or the use case. Figure 2.5 shows how data update strategies like Copy-On-Write (CoW) and Merge-On-Read (MoR) affect the performance of different open table formats.

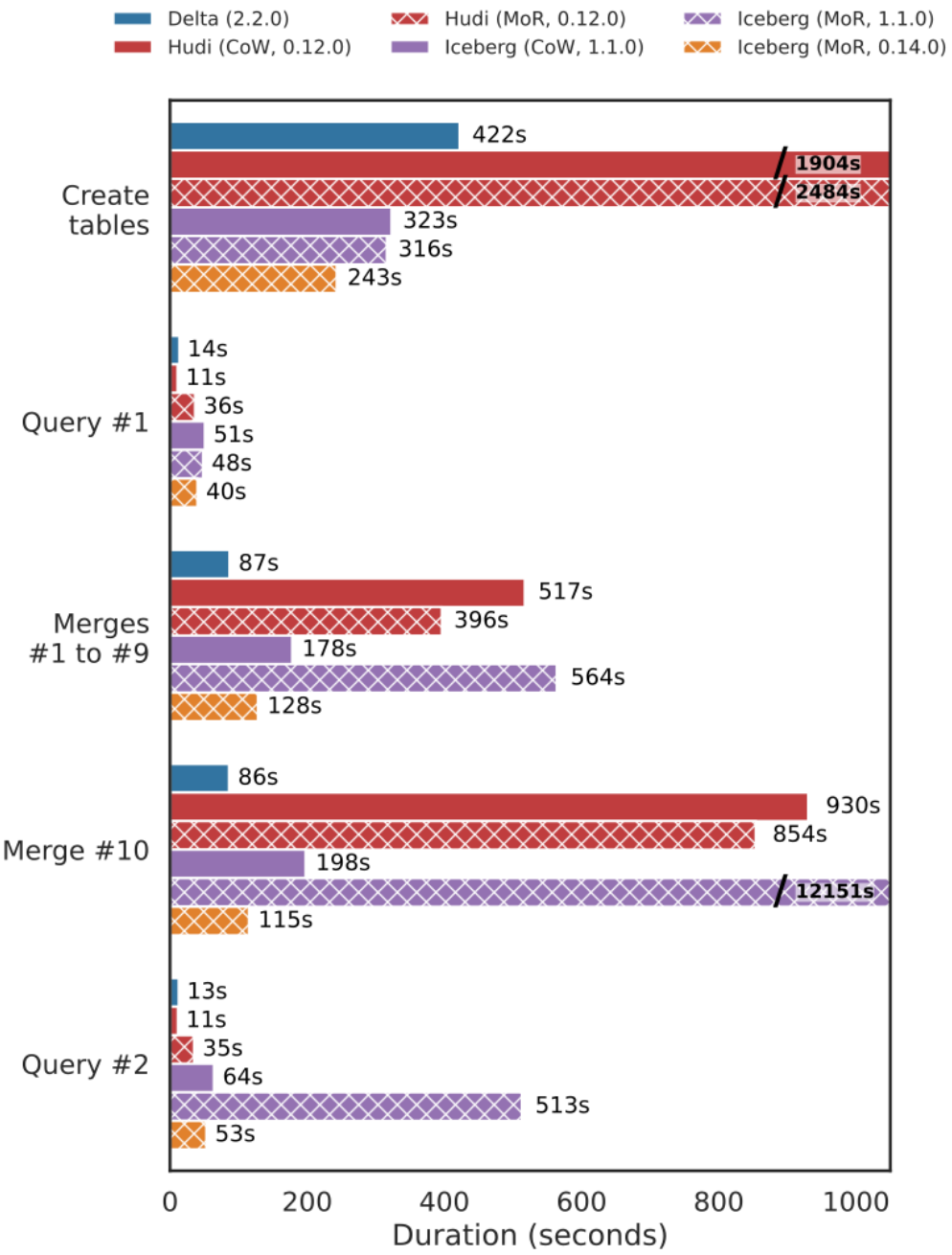


Figure 2.5: How data update strategies like Copy-On-Write (CoW) and Merge-On-Read (MoR) affect open table formats performance (Jain et al., 2023, fig. 2).

2.4.4 Comparison

Figure 2.6 and Table 2.1 show a comparison among DWH, DL, and DLH, which illustrates how DLH provides the best-of-breed based on DWH and DL properties. However, in addition to all the pure technical advantages provided by DLH, one of the main strategic advantages for many enterprises is using standard open data format (e.g. Apache Hudi, Apache Iceberg, and Delta Lake), which provides multiple benefits for different use cases, like enabling seamless access to the data by various analytics engines, including machine learning systems. Also, avoiding vendor lock-in is a pivotal aspect of data management to move data across different systems and platforms, primarily where the Cloud vendors' capabilities are used to handle the Big Data.

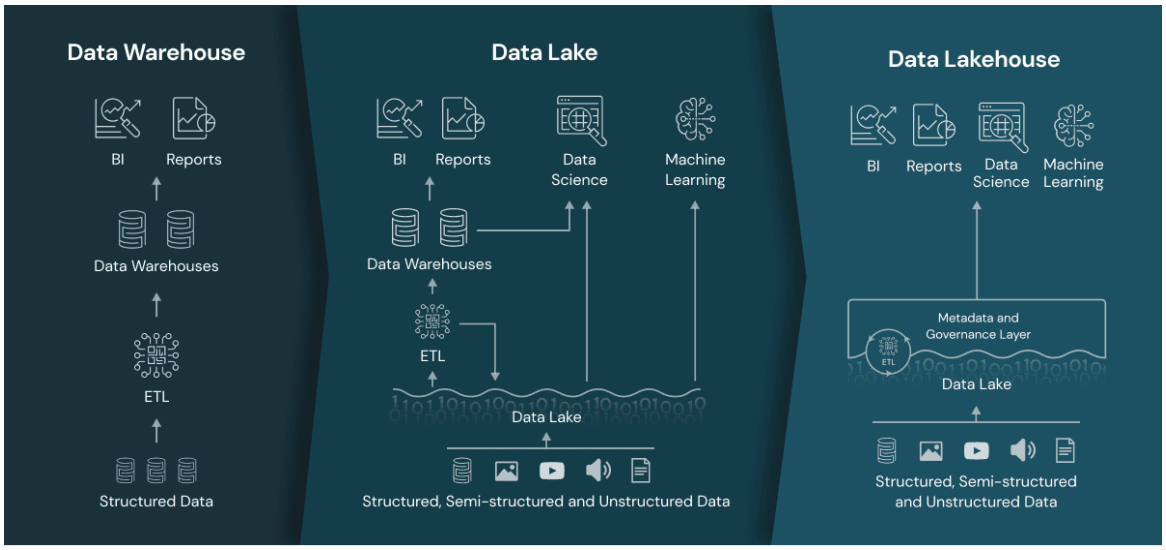


Figure 2.6: A high-level architectural comparison of
Data Warehouse, Data Lake, and Data Lakehouse (Lorica et al., 2020).

	Data Warehouse	Data Lake	Data Lakehouse
Data Types	Structured data and processed data	Structured, semi-structured and unstructured raw data	Structured, semi-structured, and unstructured, both processed/raw data
Data Format	Closed, proprietary format	Open format	Open format

Purpose	Optimal for data analytics and business intelligence use-cases	Suitable for machine learning and artificial intelligence workloads	Suitable for all use-cases (data analytics, business intelligence, machine learning and artificial intelligence)
Cost	Storage is costly and time- consuming	Storage is cost-effective, fast, and flexible	Storage is cost-effective, fast, and flexible
Users	Business professionals	Business analysts, data scientists, data engineers	Everyone in the business environment
Scalability	Scaling might be difficult because of tightly coupled storage and compute	Scaling is easy and cost- effective because of the separation of storage and compute	Scaling is easy and cost- effective
Agility	Less agile, fixed configuration	Highly agile, adjustable configuration	Highly agile, adjustable configuration
Analytics	Reporting, BI, dashboards	Advanced analytics	Suitable for all types of analytics workflows
Ease of use	The fixed schema makes data easy to locate, access, and query	Time and effort are required to organise and prepare data for use. Extensive coding is involved	Simple interfaces are provided that are similar to traditional data warehouses together with in-built AI support
Processing	Schema-on-write	Schema-on-read	Schema-on-write and Schema-on-read
ACID compliance	Records data in an ACID-compliant manner to ensure the highest level of integrity	Non-ACID compliance: updates and deletes are complex operations	ACID-compliant to ensure consistency as multiple parties concurrently read or write data

Table 2.1: High-level comparison among Data Warehouse, Data Lake, and Data Lakehouse (E. Janssen, 2022, p. 44).

2.5 Cloud Computing

Cloud computing refers to a service delivery model for information technology in which resources such as software, storage, and computing power are made available over the internet to users on demand. In this model, users do not own or manage the physical infrastructure that supports these resources but instead access them as a utility provided by a third-party service provider. Therefore, Cloud computing allows users to access and use these resources on demand rather than purchasing and maintaining their physical hardware and software (Ruparelia, 2016, pp. 1-4). Cloud computing could be categorised into three categories (Ruparelia, 2016, pp. 30-33):

- Public Cloud is a computing environment owned and operated by a third-party Cloud services provider, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform. Public clouds are designed to be accessed over the internet and are generally available to any organisation or individual that wants to use them. They offer a variety of services, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Public clouds are generally more cost-effective and easier to use than private clouds, but they may not offer the same level of security or control.

- Private Cloud is a computing environment owned and operated by a single organisation for exclusive use. Private clouds can be implemented on-premises or hosted by a third-party provider. They offer the benefits of Cloud computing, such as scalability and flexibility, but with the added security and control of an on-premises environment. Private clouds are generally more expensive and require more resources than public clouds, but they may be necessary for organisations with strict security or compliance requirements.

- Hybrid Cloud is a computing environment that combines both public and private clouds. It allows organisations to use the best of both worlds by using public clouds for certain workloads and private clouds for others. For example, an organisation might use a public Cloud for non-critical workloads that do not require high security and a private Cloud for

sensitive workloads that need to be kept in-house. Hybrid clouds offer the benefits of both public and private clouds, but they can be more complex to manage and require careful planning to ensure that workloads are placed in the appropriate environment.

In addition, when it comes to the management modes, there are three main types of Cloud computing services management vary depending on the characteristic of each of them as follows (Ruparelia, 2016, pp. 27–29):

- **Infrastructure as a Service (IaaS)** provides users with access to virtualised computing resources, such as virtual machines, storage, and networking.
- **Platform as a Service (PaaS)** provides users access to a platform for developing, testing, and deploying applications.
- **Software as a Service (SaaS)** provides users with access to software applications that can be used over the internet without the need to install and maintain the software on their computers.

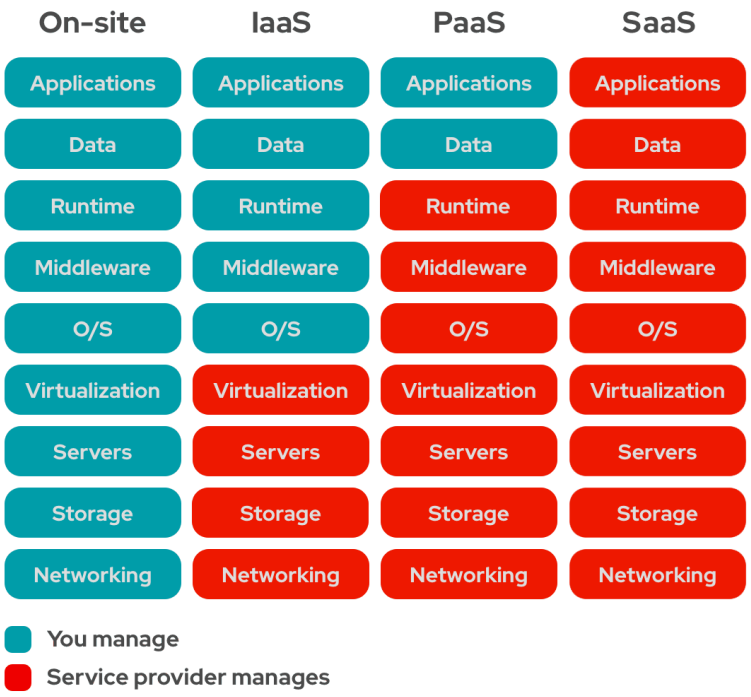


Figure 2.7: Management responsibility in different Cloud service models (IaaS Vs. PaaS Vs. SaaS, 2020).

Because of all the benefits of the Cloud, the adoption rate has soared over time, especially in recent years (More Than Half of Enterprise IT Spending in Key Market Segments Will Shift to the Cloud by 2025, 2022). Nevertheless, many enterprises have challenges and obstacles using Cloud services, especially data-related ones. According to research conducted on UK organisations, it showed that "data portability and interoperability concerns were the most discussed theme in relation to vendor lock-in" (Opara-Martins et al., 2016, p. 4) and "while security and governance concerns often can be answered by encryption, and cost concerns can be answered by cost-based decision-making models, vendor lock-in problems stay" (Kratzke, 2014, p. 2). The same research from Opara-Martins et al. (2016, p. 10) showed how using a data architecture that relies on standard open data format could play a key role in avoiding vendor lock-in where "overall, the results indicate that these challenges closely relate to interoperability and data portability issues prevalent in the Cloud environment". Furthermore, when questioned about the best ways to minimise the risks of vendor lock-in during Cloud migration, the majority of business participants identified the following strategies as the most effective means of mitigation:

A. Making well-informed decisions before selecting vendors and signing Cloud contracts (66.4%).
B. The need for an open environment for continuous competition between providers in the Cloud service market (52.3%).
C. Using standard software with industry-proven interfaces (39.3%).

2.6 Cloud-Native Software

Cloud-Native refers to the architecture and design of an application built specifically to take advantage of the Cloud computing model. In other words, a Cloud-Native application is designed to be scalable, resilient, and take advantage of the automatic provisioning of resources provided by Cloud platforms. Thus, Cloud-Native applications are often designed to be run on a distributed system and are built using microservices and containers, which allow them to be easily deployed and scaled (Domingus & Arundel, 2022, p. 17).

As a result, Cloud-Native software can be beneficial for data management because it is designed to take advantage of the benefits of Cloud computing, which include (Domingus & Arundel, 2022, p. 16):

- **Elasticity:** It enables automatic scaling up or down depending on demand, thus ensuring high availability and efficient handling high traffic volume.
- **Resilience:** It is designed to be fault-tolerant and recover quickly from failures. Therefore, it is ideal for mission-critical applications that should be highly available.
- **Cost effectiveness:** It can reduce costs by only using the needed resources at any given time because it can automatically scale up and down in keeping with demand.
- **Flexibility:** It can be easily deployed to a variety of different environments, including on-premises, public Cloud, or in a hybrid Cloud environment. That makes it easy to use in different use cases.
- **Innovation:** It can be developed and deployed faster than traditional software, allowing organisations to be more agile and responsive to changing business needs.

In general, Cloud-Native software works best on Cloud-Native platforms like Kubernetes, which is an open-source platform for automating the deployment, scaling, and management of containerised applications. Kubernetes provides a way to deploy predictable and scalable applications, making managing and maintaining large and complex applications more accessible. Moreover, Kubernetes has become the de-facto standard for container orchestration and is widely adopted in private and public sectors. That is, in particular, conforming to a survey by the Cloud Native Computing Foundation (CNCF), where Kubernetes is used by more than 50% of organisations that deploy containerised applications in production (Cloud Native Survey 2021, 2021, p. 5). In addition, many major Cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, offer managed Kubernetes services, further contributing to its widespread adoption (Cloud Native Survey 2021, 2021, p. 8).

Another recent report by Mayr (2023) shows the rapid embrace of Kubernetes, where the key discovery highlights that "as Kubernetes adoption increases and it continues to advance technologically, Kubernetes has emerged as the 'operating system' of the cloud". The same report found that from 2021 to 2022, the application workloads increased by 30% and auxiliary workloads by 211%. The "auxiliary workload" generally refers to software that supports the main functionality of an application or system, such as logging, monitoring, and backup, which are important for the proper operation and maintenance of the software but are not directly related to the primary functions that the software is designed to perform. The increase in the number of auxiliary workloads number confirms that organisations are increasingly adopting sophisticated Kubernetes technologies, such as security controls, messaging systems, observability tools, and service meshes. Furthermore, they use Kubernetes for broader use cases, such as building pipelines and scheduled utility workloads. As a result, Kubernetes has become the primary platform for running almost any type of workload. Figure 2.8 illustrates the year-over-year increase in workload numbers.

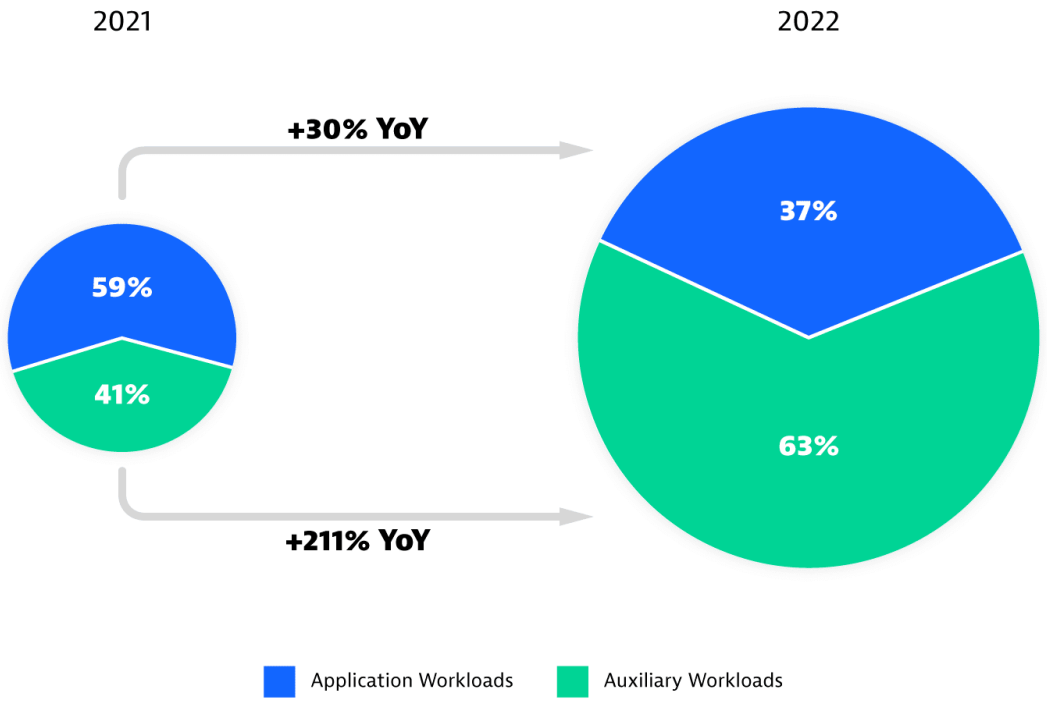


Figure 2.8: The increase in the number of main workloads versus auxiliary workloads on Kubernetes between 2021 and 2022 (Mayr, 2023).

Turning to the newest report, at the time of writing, by CNCF in 2023, it showed that the general increase in Kubernetes adoption was also combined with a usage increase in specific categories, namely, the core of the data platforms like databases (48%) and Big Data applications (35%). Moreover, "across all categories, open source projects rank among the most frequently used solutions", demonstrating how open source software and open standards play a critical role in shaping the data management landscape (CNCF Annual Survey 2022, 2023). Figure 2.9 represents the Kubernetes growth areas by category between 2021 and 2022.

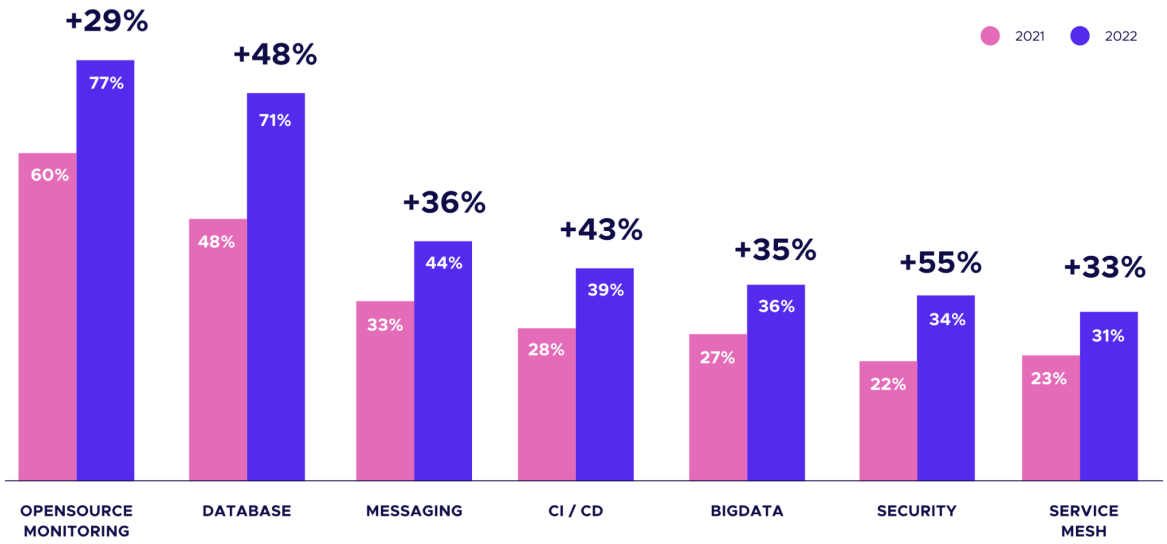


Figure 2.9: The top seven categories in which Kubernetes experienced growth during the period between 2021 and 2022 (CNCF Annual Survey 2022, 2023).

Finally, one of the most crucial Kubernetes features is that it was built with extensibility in mind. Thus, Kubernetes provides a powerful mechanism to extend its API to manage complex applications and services as if they were native Kubernetes resources. That pattern in Kubernetes is called "Operator", which enables automating "Day 2" operations and is designed to encapsulate the best practices, workflows, and knowledge of a particular application or service, allowing it to be managed in a Kubernetes-native way (Operator Pattern, 2023). Day 2 operations refer to the ongoing management and maintenance of a software application or system after the installation (Day 1). By utilising Kubernetes Operators, the effort and time required to maintain application

availability, performance, and security can be significantly reduced, allowing Kubernetes administrators and users to concentrate on business requirements instead of reinventing the wheel and creating redundant solutions (Dobies & Wood, 2020, Chapters 1, 10). Figure 2.10 shows the five Operators' capability levels according to OperatorHub.io.

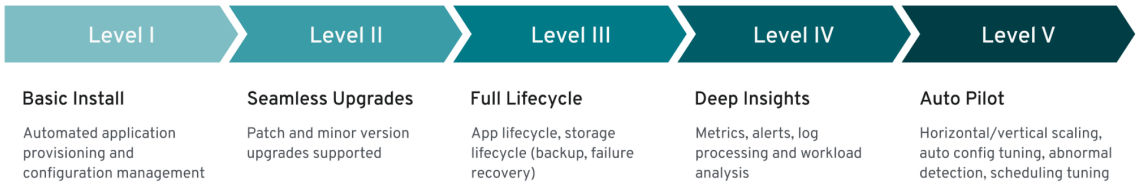


Figure 2.10: Kubernetes Operators' capability levels (Jump Start Using the Operator-SDK, n.d.).

For those reasons, Kubernetes and Cloud-Native ecosystem are promising solutions to handle the current Big Data challenges since they provide many advantages not only on the technical level but also on the strategic level, like portability and using open standards to counter vendor lock-in issues where the Cloud-Native platforms and software, "truly deliver digital capabilities anywhere and everywhere" (Costello & Rimol, 2021).

2.7 Modern Data Platform

There have been different attempts to architect and build a data platform that can cope with Big Data and fits different workloads and use cases, the so-called "Modern Data Platform" (MDP) is one of them. The term in the technology industry refers to a set of technologies and practices used to collect, store, process, and analyse large amounts of data, including Cloud computing, Big Data processing frameworks, data warehousing and business intelligence tools, and machine learning and artificial intelligence algorithms. An MDP aims to enable organisations to gain insights from their data to make data-driven decisions (Foote, 2022; LaPlante, 2020). Nevertheless, the term "Modern Data Platform" is not commonly used in academic literature, even though the concepts and technologies that make up an MDP are widely studied and discussed in academic research. For example, research on Big Data processing, Cloud Computing, Data Warehousing, and Data Analysis are all relevant to developing and implementing an MDP.

As shown in the previous sections, given that many data technologies are relatively new, like Data Lakehouse, a hybrid architecture shown for the first time in 2020, reviewing MDP research results has been limited to the years between 2020 and 2023. Using a mix-and-match technique with keywords like “modern”, “data”, “platform”, “management”, “lakehouse”, and “architecture”, the Google Scholar search engine showed that most of the related work are conference papers cover an overview of the Data Lakehouse architecture without many details about actual methods of implementing such an architecture which is a clear gap in the academic research.

For example, the article “From Data Warehouse to Lakehouse: A Comparative Review” by Harby and Zulkernine (2022) provided a comparative review of Data Warehouses and Data Lakehouses architectures. The authors highlighted the advantages and disadvantages of each architecture, as well as their suitability for different types of data and use cases. Furthermore, they compared the two architectures in terms of data integration, data modelling, data processing, scalability, and security. Even though they found that Data Lakehouse has several advantages over data warehouses, they also noted that managing it could be cumbersome because of its many moving parts.

Another related work is the paper titled “Data Lakehouse - a Novel Step in Analytics Architecture” by Orescanin and Hlupic (2021) covered the concept of a Data Lakehouse, which, as mentioned, combines the strengths of data warehouses and data lakes in a single architecture and how it provides a more scalable, flexible, and cost-effective solution for modern data analytics. The paper also covered a high-level architecture of Data Lakehouse and the advantages of each component, such as the ability to handle real-time data ingestion and processing, support for Big Data analytics, and improved data governance. Then the paper also provided examples of companies that created successful Data Lakehouses products, such as Databricks, Snowflake, and Amazon Web Services. Finally, the authors argued that Data Lakehouses are a novel step in analytics architecture that can provide significant benefits for organisations seeking to leverage their data for competitive advantage.

One interesting detailed piece of research about Data Lakehouses is the master's thesis "The Evolution of Data Storage Architectures: Examining the Value of the Data Lakehouse" by E. Janssen (2022), which explored in-depth the data storage architectures and investigated the value of Data Lakehouse architecture. The thesis began with an introduction to the history of data storage architectures, including data warehouses and data lake architecture, including their advantages and disadvantages. The author then introduced the Data Lakehouse architecture, which combines the strengths of both architectures. Furthermore, The thesis also included a case study of a company that has implemented a Data Lake and had evaluated moving to a Data Lakehouse architecture. The study concluded that the company would likely experience significant benefits from the Data Lakehouse architecture, including improved data quality, faster data processing, and increased team collaboration. Also, the research reviewed multiple neutral and vendor-specific architectures for Data Lakehouse; however, it did not address the implementation phase nor use Kubernetes as a platform for the architecture.

Finally, a paper titled "Xel: A cloud-agnostic data platform for the design-driven building of high-availability data science services" by Barron-Lugo et al. (2023) is indirectly related to the current project where the paper shares the same goals as well as using state-of-the-art tools and applications to build a portable data platform for data science services. "Agnosticism is a term used in Cloud computing to define the property of a Cloud service/application that does not depend on a given platform or infrastructure to be successfully executed" (Barron-Lugo et al., 2023, p. 88), hence; the Xel project could benefit from the MDP as infrastracture for the data science services.

According to the previous research review, no study covered building a generic data platform using DataOps, Kubernetes, and Cloud-Native ecosystem, especially a platform focusing on specific aspects like openness, portability, and averting vendor lock-in (cloud-agnostic). Hence, this research tries to cover this gap and gives an overview of the state-of-the-art in building a data platform that leverages modern practices and technology for the time being. It is essential to realise that the ultimate goal of MDP is not the technology but to build a data-driven and self-service culture within the organisation to serve the business

needs; hence, building a data platform helps to create a data-centric business where different personas smoothly interact with the data platform. Figure 2.11 illustrates a high-level architecture of a modern functional data platform, and Figure 2.12 shows the major personas that are part of the data team in a data-centric business.

