Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Data Lineage for ETL pipelines in Production

with AWS Glue Studio and Spline

Author: Kaixi Ma (VU: 2658008, UvA: 12536016)

1st supervisor:	Adam S.Z. Belloum	University van Amsterdam
daily supervisor:	Wuyou Liu	Dashmote B.V.
2nd reader:	Zhiming Zhao	University van Amsterdam

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

October 16, 2022

"I am the master of my fate, I am the captain of my soul"
from Invictus*, by William Ernest Henley*

Abstract

When processing big data, data lineage enables engineers to efficiently check the data source and debug issues. The goal of this paper is to build a data lineage subsystem for fast-changing ETL jobs in an industry setting. We use AWS Glue Studio to create ETL jobs and then integrate with Spline to automatically track lineage information. Data lineage enables data engineers to debug more efficiently, provides a clear data processing structure, and maintains consistency in how data is transformed.

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface

AWS Amazon Web Services

CloudWatch An AWS service for monitoring and managing AWS resources and applications

Docker A software platform for rapidly building, testing, and deploying applications.

EC2 Elastic Compute Cloud, a web service that provides resizable computing capability

ECR Amazon Elastic Container Registry, a service used for storing and managing the Docker image

EMR Elastic MapReduce, a managed cluster platform that simplifies running big data frameworks

ETL Extract, transform, load

REST REpresentational State Transfer, an architectural style for providing standards between computer systems on the web

S3 Simple Storage Service, an object storage service offered by AWS

UI User Interface

1

Introduction

In the big data era, raw data is gathered from numerous sources and processed with multiple data transformations. When processing big data, data engineers are faced with multiple questions with respect to the data: Where is the data source? Whether new data has been updated? When running data pipelines, how can they debug the problem when there are errors occur? How can they target the problem?

Data lineage, also known as "data provenance" or "data pedigree", is the description of the origins of a piece of data and data transformation (2). Because both data lineage and data provenance aim to describe data sources and the processed data, they are often used interchangeably (3)(4)(5). *Eugene Wu et al.* (6) distinguish between the two concepts by defining data lineage as a critical component of data provenance, and it is used to identify relationships between input and output data elements as well as to debug workflows. Data provenance is more advanced and is typically used to give business users an overview on the origin of the data. In contrast, data lineage focuses on data transformation and life-cycle management (3).

Data lineage brings many benefits to industrial production. It gives the company a bird's-eye view of the entire data processing lifecycle, covering data sources and all the data transformations which led to a given output data. It also helps developers find problems efficiently when errors occur. While there are many benefits to engineers in developing big data provenance in production, but there are also some challenges. For example, how to save significant amounts of data, maintain computational efficiency, etc.

The work described in this thesis has been developed as an internship at *Dashmote*[1], which provides big data analytics services to food and beverage companies, such as Red Bull, Coca-Cola, etc. As a data-driven company, there are multiple data pipelines used

[1] `https://dashmote.com/`

for processing big data, including data sourcing, pre-processing, machine learning, and post-processing. When designing and executing data processing pipelines, it is frequent to face errors. It could be caused by the source data, which then affects the entire data transformation. Data lineage can help engineers conduct in-depth data analysis to find problems, which benefit the efficiency of the industrial production.

This thesis aims to design and implement data lineage systems at *Dashmote*. Data lineage is introduced at *Dashmote* to improve the quality of the existing data production processing pipelines.

This thesis describes the process of designing and implementing data lineage using AWS Glue Studio and Spline. The rest of the thesis is structured as follows: Chapter 2 introduces the application scenarios, benefits, and challenges in data lineage, then states the problems encountered and research questions. In Chapter 3 introduces techniques and tools of data lineage. Chapter 4 elaborates the project implementation process, including the project requirements, data architecture, how AWS Glue integrates with Spline, etc. Chapter 5 discusses this graduation project's evaluation, reflection and future work. The conclusion and answers to research questions are presented in Chapter 6.

2

Background

In this chapter, we introduce application scenarios of data lineage in scientific and business domains. Then, we elaborate on the benefits and challenges of data lineage. They serve to illustrate the problem statement, which requires data lineage systems. At the end of the chapter, we list and discuss the research questions which will be driving our work.

2.1 Application Scenarios in Data Lineage

Simmhan et al. (7) introduced the application of data provenance in several scientific and business fields. In the following section, we summarize a few examples in both scientific and industrial domains.

2.1.1 Scientific Domains

Digital Object Identifier (DOI) The Digital Object Identifier is a string of numbers used in scientific publications/research to identify a specific paper or experiment record, including experimental procedures and results, providing readers with an understanding of how these papers relate to one another and forming a data lineage.

Geographic Information System (GIS) The Geographic Information System is a computer system used to collect, store, manage, analyze, and track geographic information. In GIS, data lineage is often used to describe the history of the source data, the method of obtaining the data, and how data is transformed. Users of GIS can generate new spatial data by manipulating and combining existing data, providing new perspectives on the data. In addition, Data lineage information assists developers in determining data quality.

Materials Science Data lineage is also critical for materials science, such as in the aerospace industry. Low-quality materials or inaccurate data can have highly negative consequences and impact operational performance. Sometimes it can be difficult for engineers to detect data quality because the data may look similar. However, by leveraging data sources, materials engineers can track whether the data is trustworthy, which can help detect faulty components and avoid production failures.

Life Science Research Data lineage can also be used in life science research, it has three significant advantages. First, the traditional method of sharing biological knowledge is through the publication of a paper. However, data sources can provide an analog of citations, allowing researchers to more effectively share biological and biomedical information (8). Second, data lineage can improve data reliability by tracking data sources and transformations. Third, data can be automatically validated and updated, ensuring that less relevant data becomes obsolete.

Astronomical Science Astronomers are working to build an international virtual observatory and provide the computing resources needed for data science. As astronomy has advanced, astronomers' work has grown from an individual to an increasingly collaborative one. Therefore, they rely not only on data from them but also on data from other sources, making the source and lineage of the data more important to them (9). Data lineage can help astronomers assess the trustworthiness and quality of data from third-party sources, provide semantic meaning and assist scientists in integrating it into their data processing system.

2.1.2 Business Domain

Business users often work with well-organized data schemas and trusted data sources. However, dirty sources will impact the entire data process. Correcting the procedure becomes expensive and time-consuming. The information on data lineage can assist data analysts in tracing the incorrect data back to the source and checking the data type. It can be modified and updated based on data source changes and transformations.

2.2 Benefits of Data Lineage

There are some benefits of data lineage for business, engineering, and science domains. *Webjørnsen et al.*(10) states some data lineage benefits in the data warehouse, which are summarized below:

In-depth Data Analysis If the data is traceable, data scientists and engineers will better understand the dataset. The ability to view and track data flow from source to destination helps data scientists understand the quality and lineage of a particular field or dataset. Data engineers gain greater insight into the entire data pipeline, and the dependencies between each dataset (11). Xavier (12) claims another case in business scenarios: sometimes, strange graphs may appear on the dashboard, such as extremely low values on a line chart. The cause can be a data pipeline failure or accidental deletion of a column. Using data lineage techniques, the data can be traced back to its source to determine where the data came from and what went wrong.

Impact Analysis Suppose data in a table needs to be changed. In that case, data lineage technologies allow us to examine the impact of the change before implementing the change or breaking the entire data pipeline. Impact analysis can also be used to determine which tables, columns, and processes will be affected by changes.

Debugging When unusual data occurs, or data pipelines fail unexpectedly, data lineage can help engineers determine if the upstream data pipeline is corrupt or if the code has a problem. At the same time, we can understand who is using current data and who is affected by pipeline failures.

Assisting with Data Mining If the input data is accurate and trustworthy, the output data will be reliable. Thus, knowing where the data comes from can improve data mining and discovery by increasing the credibility of the data.

2.3 Challenges of Data Lineage

Implementing data lineage can provide many benefits, but also some challenges. In this section, some main challenges and possible solutions will be elaborated on.

Large Data Volume *Wang et al.* (13) claim that the data lineage contains an excessive amount of data. The amount of data is much more significant than the data which needs to be processed because data lineage includes data flow paths from source to destination. A possible solution might be to save the data more efficiently or to reduce the amount of data that does not contain that target feature. Since big data processing is crucial for data lineage technology, traditional methods cannot handle massive data (14). Another solution is an approximate lineage, which compresses data by only keeping track of the essential

derivations (15). This method not only improves query performance but also minimizes data storage.

