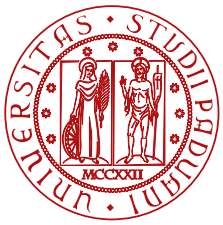

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTERS DEGREE IN COMPUTER ENGINEERING

Re-engineering Legacy Data Platforms with Cloud-Native Technologies

Advisor
Prof. Carlo Ferrari

Candidate
Vaidas Lenartavicius

Company Tutor
Mariagrazia Cardile

ACADEMIC YEAR 2023-2024

Graduation Date 27/11/2024

”You can have data without information,
but you cannot have information without data.”
– Daniel Keys Moran

Summary

This thesis is based on the work done during the internship at Data Reply, a consulting company part of the Reply group, focused on Big Data Engineering and Data Science.

The main objective of the work is the re-engineering and migration of a legacy data platform for an IT services company using state of the art cloud computing infrastructure and tools. The project is motivated by the increasing need for scalable, flexible, and cost-efficient data solutions in modern enterprises, where legacy systems often fall short in meeting current demands.

The Snowflake data warehousing platform is a particular focus of this thesis, because this project is the first instance of Data Reply opting to utilize Snowflake over its traditional competitors. As a result, a substantial portion of the project involved an extensive process of feature discovery and evaluation. This was key not only for understanding Snowflake's unique capabilities but also for ensuring that the platform could meet the specific needs of the client's data environment. The exploratory nature of this work added a layer of complexity to the project, as it required thorough research, testing, and documentation to establish best practices and optimize the platform's use in a real-world context.

Contents

List of Figures

Chapter 1

Introduction

Like most fields concerning information technology, the landscape of data storage, processing, and analytics is continuously and rapidly evolving. This evolution is driven by two equally important causes: the increase of resources and tools available to developers of solutions, and the emergence of new kinds of problems to be solved.

Developers and engineers who design solutions in this field have to strike a balance: on one hand, there is the experience accumulated day by day on what has and hasn't worked in the past, while on the other hand, new tools and approaches are entering the landscape at an ever increasing pace.

The work described in this thesis is about finding the right balance between leveraging innovative technologies and relying on proven, established approaches within the context of an enterprise project for a client company.

The project focuses on the re-engineering and modernization of a legacy on-premises data platform into a scalable, cloud-based solution. This transformation aims to enhance data accessibility, improve processing speeds, and reduce operational costs by leveraging cloud technologies. The existing platform, built on outdated infrastructure, faces challenges related to performance, maintenance, and limited scalability. By moving to a cloud environment, the project seeks to streamline data workflows, integrate advanced analytics capabilities, and provide the client with a more agile and resilient data architecture that can adapt to evolving business needs.

1.1 Personal Effort

The work performed for this thesis can be divided into the following phases:

- Feature and design discovery of various tools and systems that were considered for the project. The Snowflake data warehousing platform required the most in-depth exploration, since this was the first time Snowflake was being implemented in a company

project, I was tasked with becoming the in-house expert, conducting thorough research, hands-on experimentation, and feature analysis to ensure its successful integration and use.

- Analysis and reverse engineering of the existing legacy data platform to fully understand its architecture and workflows. This involved dissecting undocumented processes, identifying key data flows, and evaluating performance bottlenecks, all with the goal of ensuring that the new solution would effectively address the limitations of the legacy system while maintaining critical functionalities.

- Design and implementation of the new data platform, focusing on creating a scalable, efficient architecture that addressed the shortcomings of the legacy system. Definition of data ingestion pipelines, optimization of query performance, and seamless integration with existing downstream tools, all done incorporating best practices for cloud-based data warehousing.

- Validation, testing, and performance analysis of the new solution to ensure it met the project's requirements. Done by assessing data accuracy, query performance, and scalability to verify the platform's ability to handle production workloads.

1.2 Data Warehousing

A data warehouse is an enterprise system used for the analysis and reporting of structured and semi-structured data from multiple sources, such as point-of-sale transactions, marketing automation, customer relationship management, and more. It is suited for ad hoc analysis as well as custom reporting and can store both current and historical data in one place. Data warehouses are designed to give a long-range view of data over time, making it a primary component of business intelligence [1].

While the term ”data warehouse” was first introduced in the late 1980s, wide adoption of data warehouses as part of the business intelligence pipeline started in the 2000s. There are two main reasons for this timing[2]:

- With the Internet becoming more and more mainstream, companies of all sizes started increasingly relying on web-based operations, and as a consequence started collecting and generating large amounts of potentially useful user data.

- Before the emergence of cloud-based computing, data warehouses were financially viable exclusively for very large corporations with proportionately large IT budgets, due to the costs of acquiring and maintaining a large scale physical infrastructure.

The emergence of cloud computing in itself was not enough to enable businesses to start their own data warehouses. Data warehousing software that pre-dates the cloud was designed for small static clusters and would fit the architecture of the new cloud-based platforms poorly. In addition, the data itself had changed, a much larger portion of it came from sources with much less predictable structure, volume, and rate compared to the ERP applications and the like that fed the previous generation of data warehouses [3].

Many companies turned to ”Big Data” platforms, such as Hadoop and Spark. Their biggest drawback would be the engineering effort required to set them up and use them [4].

1.3 Snowflake

Figure 1.1: The Snowflake logo

Snowflake was founded, in 2012, with the belief that there were many use cases that would benefit from the advantages of cloud computing, but were not well served by either the traditional data warehouses or by ”Big Data” platforms[3].

In contrast to many other systems in the cloud data management space, Snowflake is not based on Hadoop, PostgreSQL, or the like. The processing engine and most of the other parts have been developed from scratch.

1.3.1 Design and Features

These are the principles that guided Snowflake’s design and development:

- **Pure Software-as-a-Service (SaaS) Experience**: Fully managed, requiring no hardware or software installation, with automatic scaling and updates, allowing users to focus solely on data operations.
- **Relational**: Provides a traditional SQL-based environment for structured data, supporting ACID-compliant transactions and familiar database operations.
- **Semi-Structured**: Natively handles semi-structured data formats like JSON, Avro, and Parquet, allowing for efficient querying and storage without complex transformations.

- **Elastic**: Automatically scales up or down based on workload demands, ensuring performance without manual intervention, and allowing for seamless concurrency and resource optimization.

- **Highly Available**: Built-in redundancy across multiple availability zones ensures minimal downtime and high availability for mission-critical applications.

- **Durable**: Ensures data integrity with multiple layers of protection, replicating and storing data redundantly to safeguard against data loss or corruption.

- **Cost-efficient**: Pay-per-use pricing model with separate compute and storage, allowing businesses to optimize costs by scaling resources up or down as needed.

- **Secure**: Provides robust security features including end-to-end encryption, role-based access control, and compliance with key regulatory standards such as GDPR and HIPAA.

Snowflake is a fully cloud-agnostic data platform, meaning it can be deployed on all three major cloud providers: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). When setting up a Snowflake account, the cloud provider that Snowflake will run on can be chosen. Regardless of the provider chosen, Snowflake's architecture and functionality remain the same.

Choosing to run Snowflake on the same cloud provider that is already used for other parts of your data pipeline can offer significant advantages:

- **Data Egress Costs**: When data is stored on one cloud provider (e.g., AWS S3, Azure Blob Storage, or Google Cloud Storage) and is processed on a different platform, typically data egress charges are incurred for transferring data between clouds. By deploying Snowflake on the same cloud where the data is stored, these transfer fees are eliminated, as all data movements stay within the same cloud infrastructure.

- **Network Latency and Performance**: Running Snowflake on the same cloud provider as other services in the pipeline (such as data sources, BI tools, or ETL processes) reduces network latency, leading to faster data processing and better query performance. This can result in less compute time and, ultimately, lower costs.

- **Integrated Cloud Ecosystem**: By leveraging a single cloud provider, built-in integrations between Snowflake and other native services (e.g., AWS Lambda, Azure Functions, or Google Pub/Sub) can be employed. These integrations eliminate the need for third-party tools or custom APIs.

1.3.2 Architecture

Snowflake is a service-oriented architecture composed of highly fault tolerant and independently scalable services. These services communicate through RESTful interfaces and fall into three architectural layers:

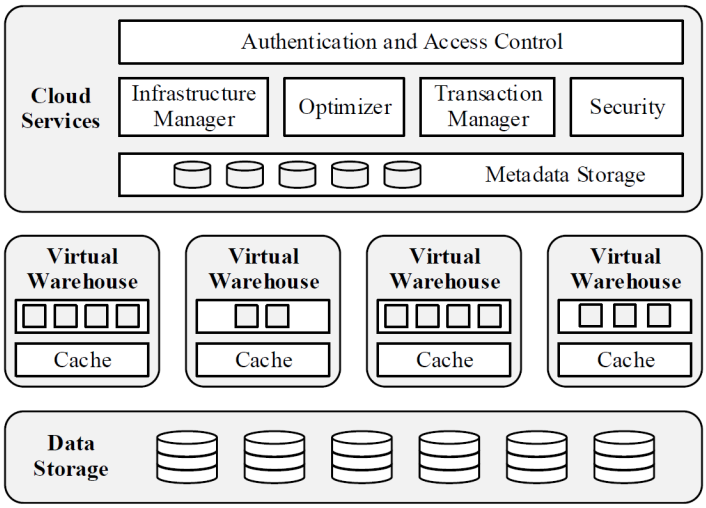

Figure 1.2: Snowflake's three tier architecture

- **Data Storage**: The Data Storage layer in Snowflake is designed to handle large volumes of structured and semi-structured data. When data is loaded into Snowflake, it is automatically divided into micro-partitions and stored in a columnar format, optimized for fast retrieval and compression. Unlike traditional systems, Snowflake separates storage from compute, meaning data is stored independently of the compute resources used to process it. This layer is entirely managed by Snowflake, abstracting away the complexity of managing indexes, partitions, or any physical storage optimizations. Data stored in this layer is automatically compressed, encrypted, and redundantly stored across multiple availability zones for durability and reliability, ensuring high availability and protection against data loss.

- **Virtual Warehouses**: The Virtual Warehouses layer represents the compute resources in Snowflake, responsible for executing queries and performing data processing tasks. Each virtual warehouse is an independent MPP (Massively Parallel Processing) compute cluster that scales dynamically based on workload demands. Users can spin up multiple warehouses of varying sizes depending on the performance requirements, allowing for concurrent query processing without contention. Importantly, virtual warehouses can be resized or paused to manage costs, and because Snowflake separates compute from storage, multiple warehouses can access the same underlying data simultaneously without

impacting performance. This layer provides the flexibility to adjust compute resources elastically, ensuring optimal performance for various workloads, including data loading, querying, and complex analytics.

- **Cloud Services**: The Cloud Services layer in Snowflake coordinates and manages all the activities within the platform, acting as the control layer. It handles tasks such as metadata management, query optimization, authentication, security enforcement, and transaction management. This layer leverages Snowflake's central repository of metadata to optimize query execution plans, ensuring efficient data access and minimizing compute resource usage. It also provides services like auto-scaling, concurrency management, and monitoring, which are automatically handled without user intervention. Built on top of cloud infrastructure, this layer ensures Snowflake operates seamlessly across various cloud providers (AWS, Azure, and GCP) while providing a consistent and secure user experience.

1.3.3 Evolution

Snowflake is a rapidly evolving data platform that has undergone significant growth and innovation since its initial release. While it was originally designed as a cloud-native solution for managing structured and semi-structured data, the platform has continuously expanded its capabilities to address a broader range of use cases, making it one of the most versatile data warehousing solutions available today.

Figure 1.3: A High-Level Snowflake Overview (Source: John Ryan, 2024)

As the demands of modern data ecosystems have evolved, Snowflake has kept pace by introducing new features and enhancing its core functionality, allowing it to stay at the forefront of the industry.

In addition to the foundational capabilities described in earlier sections, Snowflake now boasts a rich set of advanced features that cater to various aspects of data management, processing, and security. These enhancements have not only extended the platform's usability but have also improved its performance, flexibility, and integration with other technologies.

- **Snowpark**: At its core Snowflake is an SQL-based platform. As a declarative language, SQL is very powerful in allowing users from all backgrounds to ask questions about data. However, the complex logic of large-scale applications and pipelines can become difficult to understand in SQL alone.

  Snowpark is a development framework added to Snowflake that allows users to write data pipelines and complex data transformations in languages like Java, Scala, and Python. This feature enhances the platform's flexibility by enabling developers to work in their preferred programming environments. Most importantly, Snowpark helps developers leverage Snowflake's computing power to ship their code to the data rather than exporting data to run in other environments where big data is a second-class citizen. This can be a major optimization[5].

- **Time Travel**: A feature which provides the ability to access and analyze data from a previous state, enabling administrators to easily recover from accidental data modifications or deletions. It also simplifies auditing and compliance processes by allowing users to track changes over time and revert to specific points in the data's history. This feature offers flexibility and security, reducing the need for complex backup strategies while maintaining data integrity[6].

- **Unstructured Data Support**: Initially focused on structured and semi-structured data, Snowflake added support for unstructured data storage and processing, including files like PDFs, images, and videos. This development expands Snowflake's use cases to applications such as media analytics and broader data lakes[7].

- **Data Cloning**: A way to create a copy of any table, schema, or an entire database without incurring any additional costs as the derived copy shares the underlying storage with the original object[8].

- **Dynamic Data Masking**: Tools to control and secure sensitive data, allowing administrators to define which data should be obfuscated based on user roles. This feature strengthens Snowflake's data governance and compliance capabilities, enabling its use in industries with strict security requirements[9].

- **Snowflake Data Marketplace**: A centralized space on the platform that allows third party providers to come up with and publish their own data product offerings to the entire Snowflake Data Cloud.

  Providers have a way of showcasing their offerings to multiple consumers at once, along with a provision of providing a sample of their service offerings and charge users if they want complete access, thus having a chance of revenue generation.

  Consumers can access raw datasets that they can transform and perform analysis in combination with their own organizational data. A key benefit of this arrangement is that while consumers do gain access to the database, it isn't a static version of the dataset that they use. Instead, Snowflake employs the concept of "shared datasets," through which the database is shared from the provider's account and appears alongside the user's databases. This feature allows consumers to have access to the most recent version of the data[10].