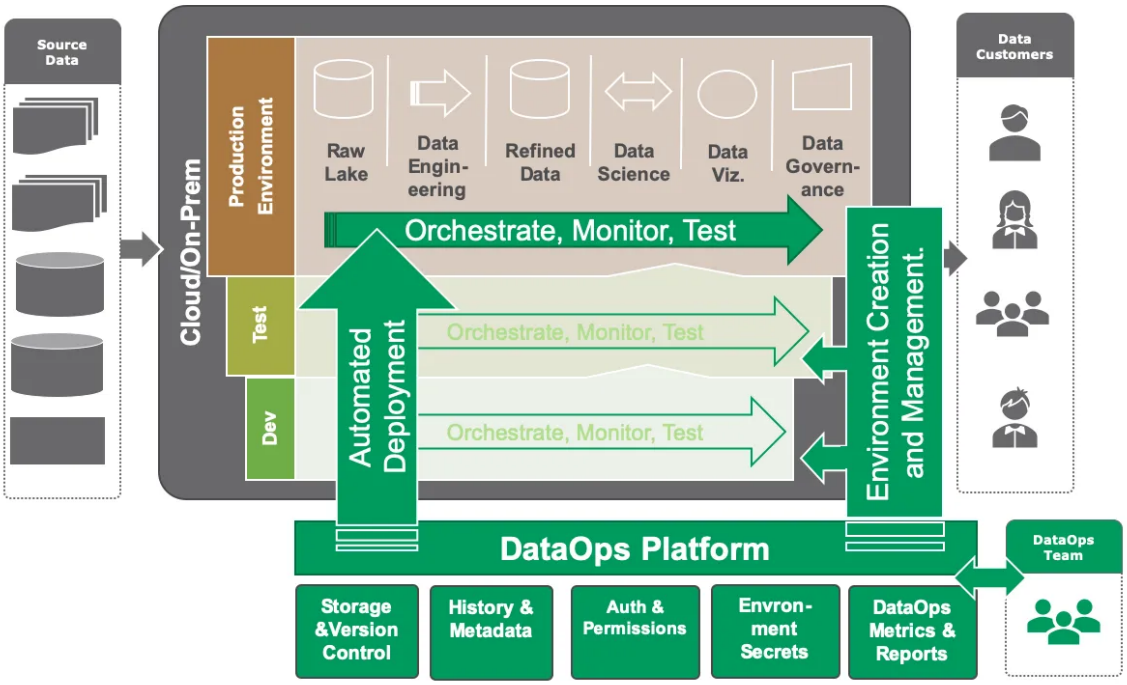


Figure 2.11: High-level architecture of a modern functional data platform (Strod, 2019a).

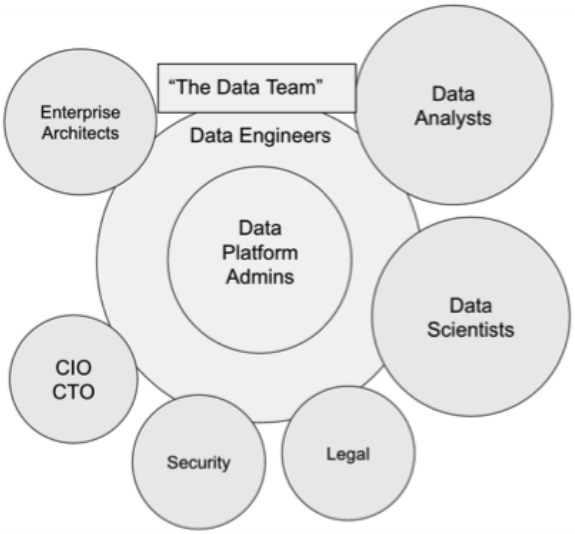


Figure 2.12: The core job role profiles of the data team in a data-centric business (Thusoo & Sen Sarma, 2017, figs. 4-2).

2.8 Summary

In the last two decades, Big Data has been a major driver of the paradigm shift in data management, and with the increasing amount of data generated, traditional data management systems and approaches need to be improved or even replaced. Hence, new technologies and architectures, such as Data Lakehouse, have emerged to handle the volume, velocity, and variety of data. These technologies enable organisations to store and process large amounts of data cost-effectively, allowing them to gain new insights and make more informed decisions. Consequently, organisations can effectively manage the challenges posed by Big Data with greater flexibility and efficiency by leveraging different technologies like Cloud computing, DataOps, and Cloud-Native software and platform such as Kubernetes. This approach facilitates the extraction of value from Big Data, leading to improved data quality, enhanced collaboration, and faster time-to-market for data-driven products and services. In light of that, building a tailored data platform solution based on modern technologies will enable many companies and organisations, aside from their size or domain, to manage and benefit from Big Data.

Chapter Three:
Specifications

3.1 General Goals

This section aims to define the specifications related to the modern data platform, like how it should work, the core requirements, and the focus areas for the initial implementation. In order to do that, first, we need to shed light on the high-level goals of such a platform. The aspiration is commonly driven by various business objectives, including enhancing decision-making, uncovering customer preferences and patterns, and increasing operational efficiency. The vast majority of enterprises (97%) have developed a documented data strategy to become data-driven, showing a widespread acknowledgement of the value of data strategies. However, only a small fraction of businesses (31%) have succeeded in transforming into data-driven organisations or built a “data culture” (28%). In fact, firms tend to view themselves as failing to transform their businesses in all areas except driving innovation with data. Specifically, over half of the businesses (53.1%) admit to not yet treating data as a business asset, and an even more significant percentage (52.4%) declare that they are not competing on data and analytics. The lack of success in creating a data-driven culture is highlighted by the fact that over two-thirds of firms (69%) admit to not having achieved this objective, with a similar proportion (71.7%) stating that they have not forged a data culture (LaPlante & Safari, 2020).

Nevertheless, adopting a data-driven approach touches many areas in an organisation to achieve efficiency, which requires organisational transformation and change management. There are multiple frameworks to handle such a change; one of them is the “Golden Triangle” framework, which Harold Leavitt invented in the 1960s. The Golden Triangle, or the People, Process, Technology (PPT) framework, highlights the importance of balancing all three components to achieve success and optimal outcomes. Even though the PPT framework is a popular model used in business management and IT, it can be applied to any industry to help organisations improve their business processes and achieve their strategic objectives (Simon, 2019). Figure 3.1 conceptualises the relation between the triangle’s sides where the change in one side affects the others.

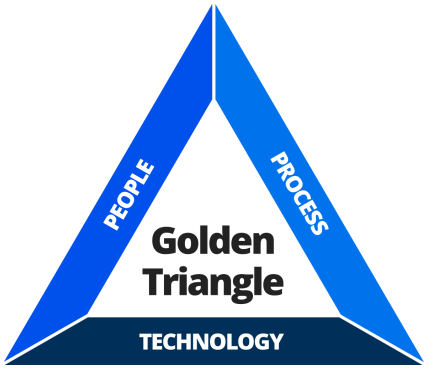


Figure 3.1: The Golden Triangle can be envisioned as a structure comprising three pillars, where the equilibrium is contingent on the stability of each individual pillar (Simon, 2019).

This research's primary focus is two sides of the framework, which are Technology (Kubernetes and Cloud-Native software) and the related Process (DataOps) to build a proof-of-concept of a modern data platform according to the typical business drivers, and at the same time, overcomes the concerns from previous research regarding Cloud adoption (Opara-Martins et al., 2016), vendor lock-in (Kratzke, 2014), and data management processes (Gür, 2021).

3.2 Requirements

Reviewing previous attempts to build a data management platform, as discussed in section 2.7 Modern Data Platform, showed that most data platforms share some functional and non-functional requirements. Functional requirements describe what actions a system must perform, and non-functional requirements describe qualities a system must possess (Robertson & Robertson, 2012, p. 9).

3.2.1 Functional

- **Data Ingestion:** The platform must have the capability to ingest data from various sources, including structured, semi-structured, and unstructured data, in real-time and batch modes. The platform must support multiple data formats and can handle high volumes of data.
- **Data Storage:** The platform must have efficient and scalable data storage capabilities, including different types of databases like SQL and NoSQL databases as well as provide data indexing, search, and retrieval capabilities. Also, the data platform must support multiple storage technologies and systems, including classic plain text formats like JSON and CSV, in addition to modern table formats like Apache Hudi, Apache Iceberg, and Delta Lake.

- **Data Processing:** The platform must have robust data processing capabilities, including batch processing, real-time processing, and stream processing. Furthermore, the platform must support multiple data processing frameworks, including Apache Spark, Apache Flink, and Apache Storm, and provide a flexible and extensible processing pipeline.
- **Data Analytics and Collaboration:** The platform must have advanced data analytics capabilities, including machine learning, predictive analytics, and data visualisation. Also, the platform must support business analytics visualisation tools, such as charts, graphs, and dashboards, to help users understand and analyse data as well as support scientific analytics tools like multiple machine learning frameworks, including TensorFlow, PyTorch, and scikit-learn, and provide a library of pre-built models and algorithms.
- **Data Governance:** The platform must have robust data governance capabilities, including data discovery, data quality, data lineage, and data auditing, to ensure the accuracy, consistency, and security of the data being processed and stored.

3.2.2 Non-functional

- **Portability:** The platform should mitigate the risks associated with vendor lock-in, enabling the user to seamlessly transition to alternative data processing technologies or vendors with minimum disruption to the existing infrastructure. Furthermore, the platform should support open data standards and provide APIs for easy integration with other tools and systems. Also, it should be designed for Cloud computing environments, leveraging Cloud computing capabilities and easily moving across different Cloud providers.
- **Extensibility:** The platform should be easy to extend or modify to accommodate new functionality or components to handle changing requirements without requiring significant redesign or refactoring. Which is necessary for any rapidly evolving organisation to be data-driven and remain competitive. Furthermore, the platform should be designed with modularity in mind, using open standards and APIs, separating concerns, and prioritising interoperability. Therefore, organisations can build a data platform that adapts to changing business needs and scales as the volume of data grows.

- **Scalability:** The platform should be scalable and handle the increasing data as the demand grows. Also, it should support horizontal scaling, allowing users to add more resources as needed, and vertical scaling, allowing users to add more processing power and storage capacity to existing resources.
- **Performance:** The platform should provide fast and efficient processing and retrieval of Big Data, with response times that meet the user's needs. It should also be able to process large amounts of data in parallel, leveraging distributed computing capabilities to optimise performance.
- **Availability:** The platform should be highly available, ensuring that data is always accessible, even during hardware or network failures. It should support disaster recovery and business continuity strategies and be able to recover from failures automatically, and ensure data availability.
- **Reliability:** The platform should be reliable, with minimal downtime and data loss. It should have built-in fault tolerance and resilience mechanisms and implement best practices for data backup and recovery practices.
- **Cost-Effectiveness:** The platform should be cost-effective, providing a good return on investment and flexible pricing options for cost optimisation as demand changes. It should leverage Cloud computing capabilities to optimise costs and can automatically provision and de-provision resources depending on demand.
- **Usability:** The platform should be easy to use, with a user-friendly interface that is intuitive and accessible to all users. It should support self-service capabilities, allowing users to manage their own data processing workflows without relying on IT support.

3.3 Focus Areas

Taking into consideration the mentioned high-level goals and requirements of the data platform as a starting point, the main focus of this project will be (a) building a resilient infrastructure for the data platform, (b) applying at least one of the DataOps principles, (c) implement the core of the DLH architecture. Accordingly, an iterative approach framework will be used to build a Minimal Viable Product (MVP), to have better quality and reduce failure rates. This approach involves developing the system's essential features first and then gradually adding additional functionalities in subsequent iterations.

To achieve that, the “Must, Should, Could, and Would haves” (MoSCoW) prioritisation method will be used to prioritise which features to include in each iteration. This method categorises features into four groups: Must-haves, Should-haves, Could-haves, and Would-haves (also known as Will-not-haves). Must-haves are essential features for the system to function, while Should-haves are necessary but not essential. Could-haves are desirable but not vital, and Would-haves are features that are needed but explicitly not included in the current iteration (Del Sagrado & Del Águila, 2020, p. 171).

The MoSCoW prioritisation method will ensure that the most important features are included in the first MVP, and at the same time, less critical features can be deferred to later iterations. That allows for a more focused and efficient development process while ensuring that the MVP meets the minimum requirements for the system to be usable. As a result, the following are features of the initial version of the data platform in keeping with the MoSCoW method.

Must Haves:

- MH1: Cloud-Native architecture.
- MH2: Scalable and Cloud-Agnostic infrastructure orchestration system.
- MH3: Open-source software and open standard formats.
- MH4: Data Lakehouse solution as a core of the data platform.

Should Haves:

- SH1: A declarative approach for configuration management.
- SH2: A data pipeline to ingest data from an external source into the platform in plain text formats like JSON or CSV.
- SH3: Self-service capabilities.

Could Haves:

- CH1: An easy way to rebuild the system with minimal manual actions.
- CH2: An open format table like Apache Iceberg from one of the ingested JSON/CSV files.

Would Haves:

- WH1: Production grade quality like high availability, security hardening, or data quality validation.
- WH2: The reset of the components not directly related to the core of the Data Lakehouse.

In light of the defined focus area in line with the MoSCoW method, the next chapter will cover architecting the prioritisated specifications.

Chapter Four:
Architecture

4.1 Holistic View

The literature review showed the evolution of data management architectures like DWH, DL, and finally, DLH, where the DLH architecture considered the new recommended setup to cope with Big Data. In the manner of the following criteria to architect the initial version of the data platform, the architecture should be (a) provider agnostic, (b) generic or cover different use cases, (c) detailed, and (d) up-to-date. By review of six architectures of data platforms created between the years 2020 and 2023 (DataLakeHouse, 2020; Ma et al., 2020, p. 3; MongoDB, 2021; Desai et al., 2022; Bornstein et al., 2022; Oppermann, 2022) showed that the “Unified Data Infrastructure v2.0” (UDI v2.0) architecture by Bornstein et al. is the best match of the defined criteria. Figure 4.1 shows that UDI v2.0 architecture is detailed, modular, and covers all functional requirements, which could work as a general-purpose data platform and allows the creation of different blueprints in line with different use cases (e.g., artificial intelligence, machine learning, multimodal data processing, and business intelligence). Moreover, Table 4.1 lists the definitions of UDI v2.0 layers, which are input sources, ingestion/transformation, storage, query/processing, transformation, analytics, and output.

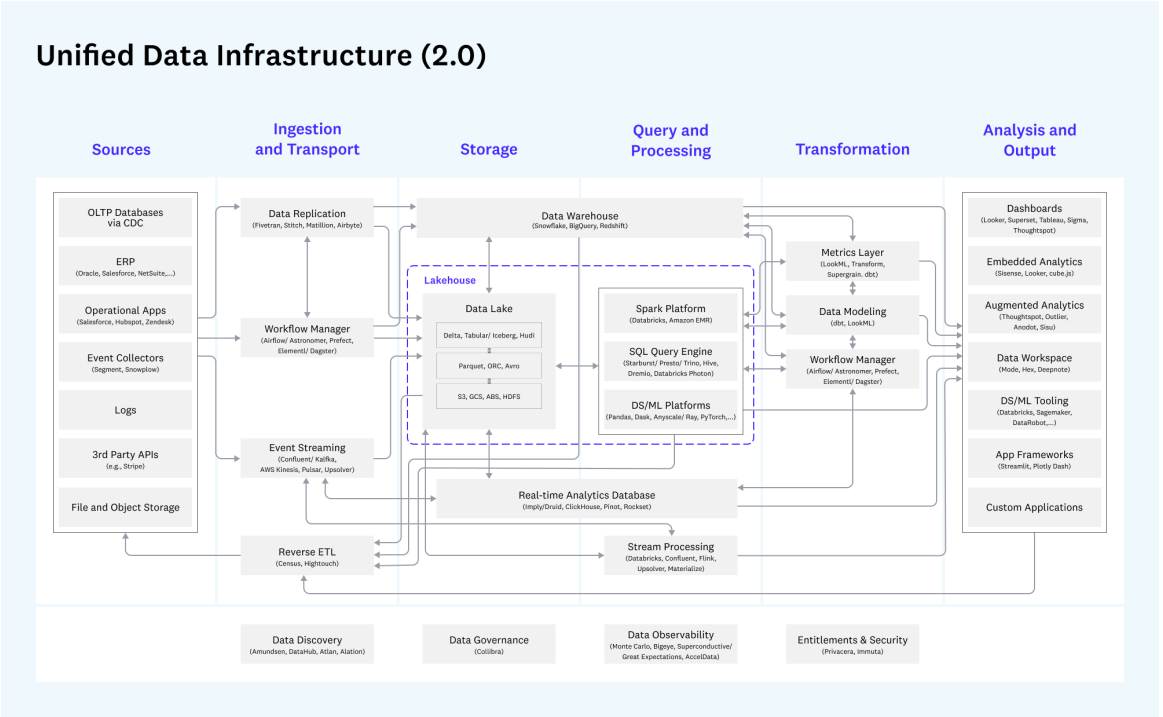


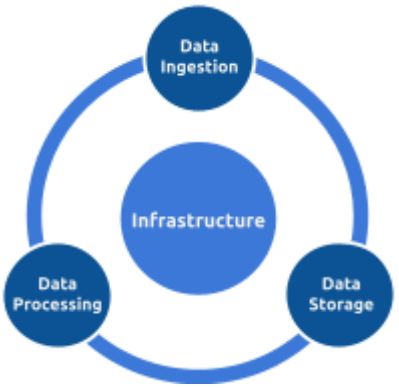
Figure 4.1: The Unified Data Infrastructure v2.0 architecture diagram (Bornstein et al., 2022).

Section	Definition
Sources	Generate relevant business and operational data.
Ingestion and Transport	Extract data from operational systems (E). Deliver to storage, aligning schemas between source and destination (L). Transport analysed data back to operational systems as needed.
Storage	Store data in a format accessible to query & processing systems and optimise for consistency, performance, cost, and scale.
Query and Processing	Translate high-level code (usually written in SQL, Python, or Java/ Scala) into low-level data processing jobs. Execute queries and data models against stored data, often using distributed computing. Includes historical analysis - describing what happened - and predictive analysis - describing expectations for the future.
Transformation	Transform data into a structure ready for analysis (T) Orchestrate processing resources for this purpose.
Analysis and Output	Provide an interface for analysts and data scientists to derive insights and collaborate. Present analysis results to internal and external users and embeds data models into user-facing applications.

Table 4.1: The definitions of the Unified Data Infrastructure v2.0 (Bornstein et al., 2022).

4.2 Core Components

To have an MVP for the data platform based on the Unified Data Infrastructure v2.0, the following components should be implemented, and more components could be added later. The core components covered in this section are infrastructure, data ingestion, storage, and processing. In addition, the initial architecture will be defined afterwards.



4.2.1 Infrastructure

As stated in the literature review, Kubernetes is a container orchestration platform that automates containerised applications' installation, expansion, and administration. Given the fact that it is used by more than 50% of organisations that deploy containerised applications in production (Cloud Native Survey 2021, 2021, p. 5), it is considered the most popular container management system. Its features include self-healing abilities, service discovery and load balancing, storage orchestration, configuration and sensitive data management, resource management, and batch execution.

The extensible capabilities offered by Kubernetes can be tailored to meet the unique requirements of the application or organisation, aiming to make managing and scaling containerised applications simple and efficient in distributed environments. Since Kubernetes is Cloud-Native and Cloud-Agnostic by design, it is the ideal infrastructure orchestration system for data platforms because it offers a shared layer of abstraction that hides the underlying details to deploy and manage containerised applications easily. Kubernetes can run in various environments like on-premises, Cloud providers, including Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), and many others. Figure 4.2 shows Kubernetes architecture which runs the same way on any Cloud, on-premises, or local setup to manage containerised applications.

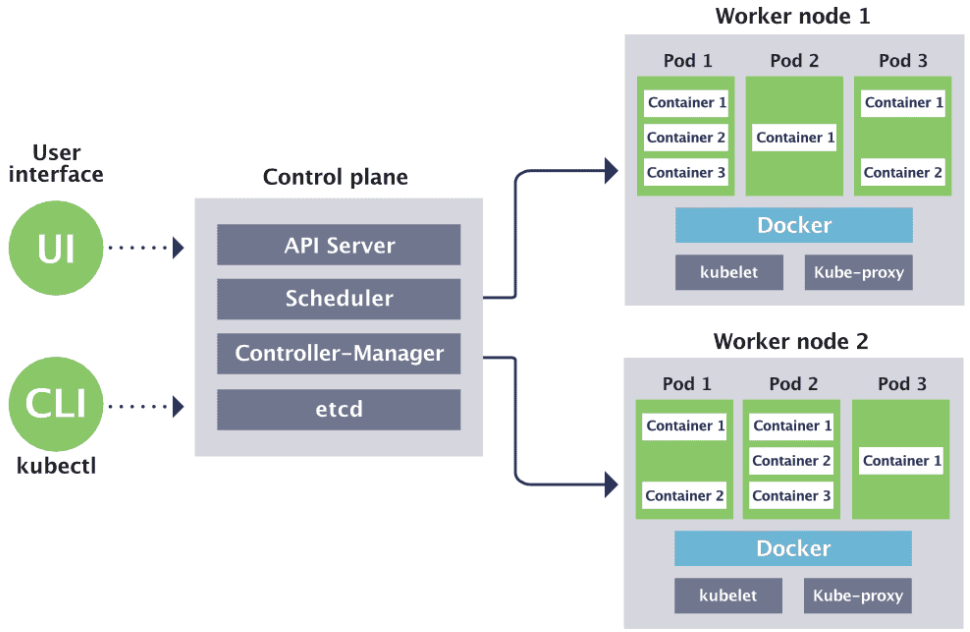


Figure 4.2: Kubernetes high-level architecture
(Kubernetes Architecture, How Kubernetes Works, 2019).

In section 2.6 Cloud-Native Software, it was noted that Kubernetes offers a unified approach, allowing workloads to leverage its capabilities effectively, making it one of its most prominent features. Namely, the Operator pattern, which encapsulates the operational knowledge as code, simplifies the deployment process, improves resource utilisation, increases flexibility, and enhances the security of the applications (What Is a Kubernetes Operator?, 2022). For those reasons, applications that have Kubernetes Operator should be preferred whenever possible. Figure 4.3 illustrates the flow of the Operator within the Kubernetes cluster, which takes actions depending on the Operator logic. Then, Figure 4.4 illustrates the Operator reconciliation loop, which observes, analyses, and acts to match the current state with the desired state for a specific resource.

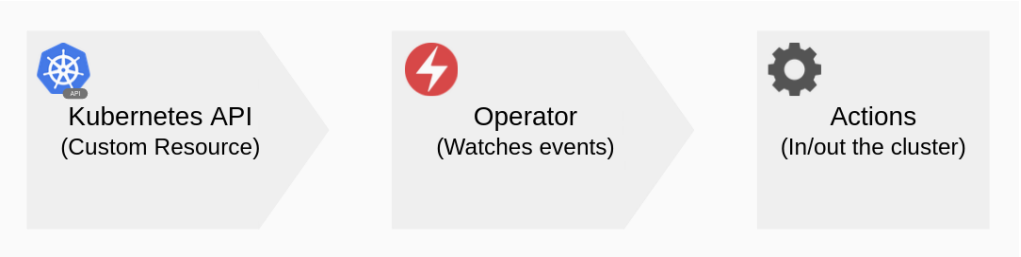


Figure 4.3: Kubernetes Operator flow which takes action inside and/or outside the cluster.

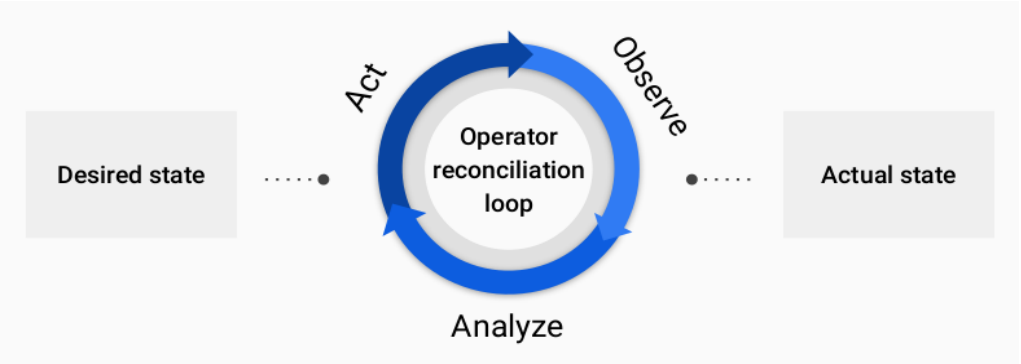


Figure 4.4: Kubernetes Operator reconciliation loop.

Another key feature of Kubernetes and its ecosystem is using declarative configuration as code. In this approach, the applications' configuration is defined as code declaratively rather than imperatively, providing many benefits like consistency, version control, automation, simplicity, and portability. Hence, it helps to streamline software development processes, increase reliability, and reduce errors and downtime. The declarative approach It is a powerful approach to managing complex systems and applications, and it is widely used in the modern software development (Declarative Management of Kubernetes Objects Using Configuration Files, 2023).

Accordingly, Kustomize, the official Kubernetes native declarative configuration management tool, will be used in this project to manage the Kubernetes manifests. Moreover, to reduce toil and repetitive work, the Kustomize built-in plugin "HelmChartInflationGenerator" will be used whenever possible to consume the upstream Helm packages known as "Helm charts". Helm is a package manager for Kubernetes that makes deploying, managing, and upgrading applications on Kubernetes clusters easier by providing a standardised

way to package and distribute them instead of creating all Kubernetes manifests from scratch to saving time and effort, avoiding reinventing the wheel. Figure 4.5 presents the core idea of Kustomuize, which provides a template-free way to customise Kubernetes manifests as layers. By convention, Kubernetes manifests are in YAML format, which is a human-friendly data serialisation language, but JSON is also accepted (Managing Resources, 2022).

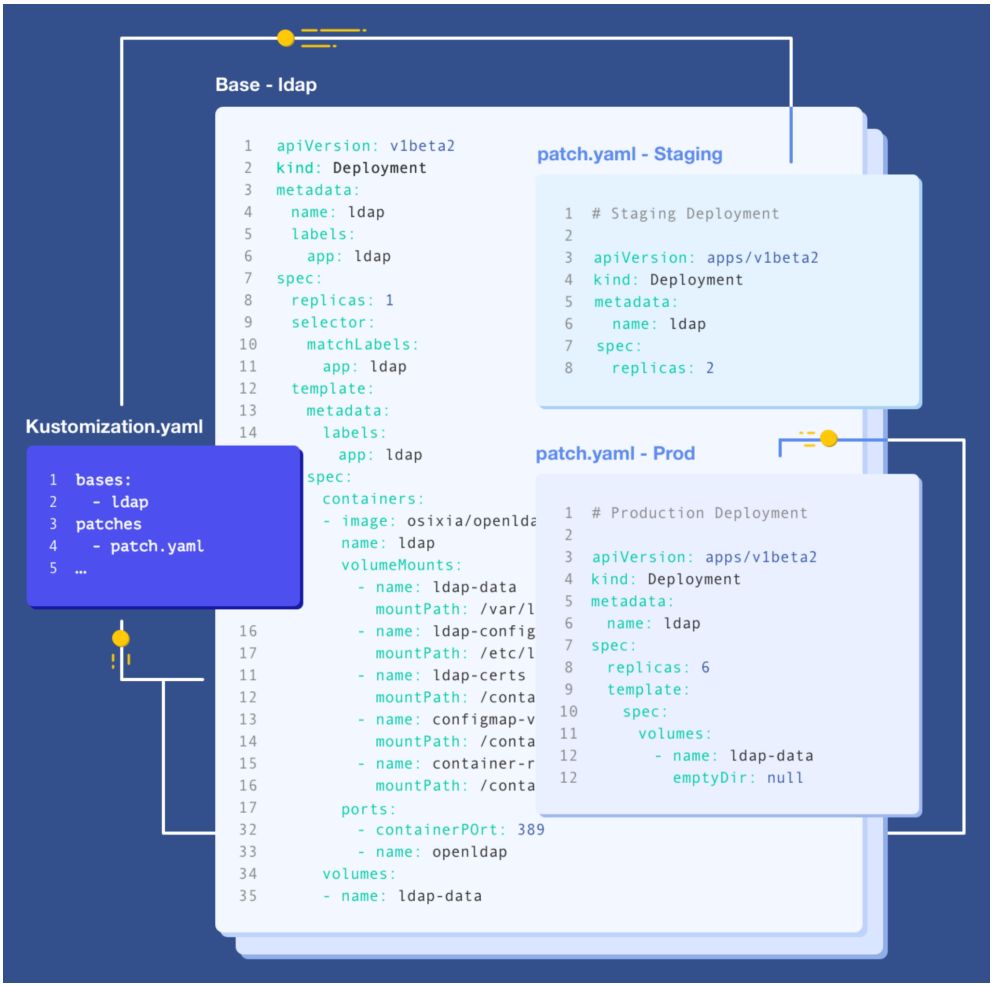


Figure 4.5: Kustomize traverses a Kubernetes manifest to add, remove or update configuration options (Kubernetes Native Configuration Management, n.d.).

After covering the infrastructure orchestration platform that matches the goals and requirements, now moving to the data platform application components starting with data ingestion.

4.2.2 Data Ingestion

In order to select a solution for ingestion and transport that matches the goals and requirements, the following inclusion and exclusion criteria were formulated:

- **IC1:** The solution provides modern ETL and workflow capabilities.
- **IC2:** The solution is Cloud-Agnostic and uses Cloud-Native approaches.
- **IC3:** The solution is customisable and fits different use cases.
- **EC1:** The solution is not easy to deploy on Kubernetes.
- **EC2:** The solution does not use a declarative style.
- **EC3:** The solution is complex or has a high learning curve.