Computational Efficiency Data lineage techniques can be resource-consuming, and data-intensive (14). Lineage approaches must be designed and implemented with minimal computational overhead to avoid affecting the target system's performance. There are two methods for computational efficiency. One approach is to compute the lineage only when needed, known as the lazy lineage model. In contrast, the eager lineage model computes the lineage each time the data is transformed. Both models have their own set of benefits and drawbacks.

Sheikh et al. (16) propose another technique to improve computational efficiency called provenance inference. It has very little storage overhead when compared to explicit provenance. Rather than obtaining all the data from a data lake, which would incur additional storage costs, provenance inference can save execution time and money by utilizing fewer computing resources (17).

Quality: Accuracy and Consistency Accuracy and consistency comprise the lineage quality factor. Because data lineage is mined from log files, accuracy is one of the most important factors when designing a data lineage approach. Sometimes the lineage tool receives an event that it does not know how to process or intentionally ignores because it "thinks" there is no helpful information associated with it from the perspective of lineage tracking. The lineage may not be captured as expected in such cases. If there are some inaccurate captures, there may be some errors in lineage tracking. Capturing inaccurate, duplicate, or conflicting lineage records can also result in inaccurate lineage.

Consistency is another quality challenge for data lineage. It may occur if two lineage traces are supposed to form one (18). Timestamps are a good example because they vary in different regions. If two lineage traces have both MM/DD/YYYY and DD/MM/YYYY formats, the lineage may be confused because it does not know which format it should be. According to their findings, the quality dimension is used to increase data lineage quality details.

Resource Utilization Resource utilization refers to CPU utilization, memory usage, etc. Based on the summary of data lineage challenges by *Sheikh et al.* (16), resource utilization should not be ignored because CPU utilization increases when data lineage is captured. However, few researchers think this is important.

2.4 Problem Statement

Data pipelines play an essential role for companies in dealing with big data. However, we might encounter problems when using data pipelines. Following are some problems which can be solved by introducing the data lineage subsystem.

All data transformations lack clarity without visualization or a method to examine how the data is processed. At *Dashmote*, ETL pipelines consist of PySpark[1] components running on AWS EMR[2] orchestrated by Apache Airflow[3]. There is no data lineage system implemented. Engineers sometimes use Spark UI (19) to investigate issues encountered in the data processing (Figure 2.1). Spark UI is the web interface that allows users to view and examine Spark job executions while a Spark application is running. It can provide execution job details. However, it is difficult to extract data lineage with Spark UI. In addition, the Spark UI might be too complex for technical users and sales departments.

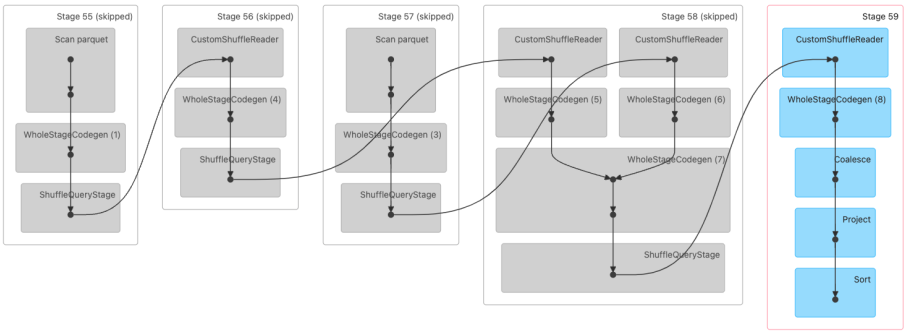

Figure 2.1: Spark UI example of a Spark job

At *Dashmote* we have noticed that new engineers struggle to understand how data is being used. In other words, when engineers want to check how data is produced, they have to read the code repositories to understand the process, which is time-consuming, complicated, and unnecessary because they need to understand the logic behind the code to understand the details of data transformations. Reading the code of each repository is sometimes not enough to understand all the inputs and outputs and the processing steps in between.

Sometimes, companies use diagrams to describe their assets, which are difficult to maintain. Post-processing is the final step before delivering data to clients, and it creates a custom output DataFrame after the standard output. Figure 2.2 describes the structure

[1] `https://spark.apache.org/docs/latest/api/python/`
[2] `https://aws.amazon.com/emr/`
[3] `https://airflow.apache.org/`

of post-processing steps, including how input data is processed in different post-processing components. Because if we need to make some changes or improvements, we have to understand the order of the whole graph and program before making changes, it is more efficient to automatically generate data lineage based on changes in data transformations.

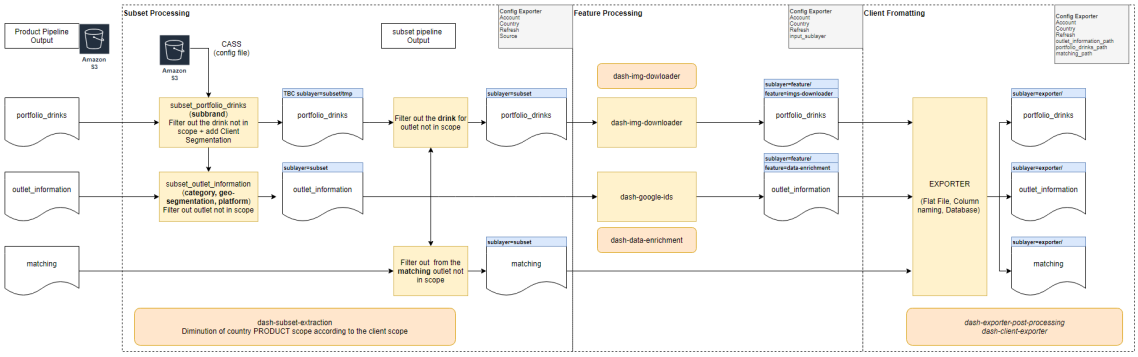

Figure 2.2: The structure of post-processing, including input and output data, and data transformations in between.

Dashmote is currently using Apache Airflow to orchestrate its ETL pipeline. Airflow UI provides visualization of ETL DAGs (Directed Acyclic Graph). However, the visualization of DAGs highly depends on the implementation of airflow operators. It provides limited information on data lineage.

As a data-driven company, *Dashmote* needs to run many components or ETL pipelines to process the data daily. Data pipelines and machine learning models are widely used in industries to process big data and complete business analysis. Suppose issues are detected in one of the company's data pipelines, or data engineers want to know how to change some data from the previous step. In that case, it is difficult for them to identify the source of the problem efficiently.

2.5 Research Questions

After reviewing the most critical problems encountered at *Dashmote*, and we have decided to focus our work on solving the following research questions and build our work on the state-of-the-art research and technology in data lineage.

Research Question 1: *How to build a versatile data lineage subsystem for a highly complex, fast-changing data pipeline in an industrial setting?*

- This is the main research question of this paper. Since there is no data lineage system implemented and no visualization of how data is processed, we want to provide the

most suitable data lineage solution for *Dashmote* and analyze its advantages and disadvantages.

Research Question 2: *What techniques, tools, frameworks can we use to implement the data lineage?*

- A survey of the state-of-the-art of existing data lineage techniques, tools, and frameworks will help us to buildup on recent achievements in Science and technology on the topic of data lineage and will help us to focus our contribution to the most important and unsolved problems.

Research Question 3: *What are the challenges when developing data lineage?*

- Since the topic of data lineage is new *Dashmote*, our work will serve also to explore and discover the challenges for developing a data lineage system. Some of these challenges will be addressed in our work, others will be the focus of future projects within the company.

3

Related Work

In this chapter, some related work on data lineage is discussed. Then, the standard techniques used to implement data lineage are elaborated, including pattern-based lineage, parsing, data tagging, and self-contained lineage. Furthermore, some implementation challenges and possible solutions are explained. Finally, some data lineage tools are summarized with their benefits and drawbacks.

3.1 Techniques

Some common techniques which are used to implement data lineage are mentioned below:

Pattern-Based Lineage The pattern-based lineage does not need to deal with code that transforms the data. Metadata for tables, columns, and business reports are evaluated, and it employs patterns in the metadata to perform lineage (20). The main advantage of pattern-based lineage is that it only processes data, but not any algorithms. It can be applied to any database technology, such as Oracle, MySQL, or Apache Spark. The downside, however, is that it is not always accurate. In some cases, it might miss the connection between datasets or transformations if the data processing logic is not in human-readable metadata or hidden in code. It can lose the connection between datasets or transformations.