These examples represent a small selection of the many features that have been introduced to Snowflake since its release. As the platform continues to evolve, new capabilities are regularly added, enhancing its ease of use, scalability, and number of use cases.

1.4 Cloud Computing

Cloud computing has become a foundational technology in modern IT[11], providing businesses with flexible access to computing resources over the internet. Instead of maintaining on-premises hardware, organizations can now rent computing power, storage, and other services from cloud providers. This shift has significantly changed the way IT infrastructure is managed, making it easier to scale operations and reduce the need for extensive in-house infrastructure.

Cloud services are typically categorized into different models based on the type of service offered, such as[12]:

- **Infrastructure as a Service (IaaS)** provides virtualized computing resources, such as servers, storage, and networking, giving users full control over their infrastructure without the need to manage physical hardware.
- **Platform as a Service (PaaS)** builds on IaaS by offering development platforms that allow users to create, manage, and deploy applications without worrying about the underlying infrastructure.
- **Software as a Service (SaaS)** offers fully developed applications hosted and maintained by a cloud provider, allowing users to access the software through a browser or client application without handling any backend management.

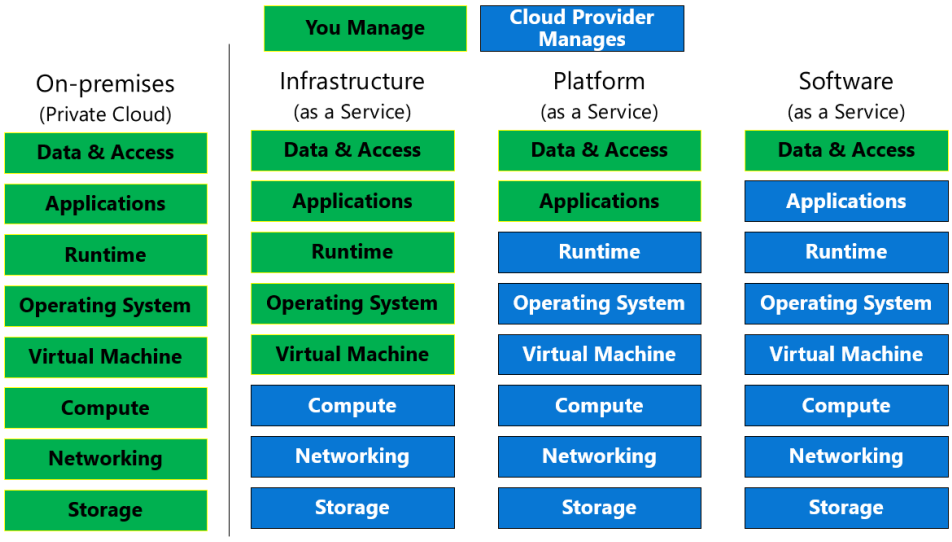

Figure 1.4: IaaS, Paas, SaaS Overview (Source: Jiadong Chen, 2020)

The major players in the cloud computing market are Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), each offering a range of different solutions. There is significant overlap between the services they offer, but each provider has its strengths and weaknesses. These providers enable companies to access scalable, on-demand resources, supporting a broad spectrum of needs from simple storage solutions to complex application hosting and machine learning workloads.

By reducing the need for physical infrastructure, cloud computing has led to more agile IT operations and enabled businesses to focus on core objectives rather than managing servers and data centers. As cloud technology continues to mature, it is becoming an integral part of how organizations operate and evolve.

1.5 Amazon Web Services (AWS)

Amazon Web Services (AWS) is currently the largest and most widely adopted cloud platform[11], offering a broad set of tools and services to enable organizations to build scalable and cost-efficient applications.

AWS's global network of data centers ensures high availability, fault tolerance, and geographic redundancy. In this project, several key AWS services play an integral role in managing data flows, performing compute tasks, and ensuring seamless communication within the cloud ecosystem.

Figure 1.5: Amazon Web Services Logo

1.5.1 Amazon S3

Amazon Simple Storage Service (S3) is a highly scalable object storage service designed to store and retrieve any amount of data from anywhere on the web[13]. S3 is widely used for storing a variety of data types, including backups, data lakes, and application assets. It ensures high durability by automatically replicating data across multiple Availability Zones (AZs) and offers various storage classes to optimize costs depending on access patterns (e.g., S3 Standard, S3 Glacier for long-term archival).

Data stored in S3 is accessible over the internet via RESTful APIs, and it integrates seamlessly with other AWS services like Lambda, EC2, and with Snowflake. S3's built-in security features, such as encryption and access control policies, make it a popular choice for businesses with stringent compliance requirements.

1.5.2 AWS Lambda

AWS Lambda is a serverless compute service that allows users to run code in response to events without provisioning or managing servers[14]. Lambda automatically scales to handle incoming requests, charging users only for the compute time consumed, making it cost-effective for short-lived tasks and infrequent workloads. Lambda functions can be triggered by a wide range of AWS services, including S3, SNS, and DynamoDB, allowing for automation across the cloud infrastructure. This service supports multiple programming languages, including Python, Java, and Node.js, and can integrate easily with other AWS tools.

1.5.3 AWS Database Migration Service (DMS)

AWS Database Migration Service (DMS) is designed to facilitate the seamless migration of data from on-premises or other cloud-based databases to AWS[15]. DMS supports a variety of

database engines, including Oracle, MySQL, PostgreSQL, and SQL Server, and can handle both homogeneous and heterogeneous migrations. One of the key features of DMS is continuous data replication, which allows real-time syncing between source and target databases, minimizing downtime during migrations.

1.5.4 Amazon Simple Notification Service (SNS)

Amazon Simple Notification Service (SNS) is a fully managed messaging service that enables the decoupling of microservices and distributed systems[16]. It supports both push-based (publish-subscribe) and pull-based (message queuing) messaging models, allowing for flexible communication between different components of an application. SNS allows users to send notifications to a variety of endpoints, including email, SMS, HTTP/S, and AWS Lambda functions. With its built-in fault tolerance and scalability, SNS is often used to send alerts, trigger workflows, or distribute messages across services.

1.6 Terraform

Figure 1.6: Terraform Logo

Terraform is an open-source infrastructure as code (IaC) tool developed by HashiCorp that enables users to define, provision, and manage cloud infrastructure using declarative configuration files[17]. This approach allows teams to automate the process of setting up and maintaining their infrastructure, promoting consistency and reducing the potential for human error. By treating infrastructure as code, Terraform enables version control, collaboration, and auditing of infrastructure changes, much like how software development practices are applied to application code.

One of the key features of Terraform is its ability to work with a wide range of infrastructure providers, including major cloud platforms such as AWS, Azure, and Google Cloud, as well as on-premises environments and third-party services. This flexibility allows users to define infrastructure of any kind—networking components, virtual machines, databases, and more—using a unified approach. Additionally, Terraform's modular design allows users to create reusable components, streamlining the process of managing complex infrastructures.

In the context of data warehousing and analytics, Terraform can also be used to define and provision database schemas and structures within platforms like Snowflake. This capability is facilitated through the use of Snowflake's DDL (Data Definition Language), enabling users to automate the creation of tables, views, and other database objects directly from Terraform configurations. By integrating Snowflake DDL into Terraform scripts, organizations can ensure that their data architecture is consistently provisioned and maintained across different environments, such as development, staging, and production. This not only enhances operational efficiency but also supports best practices in version control and collaboration among data engineers and developers.

Overall, Terraform serves as a powerful tool in the modern DevOps toolkit, enabling teams to manage infrastructure in a way that is efficient, repeatable, and adaptable to changing business needs.

1.7 Azure DevOps

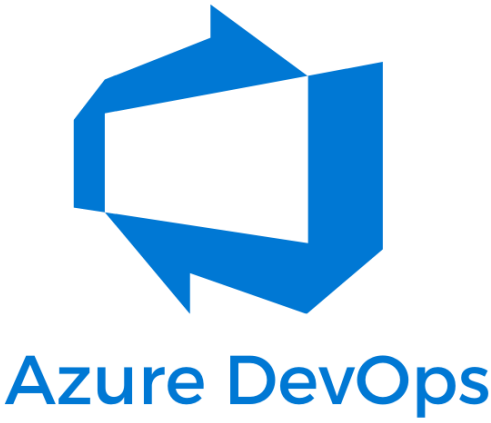

Figure 1.7: Azure DevOps Logo

Azure DevOps is a cloud-based service from Microsoft that provides a comprehensive set of tools for managing software development projects throughout the entire application lifecycle[18]. It encompasses a wide array of functionalities, including source control, continuous integration and deployment (CI/CD), project management, and collaboration features, all designed to facilitate agile development practices. Azure DevOps supports a variety of programming languages and platforms, making it a versatile choice for teams operating in diverse technology stacks.

One of the core components of Azure DevOps is Azure Repos, which offers Git repositories for source control, allowing teams to manage code versions and collaborate effectively.

In conjunction with Azure Repos, Azure Pipelines provides powerful CI/CD capabilities, enabling automated build and deployment processes across multiple environments. This ensures that code changes are continuously tested and deployed, reducing the risk of errors and accelerating delivery timelines. Additionally, Azure Boards offers robust project management tools, including Kanban boards, backlogs, and sprint planning features, facilitating effective tracking of work items and team progress.

In the context of data engineering and analytics, Azure DevOps plays a crucial role in automating workflows for data pipelines and infrastructure provisioning. By utilizing Azure DevOps in conjunction with infrastructure as code (IaC) tools like Terraform, teams can manage the deployment of data platforms, databases, and other essential components in a consistent and repeatable manner. This integration allows for better collaboration between data engineers and software developers, streamlining processes such as data ingestion, transformation, and analytics.

1.8 Thesis Structure

This chapter introduced the project and the main technologies employed in its design and implementation.

The following chapters are structured as follows.

Chapter 2 provides an analysis of the existing on-premises data platform, detailing its architecture, components, and the challenges it poses in terms of scalability, usability, and maintenance.

Chapter 3 outlines the architecture and components of the newly implemented cloud-based data platform, highlighting the technologies and methodologies employed during the re-engineering process to enhance data accessibility and operational efficiency.

Chapter 4 presents the evaluation of the new system, analyzing its scalability, reliability, and overall efficiency.

Finally, Chapter 5 summarizes the key findings of the thesis, reflecting on the lessons learned from the migration project and discussing potential avenues for future enhancements and ongoing development of the cloud-based data platform.

Chapter 2

Understanding the Legacy System

The legacy data platform at the heart of this project did not begin as a carefully planned and structured solution. Instead, it emerged organically from a proof of concept—a vertical slice system that was initially designed to address a narrow set of data processing needs. Over the years, as the company's data requirements grew, the system expanded incrementally. New tools and components were added, often in an ad hoc manner, without a clear overarching plan or architecture. This piecemeal growth resulted in a platform that, while functional, is characterized by a lack of consistent documentation, questionable design choices, and the absence of standardized best practices. What started as a simple system evolved into a complex, sprawling architecture that is difficult to manage and scale, necessitating the exploration of modern alternatives.

2.1 Functional Description

The primary function of the legacy data platform is to ingest, process, and analyze data from various external sources to gain insights into user behavior across the company's web portals. The system ingests logs and user information from multiple sources—Elasticsearch clusters for log data and relational databases for user profiles and interaction details. Once ingested, this data is processed to track user interactions, map out user journeys, and calculate various statistics related to user engagement and behavior.

These insights are crucial for understanding how users interact with the company's digital offerings, allowing for the optimization of user experiences and the identification of trends or issues. The processed data is then fed into downstream systems, such as Business Intelligence (BI) dashboards, where it is visualized and analyzed by stakeholders to inform business decisions.

Despite its utility, the legacy system's convoluted architecture and fragmented processes make it increasingly challenging to meet the evolving needs of the company efficiently and

effectively.

2.2 Architectural Description

Figure 2.1: Cloudera Logo

The legacy data platform employed by the client company is an on-premise Cloudera distribution[19], built to handle large-scale data processing and storage tasks typical of enterprise environments. This chapter provides a detailed overview of the system's architecture, highlighting the key components and their roles within the data ecosystem. The platform's primary objective was to provide an environment for managing and processing vast amounts of data generated by the company's operations. Despite its initial effectiveness, the system has begun to show limitations in scalability, performance, and maintainability, prompting the need for modernization.

2.2.1 Data Storage with HDFS

Figure 2.2: Hadoop Distributed File System Logo

At the core of the legacy system is the Hadoop Distributed File System (HDFS), which serves as the primary storage layer. HDFS is designed to store large files across multiple machines, ensuring redundancy and fault tolerance. The system's data is distributed across a cluster of commodity hardware[20]. However, while HDFS provides reliable storage, it has become increasingly cumbersome as the volume of data has grown. Issues such as prolonged data retrieval times and difficulties in managing storage resources have emerged, highlighting the limitations of the on-premise infrastructure in coping with the company's evolving data needs.

Figure 2.3: Apache Sqoop Logo

2.2.2 Data Ingestion with Sqoop

Data ingestion from external relational databases into the legacy platform is managed using Apache Sqoop. Sqoop is a tool designed to efficiently transfer bulk data between Hadoop and structured datastores such as MySQL, Oracle, and SQL Server[21]. Within the legacy system, Sqoop is employed to import data from external databases into HDFS, where it can then be processed and analyzed using Hive, Spark, or Impala. The tool supports both full and incremental data imports, providing flexibility in handling various data integration scenarios. However, as the volume of ingested data has increased, the process has become more resource-intensive and time-consuming, often leading to delays in data availability for downstream processing. Additionally, the need to configure and manage Sqoop jobs adds another layer of complexity to the data ingestion pipeline, contributing to the overall maintenance burden of the legacy system.

2.2.3 Querying and Data Processing with Hive and Impala

Figure 2.4: Impala and Apache Hive Logos

Data querying and processing within the legacy system are primarily conducted using Apache Hive and Impala. Hive, a data warehouse infrastructure built on top of Hadoop, facilitates querying of large datasets stored in HDFS through a SQL-like language[22]. It is particularly useful for batch processing tasks but tends to be resource-intensive and slow, especially

with complex queries.