Solutions	Inclusion criteria			Exclusion criteria		
	IC1	IC2	IC3	EC1	EC2	EC3
Apache Airflow	⚫	⚫	⚫	⚫		⚫
Argo Workflows	⚫	⚫	⚫			
Dagster	⚫	⚫	⚫			⚫
Kubeflow Pipelines	⚫	⚫	⚫		⚫	⚫
Prefect	⚫	⚫	⚫	⚫	⚫	
Tekton	⚫	⚫	⚫			⚫

Table 4.2: Comparison between different ingestion and transport solutions based on the inclusion and exclusion criteria.

According to the focus areas in the specifications section and the matching of the inclusion and exclusion criteria in Table 4.2, Argo Workflows appears to be the best fit for this project as an ingestion and transport solution. Argo Workflows is an open-source Kubernetes-native workflow engine for orchestrating parallel and distributed tasks. It provides a simple and powerful way to define, manage, and execute complex workflows, including data processing, machine learning, and continuous integration pipelines. By default, it uses the Kubernetes Operator pattern to define workflows as declarative YAML files, making it easy to follow DataOps principles. Furthermore, it supports programmatically writing workflows via its Software Development Kit (SDK) and built-in templates to define common

workflow patterns for flexibility and extensibility. Figure 4.6 illustrates Argo Workflows' full architecture.

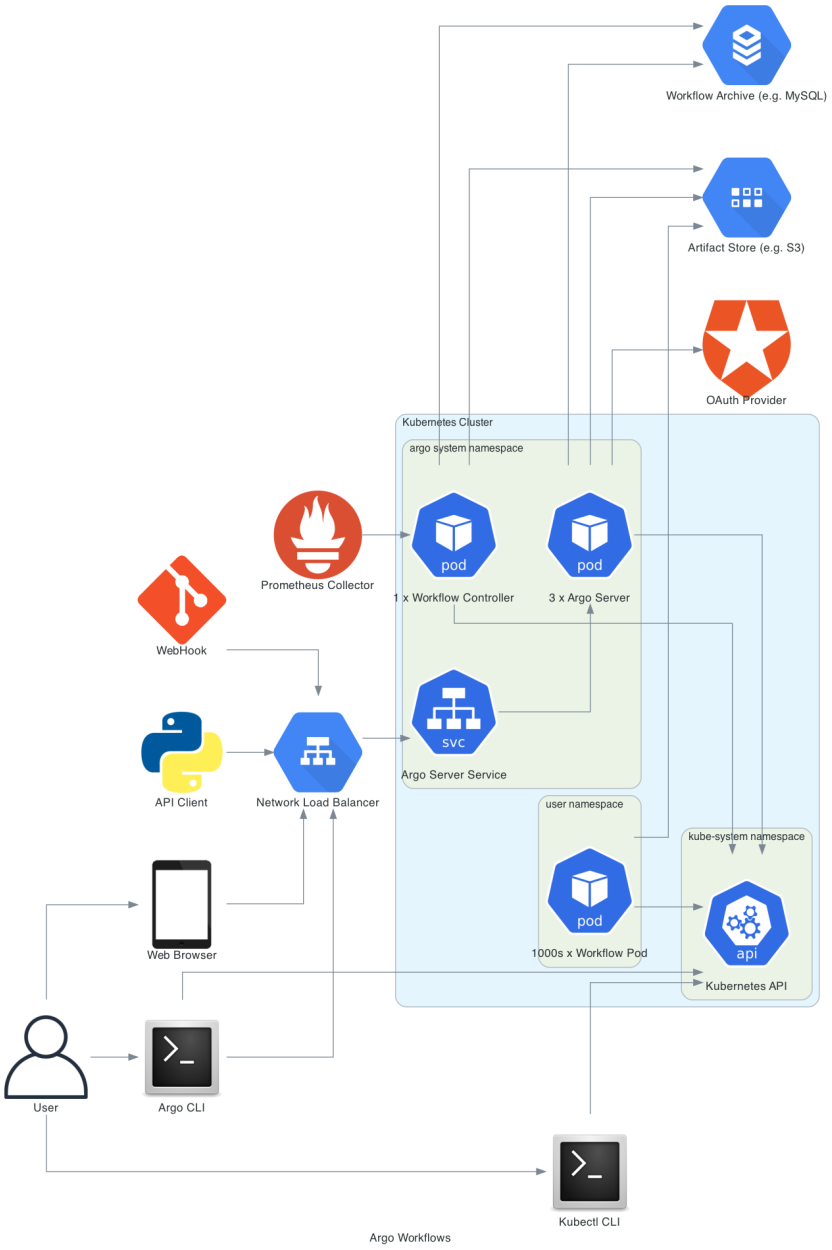


Figure 4.6: The components of the Argo Workflows architecture (Argo Workflows Architecture, n.d.).

4.2.3 Data Storage

In order to select the storage solution that matches the goals and requirements, the following inclusion and exclusion criteria were formulated:

- **IC1:** The solution provides scalable, high-performance, and distributed object storage capabilities.
- **IC2:** The solution is Cloud-Agnostic and uses Cloud-Native approaches.
- **IC3:** The solution is compatible with the standard Amazon S3 APIs.
- **EC1:** The solution is not easy to deploy on Kubernetes.
- **EC2:** The solution is not easy to manage and maintain.
- **EC3:** The solution does not have a large community and support.

Solutions	Inclusion criteria			Exclusion criteria		
	IC1	IC2	IC3	EC1	EC2	EC3
Ceph	⚫	⚫	⚫		⚫	
MinIO	⚫	⚫	⚫			
OpenIO	⚫	⚫	⚫	⚫		⚫
Riak CS	⚫	⚫	⚫	⚫		⚫
Zenko	⚫	⚫	⚫			⚫

Table 4.3: Comparison between different object storage solutions based on the inclusion and exclusion criteria.

According to the focus areas in the specifications section and the matching of the inclusion and exclusion criteria in Table 4.3, MinIO appears to be the best fit as a storage solution for this project. MinIO provides several key features, making it a popular choice for building object storage servers for Cloud-Native applications and infrastructure. Specifically, it provides a highly scalable and distributed storage system to store massive amounts of structured and unstructured data. Moreover, MinIO is Amazon S3 compatible and built on top of the Amazon S3 API, which means it can be easily integrated with various applications and tools. Figure 4.7 illustrates the architecture for MinIO on Kubernetes and multi-tenant.

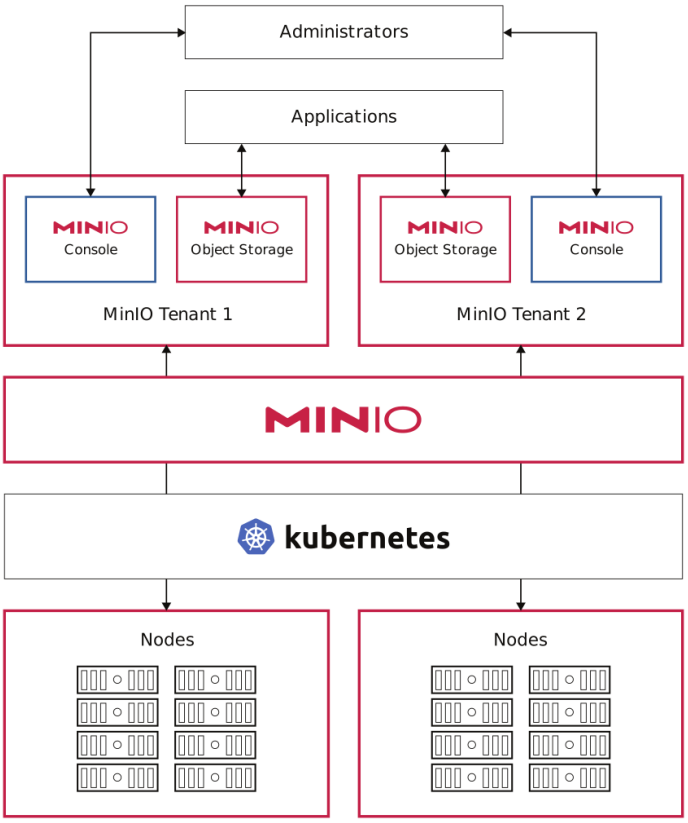


Figure 4.7: MinIO architecture on Kubernetes with multi-tenant
(MinIO High-Performance Multi-Cloud Object Storage, 2022, p. 16, adapted version).

4.2.4 Data Processing

In order to select the query and processing solution that matches the goals and requirements, the following inclusion and exclusion criteria were formulated:

- **IC1:** The solution provides query and processing capabilities.
- **IC2:** The solution is Cloud-Agnostic and uses Cloud-Native approaches.
- **IC3:** The solution has an open-source and enterprise version.
- **EC1:** The solution does not position itself as a DLH platform.
- **EC2:** The solution is not easy to deploy on Kubernetes.
- **EC3:** The solution does not support modern open standards like Apache Hudi, Apache Iceberg, and Delta Lake.

Solutions	Inclusion criteria			Exclusion criteria		
	IC1	IC2	IC3	EC1	EC2	EC3
Cloudera Data Platform	⚫	⚫	⚫		⚫	
Dremio	⚫	⚫	⚫			
Presto	⚫	⚫	⚫	⚫		
Trino	⚫	⚫	⚫	⚫		

Table 4.4: Comparison between different query and processing solutions based on the inclusion and exclusion criteria.

In line with the focus areas in the specifications section and the matching of the inclusion and exclusion criteria in Table 4.4, Dremio appears to be the best fit as a query engine solution for this project, and it will be the cornerstone of the DLH. Dremio Cloud-Native architecture build blocks are open-source technologies such as Apache Arrow, Gandiva, Apache Arrow Flight and Apache Iceberg. These technologies provide several key features for building DLH, including a high-performance query engine, self-service data access, built-in data governance and security. Both open-source and enterprise versions of Dremio are offered, with the latter option offering more features and functionalities for enterprises. For those reasons, Dremio is a popular choice for organisations looking to build a self-service DLH platform that can provide fast and efficient access to a wide range of data sources. Figures 4.8 and 4.9 show Dremio deployment and functional architecture (respectively).

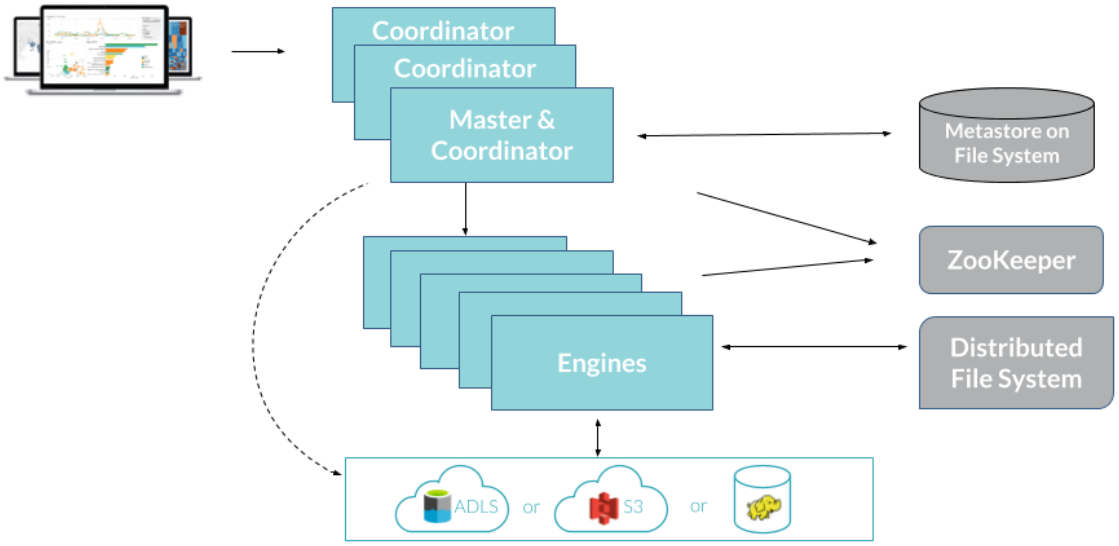


Figure 4.8: Dremio deployment architecture (Dremio Architecture Guide, 2020, p. 7).

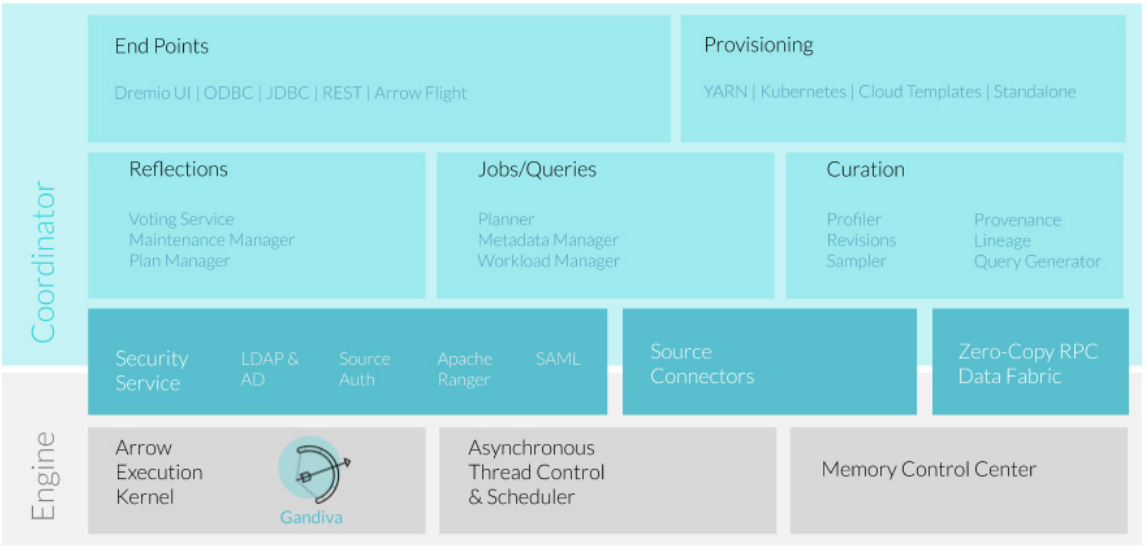


Figure 4.9: Dremio functional architecture (Dremio Architecture Guide, 2020, p. 7).

4.3 Initial Design

This section covers the initial design after defining the core components that match the goals and focus on areas traits via portability to avoid vendor lock-in, scalability to deal with massive amounts of structured and unstructured data, and extensibility by using open standards. Table 4.5 lists the items of the initial design, and Figure 4.10 illustrates the initial data platform's architecture to be implemented in the next chapter.

Item	Role	Category
DataOps	Practices framework	Process
Kubernetes	Infrastructure orchestration	Technology
MinIO	Data object storage	Technology
Dremio	Data Lakehouse platform	Technology
Argo Workflows	Data ingestion pipeline	Technology

Table 4.5: The core items of the data platform's initial architecture.

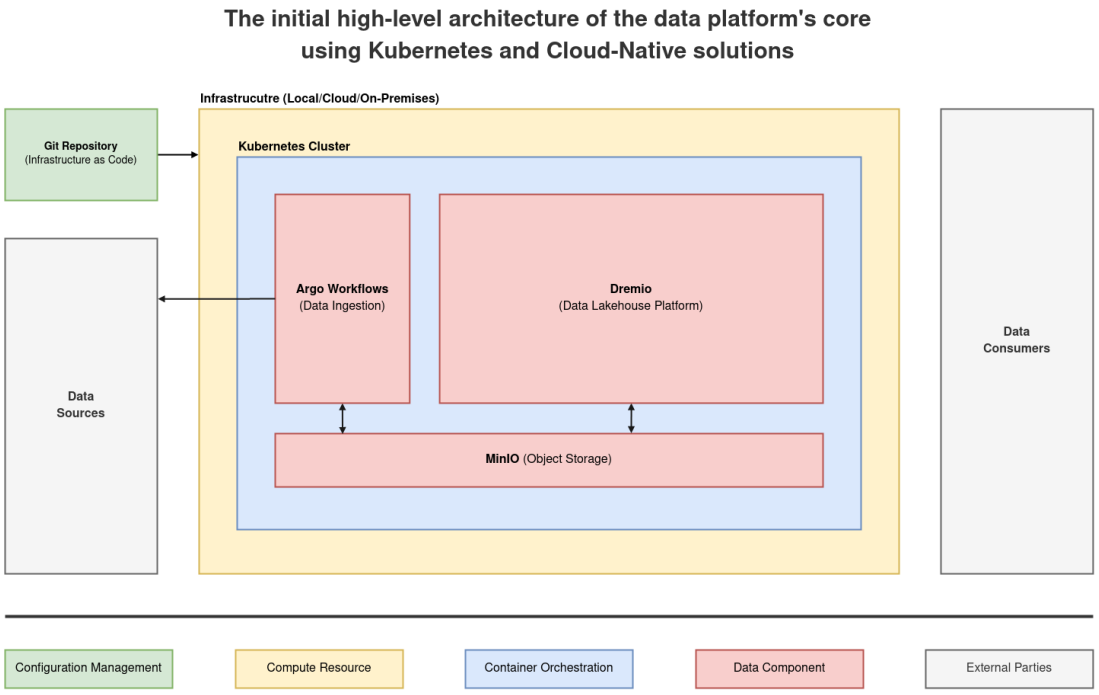


Figure 4.10: The initial high-level architecture shows the core parts of the Modern Data Platform using Kubernetes and Cloud-Native solutions.

Chapter Five:
Implementation

5.1 Approach

After defining the specifications and designing the architecture of the data platform, the next step is the implementation which follows the same approach in the previous sections. The implementation will combine MVP, Agile methodologies, and starting simple and iterating, which can help to build better solutions more efficiently. That method follows a data-driven approach by focusing on the core features needed to solve a specific problem, releasing those features quickly, and iterating in line with user feedback.

As a result, adopting iterative user-centric approaches in solution development can effectively mitigate the possibility of constructing undesired products and helps to fulfil business needs. On the technical side, this project takes a pragmatic approach to benefit from the Kubernetes ecosystem, like tools, frameworks, and resources that support deploying, managing, and scaling workloads on Kubernetes clusters. Furthermore, automation and a declarative approach will be used whenever possible while building the data platform to accelerate development.

The development will be done in a local Kubernetes cluster for faster interactions, but, at the same time, it is implemented to work seamlessly on any Kubernetes Cloud cluster, which will be used for benchmarking in the next chapter. The following sections in this chapter show the tools used to implement and develop the infrastructure as code to build and run the platform, and the Python code ingests external data into the platform. The data pipeline will use a real dataset which contains the daily subnational 14-day notification rate of new COVID-19 cases between 2020 and 2022 (Data on the Daily Subnational 14-day Notification Rate of New COVID-19 Cases, 2022).

5.2 Software

This section shows all software used in this project. As shown in the architecture chapter, the core applications are used in the platform:

- **Kubernetes:** Container orchestration platform.
- **Argo Workflows:** Data pipeline workflow engine.
- **Dremio:** Data Lakehouse platform.
- **MinIO:** Object storage.

For the tools used in the data platform development, the standard Kubernetes tools will be used as follows:

- **Docker:** The system that simulates Kubernetes nodes for local development.
- **KinD:** The tool to run local Kubernetes clusters using Docker containers as nodes, hence getting its name “Kubernetes in Docker”, which will be used for testing and local development.
- **Kubectl:** The official Kubernetes command line tool for communicating with a Kubernetes cluster’s control plane using the Kubernetes API. It will be used to deploy the developed Kubernetes manifests.
- **Kustomize:** The official tool for declarative management of Kubernetes objects, which introduces a template-free way to customise application configuration that simplifies the use of off-the-shelf applications. It will be used to manage the developed Kubernetes manifests.
- **Helm:** A package manager for Kubernetes that makes deploying, managing, and upgrading applications on Kubernetes clusters easier by providing a standardised way to package and distribute them. It will be used in conjunction with Kustomize.
- **GitHub:** A cloud-based Git repository that will be used to host the developed code for version control, and also it works as a source of truth. It is worth mentioning that, at this phase, this project does not rely on any specific features of GitHub, which means any Git service could be used.

Tables 5.1 and 5.2 show (respectively) the version of applications and tools used in the implementation.

Application	Version	Notes
Argo Workflows	3.4.5	
Dremio	24.0.0	
Kubernetes	1.24.7	
MinIO	RELEASE.2023-02-10T18-48-39Z	

Table 5.1: Data applications versions used in the project.

Tool	Version	Notes
Docker	23.0.1	
Helm	3.11.1	
KinD	0.17.0	
Kubectl	1.24.10	
Kustomize	5.0.0	

Table 5.2: Development tools versions used in the project.

5.3 Development

The entry point for this project is the infrastructure where Kubernetes will be used to manage the data applications. Then, create the deployment for the platform applications MinIO, Arog Workflow, and Dremio. Finally, develop a data pipeline that ingests external data into the platform.

5.3.1 Kubernetes Cluster

For development, a local Kubernetes cluster will be created using KinD. By default, Kubernetes services are only accessible internally within the cluster; hence, an Ingress-NGINX will be installed to access services externally. Further, to make the setup as close as possible to the production, the “local.gd” domain will be used as a replacement for real domains where it and all its subdomains like

“*.local.gd” always resolve to localhost (127.0.0.1) without any change on the local side. Figure 5.1 shows the KinD configuration file to create a local Kubernetes cluster with two nodes and the customisations to make it easy to run Ingress-NGINX to access the cluster externally. Afterwards, Ingress-NGINX will be installed using its official upstream Helm chart using Kustomize to have it as a declarative configuration as code.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4

name: modern-data-platform

networking:
  apiServerAddress: "127.0.0.1"
  apiServerPort: 6443

nodes:
- role: control-plane
  image:
kindest/node:v1.24.7@sha256:577c630ce8e509131eab1aea12c022190978dd2f745aac5e
b1fe65c0807eb315
- role: worker
  image:
kindest/node:v1.24.7@sha256:577c630ce8e509131eab1aea12c022190978dd2f745aac5e
b1fe65c0807eb315
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    listenAddress: "0.0.0.0"
  - containerPort: 443
    hostPort: 443
    listenAddress: "0.0.0.0"
```

Figure 5.1: KinD configuration for local Kubernetes cluster.

5.3.2 Data Applications

Using the same notion, the data applications MinIO, Argo Workflows, and Dremio will be installed by upstream Helm charts using Kustomize. For example, Figure 5.2 shows the Kustomize directory structure of MinIO, and Figure 5.3 shows the content of each file in the MinIO directory. The rest of the components follow the same style; the full code is available in the project Git repository.

```
minio
├── helm-chart.yaml
├── kustomization.yaml
└── namespace.yaml
```

Figure 5.2: The Kustomize directory structure of the MinIO application.

```yaml
# kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: minio
resources:
- namespace.yaml
generators:
- helm-chart.yaml
---
# namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: minio
---
# helm-chart.yaml
apiVersion: builtin
kind: HelmChartInflationGenerator
metadata:
  name: minio
name: minio
repo: https://charts.bitnami.com/bitnami
version: 12.1.7
releaseName: minio
namespace: minio
valuesInline:
  ingress:
    enabled: true
    hostname: ui.minio.local.gd
    ingressClassName: nginx
  defaultBuckets: "datalake,dremio"
  provisioning:
    enabled: true
    users:
    - disabled: false
      username: datalake
      password: datalake_poc
      policies:
      - readwrite
```

Figure 5.3: Kustomize files to deploy MinIO application.

5.3.3 Data Pipeline

The initial version of the platform has a simple data pipeline to ingest external data into MinIO to make it available for Dremio to query. The dataset belongs to the European Centre for Disease Prevention and Control (ECDC), which covers the daily subnational 14-day notification rate of new COVID-19 cases. The dataset has 454928 records, each representing a daily report from a region in a certain country between 2020 and 2022 (Data on the Daily Subnational 14-day Notification Rate of New COVID-19 Cases, 2022). The data pipeline leverages the declarative capabilities of Argo Workflows to download the dataset in JSON format, then runs a Python code which will split the data based on the reporting year and creates a file per year. Finally, the workflow will upload the generated artefacts automatically to MinIO. Figure 5.4 shows the full data pipeline.

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: covid19-subnational-data
  generateName: data-ingestion-
spec:
  entrypoint: get-ecdc-data
  templates:
  - name: get-ecdc-data
  steps:
  - - name: get-ecdc-data
      template: extract-and-load-into-datalake
      arguments:
        artifacts:
        - name: ecdc_covid19_subnational_case_daily
          http:
            url: "https://opendata.ecdc.europa.eu/covid19/subnationalcasedaily/json/"
        parameters:
        - name: year
          value: "{{ item }}"
      withItems:
      - 2020
      - 2021
      - 2022
  - name: extract-and-load-into-datalake
  inputs:
    artifacts:
    - name: ecdc_covid19_subnational_case_daily
      path: /tmp/ecdc_covid19_subnational_case_daily.json
    parameters:
    - name: year
```

```
  script:
    image: python:3.8-alpine
    command: [python]
    env:
    - name: YEAR
      value: "{{ inputs.parameters.year }}"
    - name: INPUT_FILE
      value: "/tmp/ecdc_covid19_subnational_case_daily.json"
    - name: OUTPUT_FILE
      value: "/tmp/ecdc_covid19_subnational_case_daily_{{ inputs.parameters.year
}}.json"
    source: |
      import json
      import os
      import random
      # Vars.
      year = os.getenv('YEAR')
      input_file = os.getenv('INPUT_FILE', 'input.json')
      output_file = os.getenv('OUTPUT_FILE', f'output_{year}.json')
      # Load.
      print('[INFO] Read input file {input_file} ...')
      with open(input_file, 'r') as f:
          data = json.loads(f.read())
      # Extract.
      print(f'[INFO] Filter data by year {year} ...')
      filtered_by_year = [item for item in data if item['date'].startswith(year)]
      # Verify.
      print('[INFO] Sample:', random.choice(filtered_by_year))
      # Save.
      print(f'[INFO] Write data for year {year} into {output_file} ...')
      with open(output_file, 'w') as f:
          f.write(json.dumps(filtered_by_year, indent=2))
      print('[INFO] Done')
  outputs:
    artifacts:
    - name: ecdc_covid19_subnational_case_daily_year
      path: /tmp/ecdc_covid19_subnational_case_daily_{{ inputs.parameters.year
}}.json
      s3:
        key: /ecdc/covid19/world_wide_subnational_case_daily_{{
inputs.parameters.year }}.json
      archive:
        none: {}
```

Figure 5.4: Data ingestion pipeline using Argo Workflows and Python.

After executing the data pipeline, which will be covered in the next chapter, it will produce three JSON files, a file for each year 2020, 2021, and 2022. Afterwards, Dremio will be used to create an Apache Iceberg table for the JSON file, which will unlock Dremio query capabilities. Figure 5.5 shows the Dremio query to create an Apache Iceberg table for the JSON file.

```
CREATE TABLE "world_wide_subnational_case_daily_2020" AS (
  SELECT *
  FROM datalake.ecdc.covid19."world_wide_subnational_case_daily_2020.json"
)
```

Figure 5.5: Dremio query to create an Apache Iceberg table for JSON file.

5.4 Initial Model

Based on the output of the previous sections, the implementation of the platform's core is done, which includes the Kubernetes cluster, data applications (MinIO, Argo Workflows, and Dremio), and finally, data ingestion pipeline workflow by Argo in addition to Python. All resources are managed declaratively using Kustomize and stored in a Git repository. Figure 5.6 shows the final structure of the Git repository after implementing all core parts of the platform.

```
modern-data-platform-poc
├── applications
│   ├── argo-workflows
│   │   ├── helm-chart.yaml
│   │   ├── kustomization.yaml
│   │   ├── namespace.yaml
│   │   ├── rbac.yaml
│   │   └── secret.yaml
│   ├── dremio
│   │   ├── helm-chart.yaml
│   │   ├── ingress.yaml
│   │   ├── kustomization.yaml
│   │   └── namespace.yaml
│   ├── ingress-nginx
│   │   ├── helm-chart.yaml
│   │   ├── kustomization.yaml
│   │   └── namespace.yaml
│   └── minio
│       ├── helm-chart.yaml
│       ├── kustomization.yaml
│       └── namespace.yaml
├── clusters
│   └── local
│       ├── kind-cluster-config.yaml
│       └── kustomization.yaml
└── pipelines
    └── ingestion
        └── argo-workflow-covid19-subnational-data.yaml
```

Figure 5.6: The content of the platform Git repository in a tree-like view.

To visualise the interactions of the current implementation, the C4 model will be used for that purpose. The C4 software architecture model (Context, Containers, Components, and Code) is a visual approach to describe the architecture of a software system developed by Simon Brown in 2011. It provides a set of hierarchical diagrams that describe the system's different levels of abstraction, from high-level system context diagrams to low-level code diagrams. The model consists of four diagram levels (Brown, n.d.):

- Context Diagram: The first level that shows the system in its environment, including its external dependencies, users, and other systems that it interacts with them.
- Container Diagram: The second level that shows the internal structure of the top-level system and its components and their relationships. These containers represent the major technology components or subsystems like applications and data stores. It is worth mentioning that the term "container" refers to the actual linguistic meaning, not Docker container.
- Component Diagram: The third level that shows the internal components and their relationships within each container, providing a more detailed view of the system's architecture. In programming jargon, it is like a collection of classes related to specific functionality.
- Code Diagram: The fourth level diagram that shows a view of the component implementation in detail, like classes, interfaces, objects, functions, and database tables using UML class or entity relationship diagrams. In most cases, this level of detail is not recommended to be diagrammed because it changes frequently, and most integrated development environments can generate this kind of diagram on demand.