Lineage by Parsing In contrast to the pattern-based strategy, parsing techniques rely on the logic behind the processed data. It automatically reverses the data transformation logic and then generates detailed lineage tracing (20). It is more accurate since all logic from the code can be parsed. However, the implementation is complex because it requires knowledge of programming languages as well as the tools needed to transform the data,

which could include ETL (extract, transform, load) logic, SQL solutions, Spark solutions, etc.

Lineage by Data Tagging The data tag can be tracked to create a data lineage representation if a transformation engine tags the data that is processed. This technique only works if the transformation engine is reliable to tag all the data (21). The disadvantage is that data lineage cannot be created if data tags are missing.

Self-Contained Lineage Some organizations have their data environment, providing storage, data processing logic, and master data management (MDM) to handle data sources and metadata. These environments contain a data lake where all data from all stages of their lifecycle can be stored (20). Self-contained systems can provide data lineage on their own without the need for additional data lineage tools. However, lineage relies on a self-contained system, in which case lineage cannot be executed without metadata provided in the self-contained system.

3.2 Tools

In this section, some data lineage tools will be summarized as follows, and more details can be found in (22).

SAP PowerDesigner SAP PowerDesigner is an enterprise modeling tool that can provide different models to model relational databases, such as conceptual, logical, and physical models. SQL DDL (Data Define Language) can be generated and applied based on a physical data model. Conversely, data models can be generated based on databases, and models can be converted to each other. Furthermore, it can also provide an enterprise architecture model, a requirements model, and a business process model. Those models are useful to be used to generate data lineage, create visualization graphs, check data quality, etc.

ETL Tools Some data modeling tools, such as SAP PowerDesigner, can provide data lineage information during the modeling step. SAS Data Integration Studio (SAS DI) is an ETL tool which provides visual design for the building, implementing, and managing of data integration processes[1]. It can also display the data lineage for tables and external

[1] `https://support.sas.com/en/software/data-integration-studio-support.html`

files[1]. Ab Initio is another ETL tool that provides a scalable and robust metadata system for governance and data management applications[2].

Data Lineage Extraction Tools Some tools can extract data lineage information from different types of systems. Such as, Octopai[3], D-QUANTUM[4], and Manta[5], etc. These tools can acquire data lineage information from various databases and ETL tools. It can bring many benefits from lineage information, such as in-depth analysis, debugging, data mining, etc. However, those extraction tools have some limitations.

One of these limitations is that they are unlikely to be used on all platforms. Some companies, particularly large ones, usually use multiple databases and ETL tools. A single data lineage tool cannot extract lineage information from all platform lineage. One way to solve this problem is using different platforms' tools. Additionally, some closed systems do not allow third-party tools to access and extract their metadata, making it challenging to extract metadata. There is also a question about parsing the programming code. The data lineage extraction tool obtains lineage information by parsing the code. However, if a code is used to modify the input parameters rather than the data, this will bring challenges and ambiguity for the data lineage tools.

Also, these kinds of tools' snapshot characteristics must be considered. If the data lineage information is extracted for both data warehouses and reporting tools, these paths must be extracted simultaneously. That is because the data lineage information will be inconsistent if the extraction time for the data warehouse and reporting tools is different. Therefore, the frequency of updating the data lineage is a critical issue for extraction tools.

Graph Databases Graph databases store data in nodes and edges, which are non-relational databases. It can be used for modeling and data analysis because each node represents a data element, and edges represent connections to each other. Neo4j[6] is one of the most common graph databases.

Apache Atlas Apache Atlas is a metadata management tool in Hadoop or non-Hadoop ecosystems. It can provide metadata services for Hive, Ranger, Spark, Sqoop, etc.

It has an intuition UI that can provide the data lineage for every data processing. Furthermore, the REST APIs can access and update the lineage in real-time.

[1] https://documentation.sas.com/doc/en/etlug/4.904/p13kmxhmyi0o0hn1urr3q07nu0t1.htm
[2] https://www.abinitio.com/en/data-catalog-quality-governance/metadata-management-governance/
[3] https://www.octopai.com/
[4] https://synabi.com/en/
[5] https://getmanta.com/
[6] https://neo4j.com/

Apache Atlas has some benefits for data lineage. However, it lacks the support for Apache Spark (23).

Spline Spline[1] is an open-source tool for automatically tracking data lineage. It was initially designed for financial institutions because data lineage tracking is one of the most critical problems. Many organizations also use big data technologies, such as Apache Spark[2]. Hence, Spline, as a data lineage tool, is used for capturing and storing lineage information from the internal Spark execution plan of Apache Spark. As the project expands, it becomes possible to use it with other data technologies, not only for Spark.

There are three main manners of Spline (23):

- **Lightweight**: No heavy computation is performed during the execution of the Spark application. This character solves the computation challenge which is mentioned in Section 2.3.
- **Unobtrusive**: If the lineage capture or extraction fails in Spline, the Spark job is unaffected.
- **Easy to use**: Spline is easy to be implemented by initializing the library with a single line of code.

Spline consists of three main parts, and Figure 3.1 shows the Spline architecture:

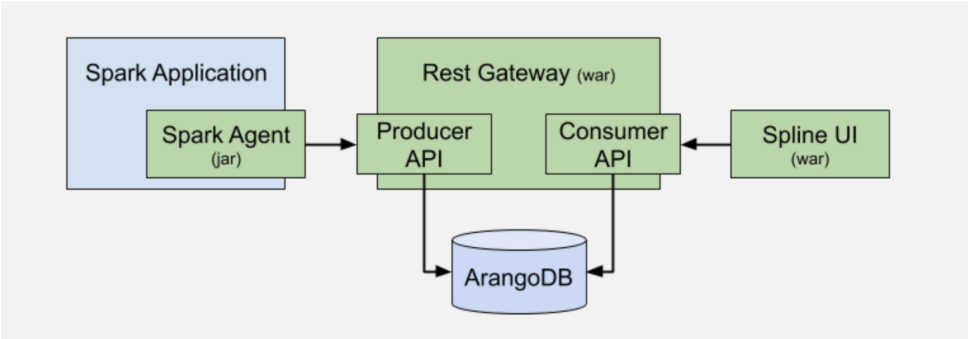

Figure 3.1: The Architecture of Spline (1)

- **Spline Server** is the core of Spline. It receives the lineage data from the *Producer API* and stores it in ArangoDB. In addition, it provides reading and querying lineage data features for *Consumer API*.

[1] `https://absaoss.github.io/spline/`
[2] `https://spark.apache.org/`

- **Spline Agents** capture the lineage and metadata from data transformation pipelines and send it to the Spline Server in a standard format using a *Producer API*. The final lineage data is processed and stored in the form of a graph and can be accessed via another REST API, also known as *Consumer API*.
- **Spline UI** is a lightweight HTTP server and is only used for data lineage visualization required by Spline UI. It can display table lineage, field lineage, and input and output schemas at various stages based on different applications.

4

Project Implementation

In this chapter, we describe the implementation process and details of data lineage. We begin by stating the project requirements based on the problem statement in Section 2.4. After that, we describe the proposed data lineage architecture design according to the project requirements. To validate our implementation, we describe the current ETL pipeline use case at *Dashmote* using Apache Airflow and the motivations to switch to AWS Glue Studio. Afterward, we describe the new ETL pipeline including the data lineage implementation details by integrating AWS Glue Studio with Spline to create data lineage. Finally, we demonstrate the visualization and utilization of the data lineage system.

4.1 Project Requirements

In Section 2.4, we list the problems and challenges facing data engineers at *Dashmote*. In this section, we provide concrete requirements we use to drive our design.

Debugging Supports It is normal to get errors when running some data pipelines during work. The reason could be data loss, incorrect data type, wrong code, etc. Engineers who cannot check information from data lineage spend much time checking where the data came from and the source data. Data lineage helps engineers keep track of data.

Consistency and Efficiency If we use graphs to describe how the data is processed, engineers need to update the graphs manually when some changes are applied, and the information cannot be updated in time. By using data lineage, information can be updated automatically and promptly, keeping lineage information consistent. Engineers can inspect components by clicking on the data, which improves efficiency.

