ECOLE
POLYTECHNIQUE
DE BRUXELLES

INFO H419: Data Warehouses

TPC-DI Benchmark of Amazon Redshift: ETL using AWS Step Functions

Authored by:

Rana İşlek

Simon Coessens

Berat Furkan Koçak

David García Morillo

Professor:

Esteban Zimányi

2023

Contents

1 Introduction

1.1 Overview of the Project

This project focuses on implementing the *TPC-DI benchmark* to test the efficiency of Data Integration workflows on *AWS (Amazon Web Services)*, specifically using *AWS Lambda*. *TPC-DI* , which stands for Transaction Processing Performance Council-Data Integration, is the first industry benchmark designed for evaluating the performance of systems in the Data Integration processes. By employing AWS Lambda, a serverless computing platform, we aim to explore the scalability, efficiency, and cost-effectiveness of serverless computing in the context of Data Integration tasks.

1.2 Objective

Our primary goal is to showcase the feasibility and performance of utilizing AWS Lambda for *TPC-DI* benchmarking. This involves evaluating scalability, efficiency, and cost-effectiveness specific to serverless computing in Data Integration. The project also tries to give insights about the advantages and challenges associated with using AWS Lambda for Data Integration workloads.

1.3 Tools

We will now present the tools to be used in this project.

1.3.1 AWS Lambda

AWS Lambda is a *FaaS (Function as a Service)* compute service that lets you run code without provisioning or managing servers. In other words, it uplifts all of the infrastructure overhead, and lets the user focus on the core tasks to be done. It also comes with the benefit of only having to pay for running time. That is the time *AWS Lambda* is actually running your code, instead of having to provision an entire virtual machine will is always running.

Key Features

Pricing: As already mentioned, it's pay-as-you-go pricing model and it's free one millions requests per month allow us to make use this service with a minimal cost.

Straightforward: It is a very simple service, you put code on it, and it's run when called, no infrastructure-management, log-management out-of-the-box, scalability, etc.

Role in *TPC-DI* Benchmarking

Lambda will be the compute service, in other words, where the code to extract, transform, and load the data will be executed.

1.3.2 AWS Step Functions

AWS Step Functions is the orchestrator for this *TPC-DI* project, and it will provide the perfect abstraction layer for architecting the different steps of this benchmark.