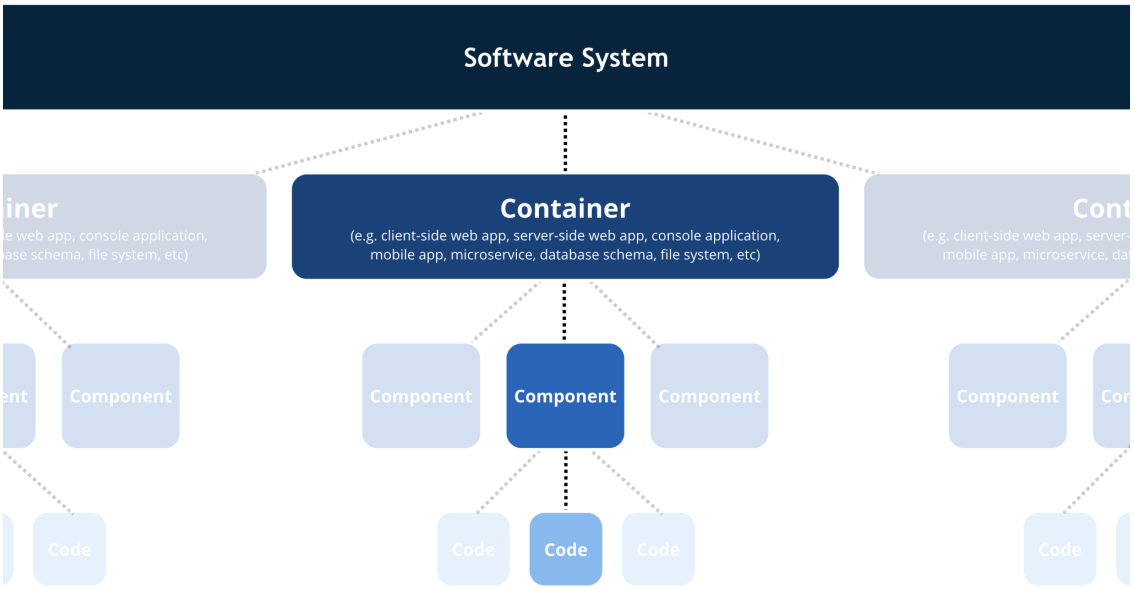


Figure 5.7: The hierarchical abstractions of the C4 model (Brown, 2023, sec. Static structure).

Figure 5.7 shows an abstract view of the C4 model, which usually creates a diagram per level. However, to avoid complexity since the project covers only the core parts of the data platform, a single-layered diagram was created to visualise the current implementation. Figure 5.8 visualises the platform architecture context, containers, and components (a simplified view where all the abstractions are combined together), which will be used in the evaluation chapter to verify and validate the implementation.

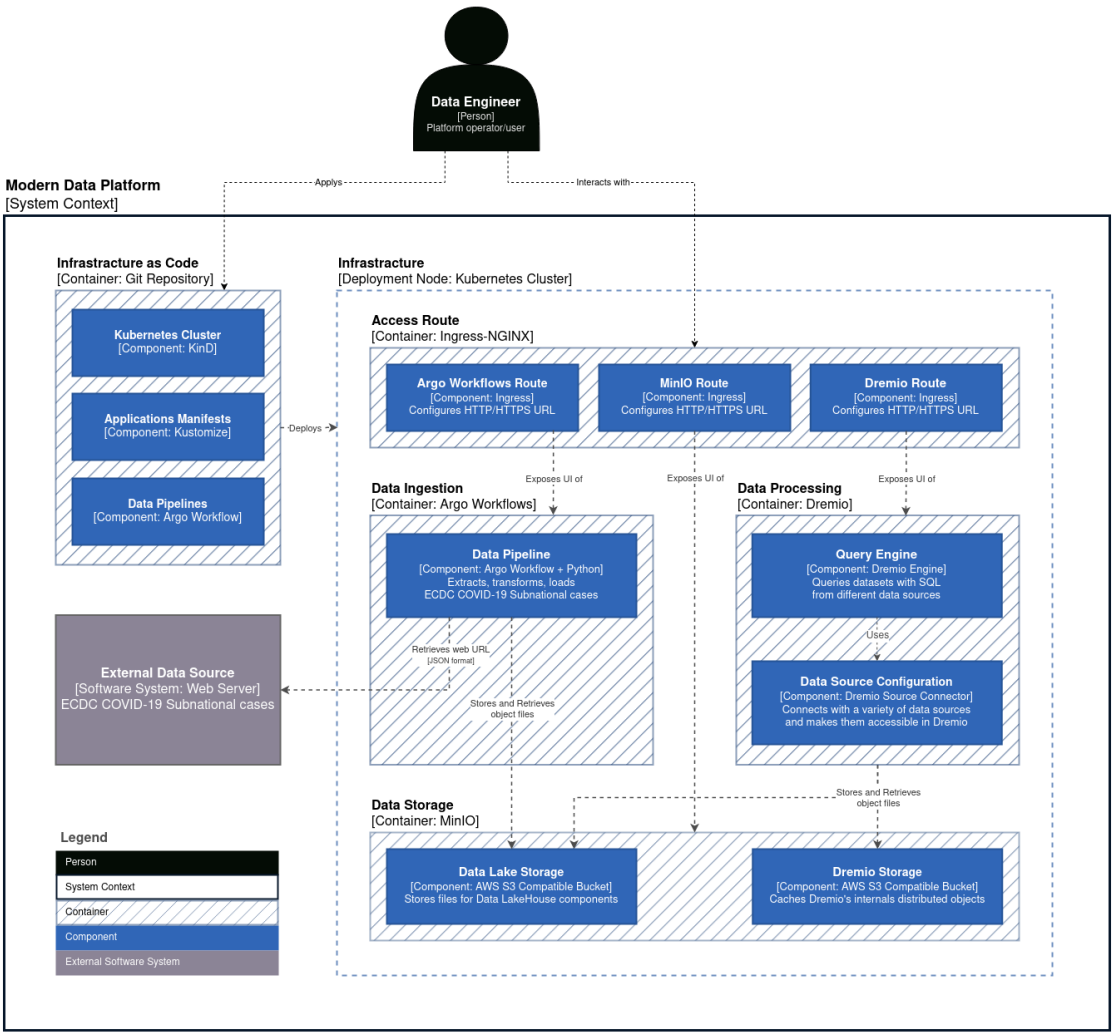


Figure 5.8: The data platform's initial model interactions following the C4 model guidelines.

The following scenarios should be conducted in the technical evaluation chapter:

- Verify the implementation against the initial architecture.
- Verify the implementation against the focus areas.
- Deploy the local Kubernetes cluster with Ingress.
- Deploy the data applications.
- Access the data application via their UI.
- Apply the data pipeline to ingest external data from an external source.
- Verify the creation of ingested data by the data pipeline.
- Query the ingested data using Dremio in SQL-like syntax.
- Create an open table like Apache Iceberg for the ingested data.
- Benchmark the query performance of the platform.

Chapter Six:
Technical Evaluation

6.1 Overview

After finishing the platform specifications, focus areas, architecture, and implementation, the technical evaluation section plays a crucial role in matching the targets and actual implementation. For that purpose, standard evaluation practices, namely verification and validation, are two essential activities in the software development life cycle to ensure that software meets the required quality standards. Verification is the process of checking whether a software product meets the specifications and requirements of the design. At the same time, validation is the process of checking whether the software product meets the needs and expectations of the customer (Fisher, 2007, pp. 3–5). Afterwards, a benchmark will be executed to assess Dremio's performance with real-world scenarios.

6.2 Verification

This section is used to assert that the core components of the platform have been implemented in line with the targeted architecture Unified Data Infrastructure v2.0, which are Infrastructure (Kubernetes), Ingestion and Transport (Argo Workflows), Storage (MinIO), and finally, Query and Processing (Dremio). Table 6.1 verifies the implementation against the focus areas in the manner of the MoSCoW prioritisation method defined in section 3.3 Focus Areas. Figure 6.1 shows the implemented components in agreement with the focus areas that provide the Data Lakehouse's minimal functionality.

Priority	Implementation
Must Haves	
• **MH1:** Cloud-Native architecture. • **MH2:** Scalable and Cloud agnostic infrastructure orchestration system. • **MH3:** Open-source software and open standard formats. • **MH4:** Data Lakehouse solution as a core of the data platform.	All of the Must-haves targets have been implemented where the initial implementation uses the Cloud-Native approach using modern applications and tools. In addition, Kubernetes, as an orchestration platform, provides scalability and portability. Finally, using Dremio and open standard formats implements the core of the Data Lakehouse platform.
Should Haves	
• **SH1:** A declarative approach for configuration management. • **SH2:** A data pipeline to ingest data from an external source into the platform in plain text formats like JSON or CSV. • **SH3:** Self-service capabilities.	All of the Should-haves targets have been implemented, where Kustomize is used to manage all Kubernetes resources in a declarative manner. Furthermore, using Argo Workflows, a data pipeline was created to ingest data from an external source and load it into the data lake storage solution. Finally, Dremio provides Self-service capabilities out of the box.
Could Haves	
• **CH1:** An easy way to rebuild the system with minimal manual actions. • **CH2:** An open format table like Apache Iceberg from one of the ingested JSON/CSV files.	All of the Could-haves targets have been implemented by using a declarative approach as well as open standard formats. Whereas all application configurations are managed as code in a Git repository which works as a source of truth to track the system changes.
Would Haves	
• **WH1:** Production grade quality like high availability, security hardening, or data quality validation. • **WH2:** The reset of the components not directly related to the core of the Data Lakehouse.	The Would-haves (also known as “will not haves”) are not implemented in the current iteration as planned and could be implemented in the next interactions.

Table 6.1: Verify the implementation against the focus areas using the MoSCoW prioritisation method.

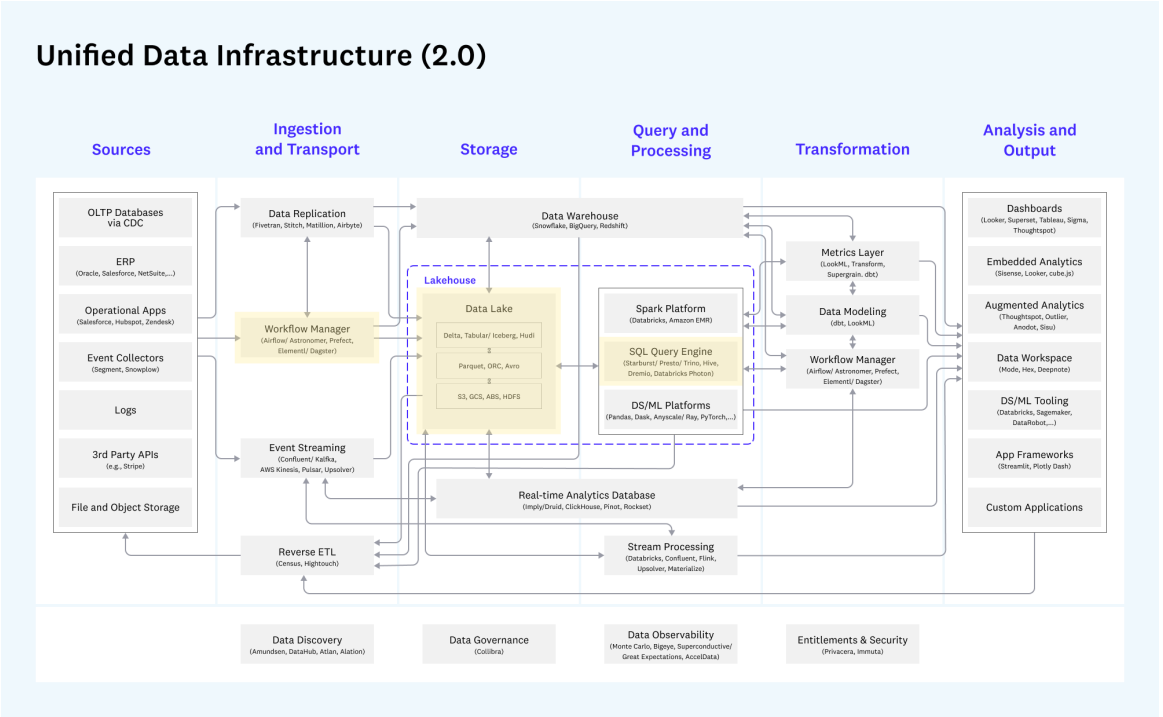


Figure 6.1: The initial implementation of the UDI v2.0 architecture.

6.3 Validation

To validate the initial design implementation, the platform will be deployed, accessed, and used the same way the end personas could use it. Hence, the next sections will perform various actions related to different components of the platform (Kubernetes cluster, data applications, and data pipeline) to ensure that the platform meets the needs and expectations of the end users. The code and configuration used in this section come from the project Git repository "modern-data-platform-poc"[2], which is accessible publicly.

6.3.1 Prerequisites

All the tools used during the implementation phase should be installed to run the validation steps. To facilitate that task, it is suggested to use "asdf"[3], a universal version manager, to install the tools which are added with their corresponding versions in the ".tool-versions" file.

[2] https://github.com/aabouzaid/modern-data-platform-poc
[3] https://asdf-vm.com

Consequently, Figure 6.2 shows how to install the pre-required tools using “asdf”, tested on the Linux operating system (Ubuntu 20.04.5 LTS with Docker Engine Community 23.0.1).

```
// Go to the directory of the project.
$ cd modern-data-platform-poc

// Install the pre-required tools.
$ asdf install

// Validate the installation.
$ asdf current
Output:
Helm          3.11.1            modern-data-platform-poc/.tool-versions
Kind          0.17.0            modern-data-platform-poc/.tool-versions
Kubectl       1.24.10           modern-data-platform-poc/.tool-versions
Kustomize     5.0.0             modern-data-platform-poc/.tool-versions
```

Figure 6.2: Install validation prerequisites tools.

6.3.2 Deployment

Since the platform uses Kubernetes as a container orchestration system to manage workloads, any Kubernetes cluster should work out of the box. However, for demonstration purposes, a local Kubernetes cluster will be used via KinD, as mentioned in the implementation chapter. Figure 6.3 shows the “kind” CLI configuration to create a Kubernetes cluster locally and then validate accessibility to the cluster using the “kubectl” tool.

```
// Create the Kubernetes cluster.
$ kind create cluster \
  --config clusters/local/kind-cluster-config.yaml

// Set context to the new Kubernetes cluster.
kubectl config set-context kind-modern-data-platform \
  --cluster kind-modern-data-platform \
  --user kind-modern-data-platform
Output:
Context "kind-modern-data-platform" modified.
```

```
// Validate the connection to the Kubernetes cluster.
$ kubectl version --output=yaml
Output:
clientVersion:
  buildDate: "2023-01-18T19:15:31Z"
  compiler: gc
  gitCommit: 5c1d2d4295f9b4eb12bfbf6429fdf989f2ca8a02
  gitTreeState: clean
  gitVersion: v1.24.10
  goVersion: go1.19.5
  major: "1"
  minor: "24"
  platform: linux/amd64
serverVersion:
  buildDate: "2022-10-26T15:06:14Z"
  compiler: gc
  gitCommit: e6f35974b08862a23e7f4aad8e5d7f7f2de26c15
  gitTreeState: clean
  gitVersion: v1.24.7
  goVersion: go1.18.7
  major: "1"
  minor: "24"
  platform: linux/amd64
```

Figure 6.3: Deploying local Kubernetes cluster and verifying the access to it.

The next step is using Kustomize and Kubectl to deploy the MinIO, Argo Workflows, Dremio, and Ingress-NGINX applications. Figure 6.4 shows the steps. This task could take a couple of minutes, depending on the Internet speed of the host machine. For a detailed view of the deployed Kubernetes objects, check Figures B.1, B.2, B.3 and in Appendix B.

```
// Deploy the applications to the Kubernetes cluster.
$ kustomize build --enable-helm clusters/local | kubectl apply -f -

// Next, verify that deployment is done.

// 1) Ingress-Nginx.
$ kubectl rollout status deployment \
  --watch --namespace ingress-nginx ingress-nginx-controller
Output:
Waiting for deployment "ingress-nginx-controller" rollout to finish: 0 of 1
updated replicas are available...
deployment "ingress-nginx-controller" successfully rolled out
```

```
// 2) MinIO.
$ kubectl rollout status deployment \
  --watch --namespace minio minio
Output:
Waiting for deployment "minio" rollout to finish: 0 of 1 updated replicas
are available...
deployment "minio" successfully rolled out

// 3) Argo Workflows.
$ kubectl rollout status deployment \
  --watch --namespace argo-workflows argo-workflows-server
Output:
Waiting for deployment "argo-workflows-server" rollout to finish: 0 of 1
updated replicas are available...
deployment "argo-workflows-server" successfully rolled out

// 4) Dremio.
$ kubectl rollout status statefulset \
  --watch --namespace dremio dremio-master
Output:
Waiting for 1 pods to be ready...
partitioned roll out complete: 1 new pods have been updated...
```

Figure 6.4: Install the applications into the Kubernetes cluster.

The final step is to ensure that the applications are accessible. Figure 6.5 shows the URL of each application. For the interface for each application accessed from the host machine, see Figures B.4, B.5, and B.6 in Appendix B.

```
// Get the URL for all applications.
$ kubectl get ingress --all-namespaces

Output:
NAMESPACE      NAME                 CLASS  HOSTS               ADDRESS        PORTS   AGE
argo-workflows argo-workflows-server nginx  ui.argo.local.gd    10.96.209.185  80      3m34s
dremio         dremio                nginx  ui.dremio.local.gd  10.96.209.185  80      3m34s
minio          minio                 nginx  ui.minio.local.gd   10.96.209.185  80      3m34s
```

Figure 6.5: Install the applications into the Kubernetes cluster.

6.3.3 Data Ingestion

This section covers applying the data pipeline to ingest data from an external source into the platform in JSON format, namely, ECDC COVID-19 dataset cases between the years 2020 and 2022. The data pipeline will download the dataset, split it per year, and save a file for each year into MinIO. Figure 6.6 shows how to apply the data pipeline, then Figures 6.7 and 6.8 show the pipeline in the Argo Workflows and MinIO interface.

```
// Apply the data pipeline to the Kubernetes cluster.
$ kubectl apply --namespace argo-workflows --filename \
  pipelines/ingestion/argo-workflow-covid19-subnational-data.yaml
Output:
workflow.argoproj.io/covid19-subnational-data created
```

Figure 6.6: Applying Argo Workflow data pipeline.

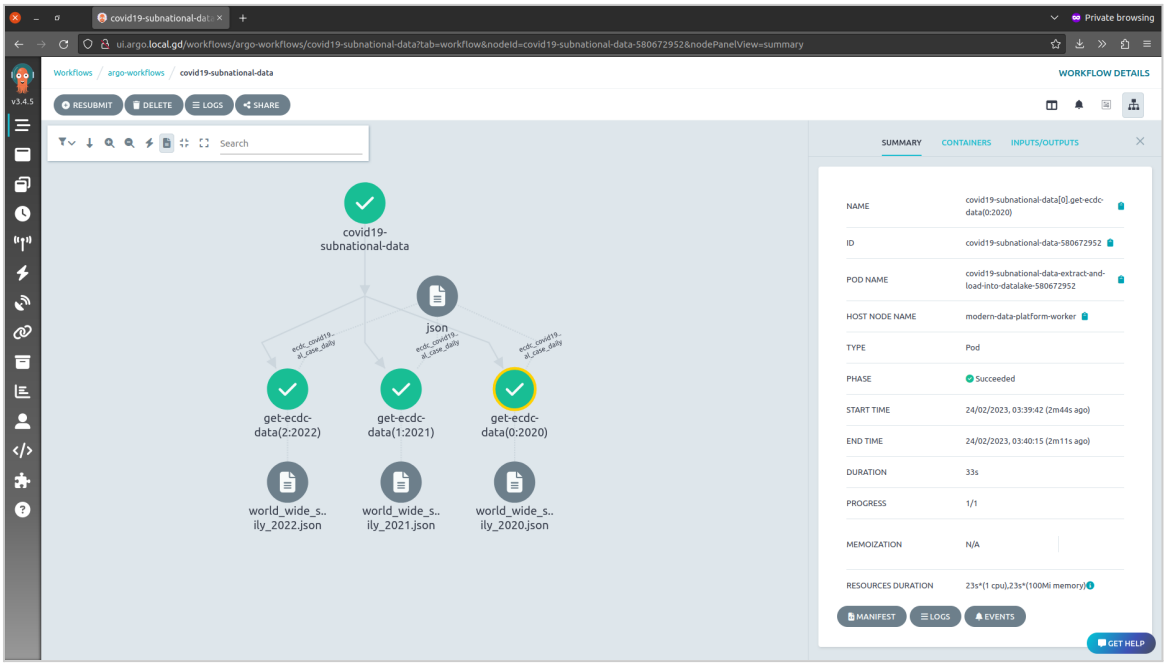


Figure 6.7: The data pipeline as shown in the Argo Workflows interface.

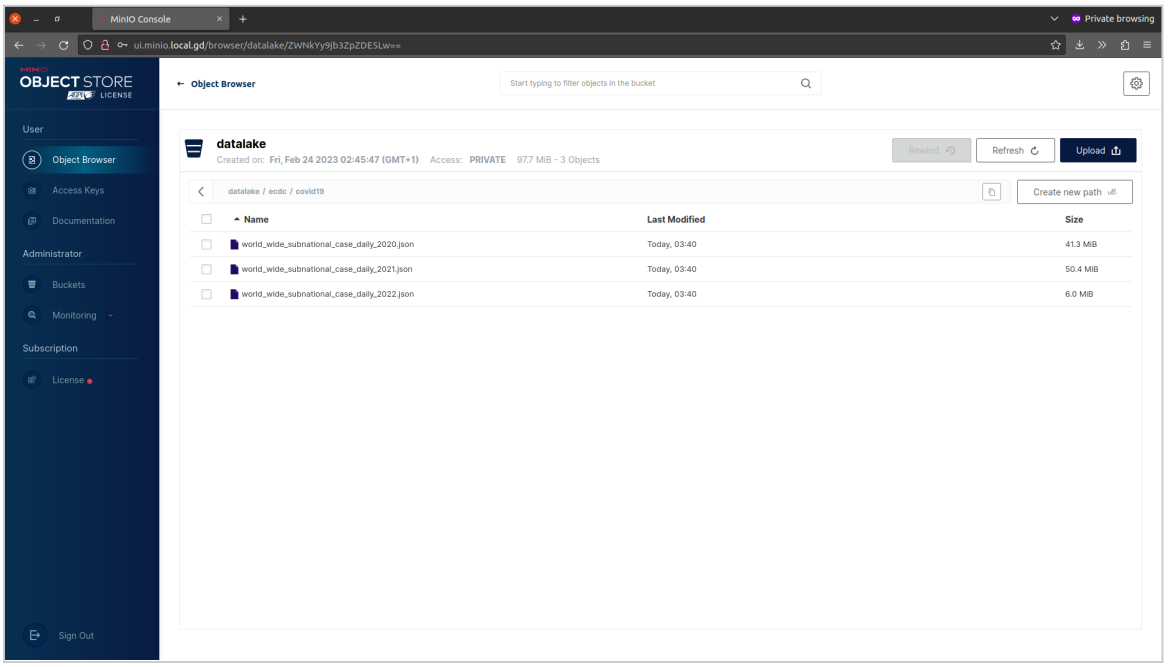


Figure 6.8: The data pipeline transformed JSON files as shown in the MinIO interface.

6.3.4 Data Processing

The last step is to validate Dremio interactions like data querying from and saving to MinIO. Dremio supports operations on plain text formats like JSON and CSV; however, many of Dremio's capabilities depend on table formats like Apache Iceberg, which in fact, does not copy the data but generates metadata for the existing plain text files. As a result, the following section covers connecting Dremio to MinIO, querying the data pipeline's transformed JSON files, and finally, creating an Iceberg table for one of the files as a proof of concept. The Dremio user interface will be used for the validation. MinIO was added as an S3-compatible plugin following Dremio documentation. See Figures B.7 and B.8 in Appendix B for details. After adding MinIO as a data source, Figures 6.9, 6.10, and 6.11 show the interactions with the ingested data JSON files. Finally, Figures 6.12 and 6.13 show the creation of an Apache Iceberg table for one of the JSON files stored on MinIO.

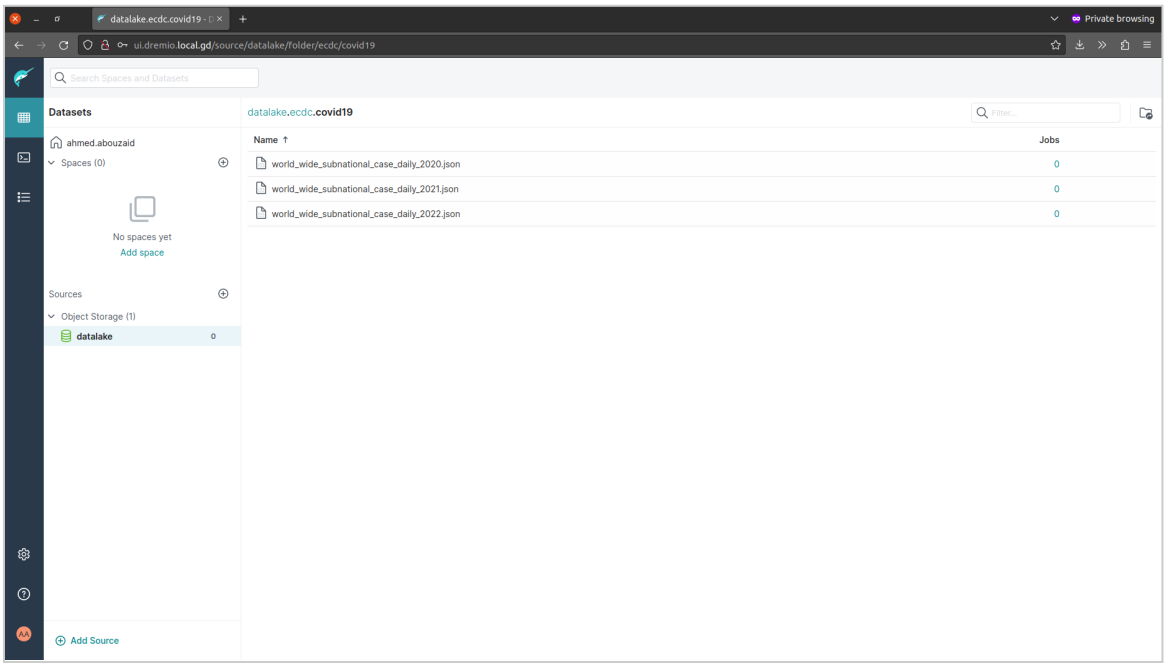


Figure 6.9: View the ingested data as plain JSON files in Dremio.

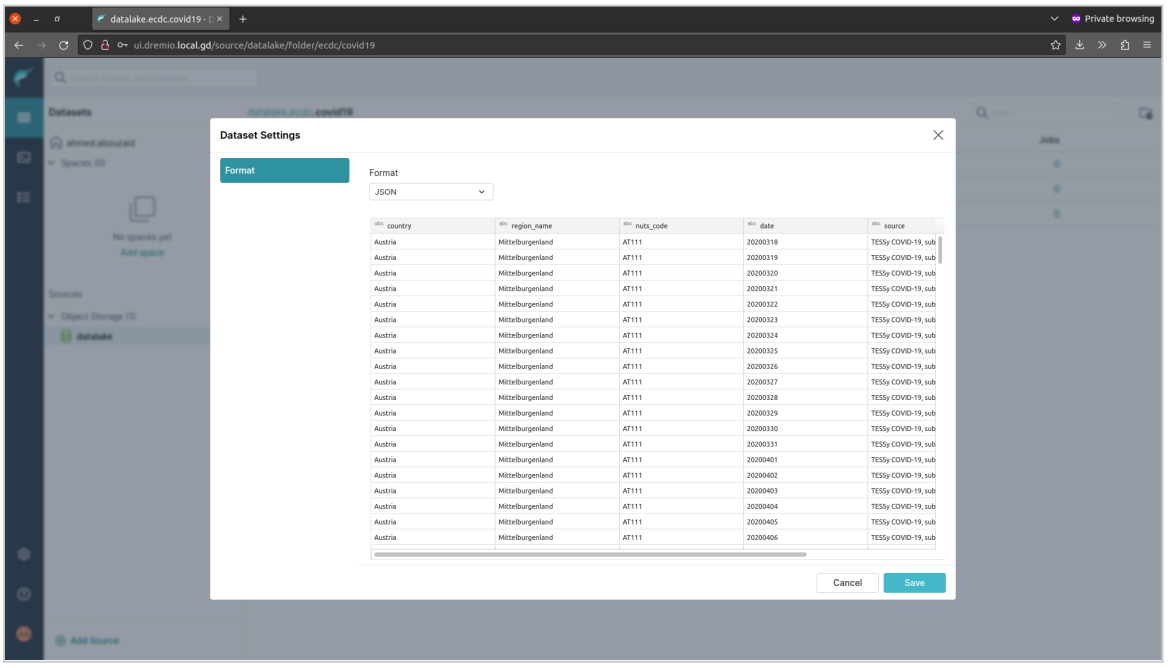


Figure 6.10: Import the ingested data as plain JSON files in Dremio.