Clear Structure of Data Processing Customers or new engineers need time to understand the entire data transformation process. If the structure is unclear, it can sometimes lead to confusion. Data lineage provides users with a clear and organized structure to understand the construction of data pipelines and even more detailed information about the data. It saves even more time when introducing the entire data processing step, and it is easy to understand by technical and non-technical users with minimal support.

Scalability and Maintainability Data lineage should be highly scalable to handle growing demands as we add more transformations or processes to the data pipeline. It should also not involve much manual maintenance and minimize the cost and effort of maintaining a data lineage system.

4.2 Data Architecture Design

Figure 4.1 illustrates the company's current ETL pipeline architecture. After sourcing the data, data engineers at *Dashmote* save it as a parquet file in an AWS S3[1] bucket. They use Apache Airflow to manipulate the data pipeline, including pre-processing, machine learning modeling, and post-processing. The pipeline collects components together using a DAG consisting of operators. However, engineers have to create GitHub repositories for each component and code the operators in Airflow, which takes time.

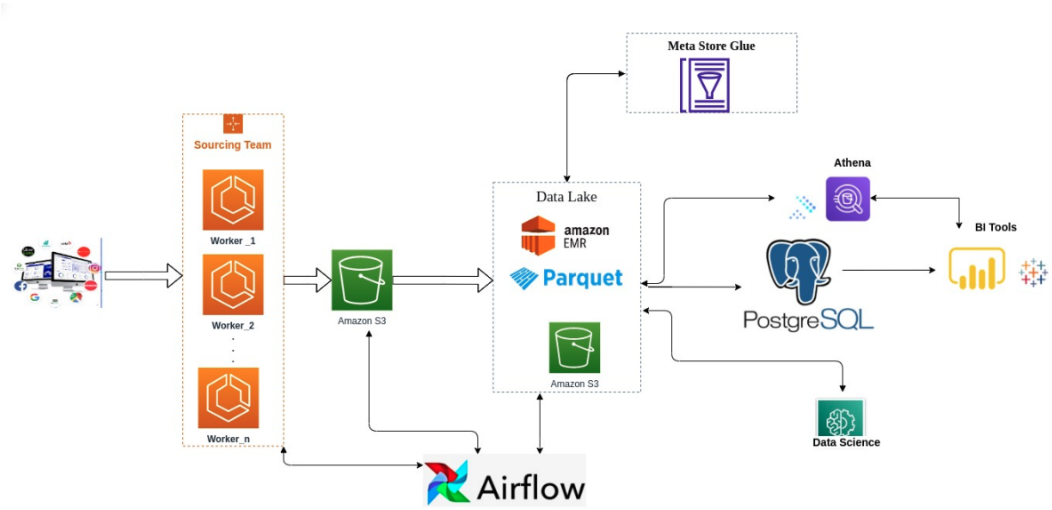

Figure 4.1: The current ETL pipeline architecture at Dashmote

[1] https://aws.amazon.com/s3/

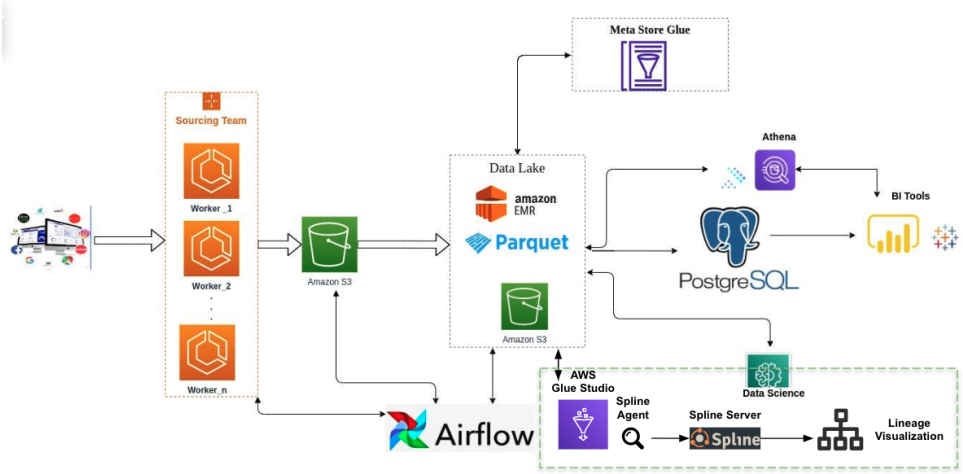

Figure 4.2: The new technical architecture design at Dashmote

To effectively deploy the ETL pipeline and solve the problems stated in Section 2.4, we decided to use AWS Glue Studio to create the pipeline and utilize Spline to generate lineage information. We apply Glue Studio to some essential functions based on the existing solution. The new technical architecture design is shown in Figure 4.2, and changes are in the dotted box in the lower right corner.

4.3 Apache Airflow vs. AWS Glue Studio

We decided to use AWS Glue Studio for the post-processing step instead of Apache Airflow because there is no complex transformation in post-processing, and Glue Studio can help us create an ETL pipeline relatively quickly. Apache Airflow is not a drag-and-drop platform, so we have to code our own DAGs. The DAG only describes the external dependencies of each task. Every task can only be run when the upstream task succeeds. Setting up an Apache Airflow architecture for production is complex process: There are three main components in post-processing, and the CC_ASEAN (a project name at *Dashmote*) post-processing pipeline is shown in figure 4.3, involving the following steps:

- Subset-extraction: extracting relevant data from the country data pool for every client, including platform scoping, brand scoping, and geolocation scoping.
- Flags-generator: generating calculated fields on the outlet, portfolio, or matching tables based on other fields of tables.

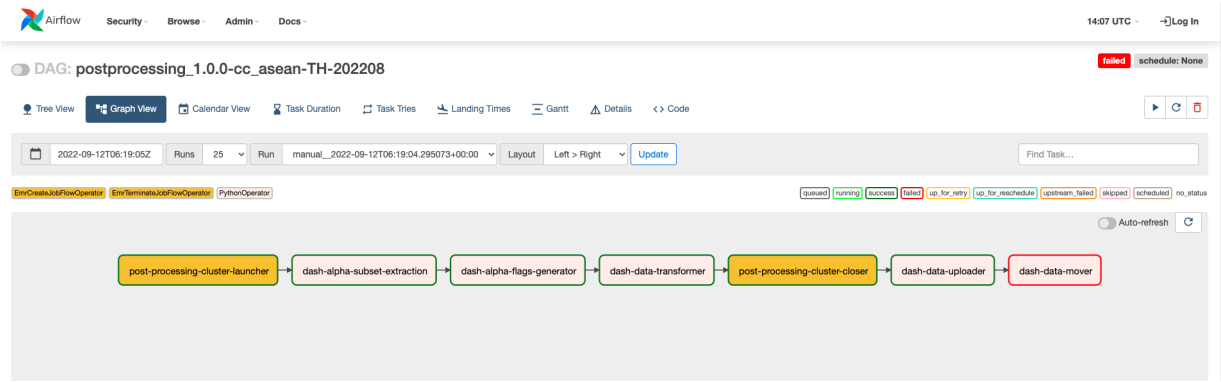

Figure 4.3: The CC_ASEAN post-processing pipeline at Dashmote

- Data-transformer: transforming the data from the company's data model to the client data model by renaming columns, changing data types, modifying the format, etc.

If we take the example of the *data-transformer*, it is only used for processing some data transformations, and it is not very necessary to run it in Airflow. Since we need to create a repository from the beginning and then define operators in Airflow. When running the pipeline, Airflow also needs to launch an EMR cluster to allow tasks to run inside, which takes time and resources. Once we encounter errors in this step, or we need to update the code for new requirements, there are many processes that we need to do:

- Push and merge the code to the repository
- Git Action build a new Docker image for the repository
- The new Docker image is uploaded to AWS ECR[1] (Amazon Elastic Container Registry), which is used for storing and managing the Docker image.
- Update the new Docker image version in Airflow repository
- Run the data pipeline in Airflow with the latest version

The process looks complicated and takes longer than creating an ETL job using AWS Glue Studio, which provides "drag and drop" features for users to create, run, and monitor ETL jobs easily.

[1] `https://aws.amazon.com/ecr/`

Port Range	Protocol	Security Groups
22	TCP	SSH for Admin
8080	TCP	Spline Rest API
9090	TCP	Spline UI

Table 4.1: AWS EC2 Security Settings

4.4 Spline: Data Lineage Tool

In this section, we describe how to set up Spline and integrate Spline with AWS Glue Studio.