To address these performance concerns, Impala is utilized for more interactive, exploratory querying. Unlike Hive, Impala allows for low-latency SQL queries directly on data stored in HDFS[23]. This makes it suitable for real-time data exploration, although its performance can still be impacted by the underlying infrastructure's limitations.

2.2.4 ETL Processes with Spark and Hive

Figure 2.5: Apache Spark Logo

The legacy system employs a combination of Apache Spark and Hive for Extract, Transform, Load (ETL) processes. Spark, known for its in-memory processing capabilities, is used for more complex and resource-intensive data transformations[24]. It handles large-scale data processing tasks more efficiently than traditional MapReduce, particularly when iterative algorithms or real-time stream processing is required. Hive, on the other hand, is used for simpler, batch-oriented ETL tasks where the emphasis is on compatibility with the existing data warehouse setup.

Despite the power and flexibility offered by Spark, the integration of these tools within the legacy system has resulted in a complex, fragmented ETL pipeline. Maintaining and optimizing this pipeline requires significant effort, particularly as the volume and variety of data continue to grow.

2.2.5 Resource Management with YARN

Resource management and job execution within the legacy platform are handled by Yet Another Resource Negotiator (YARN), a core component of the Hadoop ecosystem. YARN allows for the dynamic allocation of system resources to various applications running within the cluster[25]. It is responsible for job scheduling and the efficient management of CPU, memory, and storage resources across the distributed environment. While YARN has proven effective

Figure 2.6: Hadoop Yarn Logo

in balancing workloads, it has also introduced additional layers of complexity, particularly in configuring and tuning the system for optimal performance.

2.2.6 User Interface with Hue

Figure 2.7: Hue Hadoop Logo

Hue serves as the graphical user interface (GUI) for the legacy platform, providing users with an accessible way to interact with HDFS, Hive, and Impala. Hue simplifies the process of querying and managing data by offering a web-based interface that allows users to execute queries and browse file systems without needing to interact directly with the command line[26]. However, as the system has scaled, Hue has struggled to keep up with the increased load, resulting in slower response times and occasional system crashes, which have hindered user productivity.

2.3 Limitations of the Legacy System

The limitations of the legacy system played a critical role in shaping the design of the new architecture. As the existing platform evolved without a cohesive long-term strategy, several constraints became apparent. These shortcomings, coupled with the increasing demands of modern data processing workloads, highlighted the need for a more flexible, cloud-native solution. Identifying and addressing these limitations were key factors in guiding the architectural choices made during the re-engineering process, ensuring that the new system would not only overcome these challenges but also be better equipped for future growth and complexity.

2.3.1 Infrastructure Scalability

One of the major limitations of the legacy system lies in its inability to scale effectively with the increasing demands of new workloads and the growing volume of raw data ingested.

The on-premises hardware, which was initially sufficient for the platform's needs, now struggles to accommodate the rising data loads and processing requirements. This creates a bottleneck, forcing the business to make difficult trade-offs. In particular, the company currently tends to reduce the frequency of data processing tasks and limit the retention period of stored data to avoid exceeding hardware capacity. The alternative would be costly and time-consuming hardware upgrades. Neither of these options provides a long-term or flexible solution.

Hardware upgrades for on-premises Cloudera servers are notoriously delicate operations[27]. Some of the most common issues that can arise are:

- **Compatibility Issues**: Cloudera clusters are sensitive to hardware configurations. Adding new hardware or upgrading components (like CPUs, RAM, or storage) may result in compatibility issues with existing hardware or software, necessitating extensive reconfigurations. Balancing the performance of new and old hardware is another potential challenge.

- **Downtime and Service Interruptions**: Upgrading hardware often requires taking the system offline or limiting its availability. This can disrupt critical services and workflows, impacting business operations. Planned outages need careful coordination, and unplanned issues during upgrades can lead to extended downtime.

- **Cluster Reconfiguration**: Scaling up the hardware often requires reconfiguring the Cloudera cluster settings (e.g., YARN, HDFS). These adjustments are necessary to ensure the cluster can efficiently utilize the new hardware, but reconfiguration adds complexity and requires careful tuning to prevent performance degradation.

- **Maintenance Complexity**: More hardware means more components to monitor, maintain, and troubleshoot. This can increase the burden on IT staff, who must manage the cluster and ensure all new hardware components work seamlessly with the existing infrastructure.

- **Scaling Limits**: Even with upgrades, the physical limitations of on-premises data centers can restrict how much hardware can be added. Power and cooling capacity, physical space, and network bandwidth may all pose limitations, capping the maximum scale achievable with upgrades.

- **Software Version Dependencies**: Upgrading hardware may necessitate upgrading the Cloudera software stack to take full advantage of the new infrastructure. However, this can introduce dependency issues, where new software versions are incompatible with existing configurations or third-party tools, leading to further disruptions.

2.3.2 Maintainability and Functional Scalability

Another critical limitation of the legacy system has to do with code scalability, caused by the lack of code versioning and the absence of proper environment separation.

The system operates solely within a production environment, meaning that all maintenance tasks and feature releases are performed directly on live systems without the safety of a staging or development environment. This approach introduces a high level of risk, as any mistake or oversight in updates could cause disruptions to the production system[28]. To mitigate these risks, operations are carried out slowly and with extreme caution, resulting in a bottleneck for both maintenance and feature deployment.

Additionally, modifying any component of the system requires an in-depth understanding of the entire codebase, as it lacks modularity or documentation. Developers often retain older versions of scripts within the codebase itself, using these outdated files as a fallback in the event of failure. While this practice provides a rudimentary form of backup, it leads to disorganized and confusing file structures, making it difficult to navigate the code or understand the logic behind specific implementations. This haphazard approach to code management not only slows down development but also increases the likelihood of introducing errors and complicates future scaling efforts.

The absence of structured version control or environment separation significantly hampers the system's ability to scale, as even minor changes require careful, manual intervention, and extensive system knowledge.

2.3.3 Task Orchestration

The orchestration of tasks within the current data platform relies on a crontab file that schedules scripts to run at predefined intervals. While this approach allocates time slots for tasks based on their dependencies, it lacks the necessary safeguards to ensure that tasks in a dependency chain complete successfully before subsequent tasks are initiated. This can result in scenarios where dependent tasks are executed even if their predecessors have failed or are running slower than anticipated, potentially leading to data inconsistencies and operational issues.

Furthermore, as the number of scheduled tasks increases, the crontab file has become increasingly complex and difficult to interpret. The interdependencies among tasks become ob-

scured, complicating the management and troubleshooting processes. This lack of clarity can hinder the ability of operators to effectively monitor task execution and respond to failures, ultimately impacting the overall reliability of the data platform. The current orchestration method presents significant challenges to scalability and maintainability, necessitating a reevaluation of task management strategies to accommodate future growth.

2.3.4 Observability

The observability of the various tasks comprising the data flow in the current system is hindered by a lack of standardized logging practices. As a result, detecting and diagnosing issues becomes a challenging endeavor. Logs are often dispersed across unpredictable locations, which complicates the process of locating relevant information when problems arise. Additionally, the logs themselves may either provide insufficient detail to facilitate effective troubleshooting or, conversely, be overly verbose, rendering them difficult to read and interpret.

While the Hue GUI allows users to access data and perform simple queries effectively, it falls short when it comes to executing complex analyses. Tasks such as assessing data quality or identifying specific patterns that could indicate underlying issues in the data processing are cumbersome. This lack of robust observability features limits the ability of operators to monitor system performance effectively, impeding the timely detection of anomalies and ultimately impacting the reliability of the data platform.

2.3.5 Reliability

On-premise data solutions are inherently vulnerable to downtime resulting from power failures and network outages. These interruptions can significantly impact system availability and operational continuity. Although there are mechanisms to mitigate such risks—such as uninterruptible power supplies (UPS) that provide backup power during outages and redundant network architectures that utilize diverse communication pathways to maintain connectivity—the implementation of these solutions is often economically impractical for many organizations[29].

These challenges have underscored the need for a modern, cloud-based solution that can provide enhanced scalability, performance, and ease of use. This realization has driven the decision to explore Snowflake as a potential replacement for the legacy system, which is the focus of subsequent chapters in this thesis.

2.3.6 Critical Knowledge Dependency

The operational effectiveness of the legacy system is heavily contingent upon the specialized knowledge and experience of its current maintenance team. Due to the previously discussed

limitations, interacting with and managing the system necessitates a high level of expertise. This reliance on a small group of knowledgeable employees poses a significant risk; should any member of the team, or the team as a whole, become unavailable, the organization may face severe challenges in maintaining operational continuity.

The intricate nature of the system, coupled with a lack of comprehensive documentation and best practices, means that significant time and effort would be required to either develop a substitute solution or to train new personnel to manage the existing system effectively. This critical dependency on a limited pool of expertise highlights the urgent need for a more resilient and scalable architecture in future developments.

Chapter 3

Designing the New Cloud-Based Solution

The transition from a legacy on-premises data platform to a modern cloud-based architecture represents a significant shift in both technology and approach. This chapter delves into the design and implementation of the new system, emphasizing the strategic decisions made to leverage cloud capabilities effectively. By incorporating innovative technologies the new architecture aims to overcome the limitations faced by the legacy system.

The design process involved careful consideration of the existing data workflows, the specific requirements of the client, and the need for scalability, security, and performance. Key objectives included enhancing data accessibility for analytics, streamlining data ingestion and transformation processes, and ensuring that the new solution could adapt to the evolving needs of the business. Throughout this chapter, we will explore the architectural components of the new system, the integration of various technologies, and the methodologies employed to ensure a seamless migration from the old platform to the new. By focusing on a robust and flexible design, the new cloud-based solution is positioned to empower the client with greater agility and insights from their data.

3.1 Macro-Architecture

The macro-architecture of the new cloud-based data platform is designed to provide a cohesive and efficient framework for data ingestion, processing, and storage. Initially, the vision was to develop the entire platform utilizing only Snowflake to streamline the workflow and minimize complexity. However, as the design process progressed, it became evident that certain operations within Snowflake were cumbersome, particularly those involving data ingestion from external sources, data exporting, and monitoring tasks. To address these challenges, AWS services such as Lambda, S3, SNS, and Database Migration Service (DMS) were integrated into the architecture.

AWS Lambda functions were employed to handle lightweight ETL processes, enabling real-time data transformations and event-driven actions without the overhead of managing dedicated servers. Amazon S3 buckets were utilized for reliable data storage and as staging areas for raw data before it entered the processing pipeline. Amazon SNS facilitated efficient messaging and notifications, ensuring that relevant stakeholders were promptly informed of key events within the data workflow. Additionally, AWS DMS was chosen to simplify the migration of data from external relational databases into the cloud environment.

With these AWS components in place, Snowflake was positioned as the central hub for data processing and warehousing. By leveraging its powerful analytical capabilities, Snowflake allows for seamless querying and analysis of the ingested data. This hybrid approach not only enhances the overall functionality of the platform but also provides the necessary flexibility and scalability to accommodate future growth and evolving business requirements. The following sections will detail each component of the architecture, illustrating how they interact to create a robust data ecosystem.

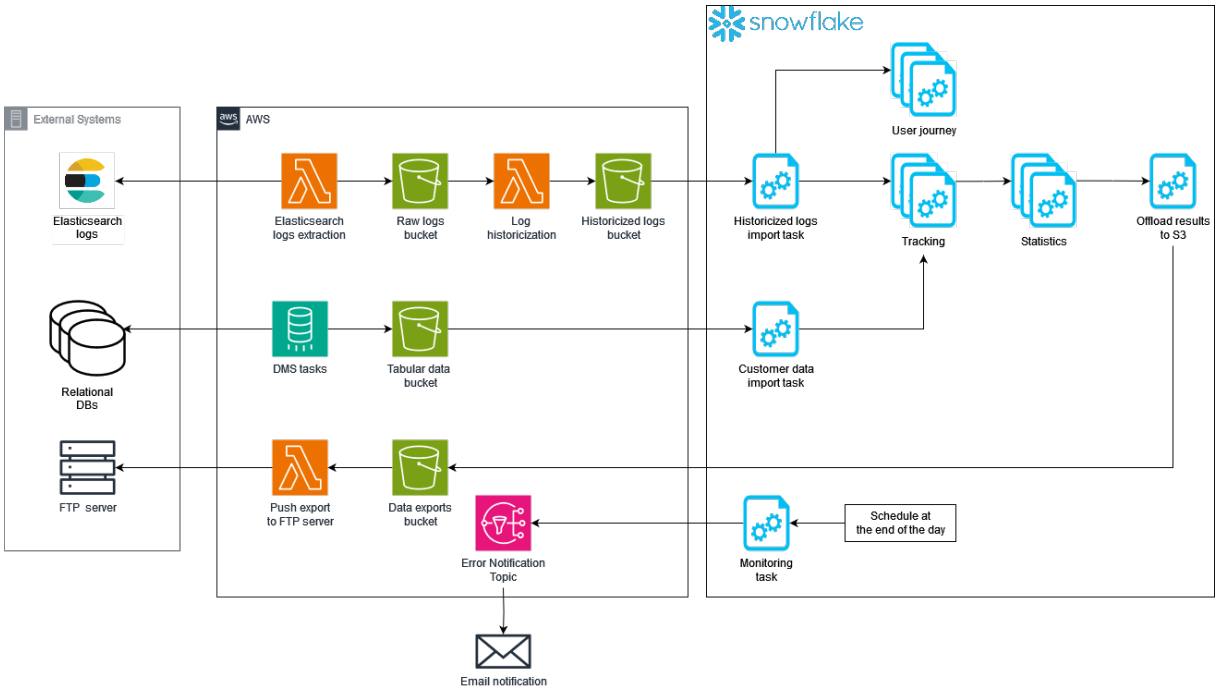

Figure 3.1: Cloud-Based Data Platform Architecture

The implementation of the new cloud-based data platform can be conceptually divided into three modules: Ingestion, Processing, and Export & Monitoring. Each of these modules encompasses a series of workflows that collectively contribute to the functionality of the overall system.