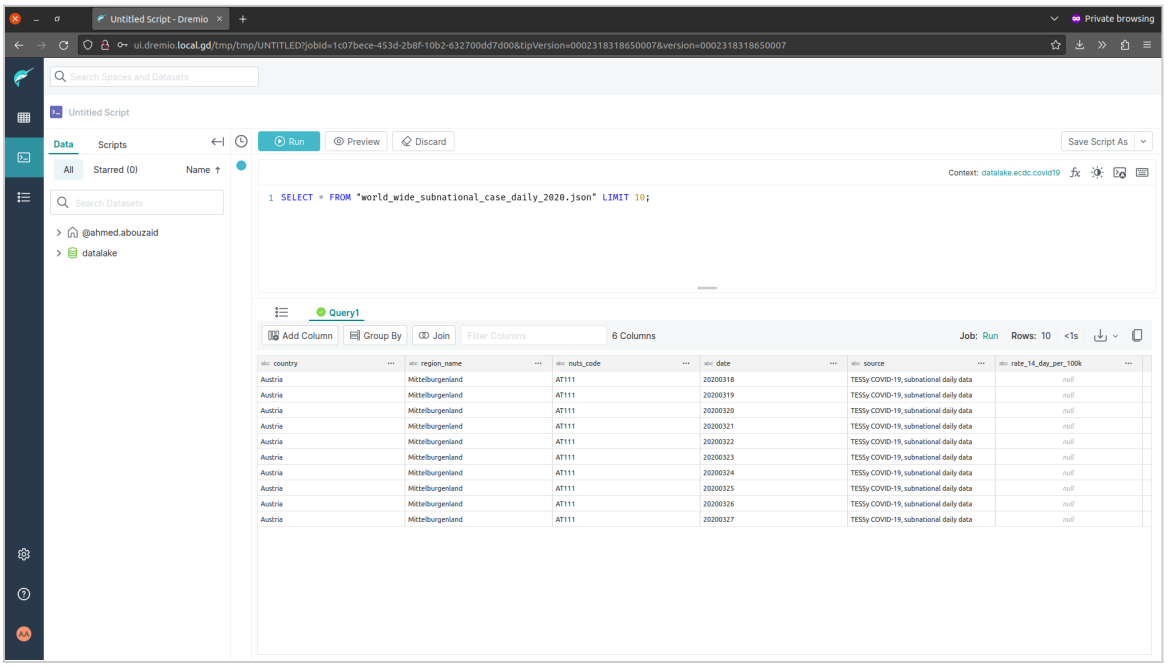


Figure 6.11: Query the ingested data as plain JSON files in Dremio.

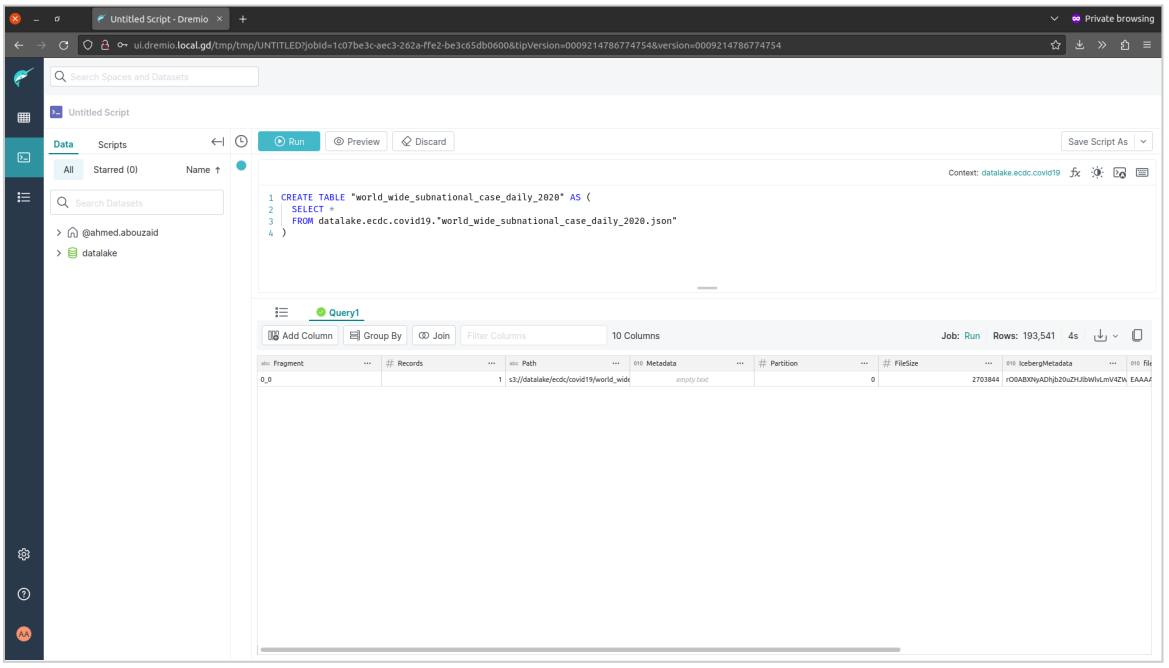


Figure 6.12: Creating an Apache Iceberg table for one of the JSON files In Dremio.

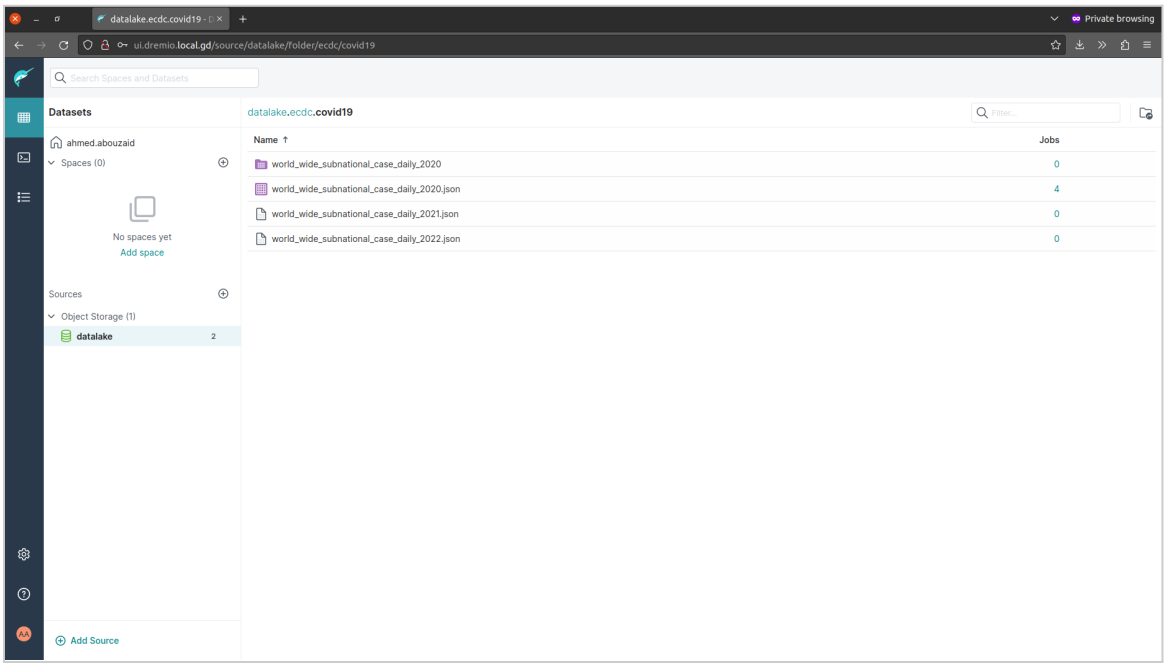


Figure 6.13: View the created Apache Iceberg table as stored on MinIO.

After validating all platform functionalities based on section 5.4 Initial Model, the next chapter focus on evaluating the platform's query performance.

Chapter Seven:
Benchmarking

7.1 Overview

Performance assessment is an essential part of the evaluation to verify the capabilities of any data solution. Benchmarking against data applications typically involves creating a standardised test environment and running a series of tests against the system using a set of predefined workloads and performance metrics. Therefore, the main goal of this section is to review the platform performance, namely, default Dremio caching capabilities like Columnar Cloud Cache. Turning now to the benchmarking process, starting with the framework used for the benchmarking, then preparation of the test environment and data generation, then the execution of the tests, and finally, the benchmark results.

7.2 Framework

The industry-standard benchmark suite, Transaction Processing Performance Performance Council for Decision Support (TPC-DS), will be used for the benchmarking purpose. The suite is designed to measure the performance of decision support systems (DSS) that examine large volumes of data and uses a complex schema and model data with various queries that simulate business intelligence queries that might be performed in a real-world scenario. Also, it includes a wide range of query complexities, data sizes, and data types, making it a robust measure of DSS performance (Poess et al., 2007, pp. 1138–1139). The TPC-DS tests include ninety-nine queries in four broad groups that characterise most decision support queries: reporting, ad-hoc, iterative OLAP, and data mining (Poess et al., 2007, p. 1140). Figure 7.1 shows the TPC-DS dataset schema.

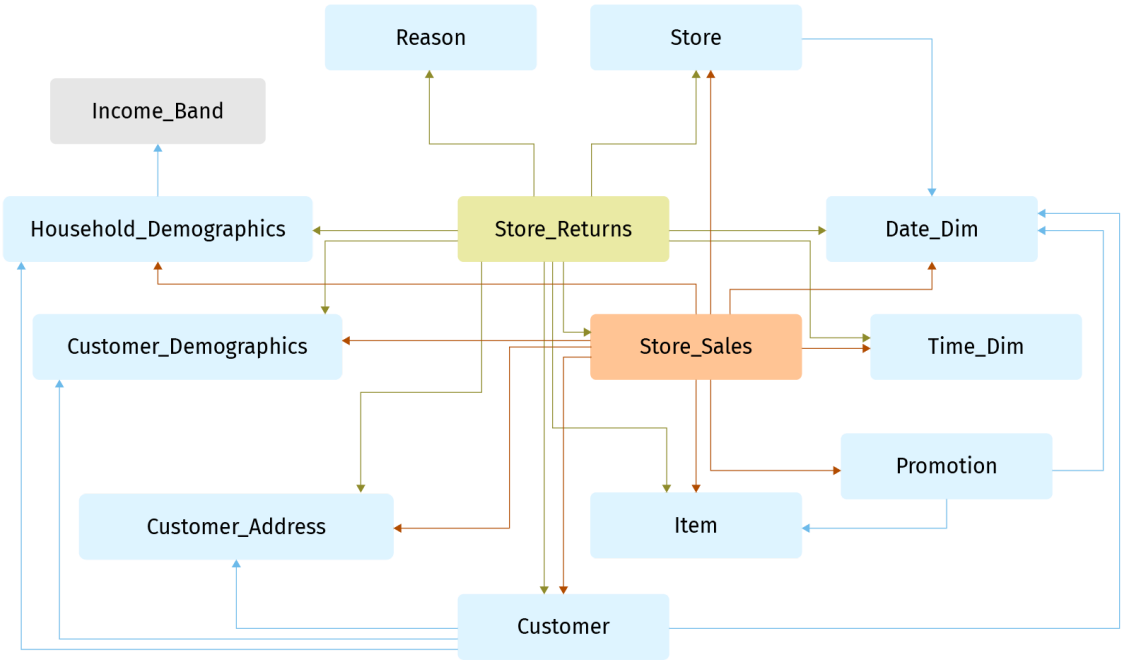


Figure 7.1: TPC-DS dataset entity relationship diagram contains two fact tables: Store_Sales and Store_Returns, with other dimension tables (TPC-DS Dataset ERD, 2020).

7.3 Resources

To ensure the accuracy of the benchmark, a Cloud Kubernetes cluster will be used during the test. Particularly Google Kubernetes Engine (GKE) in Autopilot mode, which simplifies the test process as there is no need to manage the underlying cluster (GKE Autopilot Overview, n.d.). Figure 7.2 shows the "gcloud" command used to create the cluster.

```
// Create GKE cluster in Autopilot mode.
$ gcloud container clusters create-auto "modern-data-platform" \
  --project "mdp-poc" --region "europe-west3" --release-channel "regular" \
  --network "projects/mdp-poc/global/networks/default" \
  --cluster-ipv4-cidr "/17" --services-ipv4-cidr "/22" \
  --subnetwork "projects/mdp-poc/regions/europe-west3/subnetworks/default"
```

Figure 7.2: Creating GKE Autopilot cluster using "gcloud",
which does not require defining the Kubernetes cluster resources in advance.

Only Dremio and MinIO are needed to run this benchmark on the application level. Table 7.1 shows the resources used for each component.

Component	CPU	RAM	Disk
Dremio Coordinator	4	4G	1G
Dremio Executor	4	4G	1G
Dremio ZooKeeper	1	1G	1G
MinIO	1	1G	265G

Table 7.1: Benchmarking compute resources for Dremio and MinIO.

7.4 Preparation

Since Dremio is the benchmarking target, the benchmark guide by Leontiev (2020) will be followed for data generation and preparation. As stated in the guide, only fifty-six queries worked out of ninety-nine queries without making any changes in the queries, and the rest of the queries need some logical query rewrites to work. Even so, for this research project, the benchmark will use the fifty-six out-of-the-box queries. Furthermore, the benchmarking will use a dataset of 10 Gigabytes, which is considered a relatively small dataset for Data Lake/Lakehouse, but it will provide a baseline for Dremio's performance, especially with limited computing resources. Figure 7.3 shows the command to generate the dataset using the "dsdgen" CLI tool part of the TPC-DS suite.

```
// Generate the dataset using TPC-DS tools.
$ dsdgen -scale 10 -f -dir dataset_10g -terminate n
```

Figure 7.3: Generating 10 GB of TPC-DS data schema using the "dsdgen" tool.

Based on the benchmark guide, the data generation was followed by adding CVS headers to the data files, uploading them to the MinIO bucket, loading them in Dremio, and finally creating them as datasets in Apache Iceberg format. Figure 7.4 shows the generated files, the number of rows for each file and the duration of creating their metadata in Dremio.

file	rows	duration
call_center.csv	24	00:00:08
catalog_page.csv	12K	00:00:01
catalog_returns.csv	1.4M	00:00:37
catalog_sales.csv	14.4M	00:00:41
customer_address.csv	250K	00:00:03
customer.csv	500K	00:00:07
customer_demographics.csv	1.9M	00:00:10
date_dim.csv	73K	00:00:02
household_demographics.csv	7.2K	00:00:01
income_band.csv	20	00:00:01
inventory.csv	133.1M	00:05:24
item.csv	102K	00:00:03
promotion.csv	500	<1s
reason.csv	45	<1s
ship_mode.csv	20	00:00:01
store.csv	102	<1s
store_returns.csv	2.9M	00:00:53
store_sales.csv	28.8M	00:09:59
time_dim.csv	86.4K	00:00:01
warehouse.csv	10	<1s
web_page.csv	200	00:00:01
web_returns.csv	719.2K	00:00:17
web_sales.csv	7.2M	00:05:42
web_site.csv	42	<1s

Figure 7.4: TPC-DS data and metadata creation duration as shown in Dremio.

7.5 Execution

For the benchmark execution, Apache JMeter v5.5 is used to run the test plan, which will be executed in consecutive runs to record results with or without cache available (cold and warm queries). Figure 7.5 shows the two executions of the JMeter's test plan to assess the performance when the cache is available or not in Dremio.

```
// Port-forward Dremio's port to access it locally.
$ kubectl port-forward svc/dremio-client 31010 &

// First run (cold queries).
$ jmeter -n -t DremioTestPlanTPC-DS.jmx
Created the tree successfully using DremioTestPlanTPC-DS.jmx
Starting standalone test @ 2023 Mar 13 03:15:45 CET (1678673745418)
summary +      12 in 00:01:16 =      0.2/s Avg:  6371 Min:  2204 Max: 17084 Err:
       0 (0.00%) Active: 1 Started: 1 Finished: 0
summary +      9 in 00:00:30 =      0.3/s Avg:  3316 Min:  1196 Max:  7138 Err:
```

```
       3 (33.33%) Active: 1 Started: 1 Finished: 0
summary =      21 in 00:01:46 =      0.2/s Avg:  5062 Min:  1196 Max: 17084 Err:
       3 (14.29%)
summary +      8 in 00:00:32 =       0.2/s Avg:  4037 Min:  1685 Max:  7842 Err:
       3 (37.50%) Active: 1 Started: 1 Finished: 0
summary =      29 in 00:02:19 =      0.2/s Avg:  4779 Min:  1196 Max: 17084 Err:
       6 (20.69%)
summary +      6 in 00:00:26 =       0.2/s Avg:  4360 Min:  1126 Max: 11324 Err:
       0 (0.00%) Active: 1 Started: 1 Finished: 0
summary =      35 in 00:02:45 =      0.2/s Avg:  4707 Min:  1126 Max: 17084 Err:
       6 (17.14%)
summary +      14 in 00:00:31 =      0.4/s Avg:  2224 Min:   717 Max:  6143 Err:
       3 (21.43%) Active: 1 Started: 1 Finished: 0
summary =      49 in 00:03:16 =      0.3/s Avg:  3998 Min:   717 Max: 17084 Err:
       9 (18.37%)
summary +      5 in 00:00:28 =       0.2/s Avg:  5649 Min:  2301 Max: 15477 Err:
       0 (0.00%) Active: 1 Started: 1 Finished: 0
summary =      54 in 00:03:44 =      0.2/s Avg:  4151 Min:   717 Max: 17084 Err:
       9 (16.67%)
summary +      2 in 00:00:05 =       0.4/s Avg:  2455 Min:  1329 Max:  3582 Err:
       1 (50.00%) Active: 0 Started: 1 Finished: 1
summary =      56 in 00:03:49 =      0.2/s Avg:  4090 Min:   717 Max: 17084 Err: 10
(17.86%)
Tidying up ... @ 2023 Mar 13 03:19:34 CET (1678673974988)
... end of run
```

```
// Save the first run results.
$ mv DremioSummary-tpc-ds.csv dremio-v24.0.0-tpc-ds-10g-cold.csv

// Second run (warm queries).
$ jmeter -n -t DremioTestPlanTPC-DS.jmx
Created the tree successfully using DremioTestPlanTPC-DS.jmx
Starting standalone test @ 2023 Mar 13 03:19:36 CET (1678673976530)
summary +      7 in 00:00:23 =       0.3/s Avg:  3349 Min:   808 Max:  7632 Err:
       0 (0.00%) Active: 1 Started: 1 Finished: 0
summary +      15 in 00:00:30 =      0.5/s Avg:  1977 Min:   180 Max:  5377 Err:
       3 (20.00%) Active: 1 Started: 1 Finished: 0
summary =      22 in 00:00:53 =      0.4/s Avg:  2414 Min:   180 Max:  7632 Err:
       3 (13.64%)
summary +      9 in 00:00:31 =       0.3/s Avg:  3489 Min:   380 Max:  8449 Err:
       3 (33.33%) Active: 1 Started: 1 Finished: 0
summary =      31 in 00:01:25 =      0.4/s Avg:  2726 Min:   180 Max:  8449 Err:
       6 (19.35%)
summary +      18 in 00:00:30 =      0.6/s Avg:  1674 Min:   127 Max:  5939 Err:
       3 (16.67%) Active: 1 Started: 1 Finished: 0
summary =      49 in 00:01:55 =      0.4/s Avg:  2340 Min:   127 Max:  8449 Err:
       9 (18.37%)
summary +      7 in 00:00:26 =       0.3/s Avg:  3690 Min:   332 Max: 12427 Err:
       1 (14.29%) Active: 0 Started: 1 Finished: 1
summary =      56 in 00:02:21 =      0.4/s Avg:  2508 Min:   127 Max: 12427 Err: 10
(17.86%)
Tidying up ... @ 2023 Mar 13 03:21:57 CET (1678674117494)
... end of run
```

```
// Save the second run results.
$ mv DremioSummary-tpc-ds.csv dremio-v24.0.0-tpc-ds-10g-warm.csv
```

Figure 7.5: Execute JMeter's test plan to assess the performance of cold and warm queries in Dremio.

The test outputs are the two files "dremio-v24.0.0-tpc-ds-10g-cold.csv" and "dremio-v24.0.0-tpc-ds-10g-warm.csv", which will be plotted using Pandas and Matplotlib in Python Jupyter Notebook.

7.6 Outcome

This section focuses on better understanding TPC-DS benchmark results; thus, a Python Jupyter Notebook was created to plot the query performance using Pandas and Matplotlib. The Python code is available in Appendix C, as well as the Jupyter Notebook in the project Git repository "modern-data-platform-poc" under "benchmark" directory[4]. First, starting with Figure 7.6, which lists a sample of the data after cleaning and merging the cold and warm query results and getting the difference percentage between them. Next, various charts give insights into the benchmark results.

label	duration_cold_sec	duration_warm_sec	duration_diff_pct
q73	1.838	1.009	-45.1
q63	3.334	3.135	-5.97
q71	3.511	2.291	-34.75
q1	3.777	1.85	-51.02
q56	2.723	1.813	-33.42

Figure 7.6: A sample of Dremio's TPC-DS benchmark after the clean-up.

The chart in Figure 7.7 shows the status of the TPC-DS 99 queries, which in Dremio, according to the benchmark guide by Leontiev (2020), 43 queries are unsupported, and 56 queries are supported. However, during the execution, 10 queries never succeeded, even with multiple trials, reducing the supported queries to 46. For further investigation, the IDs of the constantly failed queries are 2, 10, 35, 47, 51, 57, 59, 65, 69, and 83.

[4] https://github.com/aabouzaid/modern-data-platform-poc/tree/main/benchmark

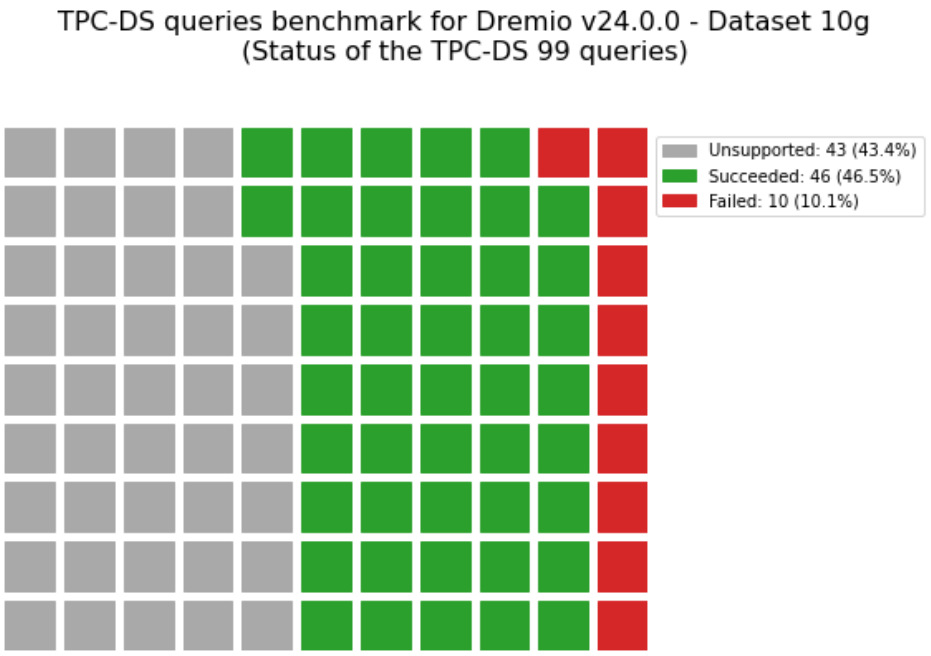


Figure 7.7: The status of the TPC-DS 99 queries by Dremio.

Turning to the test execution duration, as shown in Figure 7.8, with a decrease in the duration for the second run by 39% where the cache is used.

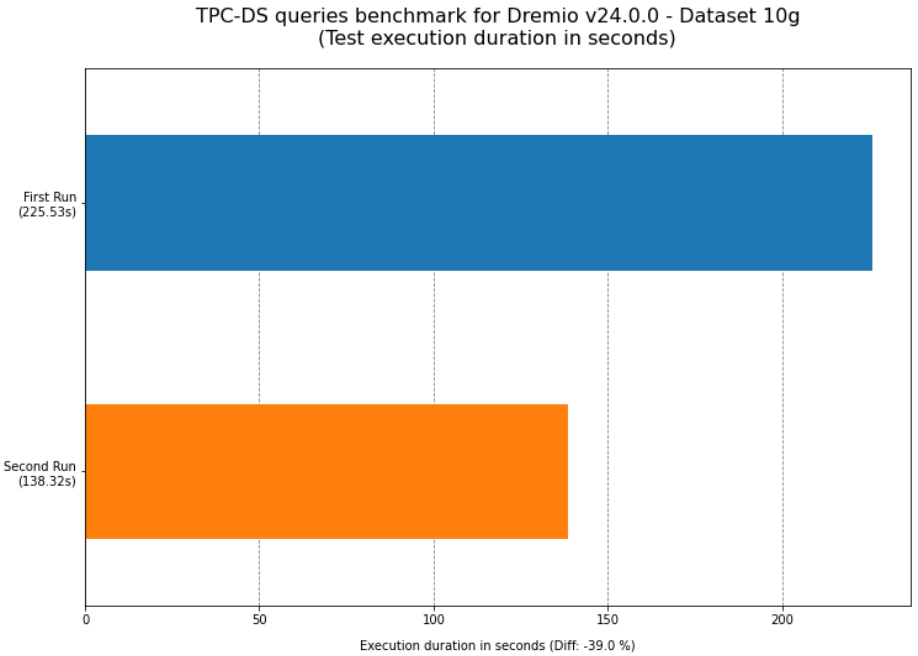


Figure 7.8: Test execution time in seconds with a decrease observed in execution duration when the cache is available.

In general, caching aims to enhance performance, but that is only sometimes the case; sometimes, caching introduces overhead and worsens performance. As a result, to validate that, the difference between cold and warm runs was calculated to identify any positive increase in duration as an indicator of worsened performance. Figure 7.9 shows the performance of the TPC-DS queries where three queries' performance worsened when they ran for the second time and leveraged the cache. With the improvement in the remaining queries, accepting those three queries would be feasible, as they account for only 6.5% of the total queries. It is worth mentioning that each query engine is different, and in real-life scenarios, the queries themselves could be optimised or rewritten based on the engine characteristics for better performance (How Semantic Management Can Reduce Your Overhead and Cut Query Costs by 50-90%, 2023).

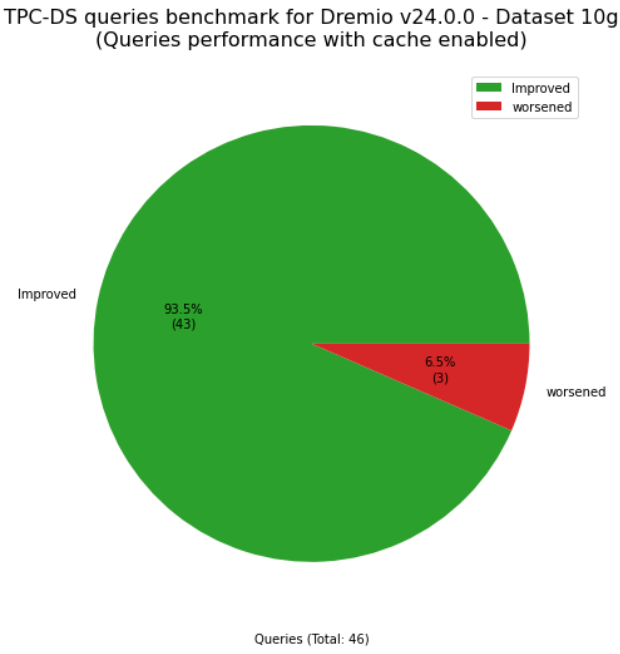


Figure 7.9: TPC-DS queries performance with cache enabled.

The performance statistics have been plotted to better understand the benchmark performance dynamics. Figure 7.10 gives a high-level overview where the best query performance was 83.05% by query ID 96, and the worst was 50.4% by query ID 79. Moreover, 25% of the queries saw improvements greater than 44%, while 75% experienced enhancements of over 17%. Finally, the median enhancement was approximately 33%.

Figure 7.11 shows the distribution of the queries in more detail, where it is clearly shown that the majority of the queries witness enhancements when the cache is used, but only two queries are far outliers with a worsened performance by using the cache.