4.4.1 Set up Spline Server on AWS EC2

We use docker-compose to containerize and deploy the Spline on EC2[1] (Elastic Compute Cloud), which is a web service that provides resizable computing capability. Docker[2] and Docker Compose[3] need to be installed in advance.

We first need to create and launch an EC2 instance with 2 CPUs, and the security settings are in Table 4.1. The rest of Spline settings are in this GitHub repository[4], including opening the SSH client, logging into the EC2 instance, installing and starting the Docker service, installing Docker-compose. The Spline version that we are using is 0.7.7.

4.4.2 Create and Run ETL jobs in AWS Glue Studio

After setting up the Spline Server, we need to create and run an ETL job in AWS Glue before implementing the data lineage. We can easily use the "drag and drop" feature to execute the job. The Glue studio has three main parts: Source, Transform, and Target.

For sourcing and saving data, users can select AWS Glue Data Catalog, Amazon S3, MySQL, PostgreSQL, etc. In our case, we store and fetch the data in AWS S3. We save data in parquet format because parquet is a column-oriented data store. It saves storage space and speeds up aggregation, and analytical queries (24), compared with some row-based data stores, such as CSV, JSON, etc.

In the transformation step, some essential functions such as Apply Mapping, Select Fields, Drop Fields, etc. We can also customize our SQL code in Glue Studio. The

[1] `https://aws.amazon.com/ec2/`
[2] `https://www.docker.com/`
[3] `https://docs.docker.com/compose/`
[4] `https://github.com/AbsaOSS/spline-getting-started/tree/main/spline-on-AWS-demo-setup`

Transform part is easy to use, such as the Select Field function, which will display the fields contained in the database, and we need to click the box to select the fields we need. Figure 4.4 shows a Glue ETL job, which is used for filtering information and saving data into the client's S3 bucket.

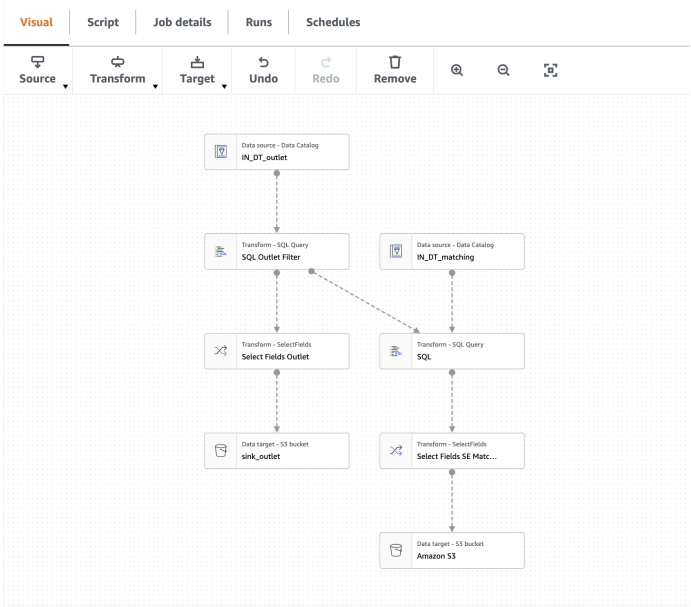

Figure 4.4: A Glue ETL job to filter information and save to an S3 bucket based on client's requirements

4.4.3 Integrate AWS Glue with Spline

The "Job details" need to be edited if we want to integrate AWS Glue with Spline. We first download the Spline Agent Spark jar from the Jar Download[1] website, which is designed to be embedded in the Spark driver. Then we save it in the AWS S3 bucket and edit the JARs path in Glue Studio.

Then the "Job parameters" in advanced properties need to be specified as the spline-related parameters. The parameters are shown in Table 4.2

We can create ETL jobs using AWS Glue Studio, and Glue can automatically generate scripts. However, some issues need to be addressed when capturing data lineage. If we capture lineage from the script provided by Glue, it will fail because Spline cannot capture RDD lineage, while Glue uses RDD under the hood and converts DataFrame to RDD internally before processing.

[1] `https://jar-download.com/artifacts/za.co.absa.spline.agent.spark`

Key	Value
--class	GlueApp
--conf	spark.spline.lineageDispatcher.http.producer.url= http://<Spline_Server_IP>:8080/producer --conf spark.spline.mode=REQUIRED --conf spark.sql.queryExecutionListeners= za.co.absa.spline.harvester.listener.SplineQueryExecutionListener

Table 4.2: Spline Server Parameters

Function	GlueContext	SparkContext
Source Data	IN_DT_outlet_node1659887173884 = glueContext.create_dynamic_frame.from_catalog(database="dash-alpha-dev-postprocessing", table_name="cc_india_data_transformer_outlet", transformation_ctx="IN_DT_outlet_node1659887173884")	IN_DT_outlet_node1659887173884 = spark.sql("SELECT * FROM 'dash-alpha-dev-postprocessing'.cc_india_data_transformer_outlet")
Save Data	AmazonS3_node1660851337315 = glueContext.write_dynamic_frame.from_options(frame=SelectFieldsSEMatching_node1660851333882, connection_type="s3", format="glueparquet", connection_options={ "path": "s3://dash-postprocessing-dev/cc_india/in_push_to_asean_matching/", "partitionKeys": ["refresh", "country"], }, format_options={"compression": "gzip"}, transformation_ctx="AmazonS3_node1660851337315",)	SelectFieldsSEMatching_node1660851333882.write\ .format('parquet')\ .mode('overwrite')\ .partitionBy('country', 'refresh')\ .option('compression', 'snappy')\ .parquet('s3://dash-postprocessing-dev/data_lineage/cc_india/in_push_to_asean_matching/')

Table 4.3: Syntax differences between GlueContext and SparkContext for the same function

The scripts generated by Glue are based on GlueContext, and Spline can capture lineage if we use Spark API directly within Glue rather than GlueContext. In that case, we first need to convert the script from GlueContext to SparkContext. Table 4.3 shows the comparison between GlueContext and SparkContext. All the logs are recorded in CloudWatch[1]

4.5 Visualization and Utilization of Data Lineage

After the Glue job has been run, a data lineage can be captured by Spline, and the lineage is shown in Figure 4.5. The input and output data information is displayed in the UI. We can also extend the lineage to check more detailed information.

Figure 4.6 provides the complete execution details of the ETL job. The transformation of the whole process is shown in the graph. By clicking on each component, we can intuitively see which transformations have been applied.

When selecting a specific data column, we can also track the data in a particular field, shown in Figure 4.7. It can help us with data source checking. If BI analysts find that the data is incorrect while doing quality assurance, engineers can use data lineage to track the source of the data and how data was transformed, which helps with debugging.

[1] `https://aws.amazon.com/cloudwatch/`

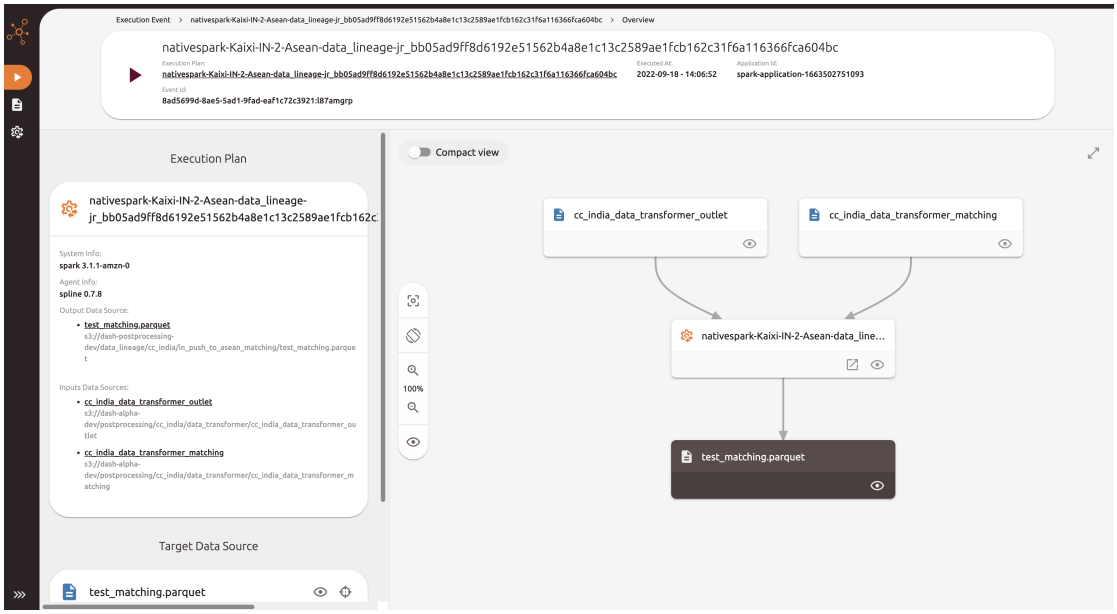

Figure 4.5: Data lineage in the spline UI. From this interface, we can clearly see where the input and output data are, and the execution time

Furthermore, if engineers want to make some changes to middleware components, they can see which components or data will be affected by viewing the data lineage information.