Key Features

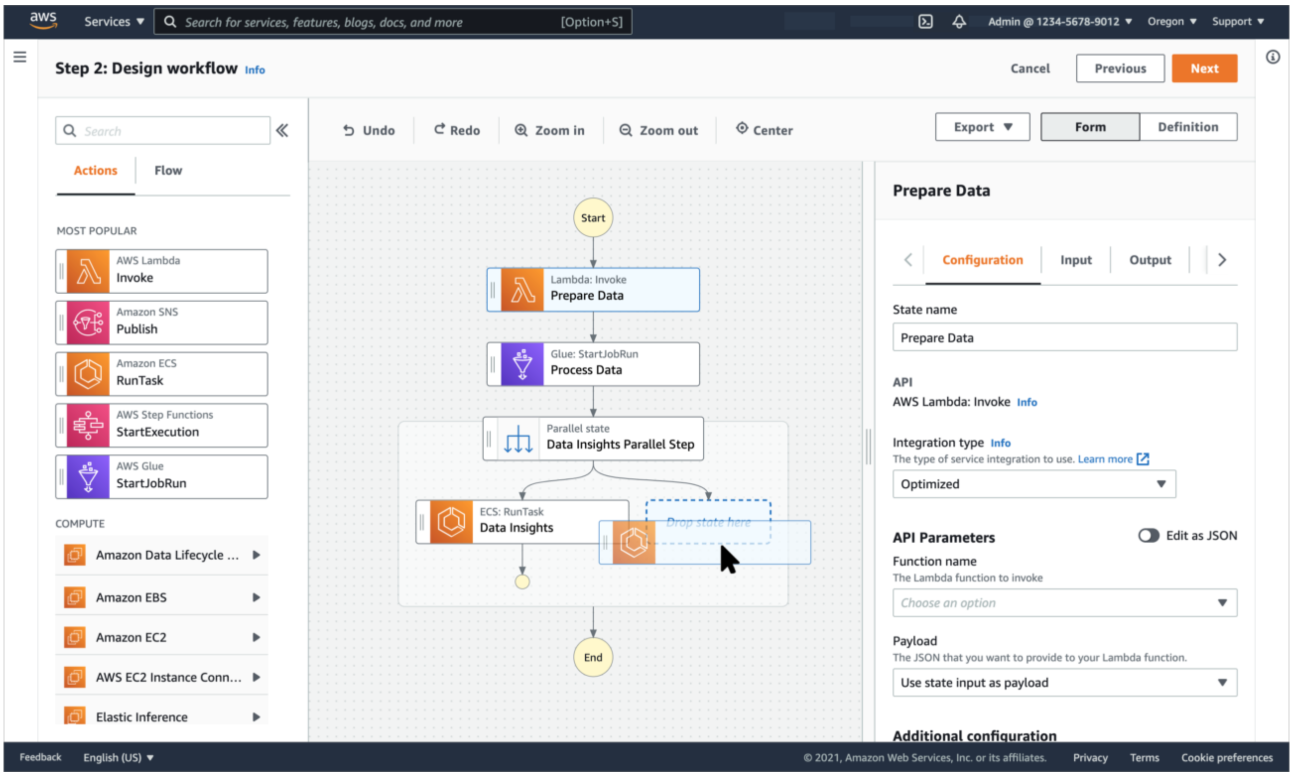
Simplicity: The main benefit of this service is how simple it is to start working with it. It provides with a simple and intuitive UI where you can just drag-and-drop the different steps that will conform your workflow. This UI is shown in 1.1

Pricing: It provides with a generous Free Tier where the first 4000 state transitions are for free. After consuming the Free Tier, every 1000 state transitions will cost $0.025.

This meant being able to use AWS Step Functions for free, as we didn't get to consume the Free Tier.

Role in *TPC-DI* Benchmarking

This service will be used to architect the different steps of the benchmark conforming a workflow, connecting the different *AWS Lambdas*, since some tables have some dependencies between them.



State name: Prepare Data

API: AWS Lambda: Invoke

Integration type: Optimized (The type of service integration to use.)

Function name: Choose an option (The Lambda function to invoke)

Payload: Use state input as payload (The JSON that you want to provide to your Lambda function.)

Figure 1.1: AWS Step Functions

1.3.3 Amazon Redshift

Amazon Redshift is a key component in our *TPC-DI benchmarking* project, serving as a fully managed Data Warehouse for efficiently storing and managing large volumes of data. Its integration ensures a robust and scalable foundation for executing the *TPC-DI* benchmark.

Key Features and Functions

Data Warehousing Capabilities: *Amazon Redshift* provides a potent solution for storing, organizing, and analyzing vast data sets through its columnar storage architecture and parallel processing capabilities.

Integration with AWS Lambda: Seamlessly integrated with *AWS Lambda functions*, Amazon Redshift facilitates the orchestration of Data Integration tasks, including extraction, transformation, and loading processes, leveraging the serverless computing paradigm.

Scalability: Amazon Redshift's ability to scale in terms of both storage and compute resources is crucial for handling the diverse workloads inherent in *TPC-DI* benchmarking, ensuring adaptability to different Data Integration scenarios.

Role in *TPC-DI* Benchmarking

Data Storage: Serving as the primary data storage solution for *TPC-DI* benchmarking, Amazon Redshift houses datasets, enabling efficient data retrieval and analysis during *TPC-DI* task execution.

Performance Evaluation: Amazon Redshift's performance is evaluated in the *TPC-DI* benchmark, monitoring metrics such as query execution times, resource utilization, and overall system throughput to assess its effectiveness in handling Data Integration workloads.