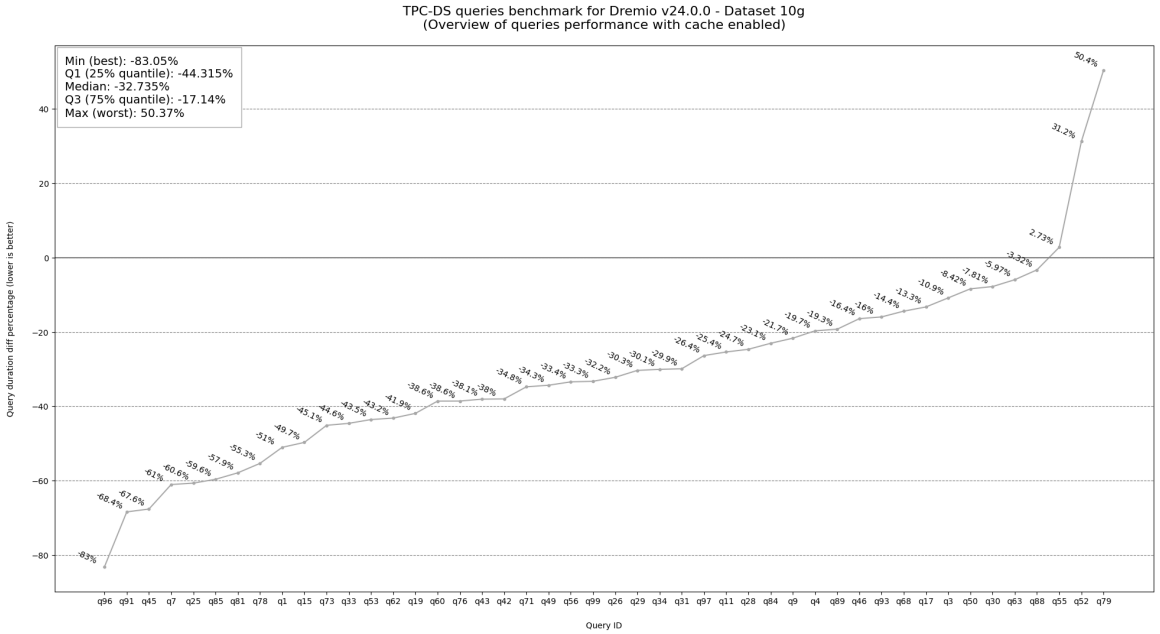


Figure 7.10: Overview of queries performance with cache enabled.

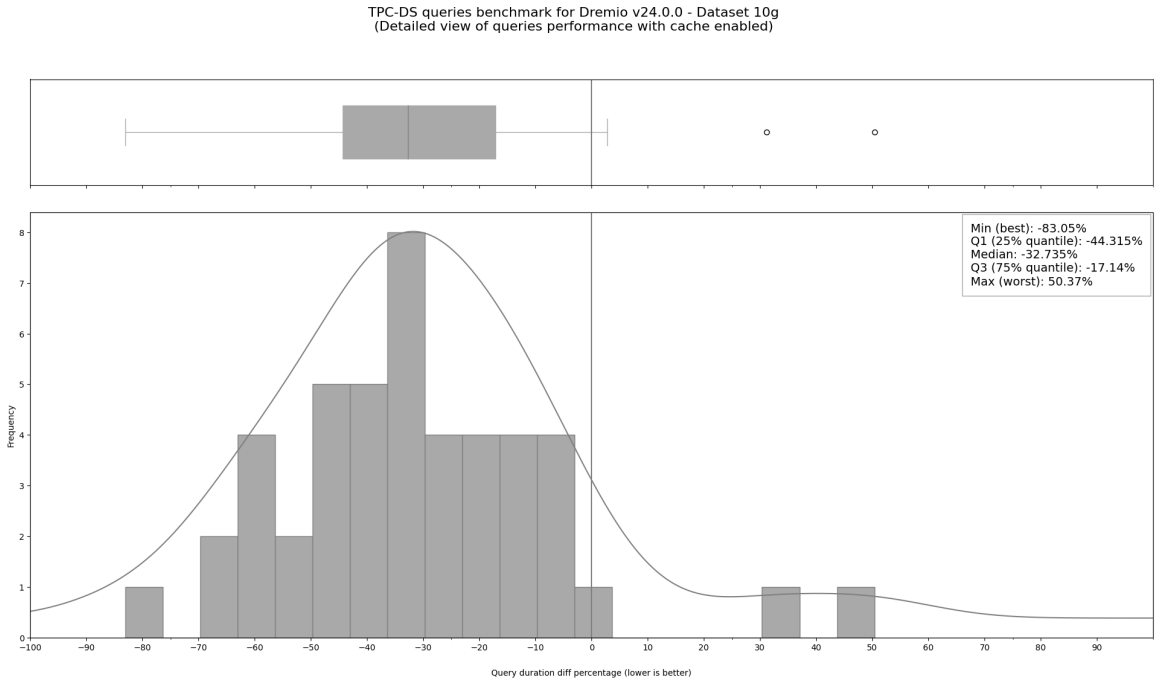


Figure 7.11: Detailed view of queries performance with cache enabled.

Finally, diving more into the individual queries performance, Figures 7.12 and 7.13 show the best and worst ten queries performance in seconds as well as the difference between the cold and warm queries in percentage.

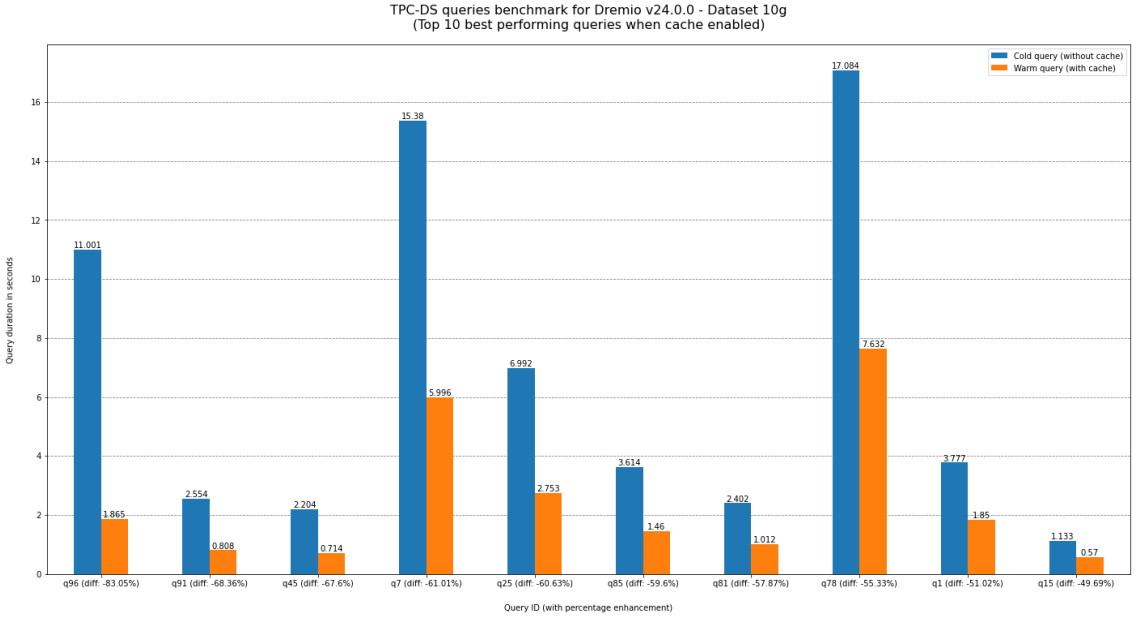


Figure 7.12: Top 10 best-performing queries when cache enabled.

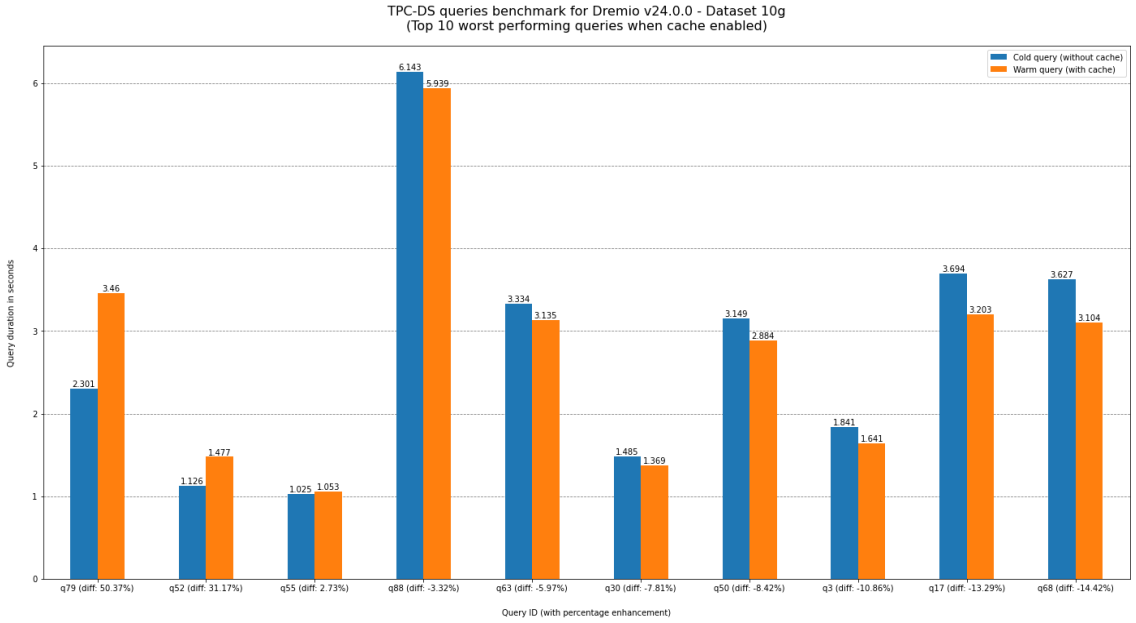


Figure 7.13: Top 10 worst-performing queries when cache enabled.

One of the findings when looking at Dremio UI showed that the query performance depends on the query itself rather than how many rows it accesses. For example, Figures 7.14 and 7.15 show two queries, one accesses 50.9 million rows in 2.17 seconds, but the other accesses 34.2 million rows in 8.41 seconds. Hence, to have an accurate performance evaluation, it is essential to have more information about the query type based on the four groups by TPC-DS. For example, it could be accepted to have a slightly slower ad-hoc query since it is not expected to run frequently compared to other query types.

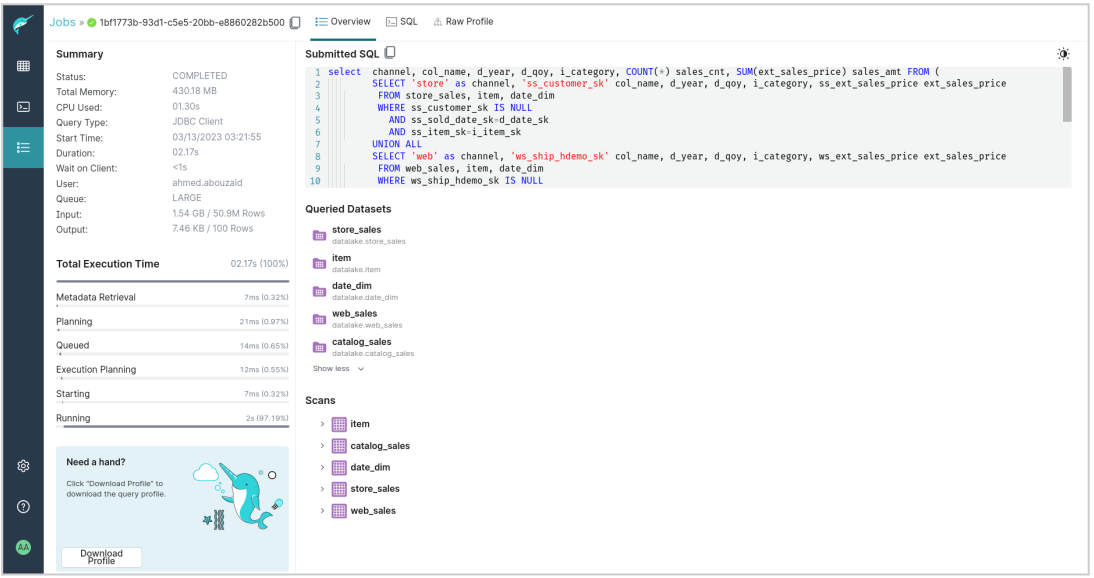


Figure 7.14: A benchmark query accesses 50.9 million rows in 2.17 seconds.

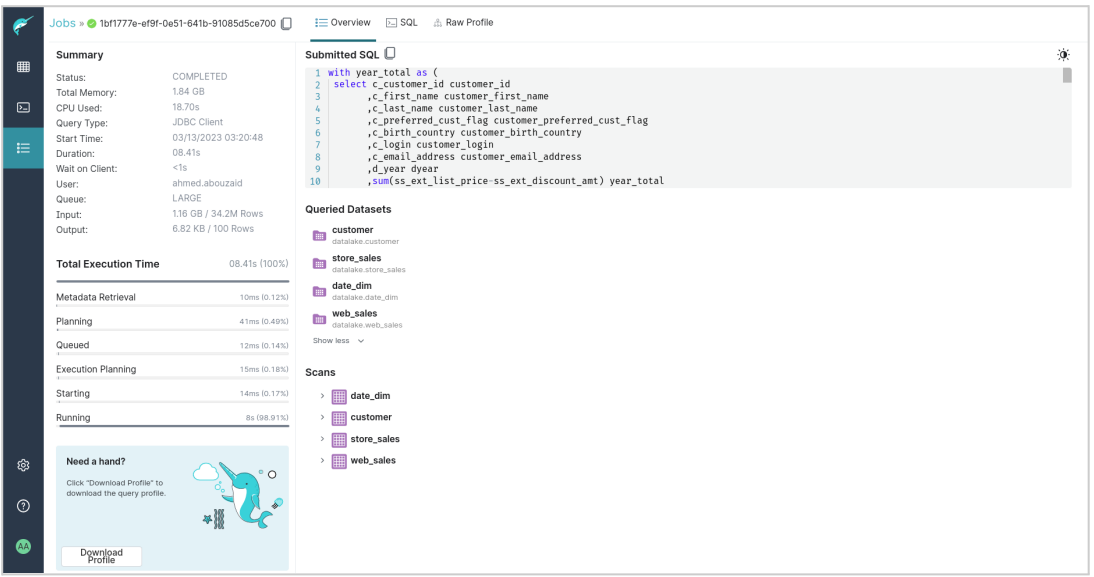


Figure 7.15: A benchmark query accesses 34.2 million rows in 8.41 seconds.

Resource utilisation is another essential benchmark aspect. By utilising the Google Cloud Metrics Explorer (which is enabled by default for GKE clusters), it is clear that caching significantly improves query time and impacts the duration of resource utilisation. On the other hand, both Dremio and MinIO were underutilised during the benchmark, which shows the platform's performance based on the assigned resources for the deployment in the 6.4.2 Resources section. Therefore, it is an indicator of benchmark accuracy due to the absence of any resource throttling that could impact benchmark results. Figure 7.16 illustrates the CPU utilisation percentages for Dremio and MinIO, with annotation showing the first and second test execution periods, where neither MinIO nor Dremio exceeded 50% of the available CPU. Figure 7.17 is the same but for memory utilisation percentages showing that both components barely exceeded 60% of the available memory.

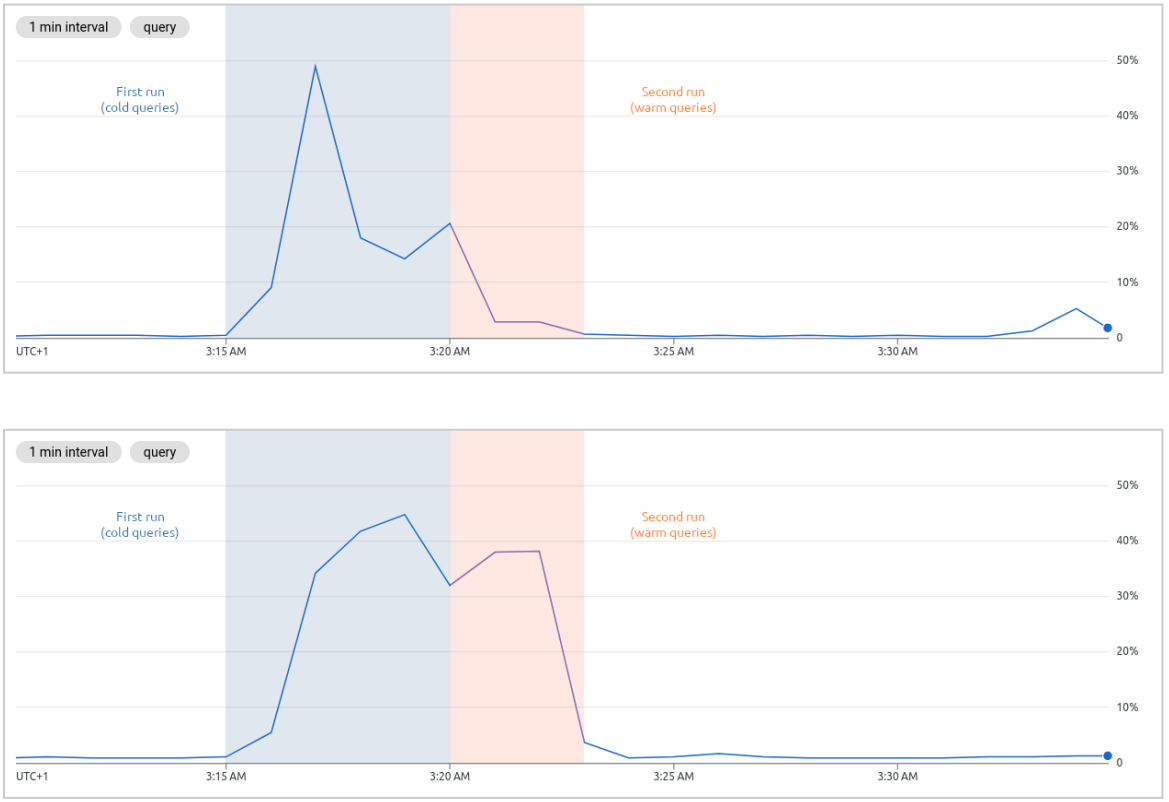


Figure 7.16: CPU utilisation percentages for MinIO (top) and Dremio (bottom).

Figure 7.17: Memory utilisation percentages for MinIO (top) and Dremio (bottom).

7.7 Summary

To conclude this chapter, the TPC-DS benchmark suite showed Dremio's ability to handle millions of records efficiently. Further, with cache available, the overall performance enhancement was confirmed by the benchmark execution duration, which is 39% less. In detail, the median query duration was enhanced by 33%, the best query duration was reduced by 83%, yet, two outliers queries (6.5% of the total queries) where the cache worsened the performance by about 50%. That enhancement in query duration reflects on resource utilisation which also saves costs. Nonetheless, there is a space for more advanced benchmarking, such as debugging the ten failed queries, assessing the performance when the data is updated, using different datasets sizes, tuning Dremio's options, or using benchmarking tools tailored for Data Lakehouse like "LHBench"[5].

[5] https://github.com/lhbench/lhbench

Chapter Eight:
Results and Discussion

8.1 Overview

At this point, the project answered all of the research's three questions; hence, this chapter states the project findings, discusses the results in-depth, and, finally, recommends focus areas for future work.

8.2 Results

RQ1: What are the main current challenges of managing Big Data?

As covered in the literature review, the massive amount of "Big Data" is not just about being "big" but all its properties: volume, velocity, variety, and veracity. Therefore, managing data with this characteristic can be challenging, requiring specialised tools and techniques. Some of the current main challenges of managing Big Data include data integration, quality, security, storage, and processing. Because the exponential increase in the amount of data brought made the old methods unable to cope with it; thus, many organisations started to investigate and adopt new solutions to handle Big Data challenges.

As a result, multiple attempts have been created in the last two decades to solve Big Data challenges, but each has limitations. In particular, the Data Lake architecture emerged around 2010 to provide large amounts of heterogeneous data in a single place, benefiting from Cloud computing and cheap commodity storage solutions; however, due to the lack of unified fine-grained access, it was easily turned from a "lake" to a "swamp". Therefore, in 2020, a new hybrid approach architecture appeared called Data Lakehouse combines the benefits of both Data Lakes and Data Warehouses to overcome the risks of the previous solution, such as missing data culture, vendor lock-in, and poor security and governance. Furthermore, in various scenarios, the adoption of DataOps and Cloud-Native software has increased significantly due to their key role in data management. Accordingly, leveraging Data Lakehouse architecture, DataOps, and Cloud-Native software could help in building an effective and resilient data management platform to handle ever-increasing Big Data challenges.

RQ2: How could DataOps methodologies help to manage Big Data?

DataOps is a set of methodologies that aim to improve the efficiency and effectiveness of data management by applying Agile and DevOps methodologies to data operations to guarantee data quality. Adopting DataOps principles undoubtedly plays a significant role in building an organisation's data culture by implementing a process-oriented approach to working with data. In fact, organisations that foster data-driven decision-making have a competitive edge by identifying and responding to opportunities faster than their competitors, thereby gaining a competitive advantage. Following the DataOps Manifesto, most principles have been applied during this project, like simplicity, daily interactions, orchestration, reusability, and reproducible and disposable environments. Using the DataOps as a framework during the development of the proposed data platform contributed to providing a robust and flexible process working with different stages of the project.

RQ3: How can Kubernetes and Cloud-Native software help to build an efficient data platform for Big Data?

Clearly shown during the implementation that leverages Cloud-Native software designed to run in the Cloud or on-premises with state-of-the-art architectures like Data Lakehouse helps build reliable and efficient data platforms to handle Big Data challenges. Using that combination brings many benefits for data management like portability to reduce the risks of vendor lock-in, enabling the organisations to seamlessly transition between providers with minimum disruption, scalability for handling the massive amount of data efficiently, flexibility for deploying different types of workloads to handle the variety of Big Data, resource efficiency to decrease the managing costs, and finally resiliency to ensure the availability of the data platform. Another vital aspect, particularly, is using Kubernetes ecosystems like Operators pattern, Kustomize declarative approach, and off-the-shelf Helm applications, which can reduce development efforts and accelerate delivery by focusing on the business logic instead of reinventing the wheel for the platform management.

8.3 Project Reflection

Before moving to the discussion, it is essential first to put this project in a broader context. This section highlights what went well and what could be done better during the project execution. Starting with the approach, using an iterative approach proved to be highly beneficial in achieving the project goals, as it allowed for continuous refinement of the methodologies based on new information or feedback. Further, regularly revisiting the project path and adjusting the methodology accordingly aid in staying on track and progressing towards the project objectives. Also, critical thinking, problem-solving, and data-driven decision-making were essential to solve ongoing challenges and achieving better research outcomes.

On the one hand, by looking back at the related works discussed in the literature review 2.7 Modern Data Platform, the project results agree with the previous research about the importance of the Data Lakehouse architecture and how it can provide a flexible and scalable solution for data management compared to previous methods like Data Warehouse and Data Lake. Nevertheless, the current project took a step further to bridge the research gap by architecting and implementing such a platform and going into detail about core components and performance assessment. Moreover, the output of this research project (building a resilient cloud-agnostic data platform based on Data Lakehouse architecture) could be directly used for other projects like the proposed “Xel” project by Barron-Lugo et al. (2023). Therefore, this project successfully verified and extended previous research as well as laid the groundwork for future research. However, it is challenging to directly compare the current research results due to the topic’s novelty (since the Data Lakehouse architecture was shown for the first time in 2020) and the research area’s originality, making it hard to establish a direct qualitative comparison or reference point.

On the other hand, some of the current project areas could be improved, which are mainly affected by time constraints. For example, although it was advised in the architecture chapter to use Kubernetes Operator whenever possible, yet, MinIO Operator has not been used in the proof-of-concept due to its complexity. Another concern is the dataset size used in the benchmarking by the TPC-DS

suite, as the 10 Gigabytes dataset may be insufficient and could potentially lead to inaccurate benchmarking results. It was not possible to use a larger dataset because it requires more time and computing resources since the test data generated by the TPC-DS suite needs some transformation before using it in Dremio. Lastly, during the benchmarking using the TPC-DS suite, approximately 50% of the test queries could not be executed on Dremio. These queries were either unsupported (43 queries) or failed without a clear reason (10 queries). As a result, it was not possible to compare the benchmarking results with the previous findings of Jain et al. (2023). If more time was available, the failed and unsupported queries could be reviewed for better benchmarking results.

8.4 Discussion

Truning now to the discussion about the data platforms in light of the research results. Data Lakehouse architecture is the cornerstone of modern data platforms, and one of its major advantages is the ability to support a wide range of data sources and formats, which is a mission-critical requirement for organisations today working with structured, semi-structured, and unstructured formats and ingest data from various sources, including sensors, social media, and customer interactions. In this sense, the Data Lakehouse architecture enables organisations to utilise their data better and gain previously deemed impossible insights by providing a unified platform for storing and processing data. Nevertheless, it is worth noting that this transformation has four crucial elements related to data: culture, infrastructure, applications, and formats.

Data culture: As revealed in the literature review, the majority of organisations acknowledge the importance of building a data culture, yet, many of them struggle to build one. That is probably because of the rapid change in data management practices as well as a lack of detailed guidance to build sustainable data practices. While the DataOps principles were introduced to address these challenges, they still need more details to be able to apply them in action. Moving to the challenges of applying DataOps methodologies, one of the main challenges is that DataOps requires a cultural shift towards a data-driven mindset, which can be challenging to achieve in organisations that are not used to working in this manner.

On the one hand, DataOps, at its core, relies on technical expertise, making it easier to implement with a skilled team with prior experience with the DataOps roots like DevOps and Agile. On the other hand, this means that DataOps can be challenging to scale across organisations, especially for professionals with non-technical backgrounds. Accordingly, by looking back at the DevOps history, where various implementation structures emerged to fulfil different use cases (Skelton & Pais, n.d.), it is expected that DataOps will follow a similar pattern to deal with this contradictory situation.

Data infrastructure: The emergence of Kubernetes and Cloud-Native software has also significantly impacted data management. Kubernetes provides a scalable and efficient platform for managing data, making it easier for organisations to manage their data infrastructure. In addition, DataOps and Cloud-Native ecosystems also offer practices and tools for building and running scalable applications in the Cloud. Hence, that combination can accelerate building and deploying applications, enabling them to iterate faster and focus on business needs. However, Kubernetes is not a trivial system, and using Kubernetes still includes complexity and overhead.

Firstly, Kubernetes cluster management is a cumbersome task. Building a production-grade cluster is time-consuming and requires many axillary workloads and configurations like network policies, access controls, managing secrets, monitoring, and logging. There are several solutions available to mitigate that issue. One is using fully managed Kubernetes services which eliminate the need to manage the actual Kubernetes cluster like Google Kubernetes Engine (GKE) Autopilot. Another is using Kubernetes distributions which are preloaded with essential axillary workloads like Red Hat OpenShift and SUSE Rancher. Both solutions allow organisations to focus on their core business workloads and significantly decrease the time to use Kubernetes in production.

Secondly, although Kubernetes has become the de-facto standard for container orchestration and is widely adopted in private and public sectors, it is considered a relatively new system (its initial release was on September 2014). That means it can be assumed that only some applications are Kubernetes-ready. Additionally,

despite the fact that various applications have integrated and endorsed Kubernetes, not all of Kubernetes' functionalities are utilised. For example, Dremio, the selected solution for the Data Lakehouse platform in this project, provides a comprehensive set of REST APIs that allows users to perform most operations programmatically; nevertheless, it does not utilise the Kubernetes Operator capabilities, and it only uses Kubernetes core primitives. Still, based on the rapid adoption of Kubernetes, which has become the primary platform for running almost any type of workload, and the number of operators on OperatorHub.io (305 operators at the time of writing), it is likely a matter of time before seeing more Operators for data applications and native integration with different data storage systems.

Data applications: Given the fast-paced evolution of solutions in the data market, it can be challenging to find a solution that adequately meets all use cases, particularly when it comes to Data Lakehouse, as outlined in section 4.2 Architecting Core Components. Thus, it ultimately comes down to making trade-offs, especially regarding portability. For example, Databricks, a company named as one of the leaders in 2021 and 2022 Gartner Magic Quadrant for Cloud Database Management Systems (Databricks Named a Leader in 2021, 2021; Databricks Named a Leader in 2022, 2022) opted for a "Cloud-Only" approach and did not offer any official support for self-hosted Kubernetes. On the other hand, the solutions that focus on openness, like Cloudera Data Platform, where the company describes its solution as "100% open—open source, open standards" (Data Lakehouse: A Modern Data Architecture, n.d.), but their solution is excessively complicated and only suitable for large enterprises. Which, in fact, reduces the available choices of Data Lakehouse solutions for small and medium-sized enterprises.

Turning to Dremio, the self-hosted version is missing some of the Dremio Cloud capabilities, where the self-hosted does not have a built-in metadata version control (git-like) which is present in the Cloud known as "Dremio Arctic". However, that could be covered by another software like lakeFS and integration with Dremio (lakeFS Integrations, n.d.). Therefore, it is important to note that there is no universal solution for data management, and it is crucial for each

organisation to evaluate carefully and identify the business needs before deciding to adopt a specific Data Lakehouse solution.