3.1.1 Ingestion

The Ingestion module focuses on the initial capture and storage of data from various external sources. It includes the following workflows:

- **Logs Extraction and Historicization**: This workflow involves extracting log data from systems such as Elasticsearch, ensuring that all relevant logs are captured, processed, and stored for future analysis. The historicization process guarantees that logs are archived systematically to facilitate retrospective queries and audits.

- **Customer Data Fetch**: This workflow is dedicated to retrieving customer-related data from external relational databases. By employing AWS DMS, the platform can efficiently ingest up-to-date customer information, ensuring that the data remains current and relevant for downstream processes.

3.1.2 Processing

Once data is ingested, the Processing module handles the various transformations and analyses needed to derive meaningful insights. This module includes the following workflows:

- **Master Data Pre-processing**: This workflow involves the cleansing and preparation of master data to ensure accuracy and consistency across the system. It is crucial for maintaining data integrity, enabling reliable analytics, and supporting decision-making processes.

- **Tracking Elaboration**: In this workflow, user interactions and activities across various portals are analyzed to create a comprehensive view of user behavior. This involves processing the ingested log data to identify patterns and trends.

- **User Journey Elaboration**: Building upon the tracking data, this workflow focuses on reconstructing the user journey through various touchpoints. By analyzing the sequence of interactions, the platform can provide valuable insights into user experience and conversion paths.

- **Statistics Evaluation**: Here, various statistics are calculated based on the processed data, providing key performance indicators (KPIs) and insights that inform business strategies. This evaluation serves as the foundation for reporting and further data analysis.

3.1.3 Export & Monitoring

The final module, Export & Monitoring, ensures that processed data is made accessible for analysis and that the system operates efficiently. This module consists of the following workflows:

- **FTP Server Data Forwarding**: Processed data is forwarded to an FTP server, enabling seamless data sharing with external systems or stakeholders. This workflow is vital for ensuring that relevant data is readily available for reporting and operational needs.
- **Monitoring Report Creation**: This workflow involves generating reports that summarize system performance, data ingestion rates, and processing times. Monitoring reports provide insights into the operational health of the platform and help identify any areas that may require attention or optimization.

3.2 Ingestion

The ingestion module is responsible for collecting raw data from various sources and transferring it into the cloud infrastructure. The module's architecture is designed for modularity, allowing new sources of data to be added with minimal overhead.

3.2.1 Logs Extraction and Historicization

The Logs Extraction and Historicization workflow is a critical component of the data ingestion process, designed to ensure that user activity logs from six different web portals are efficiently retrieved, stored, and organized for subsequent analysis. This workflow begins with the ElasticSearch logs extraction job, which connects to the designated ElasticSearch endpoint to access logs associated with user interactions across all six portals. The primary objective of this job is to extract relevant web portal logs that provide insights into user activity patterns, essential for generating key performance indicators (KPIs) and business metrics.

Once the raw logs have been successfully imported, a historicization job is initiated to structure the extracted data systematically. This job organizes the logs into a well-defined directory tree, categorizing them by source portal and date. This hierarchical organization not only facilitates easier access and consumption of the logs but also enhances the efficiency of subsequent data processing workflows.

To meet compliance requirements, all logs retrieved from ElasticSearch must be retained for a minimum of five years. To optimize storage costs and management, historic logs are persisted in cold storage, as they are infrequently accessed after the initial processing phase. In contrast, the logs stored in hot storage are limited to a more immediate timeframe, encompassing only

the last seven days of activity. This strategic approach to data storage ensures that the system remains both compliant with regulatory requirements and cost-effective, balancing the need for accessibility with the realities of data management.

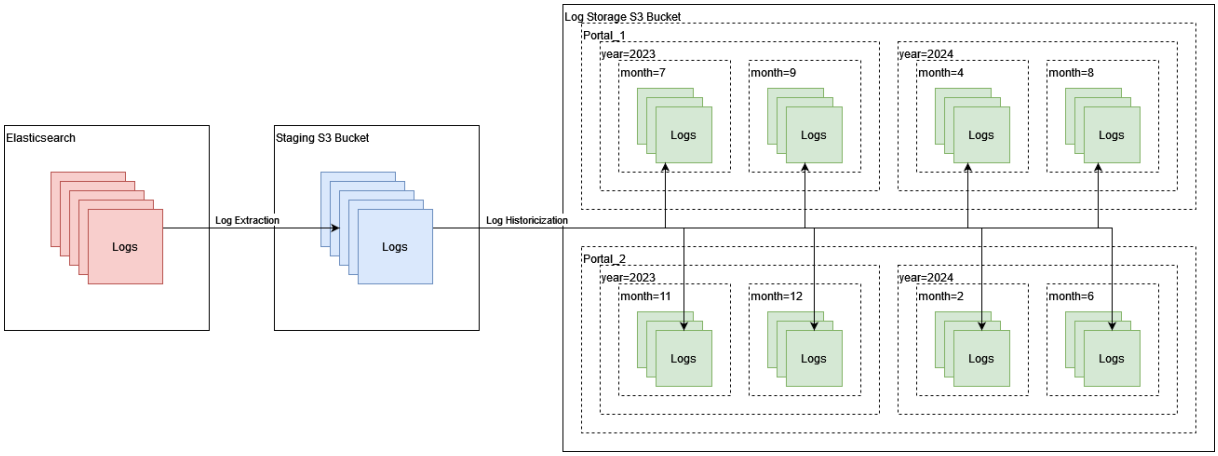

Figure 3.2: Log Extraction and Historicization Overview

The implementation of the Logs Extraction and Historicization workflow involves a two-step process leveraging AWS Lambda functions and Amazon S3 to automate the collection and organization of ElasticSearch logs.

First, ElasticSearch logs are collected using a scheduled AWS Lambda function that executes daily. This Lambda function connects to the ElasticSearch endpoint, retrieves logs pertaining to user activity from the six different web portals, and deposits them into a designated Amazon S3 bucket. The choice of a scheduled Lambda function ensures that logs are ingested regularly, maintaining up-to-date records for analysis and compliance.

Once the logs have been ingested into the S3 bucket, a second Lambda function is triggered to perform the historicization process. This function follows the same logic applied in the legacy system, ensuring consistency in how logs are organized. The historicization involves categorizing the logs into a structured directory tree within another S3 bucket, organizing them by source portal and date for easy access and retrieval. This systematic organization mirrors the structure of the existing system while taking advantage of the flexibility and scalability of cloud storage.

By using S3 for both the ingestion and historicization of logs, the implementation allows for efficient data storage and management. The separation of concerns, with one Lambda function handling ingestion and another managing historicization, not only adheres to best practices in software architecture but also enhances the system's maintainability. Overall, this implementation ensures that the logs are efficiently collected, stored, and organized, laying the groundwork for effective data processing and analysis in subsequent stages of the data pipeline.

3.2.2 Customer Data Fetch

The Customer Data Fetch workflow is responsible for importing essential data from external on-premises databases into the system, covering key business entities such as customers, subscriptions, and products. This data is crucial for enriching raw logs with contextual information, enabling a more comprehensive analysis of user behavior across the company's web portals.

The imported tables serve as foundational inputs for other data processing jobs and queries. They are used to enhance raw log data with customer-related information, allowing for more detailed user journey analysis and behavior tracking. In addition, a portion of the imported tables feeds directly into Business Intelligence (BI) dashboards, providing real-time insights to business users for operational and strategic decision-making.

There is no explicit requirement to persist older versions of the imported tables. However, retaining a few days' worth of data is recommended to ensure that the system can recover smoothly in the event of an incident or processing flow failure. For this purpose, a retention period of seven days in hot storage is deemed sufficient to support potential restarts of the data processing pipelines without causing any disruptions.

It is important to note that the imported data is not required to fulfill any disaster recovery (DR) obligations. Specifically, the system does not treat this data as a backup source for the on-premises databases, meaning the focus is solely on ensuring timely and accurate data ingestion rather than long-term preservation or recovery scenarios. This approach optimizes storage usage while ensuring that business processes remain efficient and well-supported by up-to-date customer data.

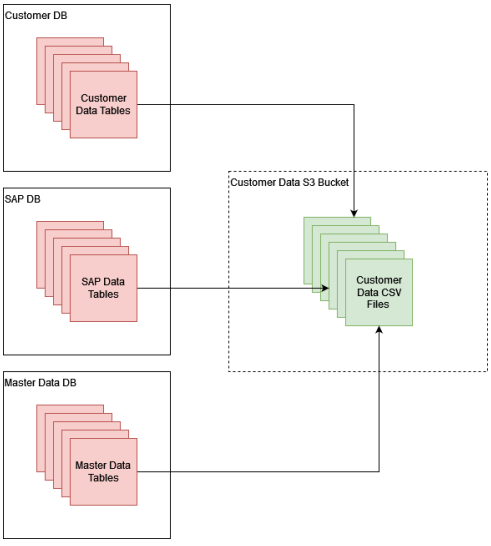

Figure 3.3: Customer Data Fetch Overview

The implementation of the Customer Data Fetch workflow utilizes AWS Database Migration Service (DMS) to automate the extraction and import of customer data from external on-

premises Oracle databases. DMS tasks are scheduled to run daily, ensuring that the most up-to-date data regarding customers, subscriptions, and products is retrieved from the source systems.

Once the DMS tasks are executed, the extracted data is deposited directly into a designated Amazon S3 bucket. This approach ensures that the customer data is efficiently ingested and stored in a highly scalable and accessible location within the cloud environment. The S3 bucket serves as a staging area where the imported tables are held temporarily before being processed and integrated into the system's tables for further enrichment and analysis.

By leveraging DMS, the platform benefits from reliable and automated data migration, reducing the need for manual intervention while ensuring that data from the Oracle databases is consistently available for downstream processing. This implementation ensures seamless integration between on-premises systems and the cloud-based platform, enabling other data processing flows to enrich logs with customer data and feeding the BI dashboards with timely information.

The workflow involves retrieving data from more than fifty tables from the external Oracle databases. While many of these are simple subsets of columns from the source tables, in some cases more complex data retrieval is required. For specific tables, join and filter queries are performed directly in the source databases as part of the DMS tasks to ensure that only data meeting certain business criteria is retrieved. This optimization reduces the volume of unnecessary data transferred and ensures that only relevant, curated information is ingested into the platform for further processing.

3.3 Processing

The processing module handles the core data transformation and analysis tasks within the new solution. It is in this module that raw ingested data is transformed into actionable insights, ready for downstream analysis or reporting. This section will focus on the processing tasks, which are executed on Snowflake.

3.3.1 Master Data Pre-processing

The Master Data Pre-processing flow is a relatively small but crucial technical process aimed at re-organizing data sourced from the Master Data database into a more accessible format. The goal of this workflow is to create a set of simplified tables by performing basic transformations, such as joins and column renaming. This restructuring makes the master data easier to use and integrate into other processes within the platform.

The outputs of the preprocessing flow include high-level information about key business entities, such as:

- **Subscriptions**: Including their current status and the period of validity.
- **Customers**: Detailing their location and associated customer segment.
- **Products**: Including information about their cluster and descriptions.

These processed tables are essential for downstream workflows that support various business operations, including statistics evaluation and reporting. Although this flow is lightweight in terms of complexity, its output provides foundational data that is referenced across multiple processes.

In terms of data retention, the requirements for this flow are directly aligned with those of the source master data. Only 7 days of data need to be kept in storage at any given time, as the information is regularly refreshed, and older versions do not need to be archived. This approach ensures that storage resources are optimized while maintaining up-to-date data for ongoing processing and reporting workflows.

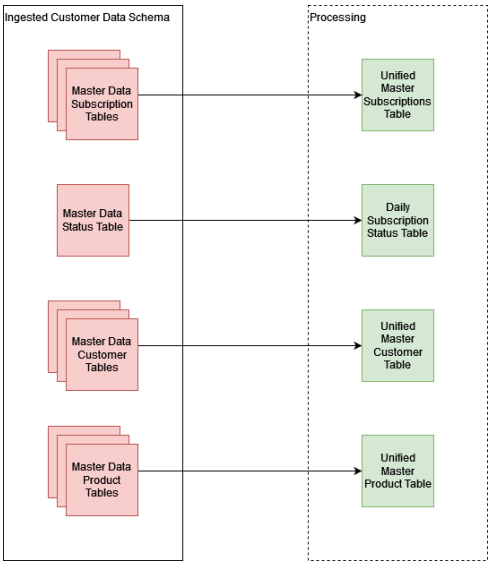

Figure 3.4: Master Data Pre-Processing Overview

The implementation of the Master Data Pre-processing flow involves executing four distinct Snowflake queries that operate independently of each other. Each query is designed to transform the source data into simplified, high-level views that facilitate easier access and usability for subsequent processes.

- **Subscriptions Overview Query**: Performs a join operation across various source tables containing subscription data. This operation generates a consolidated view that captures key attributes of subscriptions, including their current status and validity period. The resulting table serves as a high-level summary of all active subscriptions, providing essential information needed for analysis and reporting.

- **Daily Subscription Status Query**: Builds a simplified view by processing the source table that contains subscription status data. This query associates each subscription with its corresponding status on a daily basis, selecting only the relevant columns of interest. This daily snapshot enables tracking of subscription statuses over time, which is crucial for understanding customer engagement and managing renewals.

- **Customers Overview Query**: Focuses on customer data, executing a join operation across relevant source tables. This query generates a high-level view that includes critical details about customers, such as their location and segment. The output table provides a comprehensive overview of the customer base, which can be leveraged for targeted marketing and customer service initiatives.

- **Product Overview Query**: Processes the product data by performing a join operation across the source tables that contain product information. This query creates a high-level view of products, detailing their clusters and descriptions. This simplified view is valuable for understanding the product offerings and how they relate to subscriptions and customer segments.

The outputs of these four queries are stored in a structured format that enhances the accessibility and usability of master data within the system. By organizing this information into easily understandable tables, the Master Data Preprocessing flow lays the groundwork for more complex data processing workflows and analytical endeavors.