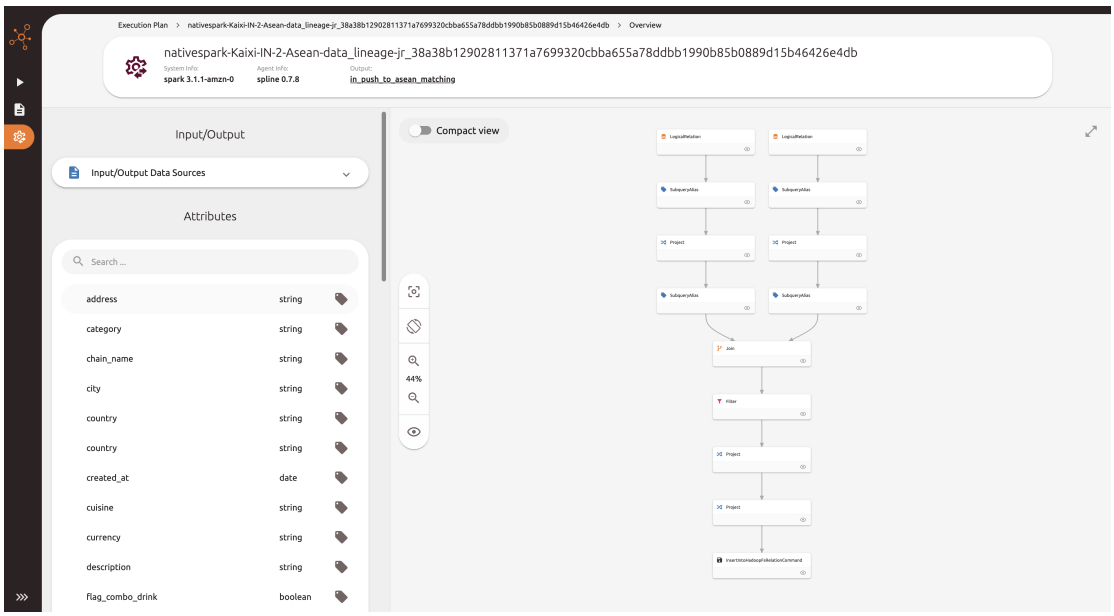

Figure 4.6: ETL execution details in Spline UI, from which we can see what transformations were processed during this job

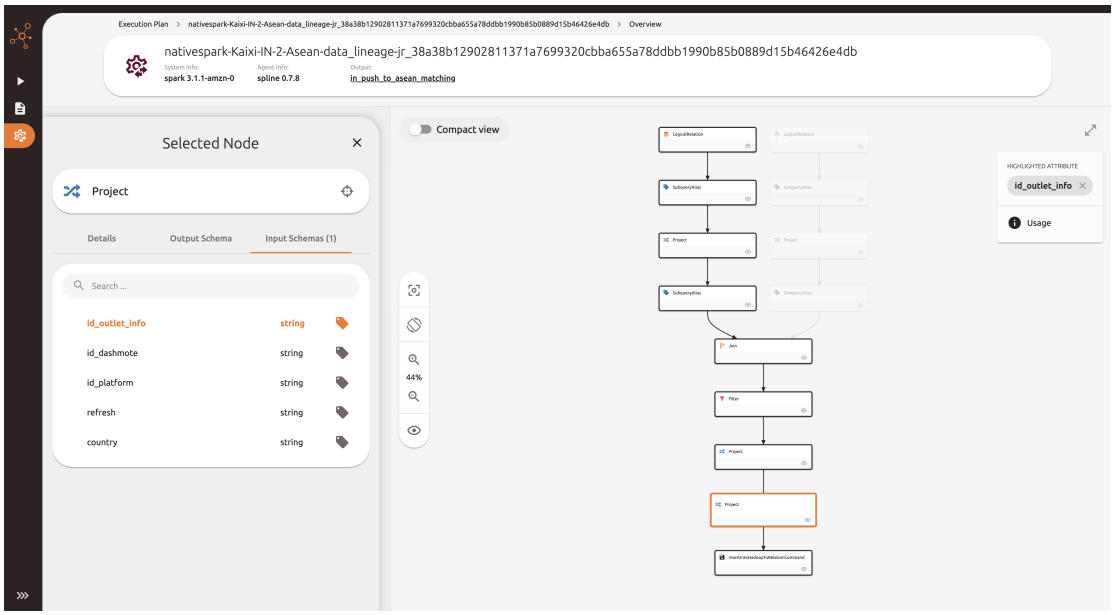

Figure 4.7: By clicking on the box, we can check where the data came from and what columns of data are included in this step

5

Discussion

In this chapter, we discuss the impact of the proposed data lineage on the ETL pipeline at *Dashmote*. in Section 5.2, we compare certain feature of the ETL pipeline before and after adding the data lineage to the processing pipeline. Finally, we analyze the project limitation and discuss future work.

5.1 Evaluation

This section evaluates the project in terms of development cost, debugging time, and business value.

Development cost Based on the creation date of the components in GitHub, it takes about 4 to 5 months to develop the three post-processing components. However, we only need two hours to complete post-processing ETL jobs using AWS Glue, which saves development time. Additionally, Glue allows us to modify transformations in ETL jobs quickly. However, if we run post-processing pipelines in Airflow, we need to update the code in the GitHub repository first and create the DAG in Airflow, which takes longer.

Debugging time Considering the time-saving perspective, the plan with Glue and Spline can save engineers time for debugging. Previously, we used Airflow to run pipelines. If some errors occur, we have to check the specific component and debug the code locally. Since the data we processed is massive (e.g., 8 GB), based on experience, usually takes us more than 1 hour to detect the error data and sometimes 30 minutes to read and load data. However, Glue and Spline can help us check the potential bug within a few minutes.

Business value Before the data lineage solution, product managers could only view the results of all transformations without a clear view of the entire data processing. With the

help of Glue and Spline, people who do not have technical backgrounds can quickly check how data is transformed between different steps.

5.2 Reflection

Before we started to work on this project, *Dashmote* used Apache Airflow to run data pipelines. Our project main contribution was to introduce a data lineage subsystem to improve the data processing pipeline. By implementing data lineage, there are several benefits to engineers, new employees, and customers. Some summaries are provided below to compare the changes between previous and current solutions.

5.2.1 Comparison

The comparison between the ETL pipeline processing at *Dashmote* before and after we introduced the data lineage subsystem is shown in Table 5.1.

ELT pipeline without data lineage	ETL pipeline with data lineage
- Not clear about how data is processed Engineers can use Spark UI to investigate issues, but it is difficult to understand the data lineage information. **- Difficult to understand how data is used** New engineers want to check the input and output of a specific component, and they have to read code repositories to understand the logic behind the code, which is time-consuming and low efficient. **- Limited lineage information:** Apache Airflow can display what the pipeline looks like, but engineers cannot check more detailed lineage information from it.	**- Clear structure of data processing** Data lineage can provide detailed information on how data is processed in the data pipeline. It can help customers or new engineers to have a clear understanding of data processing.
- Debugging in time-consuming If an error occurs in the data pipelines, engineers have to spend time tracking where the incorrect data is coming from, which takes a long time.	**- Debugging supports:** Instead of spending hours debugging code, engineers can use data lineage to check where the data source is quickly.

- No automatic updated structure graph If there are any changes in the component, a flow chart cannot be updated promptly based on the changes, and we have to modify it manually, which is inefficient.	**- Consistency and Efficiency:** For some fundamental data transformations, using Glue Studio can create an ETL job more efficient comparing Airflow. When some changes are applied in the data pipeline, data lineage can be automatically updated, which can keep the consistency of the lineage information updated efficiently.