Data formats: The emergence of the Data Lakehouse architecture has brought new ideas to the world of data management. While it presents many opportunities for organisations to use their data effectively, it also comes with its own challenges. Even though the Data Lakehouse architecture relies heavily on the Data Lake and Data Warehouse pillars and ecosystem, the new architecture is still young and evolving rapidly. Therefore, with more adoption of the new architecture, the data landscape is expected to change a lot in the near future.

This research project has shown that an entire Data Lakehouse architecture has many moving parts, which could overwhelm many organisations. Hence, organisations must carefully evaluate their data management needs and weigh the trade-offs before using a full Data Lakehouse solution. A hybrid solution could be better for different use cases with more support of the open table formats (e.g., Apache Hudi, Apache Iceberg, and Delta Lake) because of the ability of these formats to provide a standardised way of storing and managing data and metadata. That is mainly with more adoption of the major service providers (like AWS, Google, Azure, Snowflake, Databricks, Dremio, and Cloudera), where it will be easier for companies to move their data between providers and reduce vendor lock-in risk.

8.5 Future Work

This research built a prototype as a cornerstone for the modern data platform following the Data Lakehouse architecture and benefiting from DataOps, Kubernetes, and Cloud-Native software, focusing mainly on portability and openness. Nevertheless, there are multiple directions for future work, which are split into four categories. First category: It focuses on the same approach as this research, adding more components of the Unified Data Infrastructure v2.0 architecture, for example, implementing a data version control like LakeFS and integrating it with Dremio or introducing another key component like Apache Spark, which provides an analytics engine for data processing. Also, implementing

the platform's operational components, like building an operator for Dremio or introducing advanced infrastructure automation via GitOps. **Second category:** It takes a different path, where it customises the current generic architecture and implements one of the blueprints suggested by Bornstein et al. (2022) like "Modern Business Intelligence", "Multimodal Data Processing", or "Artificial Intelligence and Machine Learning". **Third category:** It focuses on investigating the integration with the public Cloud service since more of the Cloud providers now support the open table formats, which reduces the risk of vendor lock-in, and at the same time, accelerates the development and decreases the maintenance burdens. **Fourth category:** It is around conducting advanced benchmarking to assess the performance of the data platform components in-depth. Namely, rewriting the TPC-DS unsupported queries to work on Dremio, comparing Dremio with another solution, benchmarking the platform with refreshed data, using different datasets sizes, tuning Dremio's options, or using benchmarking tools tailored for Data Lakehouse like "LHBench"[6].

8.6 Conclusion

This research project shed light on some significant challenges in managing Big Data, as the traditional methods are unable to cope with the exponential increase in data sources and types. Challenges like resilience, scalability, portability, and vendor lock-in risks are the primary influencers of organisations' data strategies. Therefore, many organisations investigate the latest data architectures, technologies, and practices to address these challenges. One of the unique latest architectures is Data Lakehouse (2020) which combines features of the previous architectures like Data Lake and Data Warehouse, which is the core of "Modern Data Platform". The term Modern Data Platform is not frequently used in academic literature; however, the concepts that compose that idea are widely studied and discussed in academic research. Nevertheless, the term in the technology industry refers to the state-of-art use of technologies and practices to manage large amounts of data.

[6] https://github.com/lhbench/lhbench

As a result, this project used the Data Lakehouse architectures to build a proof-of-concept for a Modern Data Platform using DataOps methodologies for process, Kubernetes as an orchestration platform, and Cloud-Native software data applications. The resulting implementation demonstrated the effectiveness of this combination in creating a resilient data platform. In the end, with new architectures, technologies, and practices, it can be stated clearly that the data management landscape will undergo a fundamental transformation in the next few years, especially with the unification of data formats using open table standards (e.g., Apache Hudi, Apache Iceberg, and Delta Lake). Hence, the boundaries between the Cloud and on-premises could fade over time, providing more flexibility to manage massive amounts of data for organisations of all sizes and use cases.

References

1. 3 reasons why DataOps is essential for big data success | The Big Data Hub. (2014, June). IBM Big Data Hub. Retrieved January 8, 2023, from https://web.archive.org/web/20140715022431/https://www.ibmbigdatahub.com/blog/3-reasons-why-dataops-essential-big-data-success
2. Argo Workflows Architecture. (n.d.). Argo Workflows. Retrieved March 8, 2023, from https://argoproj.github.io/argo-workflows/architecture/
3. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., Van Hovell, H., Ionescu, A. M., Łuszczak, A., Świtakowski, M., Szafrański, M., Li, X., Ueshin, T., Mokhtar, M. K., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., . . . Zaharia, M. (2020). Delta lake. Proceedings of the VLDB Endowment, 13(12), 3411–3424. https://doi.org/10.14778/3415478.3415560
4. Armbrust, M., Zaharia, M., Ghodsi, A., & Xin, R. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. Conference on Innovative Data Systems Research. http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf
5. Barron-Lugo, J. A., Gonzalez-Compean, J., Lopez-Arevalo, I., Carretero, J., & Martinez-Rodriguez, J. L. (2023). Xel: A cloud-agnostic data platform for the design-driven building of high-availability data science services. Future Generation Computer Systems, 145, 87–103. https://doi.org/10.1016/j.future.2023.03.019
6. Boch, M., Gindl, S., Barnett, A., Margetis, G., Mireles, V., Adamakis, E., & Knoth, P. (2022). A Systematic Review of Data Management Platforms. Information Systems and Technologies, 15–24. https://doi.org/10.1007/978-3-031-04819-7_2
7. Bornstein, M., Li, J., & Casado, M. (2022, November 7). Emerging Architectures for Modern Data Infrastructure. Andreessen Horowitz. Retrieved February 16, 2023, from https://a16z.com/2020/10/15/emerging-architectures-for-modern-data-infrastructure/

8. Brown, S. (n.d.). The C4 model for visualising software architecture. Retrieved March 4, 2023, from https://c4model.com/
9. Brown, S. (2023, February 24). The C4 model for visualising software architecture. Leanpub. Retrieved March 4, 2023, from https://leanpub.com/visualising-software-architecture/read
10. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. (2019). Cisco Systems, Inc. Retrieved March 24, 2023, from https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf
11. Clark, L. (2023, January 13). Apache Iceberg promises to change the economics of cloud-based data analytics. Retrieved February 24, 2023, from https://www.theregister.com/2023/01/03/apache_iceberg/
12. Clarke, R. (2015). Big data, big risks. Information Systems Journal, 26(1), 77–90. https://doi.org/10.1111/isj.12088
13. Cloud Native Survey 2021. (2021). In Cloud Native Computing Foundation. Retrieved January 5, 2023, from https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf
14. CNCF Annual Survey 2022. (2023, January 31). Cloud Native Computing Foundation. Retrieved March 7, 2023, from https://www.cncf.io/reports/cncf-annual-survey-2022/
15. Costello, K., & Rimol, M. (2021, October 18). Gartner Identifies the Top Strategic Technology Trends for 2022. Gartner. Retrieved March 24, 2023, from https://www.gartner.com/en/newsroom/press-releases/2021-10-18-gartner-identifies-the-top-strategic-technology-trends-for-2022
16. Data fields overlap. (2020). Precision Artificial Intelligence Virtual Summit 2020. https://www.precisionmedicineleaderssummit.com/ai-machine-learning-data-science-in-precision-medicine/
17. Data Lakehouse: A Modern Data Architecture. (n.d.). Cloudera. Retrieved March 21, 2023, from https://www.cloudera.com/products/open-data-lakehouse.html

18. Data on the daily subnational 14-day notification rate of new COVID-19 cases. (2022). [Dataset]. European Centre for Disease Prevention and Control (ECDC). https://www.ecdc.europa.eu/en/publications-data/subnational-14-day-notification-rate-covid-19
19. Databricks Named a Leader in 2022 Gartner® Magic Quadrant<sup>TM for Cloud Database Management Systems. (2022, December 16). Databricks. Retrieved March 21, 2023, from <span>https://www.databricks.com/blog/2022/12/16/databricks-named-leader-2022-gartner-magic-quadrant-cloud-database-management
20. DataLakeHouse. (2020, June 13). DataLakeHouse Reference Architecture. Retrieved February 16, 2023, from https://datalakehouse.org/datalakehouse-platform/
21. DataOps Manifesto. (2017, July). The DataOps Manifesto - Read The 18 DataOps Principles. DataOps Manifesto - 18 DataOps Principles. Retrieved March 3, 2023, from https://dataopsmanifesto.org/en/
22. Declarative Management of Kubernetes Objects Using Configuration Files. (2023, January 4). Kubernetes. Retrieved March 7, 2023, from https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/
23. Definition of Best-of-breed - Gartner Information Technology Glossary. (n.d.). Gartner. https://www.gartner.com/en/information-technology/glossary/best-of-breed
24. Del Sagrado, J., & Del Águila, I. M. (2020). Assisted requirements selection by clustering. Requirements Engineering, 26(2), 167–184. https://doi.org/10.1007/s00766-020-00341-1
25. Desai, V., Fountaine, T., & Rowshankish, K. (2022, June 16). How to unlock the full value of data? Manage it like a product. McKinsey & Company. Retrieved February 16, 2023, from https://www.mckinsey.com/capabilities/quantumblack/our-insights/how-to-unlock-the-full-value-of-data-manage-it-like-a-product

26. Dobies, J., & Wood, J. (2020). Kubernetes Operators: Automating the Container Orchestration Platform. Van Duuren Media.
27. Domingus, J., & Arundel, J. (2022). Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud (2nd ed.). O'Reilly Media.
28. Dremio Architecture Guide. (2020). [White paper]. Dremio Corporation. https://hello.dremio.com/dremio-architecture-guide.html
29. E. Janssen, N. (2022). The Evolution of Data Storage Architectures: Examining the Value of the Data Lakehouse [Master Thesis]. University of Twente.
30. Etzion, D., & Aragón-Correa, J. A. (2016). Big Data, Management, and Sustainability. Organization & Environment, 29(2), 147–155. https://doi.org/10.1177/1086026616650437
31. Fang, H. (2015). Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER). https://doi.org/10.1109/cyber.2015.7288049
32. Fisher, M. S. (2007). Software Verification and Validation: An Engineering and Scientific Approach. Springer Publishing.
33. Foote, K. (2022, June 4). What is a Modern Data Platform? Understanding the Key Components. Databand. Retrieved January 23, 2023, from https://databand.ai/blog/what-is-a-modern-data-platform/
34. Giebler, C., Gröger, C., Hoos, E., Schwarz, H., & Mitschang, B. (2019). Leveraging the Data Lake: Current State and Challenges. Big Data Analytics and Knowledge Discovery, 179–188. https://doi.org/10.1007/978-3-030-27520-4_13
35. GKE Autopilot overview. (n.d.). Google Cloud. Retrieved March 14, 2023, from https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview

36. Gür, I., M. Sc. (2021). DataOps for Data Sharing. In B. Otto & J. Rehof (Eds.), ISST Reports (ISSN 0943-1624). Retrieved January 9, 2023, from https://www.isst.fraunhofer.de/content/dam/isst-neu/documents/Publikationen/Datenwirtschaft/ISST-Report/DataOps_Fraunhofer-ISST-Report-kl.pdf
37. Harby, A. A., & Zulkernine, F. (2022). From Data Warehouse to Lakehouse: A Comparative Review. 2022 IEEE International Conference on Big Data (Big Data). https://doi.org/10.1109/bigdata55660.2022.10020719
38. Hayes, B. (2019, February 9). How do Data Professionals Spend their Time on Data Science Projects? Business Over Broadway. Retrieved March 24, 2023, from https://businessoverbroadway.com/2019/02/19/how-do-data-professionals-spend-their-time-on-data-science-projects/
39. How Semantic Management can reduce your overhead and cut query costs by 50-90%. (2023, March 1). Single Origin. Retrieved March 15, 2023, from https://blog.singleorigin.tech/reduce-overhead-and-improve-performance-with-semantic-management/
40. IaaS vs. PaaS vs. SaaS. (2020). Red Hat. Retrieved December 31, 2022, from https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas
41. Iceberg Table's Architecture. (2022, July 27). MinIO. https://blog.min.io/lakehouse-architecture-iceberg-minio/
42. Inmon, W. H. (2005). Building The Data Warehouse (4th ed.). Wiley.
43. Jain, P., Kraft, P., Power, C., Das, T., Stoica, I., & Zaharia, M. (2023). Analyzing and Comparing Lakehouse Storage Systems. Conference on Innovative Data Systems Research (CIDR '23). https://www.cidrdb.org/cidr2023/papers/p92-jain.pdf
44. Jump Start Using the Operator-SDK. (n.d.). OperatorHub.io. Retrieved March 7, 2023, from https://operatorhub.io/getting-started
45. Kimball, R., & Caserta, J. (2004). The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data (1st ed.). Wiley.

46. Kratzke, N. (2014). Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In. Journal of Computer and Communications, 02(12), 1–7. https://doi.org/10.4236/jcc.2014.212001
47. Kubernetes architecture, How Kubernetes works. (2019, August 19). Cloud Native Computing Foundation. https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/
48. Kubernetes native configuration management. (n.d.). Kustomize. Retrieved March 8, 2023, from https://kustomize.io/
49. lakeFS Integrations. (n.d.). lakeFS. Retrieved March 2, 2023, from https://docs.lakefs.io/integrations/dremio.html
50. LaPlante, A. (2020). The Modern Cloud Data Platform: Rise of the Lakehouse. O'Reilly Media, Inc. https://www.oreilly.com/library/view/the-modern-cloud/9781492087953/
51. LaPlante, A., & Safari, A. O. M. C. (2020). Building a Unified Data Infrastructure. Van Duuren Media.
52. Leontiev, S. (2020, August 25). Dremio Benchmarking Methodology - How to Do It Yourself. Dremio. Retrieved March 14, 2023, from https://www.dremio.com/blog/dremio-benchmarking-methodology/
53. Lorica, B., Armbrust, M., Xin, R., Zaharia, M., & Ghodsi, A. (2020, January 30). What Is a Lakehouse? Databricks. Retrieved December 29, 2022, from https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html
54. Ma, R., Li, W., Ma, N., Zhang, X., & Zhang, H. (2020). Design and Research of Big Data Platform Framework for Power Enterprises. IOP Conference Series: Earth and Environmental Science, 529(1), 012009. https://doi.org/10.1088/1755-1315/529/1/012009
55. Mainali, K., Ehrlinger, L., Himmelbauer, J., & Matskin, M. (2021). Discovering DataOps: A Comprehensive Review of Definitions, Use Cases, and Tools. DATA ANALYTICS 2021, the Tenth International Conference on Data Analytics, 61–69.

56. Managing Resources. (2022, October 8). Kubernetes. Retrieved March 17, 2023, from https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/
57. Mayr, A. (2023, February 7). Kubernetes in the wild report 2023. Dynatrace. Retrieved March 7, 2023, from https://www.dynatrace.com/news/blog/kubernetes-in-the-wild-2023/
58. MinIO High Performance Multi-Cloud Object Storage. (2022). [White paper]. MinIO, Inc. https://min.io/resources/docs/MinIO-High-Performance-Multi-Cloud-Object-Storage.pdf
59. MongoDB. (2021, April). What Is A Data Platform? Retrieved February 16, 2023, from https://www.mongodb.com/what-is-a-data-platform
60. More Than Half of Enterprise IT Spending in Key Market Segments Will Shift to the Cloud by 2025. (2022, February 9). Gartner. Retrieved March 5, 2023, from https://www.gartner.com/en/newsroom/press-releases/2022-02-09-gartner-says-more-than-half-of-enterprise-it-spending
61. Munappy, A. R., Bosch, J., & Olsson, H. H. (2020). Data Pipeline Management in Practice: Challenges and Opportunities. Product-Focused Software Process Improvement, 168–184. https://doi.org/10.1007/978-3-030-64148-1_11
62. Opara-Martins, J., Sahandi, R., & Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. Journal of Cloud Computing, 5(1). https://doi.org/10.1186/s13677-016-0054-z
63. Operator pattern. (2023, January 16). Kubernetes. Retrieved March 7, 2023, from https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
64. Oppermann, A. (2022, December 19). Architecture for data platforms. Hygraph. Retrieved February 17, 2023, from https://hygraph.com/blog/data-platform-architecture
65. Orescanin, D., & Hlupic, T. (2021). Data Lakehouse - a Novel Step in Analytics Architecture. International Convention on Information and Communication Technology, Electronics and Microelectronics. https://doi.org/10.23919/mipro52101.2021.9597091

66. Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems, 24(3), 45–77. https://doi.org/10.2753/mis0742-1222240302
67. Poess, M., O. Nambiar, R., & Walrath, D. (2007, September). Why you should run TPC-DS: a workload analysis. ACM Digital Library. https://dl.acm.org/doi/10.5555/1325851.1325979
68. Robertson, S., & Robertson, J. (2012). Mastering the Requirements Process: Getting Requirements Right. Addison-Wesley.
69. Ruparelia, N. B. (2016). Cloud Computing: The MIT Press Essential Knowledge series. The MIT Press. https://ieeexplore.ieee.org/book/7580009
70. Simon, B. (2019, June 14). Complete Guide to the People, Process, Technology Framework. Smartsheet. Retrieved February 15, 2023, from https://www.smartsheet.com/content/people-process-technology
71. Skelton, M., & Pais, M. (n.d.). DevOps Team Structure and Anti-Types. DevOps Topologies. Retrieved March 24, 2023, from https://web.devopstopologies.com/
72. Späti, S. (2022, August 25). Data Lake / Lakehouse Guide: Powered by Data Lake Table Formats (Delta Lake, Iceberg, Hudi). Airbyte. Retrieved February 20, 2023, from https://airbyte.com/blog/data-lake-lakehouse-guide-powered-by-table-formats-delta-lake-iceberg-hudi
73. Strod, E. (2019a, July 3). DataOps Data Architecture. DataKitchen. Retrieved March 3, 2023, from https://datakitchen.io/dataops-data-architecture/
74. Strod, E. (2019b, September 10). Why Do DataOps. DataKitchen. Retrieved March 3, 2023, from https://datakitchen.io/why-do-dataops-2/
75. Thusoo, A., & Sen Sarma, J. (2017). Creating a Data-Driven Enterprise with DataOps. O'Reilly Media, Inc. https://www.oreilly.com/library/view/creating-a-data-driven/9781492049227/

76. TPC-DS Dataset ERD. (2020, June 30). Grid Dynamics. https://blog.griddynamics.com/edw-performance-comparison/

77. Travica, B. (2017). Big Data Aspects and Decision Making. Sixth European Academic Research Conference on Global Business, Economics, Finance and Social Sciences.

78. Watson, H. J., Goodhue, D. L., & Wixom, B. H. (2002). The benefits of data warehousing: why some organizations realize exceptional payoffs. Information &Amp; Management, 39(6), 491–502. https://doi.org/10.1016/s0378-7206(01)00120-3

79. What is a Kubernetes operator? (2022, May 11). Red Hat. Retrieved March 7, 2023, from https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-operator

80. Yugal, L. (2022). Business Analytics: Trends and Challenges. International Conference on Intelligent Emerging Methods of Artificial Intelligence & Cloud Computing, 236–243. https://doi.org/10.1007/978-3-030-92905-3_31

Appendix A:
Research Proposal

EDINBURGH NAPIER UNIVERSITY SCHOOL OF COMPUTING

MSc RESEARCH PROPOSAL

1. Student details

First name	Ahmed
Last (family) name	AbouZaid
Edinburgh Napier matriculation number	40497354

2. Project outline details

Title of the proposed project	Evaluating Modern Data Platform Practices and Architecture in the Cloud-Native Ecosystem: DataOps on Kubernetes
Name of supervisor	Dr. Peter Barclay

3. Brief description of the research area - background

According to Cisco, one of the worldwide leaders in networking, “in 2017, the annual run rate for global IP traffic was 1.5 ZB per year, or 122 EB per month” (Cisco Visual Networking Index: Forecast and Trends, 2017–2022, 2019, p. 1); hence we already live in the “Big Data” era. The data has increased in size in the last twenty years, and new data types and formats emerged from different sources. The boom in the data brought many new challenges where the old methods could not deal with that kind of data which is different in quantity and quality. It is worth emphasising what makes handling that data challenge is not just being “big” but all of its properties or what is known as “4 Vs”: Volume, Velocity, Variety, and Veracity (Etzion & Aragon-Correa, 2016, p. 148).

The boom in the data brings many new opportunities, yet with more challenges (Etzion & Aragon-Correa, 2016, p. 152). For example, “the challenges multiply when data from multiple sources are combined, particularly where the identifiers used by the underlying systems are not the same. The risks are particularly serious where the data are sensitive” (Clarke, 2015, p. 82). So the old methods could not deal with that kind of data which is different qualitatively and quantitatively. Thus, finding new ways to handle the new challenges of big data is mandatory. Especially for data management, where it is the first gate to all other data-driven activities like Artificial Intelligence, Machine Learning, and Business Intelligence. For example, the 2018 Kaggle Machine Learning & Data Science survey showed that data professionals spend most of their time cleaning data (23%) (Hayes, 2019). Therefore, investing in building a modern data platform and methods to reduce that time is crucial.

Based on Gartner's report for the top 10 data and analytics trends for 2021, DataOps will enable organisations to operate data better and integrate data in the decision-making to deliver business value (Panetta, 2021). DataOps comes to fix the current data problems benefiting from the success of DevOps in the software industry, where it touches the three parts of the organisation's golden triangle: people, process, and technology (Moilanen et al., 2022, p. 140).

On the technology side, to handle the big data challenges, there are many solutions to build modern data platforms that will probably use various Cloud computing capabilities to handle that massive data (Márquez & Lev, 2016, p. 29). However, Cloud vendors lock-in risks like integration, incompatibility, and data portability are the top blockers of cloud adoption

(Opara-Martins et al., 2016, p. 18). Kubernetes and open-source Cloud-Native ecosystem provide many advantages not only on the technical level but also on the strategic level, like portability and using open standards to counter vendor lock-in issues where they, the Cloud-Native platforms, “truly deliver digital capabilities anywhere and everywhere” (Costello & Rimol, 2021).

This project aims to discover the current big data challenges and how processes, practices, and technology could help solve those challenges. The focus will be on two sides of the golden triangle: process and technology to build a blueprint and a proof-of-concept of a modern data platform using DataOps guidelines and Kubernetes open-source cloud-native software.

4. Project outline for the work that you propose to complete

The idea for this research arose from:
The idea emerged from working in the software industry for more than ten years, where I saw many companies struggle to handle the increasing amount of data using old methods.

The aims of the project are as follows:
The project aims to review DataOps methodologies for fixing data management problems and to take a step towards a blueprint for a modern data platform using open-source cloud-native software according to DataOps best practices.

The main research questions that this work will address include:
The research will try to answer the following questions:

1. What are the current challenges of managing big data? Moreover, how the data solutions evolved over time to handle the data increase.
2. What are DataOps methodologies? And how could it help manage the current massive amount of data?
3. What is a “Modern Data Platform”, what are its properties, and how important is it to use open-source cloud-native software to build it?

The software development/design work/other deliverable of the project will be:
The project will deliver an architecture blueprint of the open-source cloud-native software to build a modern data platform. In addition, it will implement a part of the architecture as a proof-of-concept.

The project will involve the following research/field work/experimentation/evaluation:

This work will require the use of specialist software:
This project does not require specialist software since all software used is open source and available free of charge.

This work will require the use of specialist hardware:
This project does not require any special hardware since it is an architectural project.

The project is being undertaken in collaboration with:
N\A

5. References

Cisco Visual Networking Index: Forecast and Trends, 2017–2022. (2019). Cisco Systems, Inc. Retrieved October 1, 2022, from https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf

Clarke, R. (2015, December 15). Big data, big risks. *Information Systems Journal*, *26*(1), 77–90. https://doi.org/10.1111/isj.12088

Costello, K., & Rimol, M. (2021, October 18). *Gartner Identifies the Top Strategic Technology Trends for 2022*. Gartner. Retrieved October 1, 2022, from https://www.gartner.com/en/newsroom/press-releases/2021-10-18-gartner-identifies-the-top-strategic-technology-trends-for-2022

Etzion, D., & Aragon-Correa, J. A. (2016, May 10). Big Data, Management, and Sustainability. *Organization & Environment*, *29*(2), 147–155. https://doi.org/10.1177/1086026616650437

Hayes, B. (2019, February 9). *How do Data Professionals Spend their Time on Data Science Projects?* Business Over Broadway. Retrieved October 1, 2022, from https://businessoverbroadway.com/2019/02/19/how-do-data-professionals-spend-their-time-on-data-science-projects/

Márquez, F. P. G., & Lev, B. (2016). Big Data Management. Springer Publishing. https://doi.org/10.1007/978-3-319-45498-6

Moilanen, J., Luhti, T., & Niilahti, J. (2022). *Deliver Value in the Data Economy*. MindMote Oy.

Opara-Martins, J., Sahandi, R., & Tian, F. (2016, April 15). Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, *5*(1). https://doi.org/10.1186/s13677-016-0054-z

Panetta, K. (2021, March 15). *Gartner Top Data and Analytics Trends for 2021*. Gartner. Retrieved October 1, 2022, from https://www.gartner.com/smarterwithgartner/gartner-top-10-data-and-analytics-trends-for-2021

7. Ethics

Does this project have any ethical or governance issues related to working with, studying or observing other people? (YES/NO)	NO

8. Confidentiality

Does this project have any issues of confidentiality or intellectual property? (YES/NO)	NO

Appendix B:

Deployment Screenshots

B.1 Deployment Resources

View of the Kubernetes resources of the deployed application using OpenLens 6.4.0 and Lens Resource Map 1.0.1 (Lens extension).

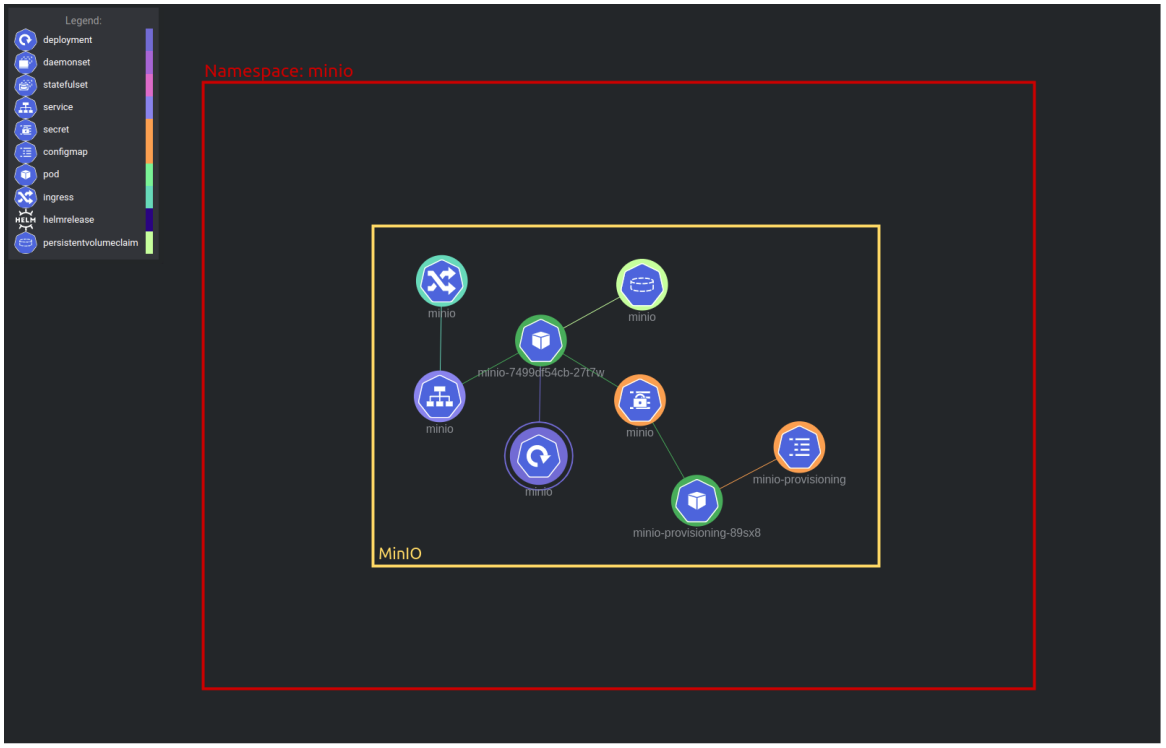


Figure B.1: MinIO deployed Kubernetes resources.