3.3.2 Tracking Elaboration

The Tracking Elaboration workflow is responsible for transforming and enriching raw server log data, which is ingested daily, with customer information. The primary objective of this process is to take raw logs from various web portals and add structure and context using customer data, enabling business users to analyze the data and extract valuable insights. This workflow provides the foundation for further analysis and statistical evaluations that support strategic business decisions.

The output from the tracking process provides detailed information on the interactions between customers and products, with explicit references to the specific web portals, subscriptions, and individual users. By processing these logs, the system generates structured data that can be used by the business to better understand user behaviors and identify trends. One of the most important business outcomes of this process is the ability to identify customers with low or no usage of their subscriptions. These customers are considered at a higher risk of churn, making them prime targets for customer engagement and retention efforts.

To support long-term business needs, the output data from the tracking flow is retained in hot storage for at least one year, allowing business users to access it through Business Intelligence (BI) dashboards and make timely decisions based on the latest available data. After the first year, the data may be moved to cold storage for cost optimization, as it is rarely accessed after the initial period. However, to meet business and compliance requirements, the data is retained in cold storage for a minimum of five years, ensuring that historical data remains available when needed for deeper analysis or reporting purposes.

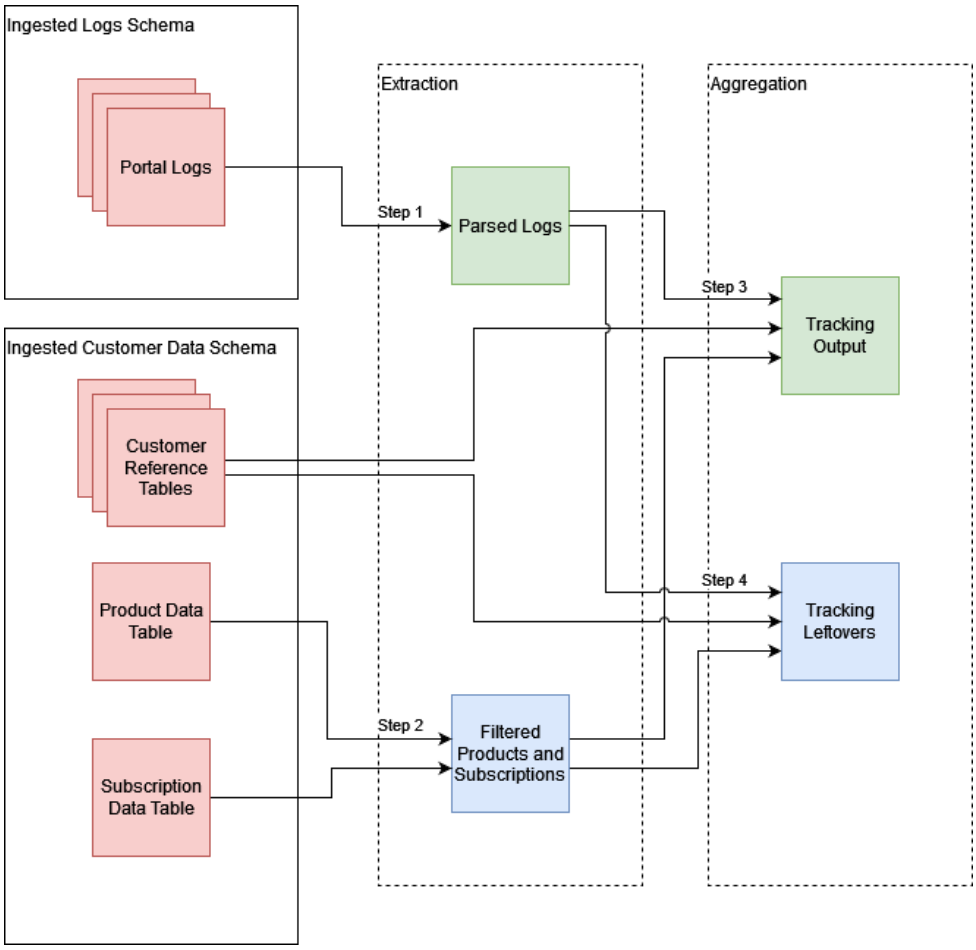

Figure 3.5: Tracking Elaboration Overview

The implementation of the Tracking Elaboration workflow is carried out using a series of Snowflake tasks that process the raw server log data, enriching it with customer information in multiple steps. The process is dependent on the successful completion of earlier data ingestion tasks, ensuring that all necessary data is available for processing. Once processed, the enriched data is stored within Snowflake's storage and made accessible to external systems and business users via appropriate interfaces, such as BI dashboards.

The workflow can be broken down into four main steps:

- **Parsing and Structuring Server Logs**: The first step involves a query that parses the server JSON logs stored in the source log tables. These logs are ingested daily and contain user activity data from the company's web portals. The query extracts the relevant fields from the logs, most of which are located within a single comma-separated field. The formatting of the logs is different for each source portal, thus logic based on regular expressions is used to parse every log correctly. The extracted fields are then transformed and formatted into separate columns, providing a structured dataset that can be further processed.

- **Filtering and Extracting Customer Data**: In this step, a query is executed on some of the customer data tables (concerning products and subscriptions) to extract only the most relevant information. The goal is to identify the latest order for each subscription, filtering the records based on subscription status and date. Two join operations are performed to narrow down the data, ensuring that only rows containing products listed in specific customer data tables are included. This filtered dataset provides the key information necessary to enrich the log data.

- **Enriching Logs with Customer Data**: In the third step, the parsed log data from step 1 is joined with the filtered customer data from step 2 and with customer reference tables. This step enriches the log data by incorporating customer details and portal information. The output of this operation is a structured dataset that tracks customer interactions, counting the number of actions performed by each customer with reference to a specific date. This enriched dataset enables business users to analyze customer activity and identify engagement trends.

- **Identifying Missing or Incomplete Data**: The final step involves an anti-join operation, where the output from steps 1 and 2 is compared against customer reference tables. The goal is to identify any logs that do not match the data from the other tables, highlighting missing or incomplete information. This ensures data integrity by flagging any discrepancies between the logs and the customer data, allowing for further investigation or data correction if necessary.

3.3.3 User Journey Elaboration

The User Journey Elaboration workflow is responsible for processing and enriching daily ingested client logs by organizing user activities into meaningful patterns. This process involves the use of macro-action groups, action counts, time deltas, and last action flags to create a structured view of how users interact with the company's web portals. The primary goal is to arrange

and enrich raw logs in a way that reveals the sequence and timing of user actions, providing actionable insights into user behavior.

The output of this process offers valuable information on user sessions, detailing what actions are performed during each session and how those actions are spaced out over time. By analyzing this data, the business can derive insights into user engagement with various product features, helping to shape marketing and product development strategies.

Marketers can leverage this data to identify the most active users of the product, enabling them to target these users with targeted sales opportunities. At the same time, management can gain an overview of feature usage, helping the team prioritize the development of features that align with user preferences and engagement patterns.

Additionally, the data allows the publishing team to understand which types of content and topics are most engaging for users, enabling more tailored content creation that resonates with the audience.

To support ongoing business analysis, the outputs from the user journey flow are retained in hot storage for a minimum of one year. This ensures that the data remains readily accessible for BI dashboards and other analysis tools used by business users. After the first year, the data can be moved to cold storage for cost optimization, while remaining accessible for up to five years to meet both business and compliance requirements. This approach ensures that historical data is preserved while minimizing storage costs for data that is seldom accessed after the initial usage window.

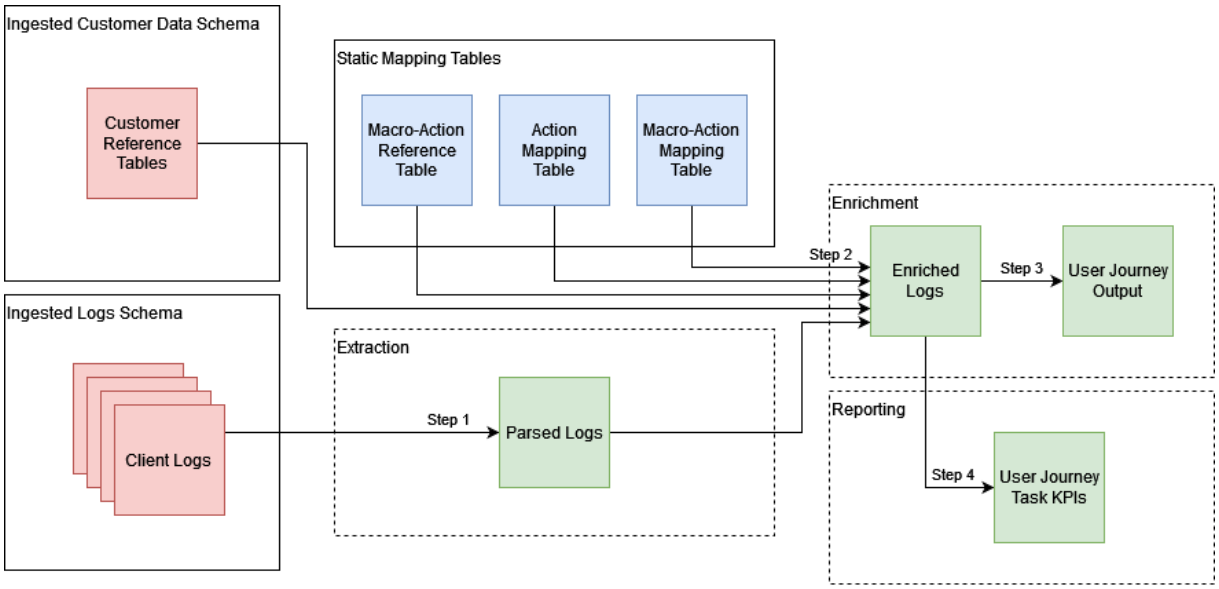

Figure 3.6: User Journey Elaboration Overview

The implementation of the User Journey Elaboration workflow mirrors that of the previous workflow, it is designed to process and structure the raw client log data using a series of

Snowflake tasks, which are dependent on the completion of earlier data ingestion tasks. The entire process is divided into several steps, with the final output stored in Snowflake and made available for external consultation through appropriate interfaces, such as BI dashboards.

The workflow is executed in the following four main steps:

- **Parsing and Structuring Client Logs**: In the first step, a query is executed to parse the JSON logs received from the client systems. These logs, stored in the source client log tables, contain unstructured data related to user activity. The query extracts the relevant fields from the raw JSON data, formatting them into separate columns that can be used in subsequent steps for analysis and enrichment.

- **Enriching Log Data with Reference Information**: The second step involves enriching the structured logs from step 1 by joining the data with various static mapping tables and customer reference tables. This enrichment process adds contextual information, such as user details and portal identifiers, to the logs. In addition, the query calculates extra fields based on timestamp comparisons and counts the number of actions per session, allowing for a deeper understanding of user behavior and session dynamics.

- **Aggregating User Sessions**: In this step, the enriched data from step 2 is aggregated to produce a single row for each individual user session. This query concatenates the sequence of actions performed by each user during their session, creating a chronological timeline of activities. The output table contains one row per session, with fields that represent the ordered list of actions, making it easy for business users to analyze the flow of events within each session.

- **KPI Calculation and Reporting**: The final step focuses on calculating key performance indicators (KPIs) for business reporting. The query aggregates the output data from step 2 again, but this time with a focus on generating metrics over specific time periods. For each user, it computes the number of actions performed within two time windows: the last 6 months and the last 12 months. This aggregation allows the business to track user engagement trends over time, helping to identify high-activity users and assess product feature usage across different periods.

3.3.4 Statistics Evaluation

The Statistics Evaluation workflow aggregates data generated by the Tracking workflow, transforming its output into structured tables that capture user interactions across varying time granularity. This process is designed to facilitate detailed analysis of user behavior during specific operational periods, enabling comparisons between different time frames. By processing user

data in this manner, the statistics job provides valuable insights into business volumes, focusing on key metrics such as subscriptions, interaction types, and subscription status.

The outputs generated from this flow serve multiple purposes for business intelligence, allowing stakeholders to evaluate trends and make data-driven decisions. Users can access these insights through BI dashboards, ensuring that relevant data is readily available for analysis over time.

To meet the business requirements, the retention period for the data produced by this flow is set at five years, stored in hot storage. This approach allows for continuous accessibility to historical data, empowering users to query and analyze trends from the past five years. By only keeping the aggregated data in hot storage for five years, excluding its sources, access to historical analysis data is maintained while keeping storage costs low.

As with other processing workflows in the data platform, the statistics flow is implemented using a series of Snowflake tasks. Each task is responsible for executing specific queries at scheduled intervals, ensuring that data is aggregated, processed, and updated continuously. This task-based approach allows the system to handle data processing in a scalable and automated manner.

The logic of the statistics flow can be conceptually divided into two main parts, each involving a series of steps designed to progressively aggregate and enrich the data to enable multi-dimensional analysis across different time frames.

The first part focuses on daily aggregations, providing a detailed breakdown of user interactions per subscription and subscription type. This part follows four key steps:

- **Daily Aggregation**: The process begins by aggregating the data output from the tracking step. Specifically, it sums the number of operations, searches, and views performed for each subscription on a daily basis. This data is further enriched by joining it with subscription metadata, such as status and type, sourced from the master subscriptions table and the daily subscription status table—both outputs of the master data pre-processing flow.

- **Aggregation by Subscription Type**: The next step involves grouping the subscriptions by their type. For each group, the query computes several key metrics, including the number of used versus unused subscriptions, the ratio of active to inactive subscriptions, and the average number of operations across all subscriptions of a given type.

- **Subscription Classification**: In this step, subscriptions are classified based on their performance relative to the average number of operations for their respective subscription type. This classification helps identify subscriptions that are performing above or below the norm.