Cost Considerations: As an integral part of the AWS ecosystem, the cost-effectiveness of Amazon Redshift is a crucial aspect of the evaluation. We analyze factors such as storage costs, query execution costs, and overall operational expenses associated with utilizing Amazon Redshift for *TPC-DI* benchmarking.

2 *TPC-DI* Benchmark

2.1 *TPC-DI* Introduction

As we also mentioned before, *TPC-DI* is **the first industry benchmark** specifically designed to measure the performance of Data Integration systems. It defines a set of standardized tasks and metrics to assess the efficiency and scalability of platforms in handling Data Integration workloads. The benchmark includes various aspects of Data Integration, including data extraction, transformation, and loading (ETL) processes.

2.2 Benchmark Goals

The *TPC-DI* benchmark aims to provide a comprehensive and standardized framework for evaluating the performance of Data Integration systems. Focusing on real-world scenarios and challenges encountered in Data Integration workflows, it offers a basis for comparing different platforms and configurations, assessing data processing capabilities, scalability, and resource utilization during Data Integration tasks.

2.3 Benchmark Components

The TPC-DI benchmark consists of several essential components, each addressing specific aspects of data integration. These components are designed to evaluate the efficiency and performance of systems in handling data integration tasks. Here's an explanation of the key components:

1. **Data Extraction**

   Data extraction is the process of retrieving information from source systems or databases.

In the context of TPC-DI, this phase involves extracting data sets that will be used in subsequent integration tasks.

The efficiency of data extraction influences the overall speed and reliability of the data integration process. It assesses how quickly and accurately the benchmarked system can retrieve data from diverse sources.

2. **Transformation**

   Data transformation involves converting and structuring the extracted data into a format suitable for the target system. It may include cleaning, filtering, and reformatting data to align with the requirements of the integration task.

   The transformation phase evaluates the system's ability to manipulate and process data effectively. It assesses how well the system can handle diverse data formats and structures during the integration process.

3. **Loading Phases**

   Loading phases involve the insertion of transformed data into the target data storage or warehouse. This could include databases, data warehouses, or other storage solutions depending on the system architecture.

   The loading phase evaluates the system's capability to efficiently load processed data into the designated storage. It assesses factors such as speed, accuracy, and scalability in handling the final step of the data integration process.

   Overall Significance The combination of these components reflects the end-to-end process of data integration in the TPC-DI benchmark. The evaluation of each phase provides insights into how well a system can handle the complexities of real-world data integration scenarios, making it a comprehensive benchmark for assessing the overall performance and capabilities of data integration systems.

3 Benchmark Preparation

3.1 Data Population

We will now cover the data managed in *TPC-DI*

3.1.1 Data Definition

The data on *TPC-DI* emulates the one from a retail brokerage business. Several sources are combined, such as for example an *OLTP* database, a Financial Newswire, a CRM, etc.

At destination, the *Data Warehouse*, a snowflake schema is used, depicted in 3.1

3.1.2 Data Generation

Data is generated using the provided *DIGen* tool provided by *TPC*. This tool will be used as a *CLI* (Command Line Interface) application to generate the data given a scale factor.

3.1.3 Data Loading

To load the data, we will use a *Redshift* client, to connect and insert the data already extracted and transformed in *AWS Lambda*.

3.2 Data Warehouse Initialization

As per section 1.2.2.1.1 of *TPC-DI*, the initialization of the Data Warehouse is not timed, so the corresponding DDL statements will be executed previous to the test on the *Data Warehouse*.