Table 5.1: The comparison between the ETL pipeline processing at Dashmote before and after we introduced the data lineage subsystem.

5.2.2 Limitations

The maintainability of this project needs to be improved. After running the job in Glue Studio, we cannot get lineage information directly in Spline because Glue Studio uses RDDs internally when processing ETL jobs. However, Spline cannot capture RDD lineage. Whenever we want to create lineage information, we first need to modify the GlueContext code using Spark syntax, and then Spline can successfully capture the lineage. Nevertheless, this process is not as efficient as we expected.

5.3 Future Work

Although the data lineage has been implemented using AWS Glue Studio and Spline to apply it to the *Dashmote* production line, and other lineage tools can also be considered. In this section, we will review some potential extension of the proposed data lineage solution. One possible extension to this work is to:

Data Lineage of Airflow In Airflow 2.2.5, Airflow Lineage[1] supports features to track data sources and data processing steps. Airflow Lineage integrate well with OpenLineage[2] and supports databases like BigQuery, SnowFlakes, etc (25). The company uses the 2.1.0 version, so we did not choose Airflow Lineage as the current solution. Since we are already using Airflow for data processing, it will be straightforward for us to implement data lineage based on Airflow.

Data Lineage of AWS EMR To get lineage information from Glue Studio using Spline, we have to convert GlueContext to Spark, which is time-consuming and cannot be applied

[1] `https://airflow.apache.org/docs/apache-airflow/stable/lineage.html`
[2] `https://openlineage.io/`

in an industrial environment. At the same time, we are also running some PySpark applications on AWS EMR and can consider capturing lineage through Spline.

6

Conclusion

In this paper, we describe the design and implement a data lineage system that helps to debug and clearly describe the various steps of the data processing lifecycle. Before our work, *Dashmote* used to apply Apache Airflow to all the company data pipelines. The data processing pipeline was complex and took a lot of time to design, because it required that data engineers code all the pipelines in Airflow. When we introduce Glue Studio, the design process has been significantly simplified as it offers "drag and drop" functionality that allows us to build ETL jobs quickly. Additionally, Airflow can only provide limited data lineage information, which is not enough for engineers to track data sources or have a clear overview of data processing. To solve this limitation, we introduced Spline to capture data lineage information from Glue Studio. After running the job in Glue Studio, Spline can automatically generate a lineage graph, which allows us to check what transformations were made, where the data came from, etc.

This project aims to add a data lineage subsystem to the current *Dashmote* data processing pipelines using state-of-the-art data processing technologies. We answer the following research questions:

- ***Research Question 1:*** *How to build a versatile data lineage subsystem for a highly complex, fast-changing data pipeline in an industrial setting?*

  In this project, we use AWS Glue Studio to deploy ETL pipelines efficiently. Moreover, we use Spline as a data lineage tool because it has three characteristics: lightweight, unobtrusive, and easy to use. Spline can capture and store lineage information from Apache Spark and display lineage visualizations in the Spline UI. Providing data lineage reduces our development cost and debugging time and brings more business

value. However, Spline cannot capture RDD lineage, and Glue Studio uses RDDs internally when processing ETL pipelines. Therefore, we need to convert the code from GlueContext to SparkConcent, which is one of the limitations of capturing lineage from Glue jobs using Spline.

- ***Research Question 2:*** *What techniques and tools can we use to implement the data lineage?*

  There are four main lineage techniques: pattern-based lineage, parsing-based lineage, data tagging, and self-contained. In this project, we implemented a parsing-based lineage technique to parse the logic behind the code, as it is more accurate comparing other techniques. There are several data lineage tools on the market. In our project, we use Spline to capture lineage because it is lightweight, unobtrusive, and easy to use. However, Spline cannot efficiently capture the lineage information from Glue Studio because we have to modify the code to Spark syntax. Therefore, we consider using Spline to capture lineage from AWS EMR or using Airflow Lineage with our current Airflow pipelines, which are mentioned in Section 5.3.

- ***Research Question 3:*** *What are the challenges when developing data lineage?*

  We summarize the challenges from two perspectives: development and management. According to the development part, there is no detailed documentation on deploying Spline in AWS Glue Studio. During the implementation, we spent much time browsing the internet as we could not find complete and detailed documentation. We found some tutorials on implementing lineage with Spline, but some details are missing. We kept getting the error after running a Glue job, and we did not know how to fix it until we found some discussion in a GitHub issue[1] where participants answered questions about RDD lineage support and gave us the Glue script on how to fix it some thoughts. Second, in the management part, we have to modify the script Glue automatically generates each time to create data lineage in Spline. The integration between Glue Studio and Spline is not efficient and brings us more workload.

To conclude, our project creates a data lineage subsystem to optimize existing data pipelines. We use Glue Studio instead of Apache Airflow to improve the efficiency of the ETL creation process. At the same time, we solved the several problems, including the unclear structure of data processing, the long and complex debugging process, and untimely updating of lineage information. It also summarizes our challenges and the techniques and

[1] `https://github.com/AbsaOSS/spline-spark-agent/issues/33`

tools we used in this project. More research is still needed to improve the efficiency of generating data lineage: using Airflow Lineage to build data lineage directly or using Spline to get lineage information from AWS EMR.

References

[1] **Spline: Data Lineage Tracking And Visualization Solution**. https://absaoss.github.io/spline/start-spline-server. [Online; accessed 2022-09-20]. iii, 13

[2] Peter Buneman, Sanjeev Khanna, and Wang-chiew Tan. **Why and Where: A Characterization of Data Provenance**. 07 2004. 1

[3] Kyle McNamara. **What is Data Lineage and Data Provenance? Quick Overview**, Sep 2021. 1

[4] Matteo Interlandi, Ari Ekmekji, Kshitij Shah, Muhammad Ali Gulzar, Sai Deep Tetali, Miryung Kim, Todd Millstein, and Tyson Condie. **Adding Data Provenance Support to Apache Spark**. *The VLDB Journal*, **27**(5):595–615, oct 2018. 1

[5] Beatriz Pérez, Julio Rubio, and Carlos Sáenz-Adán. **A Systematic Review of Provenance Systems**. *Knowl. Inf. Syst.*, **57**(3):495–543, dec 2018. 1

[6] Eugene Wu, Samuel Madden, and Michael Stonebraker. **SubZero: A fine-grained lineage system for scientific databases**. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 865–876, 2013. 1

[7] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. **A survey of data provenance in e-science**. *ACM Sigmod Record*, **34**(3):31–36, 2005. 3

[8] H. V. Jagadish and Frank Olken. **Database Management for Life Sciences Research**. *SIGMOD Rec.*, **33**(2):15–20, jun 2004. 4

[9] Bob Mann. **Some data derivation and provenance issues in astronomy**. *Workshop on Data Derivation and Provenance, Chicago*, 2002. 4

[10] Roselie Bandibas Webjørnsen. *Discovering data lineage in data warehouse: methods and techniques for tracing the origins of data in data-warehouse.* Master's thesis, 2005. 4

[11] Khoa Nguyen, Krithivasan Balasubramaniyan, and Rahul Shaurya. **Build data lineage for data lakes using AWS Glue, Amazon Neptune, and Spline**. https://aws.amazon.com/blogs/big-data/build-data-lineage-for-data-lakes-using-aws-glue-amazon-neptune-and-spline/, apr 1 2022. [Online; accessed 2022-09-20]. 5

[12] Xavier de Boisredon. **What is Data Lineage?** https://www.castordoc.com/blog/what-is-data-lineage. [Online; accessed 2022-09-20]. 5

[13] Daniel Crawl, Jianwu Wang, and Ilkay Altintas. **Provenance for mapreduce-based data-intensive workflows**. In *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*, pages 21–30, 2011. 5

[14] Alfredo Cuzzocrea. **Big Data Provenance: State-Of-The-Art Analysis and Emerging Research Challenges**. In *EDBT/ICDT Workshops*, 2016. 5, 6

[15] Christopher Ré and Dan Suciu. **Approximate lineage for probabilistic databases**. *Proceedings of the VLDB Endowment*, **1**(1):797–808, 2008. 6

[16] Umber Sheikh, Abid Khan, Bilal Ahmed, Abdul Waheed, and Abdul Hameed. **Provenance inference techniques: Taxonomy, comparative analysis and design challenges**. *Journal of Network and Computer Applications*, **110**:11–26, 2018. 6