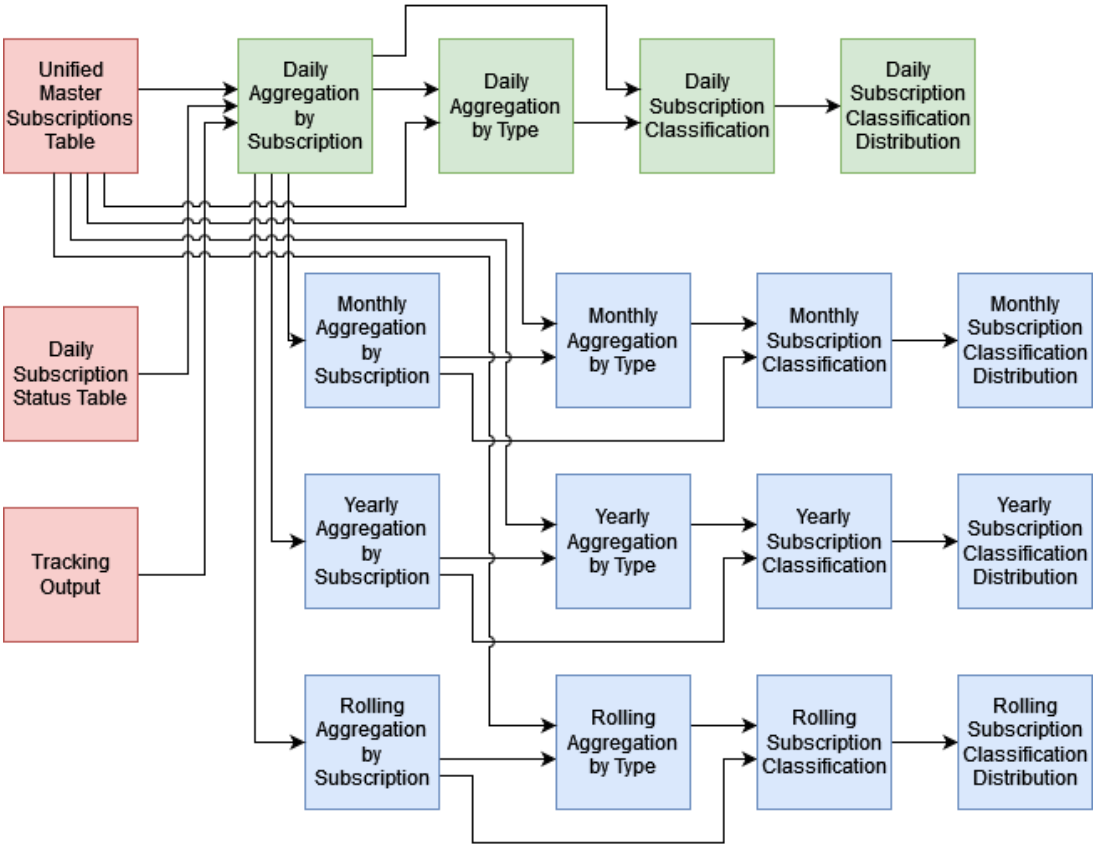

Figure 3.7: Statistics Evaluation Overview

- **Classification Distribution**: Finally, a query is run to generate the distribution of the subscription classifications for each subscription type. This provides insights into how different subscription categories perform across the user base.

The second part of the flow shifts focus to longer-term aggregations, including monthly and yearly time frames, as well as a rolling 12-month period. Similar to the first part, this portion is structured into four steps:

- **Time-Frame Aggregation**: The process begins by aggregating the daily data for three distinct time frames—monthly, yearly, and a rolling 12-month period. This enables users to analyze trends and performance metrics over varying temporal spans.
- **Aggregation by Subscription Type**: As in the first part, the subscriptions are then grouped by type, and the same key metrics (used/unused subscriptions, active/inactive subscriptions, average operations) are computed for each time-frame, with the addition of a count of days in which at least one operation was performed.
- **Extended Classification**: Building on the earlier classification from the first part, this step introduces a second classification dimension, focused on the number of active days

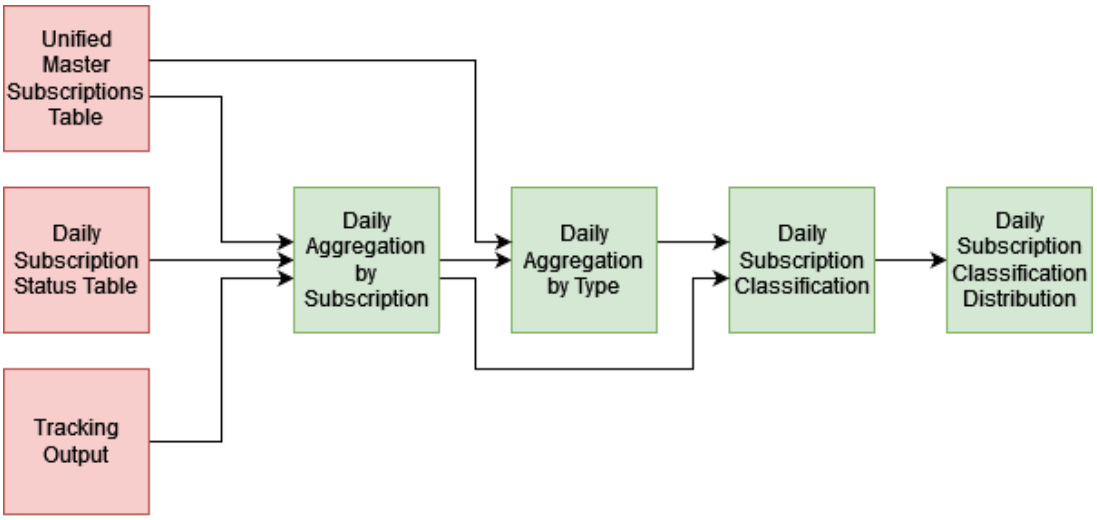

Figure 3.8: Statistics Evaluation Daily Detail

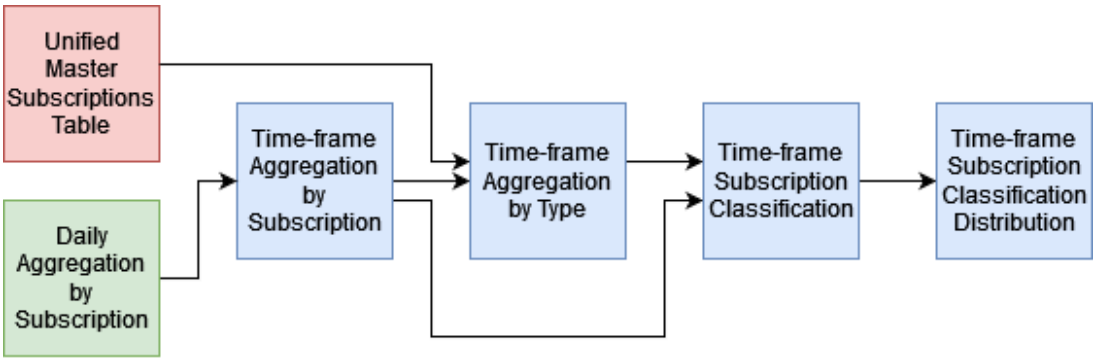

Figure 3.9: Statistics Evaluation Time-frame Detail

a subscription has within the selected time-frame. This adds further granularity to the analysis by evaluating not only the number of operations performed but also the frequency of subscription activity.

- **Classification Distribution**: The final step aggregates the classifications, providing a detailed view of the distribution for both the operational and activity-based classifications across the various time frames.

By systematically aggregating and classifying the data over different time spans, this flow equips business users with the tools to perform in-depth analyses of subscription performance and customer behavior, enabling more informed decision-making and strategic planning.

3.4 Export & Monitoring

The export and monitoring module ensures that processed data is made available to external stakeholders in the appropriate formats, while also providing oversight to ensure the system operates smoothly. This section will cover the export of data to external services such as marketing platforms, as well as the monitoring task that is implemented to check the integrity and health of the data pipeline.

3.4.1 FTP Server Data Forwarding

The FTP Server Data Forwarding workflow manages the export of key reports and statistics data into CSV format and facilitates their transfer to an on-premises FTP server. This process ensures that essential data is accessible for campaign management and analysis on the marketing platform used within the client company.

The workflow focuses on exporting two specific sets of data:

- **User Journey Task KPIs**
- **Subscription Classification and Distribution** statistics for the monthly, yearly, and rolling time-frames.

The export is necessary because the marketing platform relies exclusively on CSV files to import data, unlike other BI tools within the company that can interact with data through Snowflake's API via SQL queries.

The process begins with the Snowflake tables being exported into CSV files, which are stored in an AWS S3 bucket. From there, AWS Lambda functions are triggered to handle the automatic transfer of these files to the on-premises FTP server. This server then provides the data to the marketing platform for further processing and analysis.

3.4.2 Monitoring Report Creation

The Monitoring Report Creation workflow is responsible for performing a series of checks to ensure the proper execution of the various data processing flows within the platform. This process generates a detailed report and sends it to a designated mailing list, providing early detection of potential issues in the system's data processing or data quality.

The primary objective of this workflow is to enable prompt identification and resolution of any problems, thereby minimizing any disruption to the business's operations. By identifying failures or anomalies early, stakeholders can react quickly, ensuring that data quality and consistency are maintained.

While this workflow may detect data quality issues in the source data, it is designed strictly for monitoring purposes. It does not attempt to fix or modify any data on the analytics platform itself. Any problems discovered in the source data must be corrected by the system that originally produced the data, as the monitoring job operates solely on the processed data within the platform.

The monitoring process is implemented as a Snowflake task, designed to fail explicitly if certain anomalies or errors are detected in the data produced during the day. When the task fails, it triggers a notification via Amazon Simple Notification Service (SNS), sending alerts to a list of registered recipients, ensuring immediate awareness of the issue. This mechanism helps keep the data pipelines reliable, reducing the potential for undetected errors to affect downstream business processes.

3.5 Orchestration

The orchestration of data processing tasks in the new cloud-based solution is designed to ensure that workflows execute efficiently and in a timely manner while maintaining flexibility for future requirements. The orchestration strategy spans across both AWS services and Snowflake, integrating these environments into a cohesive system. Below is an overview of the orchestration methodology for each component of the data platform:

3.5.1 Ingestion Task Scheduling in AWS

Data ingestion tasks are initiated using the scheduling mechanisms provided by AWS. These tasks are responsible for transferring raw data from various sources into the cloud for further processing:

- **AWS Database Migration Service (DMS)**: Tasks responsible for ingesting tabular data from relational databases are directly scheduled using DMS's built-in scheduling capabilities. These schedules dictate when data extraction jobs begin, ensuring timely updates of source data.

- **AWS Lambda with Step Functions**: For other ingestion tasks, such as processing raw log files, AWS Lambda functions are orchestrated via scheduled AWS Step Functions (SFN). This approach simplifies the coordination of complex workflows involving multiple dependent Lambdas.

3.5.2 Processing Task Scheduling in Snowflake

Processing tasks within Snowflake are responsible for transforming ingested data and generating actionable insights. To ensure alignment with the ingestion workflows, these tasks are scheduled independently to run after the completion of ingestion tasks. Currently, all dataflows are executed daily, following a fixed schedule:

- **Fixed Scheduling**: Snowflake tasks are configured to start at predefined intervals, allowing sufficient time for the ingestion processes to complete.
- **AWS Lambda with Step Functions**: The architecture is designed to support event-driven execution in the future. For example, when new files are uploaded to an S3 bucket, an AWS SNS topic can trigger the appropriate Snowflake tasks automatically, using Snowflake's external functions or its event-driven capabilities. This setup will enable seamless alignment of ingestion and processing schedules for workflows with varying execution frequencies.

3.5.3 Task Dependency Management in Snowflake

Within Snowflake, processing tasks are organized into Directed Acyclic Graphs (DAGs). These DAGs ensure that task dependencies are respected, and all steps within a dataflow are executed in the correct order. The DAGs encompass:

- **Intermediate Processing Steps**: Transforming raw data into meaningful intermediate tables.
- **Final Output Generation**: Generating the final outputs required for business intelligence or further exports.
- **Data Export to AWS**: Transferring processed data back to AWS services (e.g., exporting CSV files to S3 buckets).

The DAG structure ensures efficient execution by running tasks in parallel where possible while preserving task dependencies.

3.5.4 Monitoring Task Orchestration

The dedicated monitoring workflow is scheduled to run at the end of each day, separately from the data processing tasks. This workflow performs comprehensive checks on the tasks executed during the day to validate the correct execution of ingestion and processing tasks and identify potential anomalies in data quality or task execution.

If inconsistencies or failures are detected, the monitoring workflow explicitly fails. This failure triggers an AWS SNS topic notification, which delivers a warning to registered email recipients, ensuring prompt attention to issues.

3.6 Deployment

The deployment of the new data platform leverages Infrastructure as Code (IaC) principles, ensuring that the entire architecture is declarative, reproducible, and easy to manage. The platform's infrastructure is described using Terraform, a widely used IaC tool that supports AWS and Snowflake. This approach aligns with the goal of enhancing maintainability and scalability while adhering to modern DevOps practices.

3.6.1 Terraform Modules

The Terraform codebase for the data platform is divided into two separate modules to reflect the dual environment architecture:

1. **AWS Module**:
    - Includes declarations for services like AWS Lambda functions, DMS tasks, S3 buckets, Step Functions workflows, and SNS topics.
    - Contains configuration parameters for services, IAM roles for service permissions, and associated resource policies.
    - Embeds the code for Lambda functions, written in Python, as inline files or external scripts.
2. **Snowflake Module**:
    - Contains the definitions for Snowflake tasks, DAG dependencies, warehouses, tables, and views.
    - Includes SQL scripts for queries executed by Snowflake tasks and static tables required for operations.
    - Declares roles and permissions for managing access within Snowflake.

3.6.2 Deployment Through Azure DevOps

The Terraform code is deployed using Azure DevOps build pipelines. This deployment process connects to the client company's AWS and Snowflake accounts, applying the Terraform configurations to provision and update the infrastructure.

1. **Pipeline Setup**:
    - The pipeline is configured to monitor changes in the Terraform code stored in a Git repository.
    - Upon detecting changes, the pipeline runs Terraform commands (`terraform plan` and `terraform apply`) to update the infrastructure.
2. **Environment Separation**:
    - The data platform uses three distinct environments—Development, Quality Assurance (QA), and Production.
    - Each environment corresponds to a Git branch, and changes are promoted through the branches to ensure thorough testing before reaching production.
3. **Integration with AWS and Snowflake**:
    - AWS credentials for Terraform are managed through secure Azure DevOps secrets.
    - Snowflake API keys and connection details are similarly secured, allowing Terraform to manage Snowflake resources seamlessly.