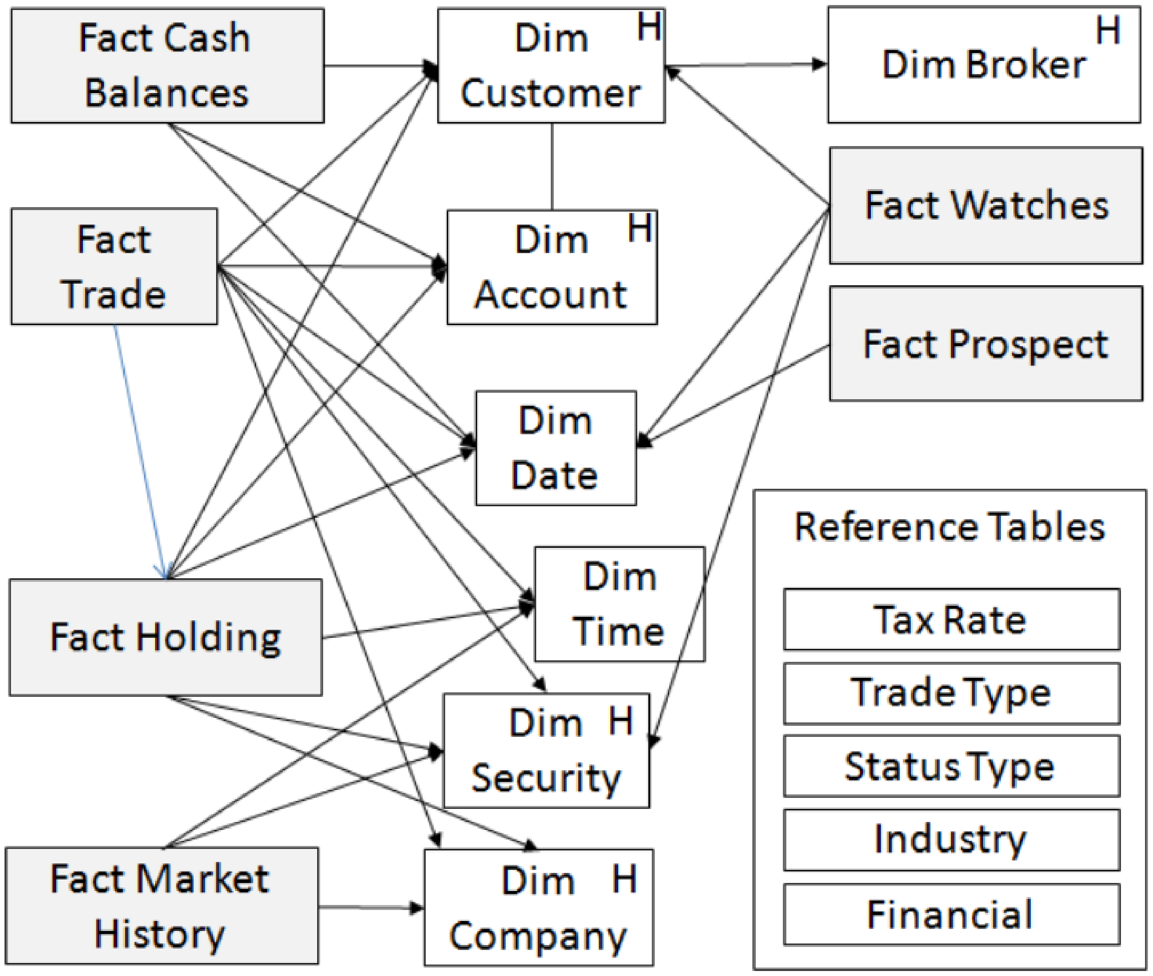


Figure 3.1: *TPC-DI* data model

4 Execution

Throughout this chapter, we will cover how *TPC-DI* is executed, step by step.

4.1 Benchmark Execution Steps

The *TPC-DI* benchmark consists of several phases. The first step, as already described before, is the Data Generation. The generation and relocating of data is not timed in the benchmark. Then in the next phase the Data Warehouse is created, also this step is not timed for the benchmark. Then we start the core phases of the *TPC-DI* benchmark starting with the Historical Load. In this phase the, initially empty, destination tables are populated with data. This phase will be timed. Then the incremental updates will be performed. These contain different transformations than in the historical load and will be performed on a smaller dataset. This phase is timed and required to complete in 30-60 minutes. In the end an automated audit on the Data Warehouse is performed where the integrity of the data in the Data Warehouse is tested. A conceptual overview of the process is given in Figure 4.1.

In the coming sections we will provide more insights into the core phases of the *TPC-DI* benchmark.