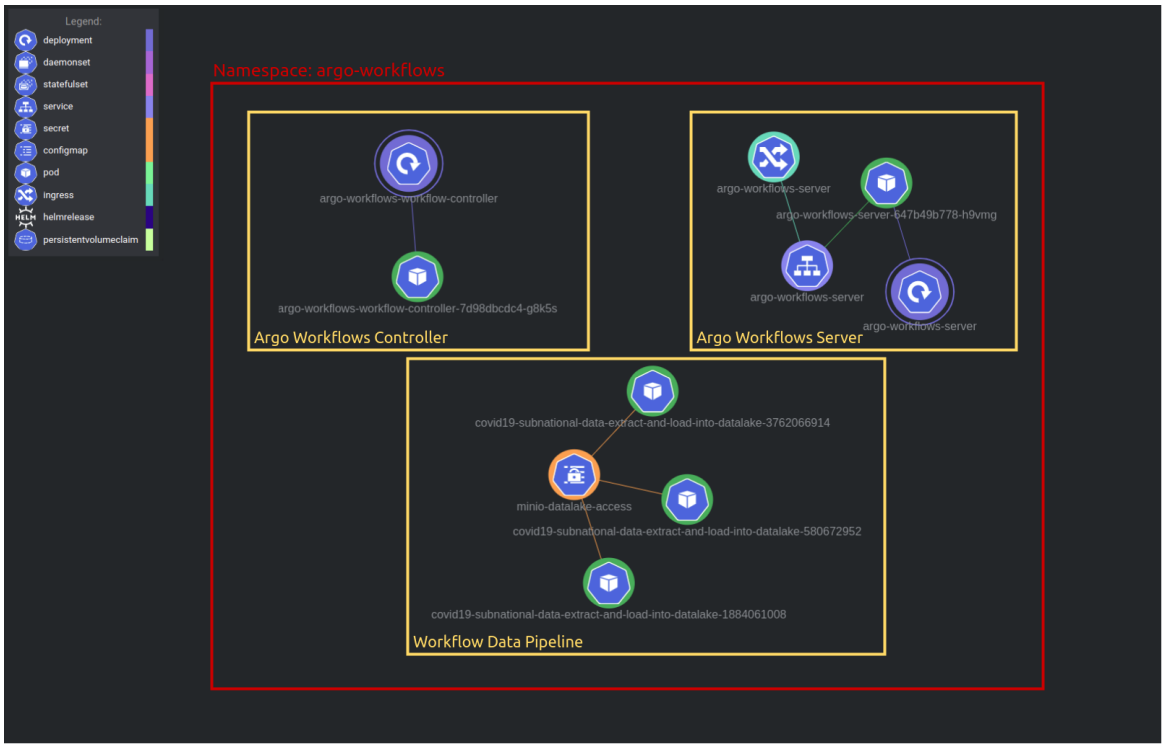


Figure B.2: Argo Workflows deployed Kubernetes resources.

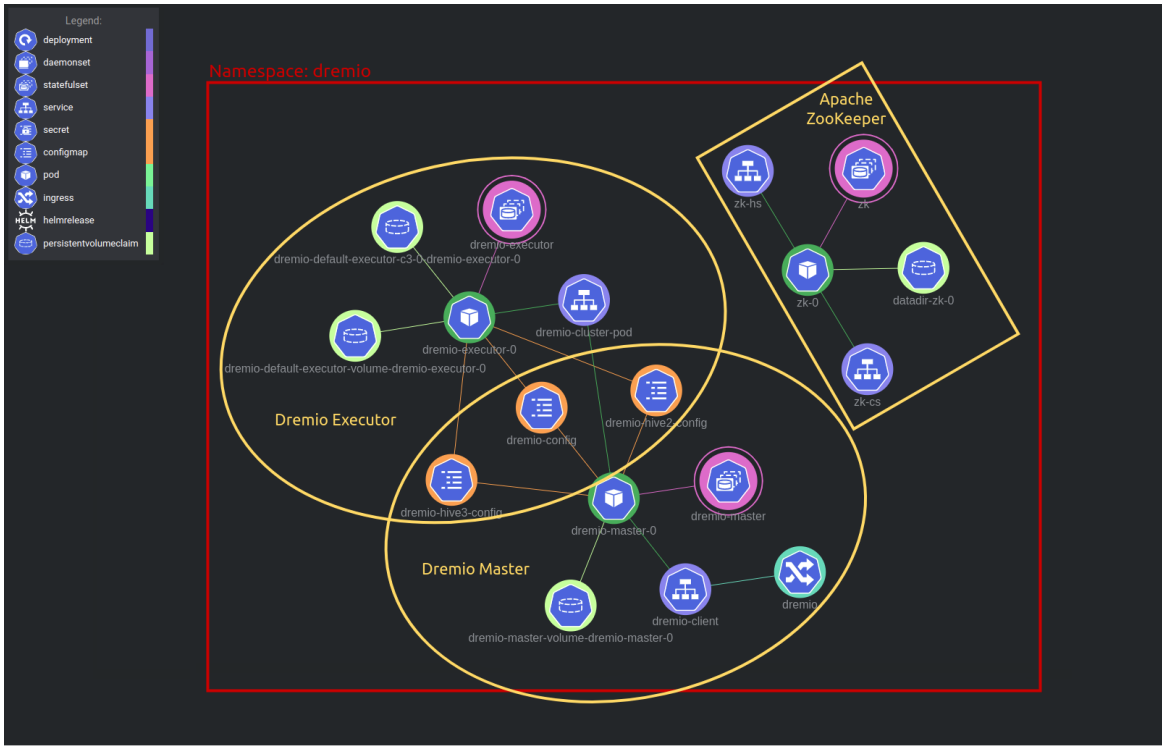


Figure B.3: Dremio deployed Kubernetes resources.

B.2 Applications Login Screen

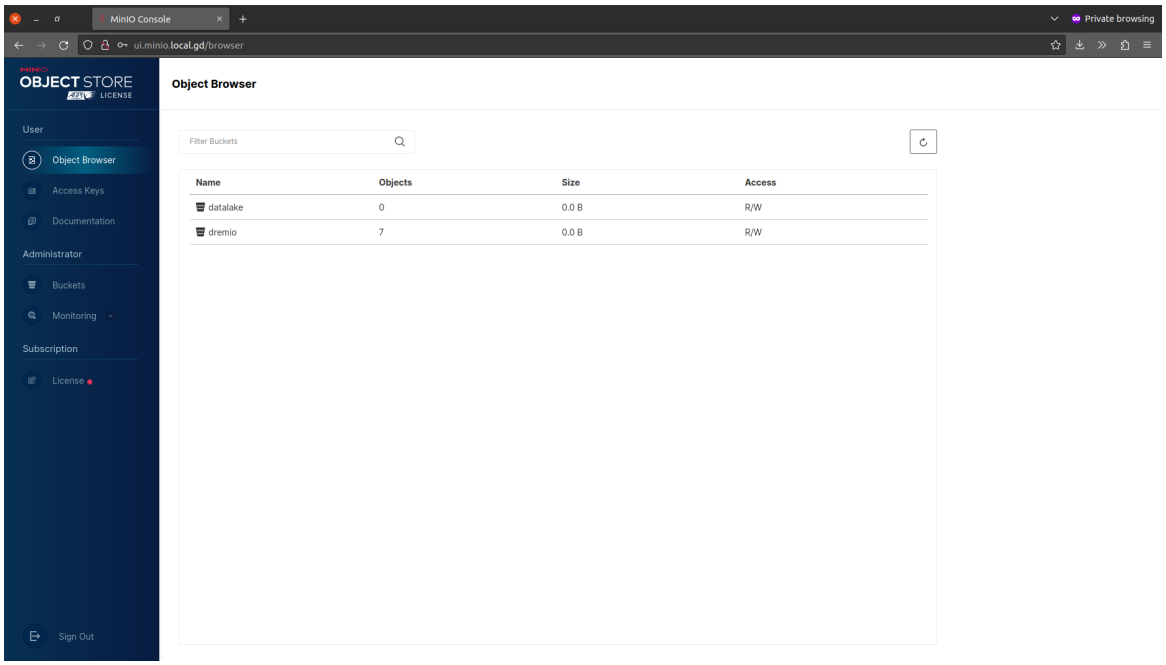


Figure B.4: MinIO interface after the login.

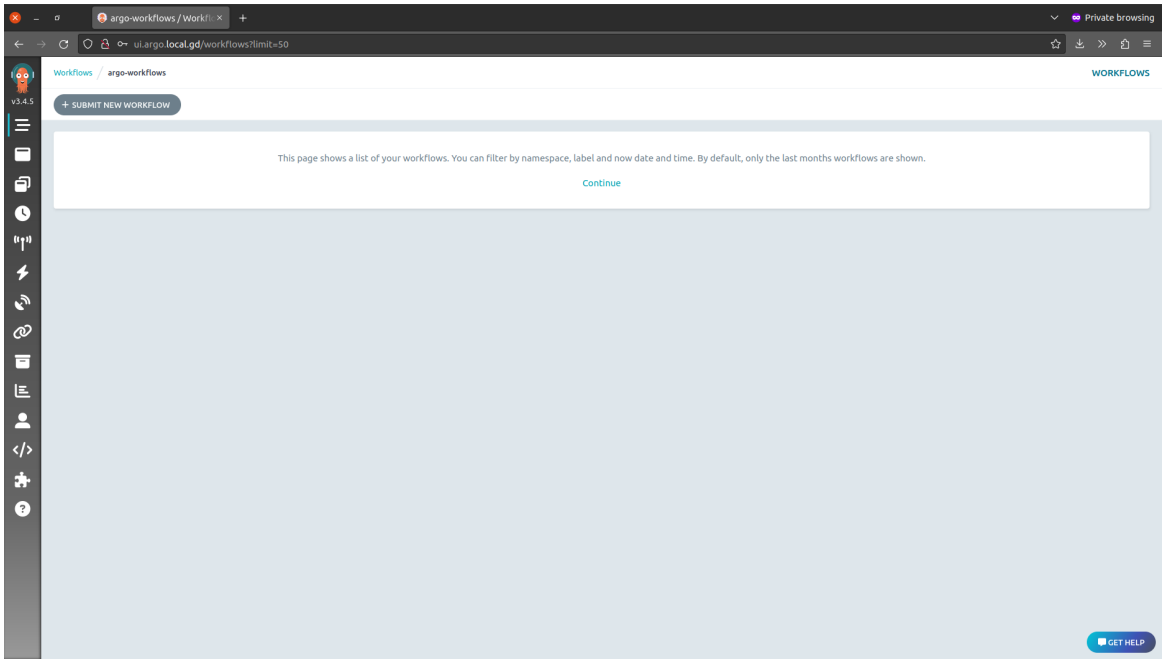


Figure B.5: Argo Workflows interface after the login.

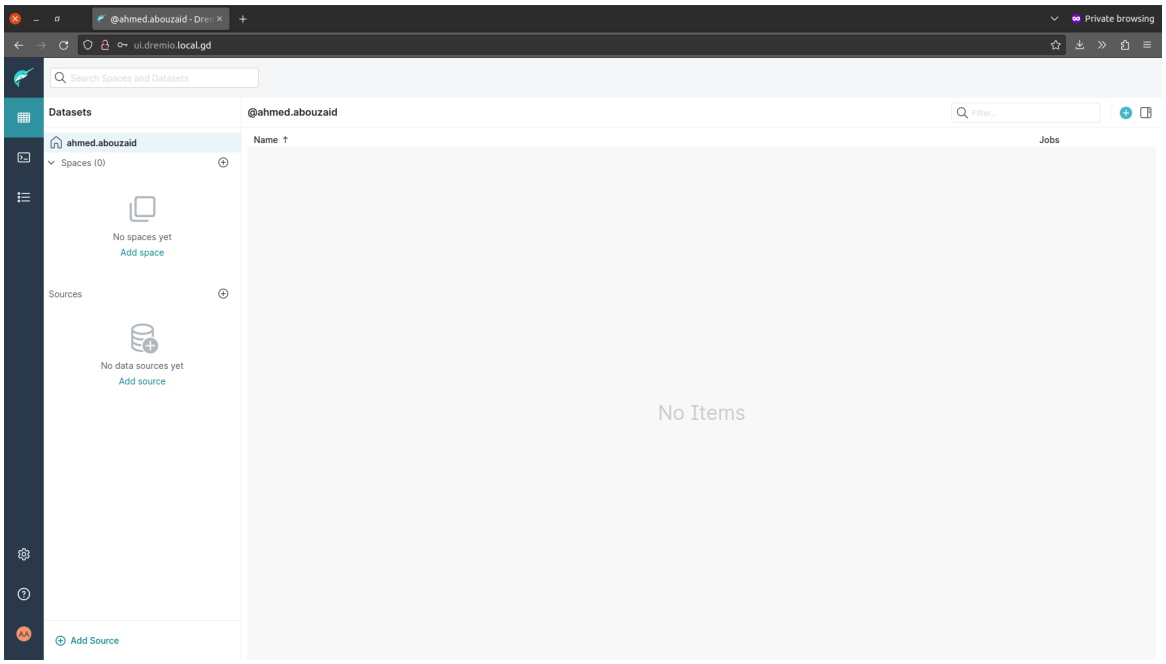


Figure B.6: Dremio interface after the login.

B.3 Dremio Source Congifureation

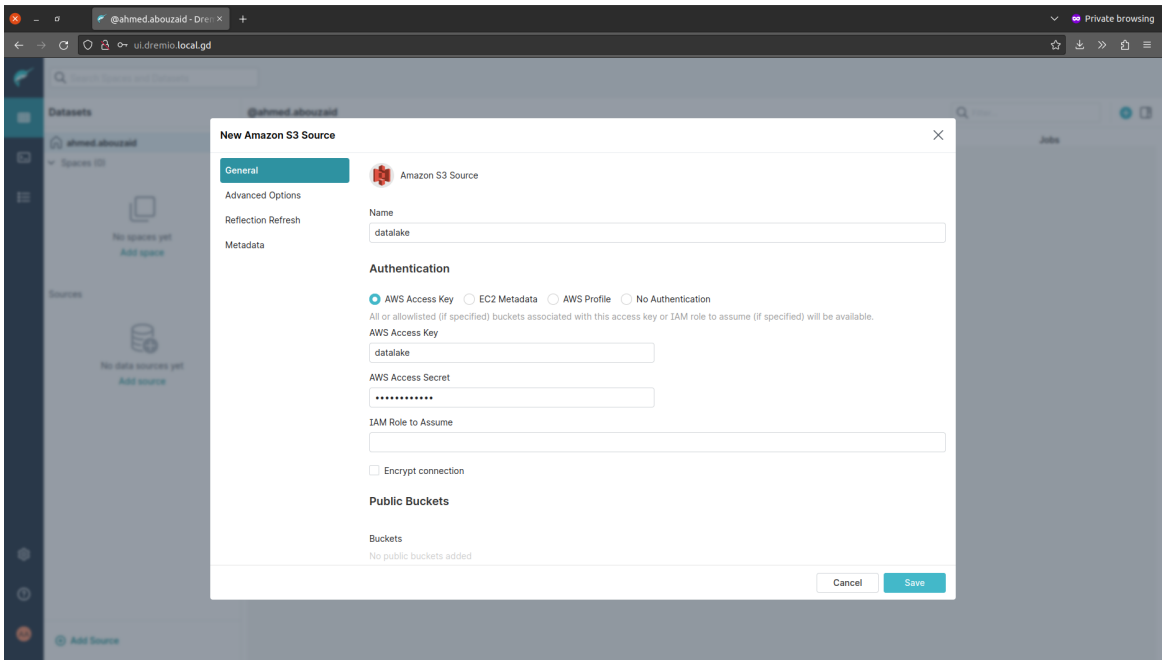


Figure B.7: Adding MinIO as a data source in Dremio - General.

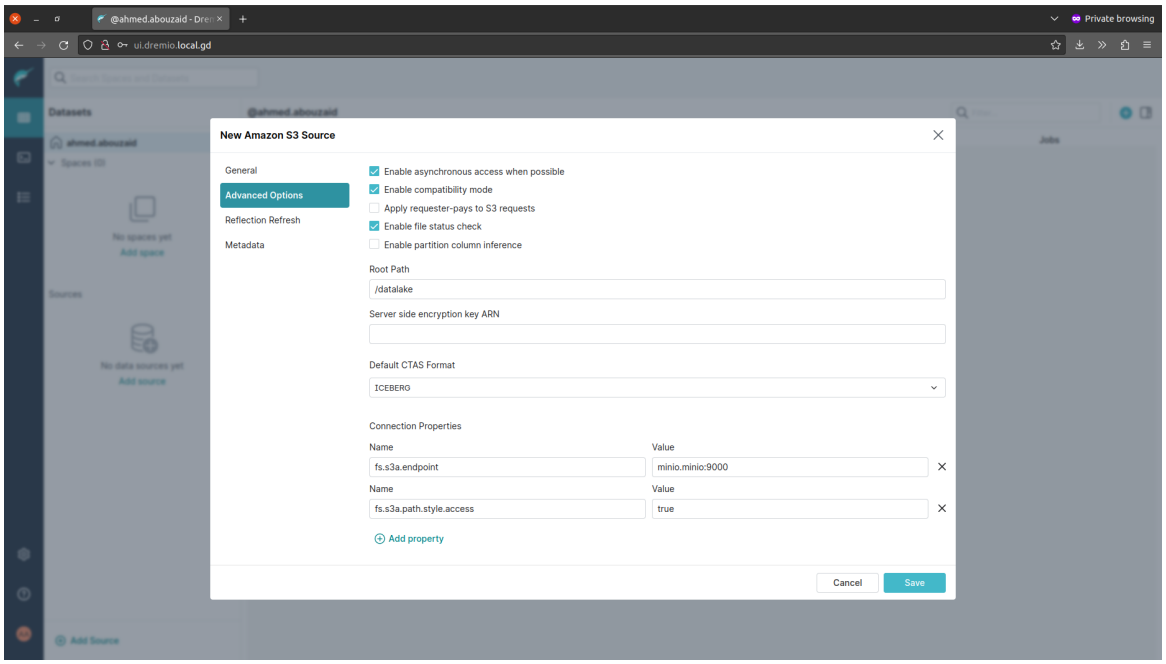


Figure B.8: Adding MinIO as a data source in Dremio - Advanced Options.

Appendix C:
Benchmarking Plots Code

```
# -*- coding: utf-8 -*-

# Install dependencies.
!pip install pywaffle

'''
This is a performance comparison between cold queries (without cache)
and warm queries (with cache) based on the Apache JMeter benchmark results.

The TPC-DS benchmark data for Dremio was generated according to this guide:
https://www.dremio.com/blog/dremio-benchmarking-methodology

Benchmark result files could be loaded from the project repo:
https://github.com/aabouzaid/modern-data-platform-poc/tree/main/benchmark
'''
import matplotlib.pyplot as plt
import pandas as pd
from io import StringIO as sio
from matplotlib.offsetbox import AnchoredText as obat
from pywaffle import Waffle

dataset_size = '10g'
dremio_version = '24.0.0'
plot_main_title = f'TPC-DS queries benchmark for Dremio v{dremio_version} - Dataset
{dataset_size}'

# --------------- #
# Load
# --------------- #

# Inline data to make it self-contained Jupyter Notebook.
# df_cold_csv = '''...'''
# df_warm_csv = '''...'''
# df_cold = pd.read_csv(sio(df_cold_csv), usecols=['timeStamp', 'elapsed', 'label'])
# df_warm = pd.read_csv(sio(df_warm_csv), usecols=['timeStamp', 'elapsed', 'label'])

df_cold = pd.read_csv(f'dremio-v{dremio_version}-tpc-ds-{dataset_size}-cold.csv',
usecols=['timeStamp', 'elapsed', 'label'])
df_warm = pd.read_csv(f'dremio-v{dremio_version}-tpc-ds-{dataset_size}-warm.csv',
usecols=['timeStamp', 'elapsed', 'label'])

# Get details about the test execution duration.
exec_duration_cold_sec = ((df_cold['timeStamp'].iloc[-1] -
df_cold['timeStamp'].iloc[0]) / 1000).round(2)
exec_duration_warm_sec = ((df_warm['timeStamp'].iloc[-1] -
df_warm['timeStamp'].iloc[0]) / 1000).round(2)

# Create a dataframe for test execution duration.
exec_duration_df = pd.DataFrame({
        'exec': [
        f'First Run\n({exec_duration_cold_sec}s)',
        f'Second Run\n({exec_duration_warm_sec}s)'
        ],
        'duration_sec': [exec_duration_cold_sec, exec_duration_warm_sec]
})
```

```
# Calculate percentage differences between the first and the second run.
exec_duration_diff_pct =
exec_duration_df[['duration_sec']].pct_change()['duration_sec'].iloc[1].round(2) *
100

#------------------------#
# Plot execution duration #
#------------------------#

exec_duration = exec_duration_df.plot.barh(x='exec', y='duration_sec', legend=False,
figsize=(12, 8))
exec_duration.invert_yaxis()
exec_duration.set_title(f'{plot_main_title}\n(Test execution duration in seconds)',
pad=20, size=16)
exec_duration.set_xlabel(f'Execution duration in seconds (Diff:
{exec_duration_diff_pct} %)', labelpad=10)
exec_duration.set_ylabel('')
exec_duration.xaxis.grid(color='gray', linestyle='dashed')
exec_duration.set_axisbelow(True)
for color, bar in zip(['tab:blue', 'tab:orange'], exec_duration.patches):
        bar.set_color(color)

# --------------- #
# Prepare
# --------------- #

# Merge cold and warm benchmark results in one dataframe.
df_combined = df_cold.merge(df_warm, on='label', how='left', suffixes=('_cold',
'_warm'))

# Calculate percentage enhancement between cold and warm results.
df_combined['elapsed_diff_pct'] = ((df_combined[['elapsed_cold',
'elapsed_warm']].pct_change(axis=1)['elapsed_warm']) * 100).round(2)

# Create a new dataframe with elapsed time in seconds (JMeter's elapsed time unit is
in milliseconds).
df_cleaned = pd.DataFrame({
        'duration_cold_sec': df_combined['elapsed_cold'].apply(lambda x: x /
1000).tolist(),
        'duration_warm_sec': df_combined['elapsed_warm'].apply(lambda x: x /
1000).tolist(),
        'duration_diff_pct': df_combined['elapsed_diff_pct'].tolist(),
        },
        index=df_combined['label']
)

print(df_cleaned.sample(5).to_csv(sep="\t"))

# --------------- #
# Queries Plot
# --------------- #

plt.rcParams.update({
        # Set background for all plots.
        "figure.facecolor": 'white'
})
```

```
#-------------------------#
# Plot TPC-DS tests status #
#-------------------------#

tpcds_tests_data = {'Unsupported': 43, 'Succeeded': 46, 'Failed': 10}
tpcds_tests_labels = [f"{k}: {v} ({v / 99 * 100:.1f}%)" for k, v in
tpcds_tests_data.items()]

tpcds_tests = plt.figure(
       FigureClass=Waffle,
       figsize=(8, 6),
       rows=9,
       columns=11,
       values=tpcds_tests_data,
       labels=tpcds_tests_labels,
       legend={'loc': 'upper left', 'bbox_to_anchor': (1, 1)},
       colors=["darkgray", 'tab:green', 'tab:red'],
)

tpcds_tests.suptitle(f'{plot_main_title}\n(Status of the TPC-DS 99 queries)',
fontsize=16)

#-------------------------------------------#
# Plot queries performance with cache enabled #
#-------------------------------------------#

query_performance_df = pd.DataFrame({
       'queries': [len(df_cleaned[df_cleaned['duration_diff_pct'] < 0]),
              len(df_cleaned[df_cleaned['duration_diff_pct'] > 0])],
       },
       index=['Improved', 'worsened']
)

# Show the percentage and actual value at the same time.
def autopct_format(pct):
       total = sum(query_performance_df['queries'])
       return '{:.1f}%\n({:.0f})'.format(pct, total * pct / 100)

query_performance = query_performance_df.plot.pie(y='queries', colors=['tab:green',
'tab:red'], autopct=autopct_format, figsize=(12, 8))
query_performance.set_title(f'{plot_main_title}\n(Queries performance with cache
enabled)', pad=20, size=16)
query_performance.set_xlabel(f'Queries (Total:
{query_performance_df["queries"].sum()})', labelpad=10)
query_performance.set_ylabel('')

#-------------------------------------------------------------#
# Plot overview of the queries performance with cache enabled #
#-------------------------------------------------------------#

diff_pct_df = df_cleaned.sort_values(by=['duration_diff_pct'])['duration_diff_pct']
diff_pct = diff_pct_df.plot.line(color='darkgray', style='.-', rot=0, figsize=(24,
12))
diff_pct.yaxis.grid(color='gray', linestyle='dashed')
diff_pct.set_axisbelow(True)
diff_pct.set_title(f'{plot_main_title}\n(Overview of queries performance with cache
```

```
enabled)', pad=20, size=16)
diff_pct.set_xlabel('Query ID', labelpad=20)
diff_pct.set_ylabel('Query duration diff percentage (lower is better)', labelpad=20)
xticks = range(diff_pct_df.count())
diff_pct.set_xticks(xticks)
diff_pct.set_xticklabels(diff_pct_df.index.tolist())
diff_pct.axhline(0, color='gray')

for index in xticks:
        diff_pct.text(index-0.8, diff_pct_df[index]+0.8,
'{0:.3g}%'.format(diff_pct_df[index]), horizontalalignment='center', rotation=-25)

# Add summary box.
diff_pct_stat = (
        f'Min (best): {diff_pct_df.min()}%\n'
        f'Q1 (25% quantile): {diff_pct_df.quantile(0.25)}%\n'
        f'Median: {diff_pct_df.median()}%\n'
        f'Q3 (75% quantile): {diff_pct_df.quantile(0.75)}%\n'
        f'Max (worst): {diff_pct_df.max()}%'
)
diff_pct_details = obat(diff_pct_stat, prop=dict(size=14), frameon=True, loc='upper
left')
diff_pct_details.patch.set(boxstyle='square', edgecolor='darkgray')
diff_pct.add_artist(diff_pct_details)

#--------------------------------------#
# Plot queries performance distribution #
#--------------------------------------#

# Base.
fig, ax = plt.subplots(2, figsize=(24, 12), sharex=True,
gridspec_kw={"height_ratios": (.2, .8)})
fig.suptitle(f'{plot_main_title}\n(Detailed view of queries performance with cache
enabled)', size=16)
plt.subplots_adjust(hspace=0.1)

# Summary.
diff_pct_details = obat(diff_pct_stat, prop=dict(size=14), frameon=True, loc='upper
right')
diff_pct_details.patch.set(boxstyle='square', edgecolor='darkgray')
ax[1].add_artist(diff_pct_details)

# Boxplot.
diff_pct_box = diff_pct_df.plot.box(color='darkgray', vert=False,
medianprops={'color': 'gray'}, patch_artist=True, widths = 0.5, ax=ax[0])
diff_pct_box.axvline(0, color='gray')
diff_pct_box.set_yticks([])

# Histogram.
diff_pct_hist = diff_pct_df.plot.hist(color='darkgray', edgecolor='gray', bins=20,
xlim=[-100, 100], ax=ax[1])
diff_pct_hist.set_xticks(range(-100, 100, 10))
diff_pct_hist.axvline(0, color='gray')
diff_pct_hist.set_xlabel('Query duration diff percentage (lower is better)',
labelpad=20)
```

```python
# Distribution line.
diff_pct_kde = diff_pct_df.plot.kde(color='gray', ax=diff_pct_hist.twinx())
diff_pct_kde.set_yticks([])
diff_pct_kde.set_ylabel('')

# Set labels with percentage enhancement to make it easy to plot bars.
df_cleaned['label_diff_pct'] = df_cleaned.assign(label=df_cleaned.index)[['label',
'duration_diff_pct']].apply(
        lambda x : f'{x[0]} (diff: {x[1]}%)', axis=1)
df_cleaned.head()

#----------------------------------------#
# Plot the best 10 queries by using cache #
#----------------------------------------#

best_queries_df = df_cleaned.nsmallest(10, 'duration_diff_pct')
best_queries = best_queries_df.plot.bar(y=['duration_cold_sec', 'duration_warm_sec'],
x='label_diff_pct', rot=0, figsize=(24, 12))
best_queries.legend(["Cold query (without cache)", "Warm query (with cache)"])
best_queries.yaxis.grid(color='gray', linestyle='dashed')
best_queries.set_axisbelow(True)
best_queries.set_title(f'{plot_main_title}\n(Top 10 best performing queries when
cache enabled)', pad=20, size=16)
best_queries.set_xlabel('Query ID (with percentage enhancement)', labelpad=20)
best_queries.set_ylabel('Query duration in seconds', labelpad=20)
for container in best_queries.containers:
        best_queries.bar_label(container)

#----------------------------------------#
# Plot the worst 10 queries by using cache #
#----------------------------------------#

worst_queries_df = df_cleaned.nlargest(10, 'duration_diff_pct')
worst_queries = worst_queries_df.plot.bar(y=['duration_cold_sec',
'duration_warm_sec'], x='label_diff_pct', rot=0, figsize=(24, 12))
worst_queries.legend(["Cold query (without cache)", "Warm query (with cache)"])
worst_queries.yaxis.grid(color='gray', linestyle='dashed')
worst_queries.set_axisbelow(True)
worst_queries.set_title(f'{plot_main_title}\n(Top 10 worst performing queries when
cache enabled)', pad=20, size=16)
worst_queries.set_xlabel('Query ID (with percentage enhancement)', labelpad=20)
worst_queries.set_ylabel('Query duration in seconds', labelpad=20)
for container in worst_queries.containers:
        worst_queries.bar_label(container)
```

Figure C.1: Python code for the benchmarking plots.[7]

[7] Also available on the project Git repository:
https://github.com/aabouzaid/modern-data-platform-poc/tree/main/benchmark