This deployment strategy represents a modern, scalable, and maintainable approach to managing a complex data platform architecture, ensuring smooth operations and future adaptability.

3.7 Continuous Integration and Continuous Delivery

To maintain and enhance the new data platform efficiently, a Continuous Integration/Continuous Deployment (CI/CD) pipeline is employed. This approach automates the deployment of infrastructure and code changes while adhering to a Git-based branching strategy inspired by the Gitflow approach.

3.7.1 Codebase Management

Gitflow is a branching model for Git repositories that organizes the development workflow into distinct, purpose-driven branches. This model is particularly well-suited for projects with multiple environments, as it provides a clear structure for managing feature development, testing, and releases.

This is how the project's branches are organized:

- **The Production branch** represents the production-ready codebase. Only stable, tested changes are merged here.

- **The Development branch** is where features are first integrated in order to verify whether they behave in the intended manner.
- **The QA branch** is an intermediate step between development and production, this is where features that should be ready for release are validated by downstream systems and end users.
- **Feature branches** are temporary branches used for developing specific features. These branch off from `Development` and are merged back when complete.
- **Release branches** are split off of `Development` to be merged into `QA`. They allow for final adjustments and bug fixes without affecting ongoing development.
- **Hotfix branches** can also be branched off directly from `Production` to address urgent issues.

Keeping with this structure, the ideal process for the integration of new features should follow this path:

- Developers create feature branches to work on new workflows, tasks, or infrastructure components.
- Changes are merged into the `Development` branch, which corresponds to the Development Environment in Terraform.
- Once features are validated in the Development Environment, the `Development` branch is promoted to the QA Environment through a Release Branch.
- After testing and validation in the QA Environment, the changes are merged into the `Production` branch, which triggers deployment to the Production Environment.

3.7.2 Automated Deployment Pipelines

Through the use of Azure DevOps, the three main branches of the Git repository are directly linked to the infrastructure environments. Here is a general overview of how the systems are linked:

- **Code Validation and Formatting**: On every push to the repository, the pipeline checks Terraform and function code for syntax errors.
- **Infrastructure Validation**: Terraform's `plan` command is used to preview infrastructure changes, ensuring no unintended modifications are introduced.

- **Deployment**: Changes in specific branches (e.g., `Development`, `QA`, `Production`) trigger the deployment of corresponding environments using Terraform's `apply` command. Static assets are uploaded to appropriate locations (e.g., S3 or Snowflake).
- **Notifications**: The pipeline sends notifications for successful deployments, failures, or required manual approvals.

By integrating Gitflow with CI/CD pipelines, the data platform development process achieves a balance between flexibility and control, ensuring efficient updates while maintaining stability and reliability. This workflow also supports scalability, as multiple developers can work on different features simultaneously without interfering with one another.

Chapter 4

Evaluation of the New Solution

This chapter evaluates the effectiveness of the new infrastructure in addressing the limitations of the old system, as described in Chapter 2. By comparing the key pain points of the legacy platform—such as scalability, maintainability, reliability, and performance—with the features and capabilities of the newly designed architecture, this analysis provides a holistic view of the improvements achieved.

The design of the new platform was guided by the specific challenges faced by the client company, including the need for better data processing efficiency, improved workflows for development and maintenance, and enhanced reliability of data-driven operations. Each of these aspects is considered in detail to determine whether the new system successfully overcomes the limitations of the previous one.

Given the motivations behind the commission of this work by the client company, most of the metrics by which the new system can be evaluated are qualitative rather than quantitative. This chapter, therefore, focuses on assessing the architectural changes and their potential impact on resolving the identified issues.

4.1 Scalability

Scalability was one of the most important issues faced by the legacy system. The limitations of the on-premises Cloudera platform significantly restricted the client company's ability to scale operations, both in terms of data storage and compute resources. The new cloud-native design overcomes these barriers, providing both vertical and horizontal scalability.

4.1.1 Vertical Scalability

Vertical scalability refers to the ability to handle increased data volumes and the processing of more complex or resource-intensive data flows.

AWS S3 is the primary storage layer for raw and intermediate data in the new architecture. As a virtually unlimited storage solution, S3 removes the capacity constraints inherent to on-premises systems. The client company can scale up its storage usage on-demand, paying only for the space actually consumed. This enables seamless accommodation of growing data volumes without requiring upfront hardware investments.

The compute resources for processing tasks in the new system are dynamically scalable. Snowflake's architecture allows independent scaling of compute resources (virtual warehouses) and storage. For resource-intensive queries, such as those that parse and transform raw logs, a larger virtual warehouse can be allocated. As referenced in Section 4.5, the runtime of a heavier query can be minimized by assigning it a larger virtual warehouse, which enables faster processing at a higher cost per compute second. This adaptability ensures that even as data volumes increase, processing times remain manageable.

4.1.2 Horizontal Scalability

Horizontal scalability refers to the system's ability to accommodate new data flows or expand existing workflows without disrupting ongoing operations.

Each data source has its own dedicated ingestion pipeline, either through AWS DMS tasks or Lambda-based workflows. This modular approach allows the addition of new ingestion pipelines with minimal impact on existing processes. Each new flow can operate independently, scaling the platform's capacity to handle diverse data sources and formats.

The orchestration in Snowflake is organized into directed acyclic graphs (DAGs), where tasks are independent but interconnected. This setup makes it straightforward to add new processing tasks or entire workflows, as the system is not tightly coupled to any specific configuration.

The combination of AWS S3 and Snowflake ensures that adding new data flows does not affect the performance of existing ones. Each new flow can use its own compute resources and leverage S3's elasticity for storage, preventing bottlenecks.

4.2 Maintainability

The limitations regarding maintainability of the legacy system were another fundamental issue, primarily due to its disorganized codebase, lack of documentation, and the absence of separate environments for development and testing. The new cloud-native design addresses these issues comprehensively by adopting modern best practices for infrastructure management and workflow development.

4.2.1 Infrastructure as Code

One of the most impactful changes is the adoption of Infrastructure as Code (IaC) through Terraform. This approach ensures that the infrastructure of the data platform is fully described in a centralized codebase, stored, and version-controlled in a git repository. With every change documented, traceable, and reversible, the chaotic and error-prone nature of ad hoc modifications seen in the legacy system is eliminated.

Terraform provides consistency and reproducibility, allowing the infrastructure to be recreated reliably across environments, reducing the risk of configuration drift and deployment errors. The Terraform codebase itself is modular, with distinct modules for AWS and Snowflake components, simplifying maintenance and enabling independent updates to specific parts of the system.

4.2.2 Continuous Integration/Continuous Deployment

Complementing IaC is a robust Continuous Integration/Continuous Deployment (CI/CD) pipeline implemented using Azure DevOps. This pipeline automates the deployment of infrastructure changes and workflow updates, integrating structured testing and validation processes.

The deployment process is streamlined across three distinct environments for development, quality assurance (QA), and production. Updates and changes are first introduced in the development environment for initial testing, then validated in the QA environment before being promoted to production. This multi-environment structure ensures that new changes can be thoroughly evaluated without risking disruption to live operations.

4.2.3 Modular and Documented Design

The design of the new system emphasizes modularity, which simplifies maintenance and reduces the risk of errors. Each processing step is implemented as an independent module with clear boundaries while maintaining logical connections with other modules in the workflow.

This modular structure improves code readability, making it easier for developers to understand and modify the system. Furthermore, the risk of unintended side effects from updates is minimized, as changes to one module are less likely to impact others. Comprehensive documentation accompanies this design, ensuring that developers can navigate and work with the system effectively, further enhancing maintainability.

4.2.4 Collaborative Development with GitFlow

The development process is further structured and collaborative due to the adoption of the GitFlow branching model for source control. GitFlow introduces a clear branching strategy where developers work on new features or fixes in dedicated feature branches.

The branches are reviewed and tested before being merged into the main development or production branches, fostering collaboration and ensuring that all changes are peer-reviewed. This structured workflow reduces the likelihood of introducing errors and supports a disciplined approach to platform evolution.

Overall, the transition from the legacy system to the new platform marks a significant improvement in maintainability. By replacing chaotic and fragile processes with structured, documented, and automated practices, the new system enables controlled and predictable evolution.

The combination of Terraform, CI/CD pipelines, modular design, and GitFlow effectively addresses the limitations of the old infrastructure, significantly reducing the operational burden on developers and ensuring that the platform remains adaptable and future-proof.

4.3 Reliability

The reliability associated with cloud-based architectures was valued significantly by the client company when discussing possible designs. Hardware failures, maintenance requirements, and the lack of redundancy sometimes lead to unexpected downtime of the on-premises infrastructure, negatively impacting business operations. The new cloud-native design directly addresses these challenges by leveraging the inherent reliability features of the AWS and Snowflake platforms.

At the core of the improved reliability is the use of AWS's globally distributed cloud infrastructure. AWS services, such as S3 for storage and Lambda for compute, are designed with fault tolerance and high availability in mind. S3 ensures data durability through replication across multiple availability zones, effectively safeguarding against hardware or regional failures. Similarly, Lambda automatically scales its execution environments to meet demand, ensuring consistent task execution without manual intervention. Snowflake, deployed in the AWS environment, inherits these advantages while also providing its own fault-tolerant architecture. The separation of compute and storage layers in Snowflake enables it to maintain operation during compute failures, as storage remains unaffected.

The use of CI/CD pipelines also contributes to reliability. Infrastructure updates and workflow changes undergo structured testing and validation before deployment. This process ensures that the production environment remains stable and functional even as the platform evolves. Furthermore, the separation of environments provides an additional layer of protection, as changes

are thoroughly vetted before being introduced to live operations.

Compared to the legacy system, the new design significantly reduces downtime and ensures uninterrupted data processing. By leveraging cloud-native technologies and adopting automation and structured workflows, the platform achieves a level of reliability that was previously unattainable. This enhancement translates directly into improved business continuity and confidence in the platform's ability to support operations.

4.4 Observability

Another advantage that comes with modern cloud-native infrastructure is the variety of observability tools available by default to its administrators. These tools enable numerous approaches to analyze the behaviors, performance and cost of single components as well as entire data flows.

4.4.1 Observability in AWS

AWS offers a suite of monitoring tools that enhance visibility into the ingestion workflows. Amazon CloudWatch is central to this effort, providing metrics, logs, and alarms for various AWS resources. For example, CloudWatch tracks execution metrics for Lambda functions, such as invocation count, duration, and error rates, giving insight into the performance and reliability of the ingestion tasks. Scheduled DMS tasks are similarly monitored through CloudWatch, where metrics like task progress, throughput, and errors are logged for review.

To address anomalies proactively, CloudWatch allows the creation of alarms based on defined thresholds for key metrics. For instance, an unusually high error rate in Lambda function executions can trigger alerts sent via SNS, notifying operators of potential issues in real time.

4.4.2 Observability in Snowflake

Snowflake provides robust observability features tailored to data processing and query performance. Query profiling is one of the most valuable tools, allowing users to analyze the execution plan and performance metrics for every query. This level of detail helps identify inefficiencies in SQL logic, resource contention, or suboptimal compute allocation.

For broader platform monitoring, Snowflake generates account usage views and performance statistics, such as warehouse activity, credit consumption, and storage usage. These metrics offer insights into resource utilization trends, enabling informed decisions about warehouse resizing or optimization. The Snowflake Information Schema complements these capabilities by exposing metadata about database objects, queries, and tasks, which can be queried for further analysis.

Error handling and anomaly detection are facilitated by task execution logs in Snowflake. Each task records its status, runtime, and any failures encountered. These logs, combined with dependency DAGs, help trace issues in workflows and assess their impact on downstream tasks.

4.5 Performance

Performance was not one of the primary motivators for the re-engineering of the data platform. The legacy system's batch-oriented processing model, which ingests and processes data daily, generally allowed sufficient time to complete tasks. However, the transition to the new architecture has introduced notable performance improvements, particularly in query execution times, and has set the stage for greater scalability in the future.

For example, a significant improvement can be observed in the execution of a query responsible for parsing raw logs into tabular format, as shown in Figure 4.1. This process involves both extracting specific fields and applying regular expressions to structure unformatted data. The Snowflake-based solution demonstrates faster execution times, even when using a "Small" virtual warehouse. By scaling up to a "Medium" virtual warehouse, execution times can be further reduced, highlighting Snowflake's ability to leverage additional compute resources effectively.

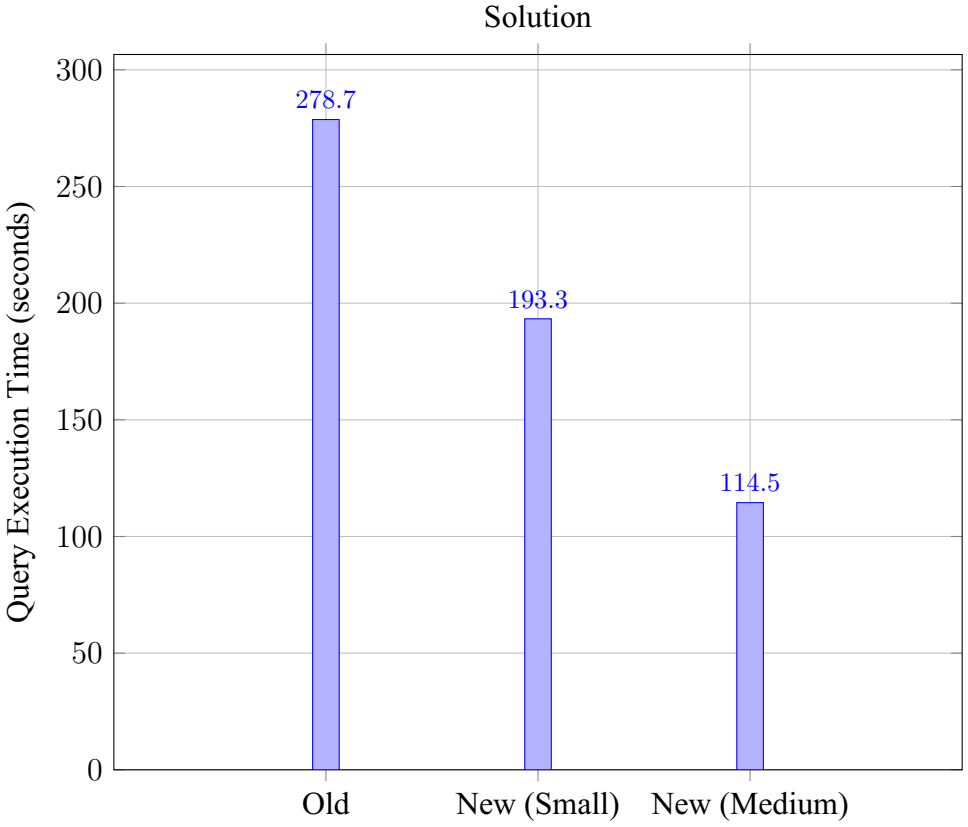

Figure 4.1: Query execution time averages over 10 runs.