4.2 Historical Load

For the Historical load, we first created a dependency graph, based on the requirements, so that we could understand the different dependencies between the tables.

Afterwards, to avoid unnecessary costs, we moved the development to our local machines, where we developed the code in *Python* and stored the processed rows into a *Postgres* database, due to the compatibility between *Redshift* and *Postgres*.

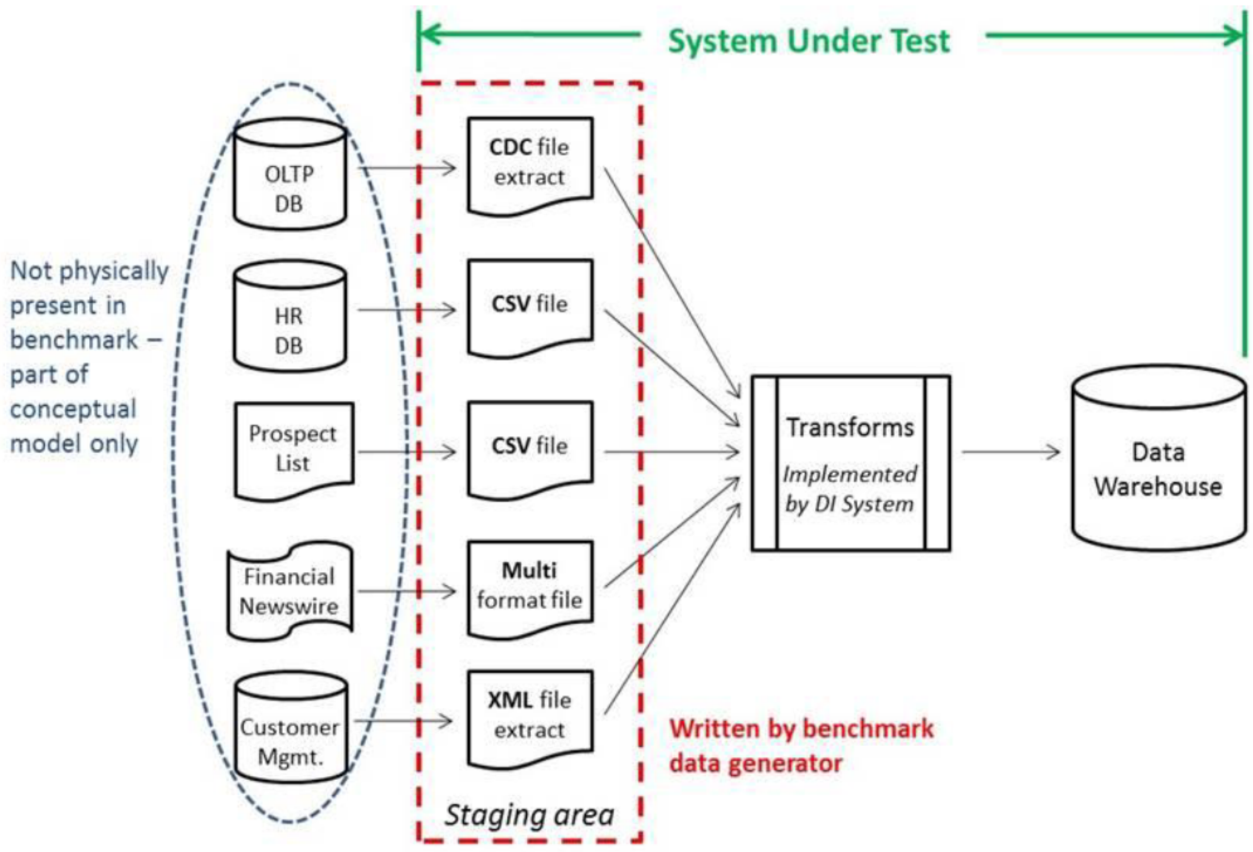


Figure 4.1: Diagram of *TPC-DI*

After successfully writing the historical load, we moved the local code to *AWS Lambda*, architecting the workflow using *AWS Step Functions*.