[17] Mohammad Rezwanul Huq, Peter M. G. Apers, and Andreas Wombacher. **An Inference-Based Framework to Manage Data Provenance in Geoscience Applications**. *IEEE Transactions on Geoscience and Remote Sensing*, **51**(11):5113–5130, 2013. 6

[18] You-Wei Cheah and Beth Plale. **Provenance Analysis: Towards Quality Provenance**. 10 2012. 6

[19] Mallikarjuna$_g$. **Webui − − − sparkapplication′swebconsolespark**. *https* : *//mallikarjuna_g.gitbooks.io/spark/content/spark* − *webui.html*. [*Online*; *accessed*2022 − 09 − 25]. 7

[20] **What is Data Lineage**. https://www.imperva.com/learn/data-security/data-lineage/. [Online; accessed 2022-09-04]. 10, 11

[21] Navdeep Singh Gill. **Data Lineage tools and its Best Practice**. https://www.elixirdata.com/blog/data-lineage-complete-guide, apr 21 2022. [Online; accessed 2022-09-05]. 11

[22] Volker Liermann and Claus Stegmann. *The Digital Journey of Banking and Insurance, Volume III: Data storage, data processing and data analysis*. Palgrave Macmillan, oct 28 2021. [Online; accessed 2022-09-08]. 11

[23] Jan Scherbaum, Marek Novotny, and Oleksandr Vayda. **Spline: Spark Lineage, not only for the Banking Industry**. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 495–498, 2018. 13

[24] **What is apache parquet**. https://www.databricks.com/glossary/what-is-parquet, oct 30 2018. [Online; accessed 2022-09-24]. 19

[25] Liangjun Jiang. **Design a end-to-end data lineage solution - Liangjun Jiang**. *Medium*, apr 21 2022. [Online; accessed 2022-09-13]. 26

Appendix

A Glue Studio jobs

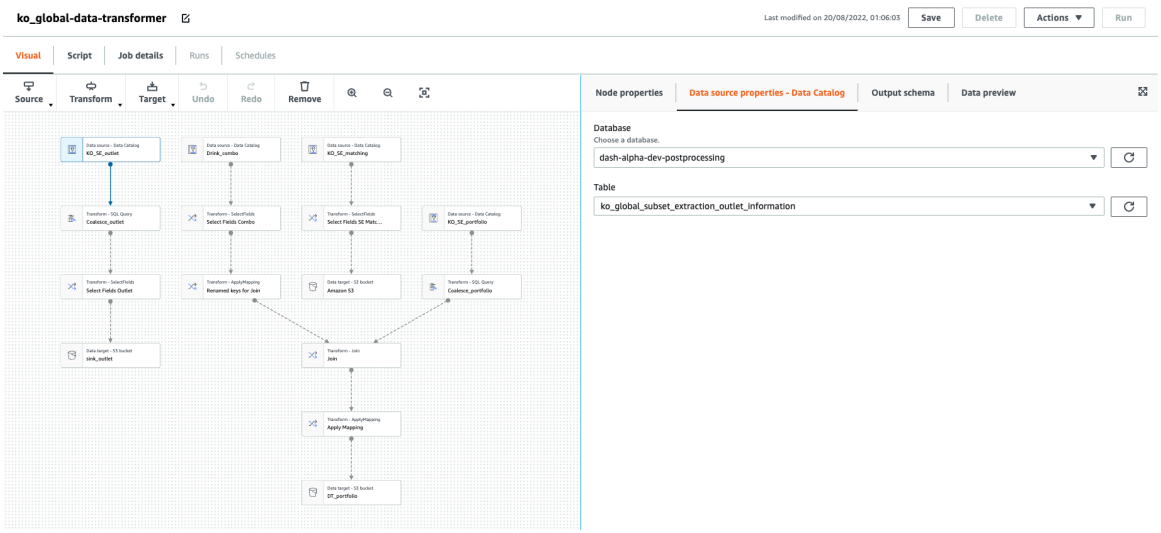

Figure 6.1: Glue Studio jobs interface with ETL pipelines

B CloudWatch logs

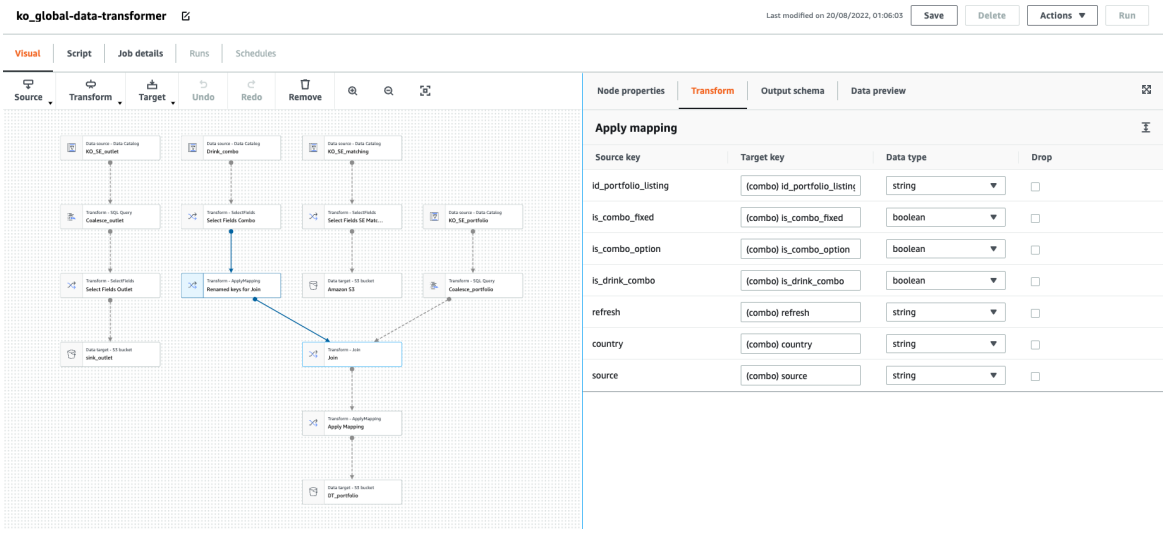

Figure 6.2: Glue Studio jobs interface with transformation details

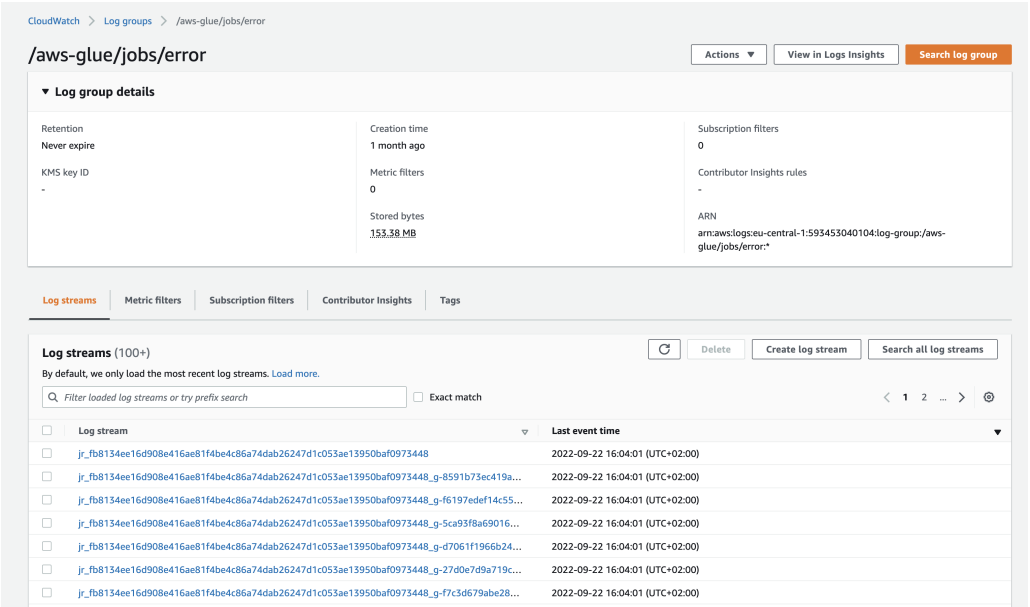

Figure 6.3: CloudWatch's error interface