This is relevant because it relates to the system's vertical scalability. Operations that are parallelizable, such as those involving multiple independent partitions of data, benefit particularly from this capability. Increasing the virtual warehouse size allocates more compute power to the queries, resulting in shorter execution times at the cost of increased resource consumption. This trade-off provides flexibility for handling increased data volumes or more computationally intensive tasks without compromising performance.

In conclusion, while performance improvements were not the primary focus of the redesign, the new architecture offers both immediate benefits and long-term potential. Snowflake's ability to manage execution times through warehouse scaling ensures that the platform remains responsive even as data volumes grow or operational needs evolve. This positions the data platform to meet future performance demands effectively.

4.6 Challenges and Limitations of the New Design

While the new data platform addresses many of the limitations of the legacy system, it introduces its own set of challenges and considerations that must be carefully managed to ensure its continued effectiveness. Cloud infrastructure proved to be an excellent fit for the client company's requirements, however, on-premises solutions still hold value in other contexts, especially when data security, regulatory compliance, or latency are critical concerns. In certain industries or specific use cases, where control over infrastructure and data is paramount, on-premise systems may still offer advantages that cloud solutions cannot fully replicate. Therefore, while cloud services are increasingly popular, on-premises solutions remain a viable option in scenarios requiring tighter control and customization.

4.6.1 Costs

One of the most significant challenges associated with cloud infrastructure is its cost[30]. While the company has not disclosed the budget for maintaining its current on-premises data platform, preliminary analyses suggest that the new solution is relatively cost-effective in its current form. However, the inherent scalability of cloud infrastructure comes with the risk of runaway costs. As the platform evolves, adding new data flows, increasing resource usage, or expanding storage, the cost structure can become increasingly complex.

It will be essential for the company to implement robust cost monitoring and management practices, such as periodic audits, usage alerts, and budgeting tools, to prevent budget inflation and ensure the platform remains financially sustainable.

However, it is important to recognize that implementing these cost monitoring and management tools and processes can themselves incur substantial costs. Budgeting tools, usage

tracking systems, and periodic audits often require specialized software, additional personnel, or consulting services, all of which add to the operational expenses.

Furthermore, the complexity of managing cloud infrastructure costs can necessitate ongoing investment in training and resources to ensure the team is equipped to effectively use these tools. As the platform evolves, the company must balance the benefits of these cost management measures with the expenses they generate, ensuring that the return on investment justifies the additional overhead.

4.6.2 Documentation and Code Management

The current design emphasizes best practices in documentation and code organization. Processes like Infrastructure as Code (IaC) through Terraform, alongside the adoption of a CI/CD pipeline and Gitflow branching strategies, set a solid foundation for maintainability and collaboration. However, these practices are inherently flexible and will require ongoing governance to ensure adherence.

As the company's teams begin managing and extending the platform, there is a risk of diverging from established conventions, potentially leading to a fragmented codebase or inconsistent documentation.

Proactive oversight, regular training, and internal reviews will be essential to maintain the platform's long-term manageability.

4.6.3 Vendor Lock-in

The decision to adopt a specific cloud provider, in this case, AWS paired with Snowflake, introduces the potential risk of vendor lock-in. The company is now reliant on these vendors' infrastructure, pricing structures, and terms of service. Should the providers increase costs or change terms in unfavorable ways, the company will face the difficult choice of accepting the new conditions or incurring significant expenses to migrate to alternative platforms.

Mitigating this risk would require ongoing evaluation of the vendors' offerings and policies, as well as exploring multi-cloud strategies or designing components that are less tightly coupled to proprietary features where feasible.

4.6.4 Operational Complexity

The new architecture introduces modern tools and paradigms, which may require a steep learning curve for existing staff. Technologies like Terraform, Snowflake, and AWS's myriad services, while powerful, demand a deep understanding to use effectively.

Without proper training and documentation, there is a risk of misconfiguration or under-utilization of the platform's capabilities. Additionally, the distributed nature of the platform, spanning AWS and Snowflake, increases operational complexity.

Ensuring that teams have access to continuous learning resources and fostering a culture of cross-team collaboration will be necessary for the long-term success of the platform.

In conclusion, while the new design resolves many of the challenges faced by the legacy system, it is not without its limitations. Conscious management, regular reviews, and proactive governance will be essential to navigating these challenges and ensuring that the platform remains a valuable asset to the organization over time.

Chapter 5

Conclusion and Future Work

In this thesis, we have explored the re-engineering of a legacy data platform, transitioning from an on-premises Cloudera-based solution to a modern, cloud-native architecture using AWS and Snowflake. The primary goal of the project was to address key pain points such as scalability, maintainability, and reliability, which were inherent in the old infrastructure. By leveraging the cloud's scalability, the flexibility of Infrastructure as Code (IaC), and the performance capabilities of Snowflake, the new design aims to provide a more efficient, reliable, and cost-effective solution for the client company's data processing needs. The re-engineered platform is poised to scale with the company's data growth, easily integrate new data sources, and be adaptable to future business requirements, all while ensuring maintainability through best practices like CI/CD and Terraform-based deployments.

The thesis also provided an examination of how the new platform improves on the old one in key areas like scalability, performance, and observability. While performance was not a central focus of the redesign, some improvements were observed, especially in the system's ability to scale vertically by adjusting compute resources. Observability features in both AWS and Snowflake contribute to enhanced monitoring and proactive issue resolution. However, challenges such as cost management, vendor lock-in, and the need for continuous maintenance remain, underscoring the need for ongoing oversight as the system evolves.

In conclusion, the project successfully met its objectives, providing a robust foundation for future developments. The new design not only resolves the issues of the old platform but also opens new avenues for leveraging advanced technologies and enhancing the company's data capabilities.

5.1 Future Work

While the new platform is a significant improvement, there are several areas for future development that can further enhance its value:

- **Development of New Data Flows**: With the new system now firmly established, there is significant potential for developing new data flows that can take advantage of the platform's ability to handle large volumes of data. The new architecture is also well equipped to deal with streaming data and provide real-time functionality.
- **Integration of ML and AI Tools**: Another potential avenue for the platform's evolution is the integration of machine learning (ML) and artificial intelligence (AI) tools provided by AWS[31] and Snowflake[32]. By incorporating these advanced analytics capabilities, the company can enhance its data-driven insights in ways that were not previously feasible, further supporting business decisions and strategic planning.
- **Cost Management Processes**: As cloud infrastructure can become expensive, particularly as the platform scales, it will be essential to establish strong cost management processes. This will involve setting up automated monitoring and alerting systems to track resource usage and costs, as well as implementing budgeting tools and conducting periodic cost audits to ensure the system remains cost-effective as it evolves.
- **Tuning Based on Long-Term Performance**: Finally, long-term performance tuning could prove beneficial to optimize the platform's resource usage. While Snowflake allows for dynamic scaling of compute resources, it is essential to continuously monitor the performance of key queries and workloads. Over time, adjustments can be made to ensure that resources are allocated efficiently, minimizing unnecessary costs while maintaining the system's performance under growing data loads.

These are some of the possible paths that would lead the platform to continue meeting the company's evolving business needs, remain efficient, and capitalize on emerging technologies to enhance data processing and analysis capabilities.

Bibliography

[1] *What is a data warehouse?* https://cloud.google.com/learn/what-is-a-data-warehouse, Accessed: 2024-06-10, Google Cloud.

[2] N. Ricks, *The rise and evolution of data warehouses*, https://www.polytomic.com/blog-posts/the-rise-and-evolution-of-data-warehouses, Accessed: 2024-06-10, Polytomic.

[3] B. Dageville, T. Cruanes, M. Zukowski, *et al.*, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, Snowflake Computing, San Francisco, CA, USA: ACM, 2016, pp. 215–226, isbn: 978-1-4503-3531-7. doi: 10.1145/2882903.2903741.

[4] V. Borkar, M. Carey, and C. Li, "Big data platforms: What's next?" *ACM Crossroads Student Magazine - XRDS*, vol. 19, Sep. 2012. doi: 10.1145/2331042.2331057.

[5] K. Smith, N. Goble, and D. Rocco, *What is snowpark — and why does it matter? a phdata perspective*, https://www.phdata.io/blog/what-is-snowpark/, Accessed: 2024-06-10, phData.

[6] B. Tuychiev, *Using snowflake time travel: A comprehensive guide*, https://www.datacamp.com/tutorial/using-snowflake-time-travel-a-comprehensive-guide, Accessed: 2024-06-10, Datacamp.

[7] S. Shah and S. Teal, *Snowflake launches unstructured data support in public preview*, https://www.snowflake.com/en/blog/snowflake-launches-unstructured-data-support-in-public-preview/, Accessed: 2024-06-10, Snowflake.

[8] *Snowflake zero copy cloning*, https://thinketl.com/snowflake-zero-copy-cloning/, Accessed: 2024-06-10, ThinkETL.

[9] H. Roy, *What is snowflake dynamic data masking?* https://www.phdata.io/blog/what-is-snowflake-dynamic-data-masking/, Accessed: 2024-06-10, phData.

[10] S. Vemula, *An introduction to snowflake's marketplace*, https://interworks.com/blog/2024/07/16/an-introduction-to-snowflakes-marketplace/, Accessed: 2024-06-10, interworks.

[11] M. Michalowski, *55 cloud computing statistics for 2024*, https://spacelift.io/blog/cloud-computing-statistics, Accessed: 2024-06-13, spacelift.

[12] S. Wickramasinghe, *What are saas, paas, iaas?* https://www.splunk.com/en_us/blog/learn/cloud-service-models-saas-vs-paas-vs-iaas.html, Accessed: 2024-06-13, Splunk (Cisco).

[13] *Amazon S3 Documentation*, https://docs.aws.amazon.com/s3/index.html, Accessed: 2024-06-13, Amazon Web Services.

[14] *AWS Lambda Documentation*, https://docs.aws.amazon.com/lambda/index.html, Accessed: 2024-06-13, Amazon Web Services.

[15] *AWS Database Migration Service Documentation*, https://docs.aws.amazon.com/dms/index.html, Accessed: 2024-06-13, Amazon Web Services.

[16] *Amazon Simple Notification Service Documentation*, https://docs.aws.amazon.com/sns/index.html, Accessed: 2024-06-13, Amazon Web Services.

[17] *Terraform Docs Overview*, https://developer.hashicorp.com/terraform/docs, Accessed: 2024-07-22, HashiCorp.

[18] *Azure DevOps Documentation*, https://learn.microsoft.com/en-us/azure/devops/, Accessed: 2024-07-22, Microsoft.

[19] *Cloudera Documentation*, https://docs.cloudera.com/, Accessed: 2024-06-03, Cloudera.

[20] *Hadoop HDFS Documentation*, https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, Accessed: 2024-06-03, Apache Software Foundation, 2024.

[21] *Apache Sqoop Documentation*, https://sqoop.apache.org/docs/1.99.7/, Accessed: 2024-06-03, Apache Software Foundation.

[22] I. Cook, *Apache Hive Language Manual*, https://cwiki.apache.org/confluence/display/Hive/LanguageManual, Accessed: 2024-06-03, Apache Software Foundation, 2018.

[23] *Introducing Apache Impala*, https://impala.apache.org/docs/build/asf-site-html/topics/impala_intro.html, Accessed: 2024-06-03, Cloudera.

[24] *Apache Spark - A Unified engine for large-scale data analytics*, https://spark.apache.org/docs/3.5.3/, Accessed: 2024-06-03, Apache Software Foundation.

[25] *Apache Hadoop YARN*, https://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/YARN.html, Accessed: 2024-06-04, Apache Software Foundation, 2021.

[26] *About using Hue*, https://docs.cloudera.com/cdw-runtime/cloud/using-hue/topics/hue-using.html, Accessed: 2024-06-04, Cloudera.

[27] *Data center management – knowing when to upgrade hardware*, https://siteltd.co.uk/blog/data-center-management-hardware-upgrade/, Accessed: 2024-06-07, Secure I.T. Environments Ltd.

[28] *Separation of environments: Keeping production data safe in development*, https://www.synthesized.io/post/separation-of-environments-keeping-production-data-safe-in-development, Accessed: 2024-06-08, Synthesized.

[29] J. Nduhiu, *What is service continuity management?* https://www.splunk.com/en_us/blog/learn/service-continuity-management.html, Accessed: 2024-06-10, Splunk (Cisco).

[30] K. Lange, *Cloud Costs: Cloud Cost Management Strategies*, https://www.splunk.com/en_us/blog/learn/cloud-cost-management.html, Accessed: 2024-09-09, Splunk (Cisco).

[31] *Leveraging new technologies (AI/ML/analytics)*, https://docs.aws.amazon.com/prescriptive-guidance/latest/mes-on-aws/ai-ml.html, Accessed: 2024-10-17, Amazon Web Services.

[32] *Snowflake ML: End-to-End Machine Learning*, https://docs.snowflake.com/en/developer-guide/snowflake-ml/overview, Accessed: 2024-10-17, Snowflake.