The final architecture for the historical load is shown in 4.2

4.3 Incremental Load

For Incremental Updates TPCDI Benchmark uses the file extracts called Changed Data Capture (CDC) extracts, that provides the changes to the tables since the last extract. These files contain additional "CDC_FLAG" and "CDC_DSN" columns at the beginning of each row.

The CDC_FLAG is a single character I, U or D that tells whether the row has been inserted, updated or deleted since the last change. For updates there is no indication as to what in the row has been changed. [TPC14]

Incremental updates represent the load of new data into an existing Data Warehouse. There are many different rates at which incremental updates may occur and the most common one is daily updates. For TPCDI Benchmark, Incremental Updates happen two times, one after the other. By requiring more than one Incremental Update, the benchmark ensures repeatability.

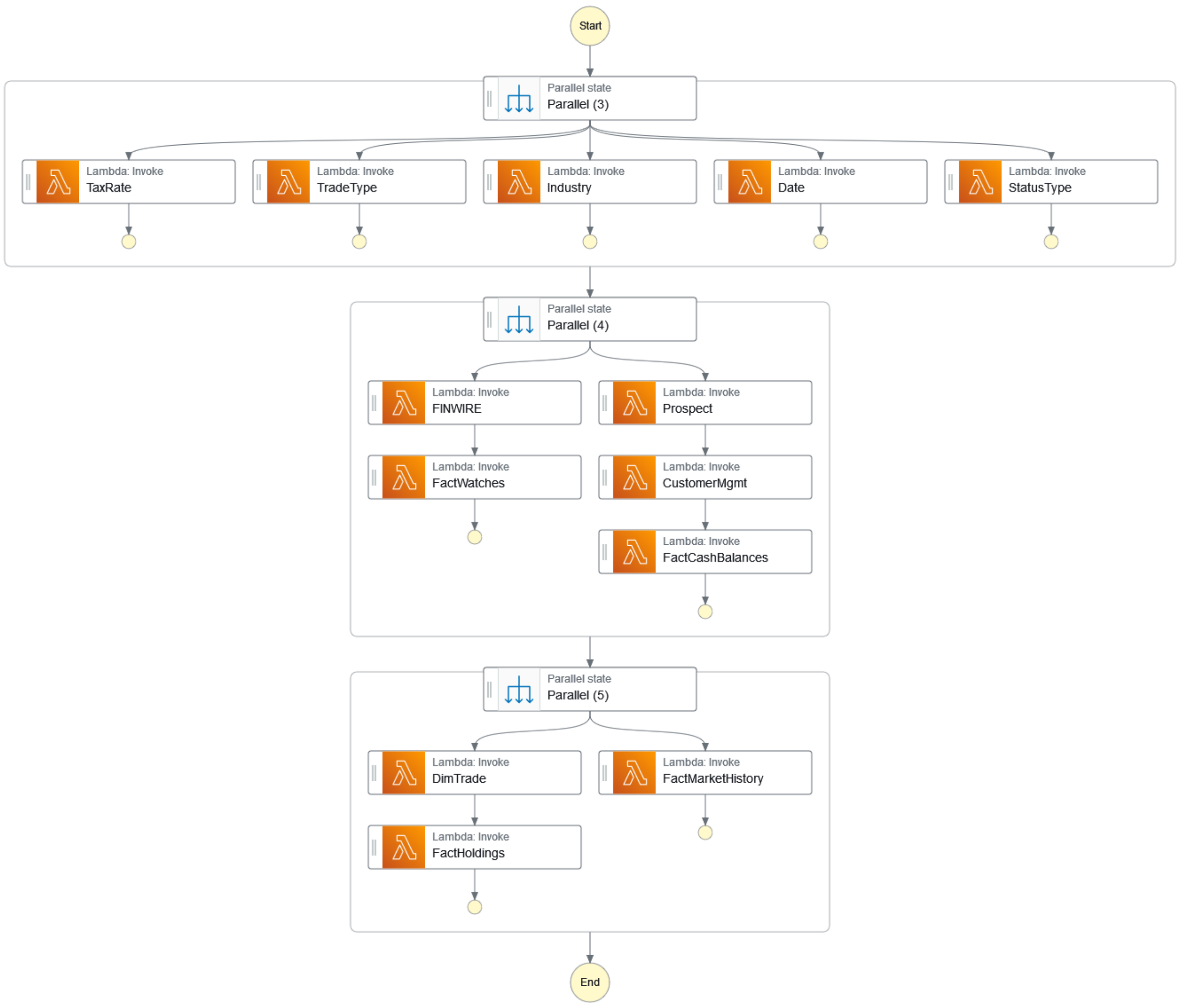


Figure 4.2: Historical load workflow

The CDC file extracts used for incremental updates are listed below:

- Account.txt
- Customer.txt
- Trade.txt
- CashTransaction.txt
- HoldingHistory.txt
- DailyMarket.txt
- WatchItem.txt
- Prospect.csv

For the transformation details, Chapter 4.6 "Transformation Details for Incremental Updates" of TPCDI specification is used.

We performed an ETL process following this structure:

- **Extract:** Reading the relevant CDC extract file from the staging area inside the folders Batch2 and Batch3 for two incremental updates respectively.
- **Transform:** There are various transformations required specific to each table. We followed the TPCDI specificiation Chapter 4.6 for all the transformation rules.
- **Load:** According to the given CDC_FLAG, either an update or an insertion should be performed on the relevant table. Moreover, for the updates in history tracking tables, propogation of these changes to the other existing tables having dependency on the updated column is implemented.

Changes are made to these tables during incremental load phase:

- DimCustomer
- DimAccount
- DimTrade
- FactCashBalances

- FactMarketHistory
- FactHolding
- FactProspect
- Fact Watches

These tables have foreign keys to the surrogate keys of different tables, which creates dependencies. Here are the dependencies presented by TPCDI Benchmark specificaton:

Table 4.1: Dependent-Source Table

Dependent Table	Dependent Column	Source Table	Source Column
DimAccount	SK_BrokerID	DimBroker	SK_BrokerID
DimAccount	SK_CustomerID	DimCustomer	SK_CustomerID
DimSecurity	SK_CompanyID	DimCompany	SK_CompanyID
DimTrade	SK_BrokerID	DimBroker	SK_BrokerID
DimTrade	SK_CreateDateID	DimDate	SK_DateID
DimTrade	SK_CreateTimeID	DimTime	SK_TimeID
DimTrade	SK_CloseDateID	DimDate	SK_DateID
DimTrade	SK_CloseTimeID	DimTime	SK_TimeID
DimTrade	SK_SecurityID	DimSecurity	SK_SecurityID
DimTrade	SK_CompanyID	DimCompany	SK_CompanyID
DimTrade	SK_CustomerID	DimCustomer	SK_CustomerID
DimTrade	SK_AccountID	DimAccount	SK_AccountID
FactCashBalances	SK_CustomerID	DimCustomer	SK_CustomerID
FactCashBalances	SK_AccountID	DimAccount	SK_AccountID
FactCashBalances	SK_DateID	DimDate	SK_DateID
FactCashBalances	SK_TimeID	DimTime	SK_TimeID
FactHoldings	SK_TradeID	DimTrade	SK_TradeID
FactHoldings	SK_CurrentTradeID	DimTrade	SK_CurrentTradeID
FactHoldings	SK_CustomerID	DimCustomer	SK_CustomerID
FactHoldings	SK_AccountID	DimAccount	SK_AccountID
FactHoldings	SK_SecurityID	DimSecurity	SK_SecurityID

Continued on the next page

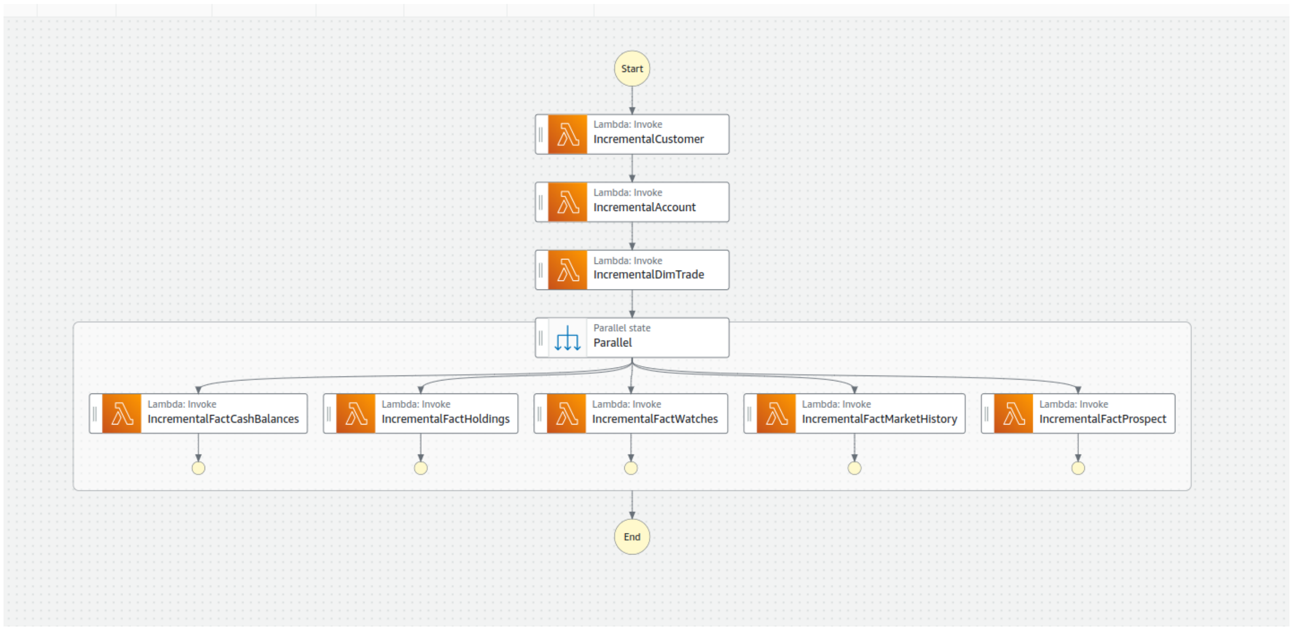


Figure 4.3: *TPC-DI* Incremental Load Process

Table 4.1 – continued from previous page

Dependent Table	Dependent Column	Source Table	Source Column
FactHoldings	SK_CompanyID	DimCompany	SK_CompanyID
FactHoldings	SK_DateID	DimDate	SK_DateID
FactHoldings	SK_TimeID	DimTime	SK_TimeID
FactMarketHistory	SK_SecurityID	DimSecurity	SK_SecurityID
FactMarketHistory	SK_CompanyID	DimCompany	SK_CompanyID
FactMarketHistory	SK_DateID	DimDate	SK_DateID
FactWatches	SK_CustomerID	DimCustomer	SK_CustomerID
FactWatches	SK_SecurityID	DimSecurity	SK_SecurityID
FactWatches	SK_DateID_DatePlaced	DimDate	SK_DateID
FactWatches	SK_DateID_DateRemoved	DimDate	SK_DateID
Prospect	SK_UpdateDateID	DimDate	SK_DateID

According to these dependencies, the order of incremental updates are of big importance. Figure 4.3 shows the execution order for each incremenatl update.

Update Propogation:

In many parts of the report, cruicial details such as "Updates to associated customer records require history-tracking updates to all of the customer's accounts where IsCurrent=1. The value of SK_CustomerID must be set to the value of DimCustomer.SK_CustomerID in the newest customer record" is given. This means according to the dependencies given above, updates in a history tracking table must be propagated to the dependent tables. This is performed by updating the surrogate key references they contain.

The difference between incremental load 1 and 2 is that the data source comes from S3 folders Batch2 and Batch3 respectively.

5 Results

We will now go over the different metrics used in *TPC-DI* to calculate the score of the benchmark.

5.1 Performance Metrics

There are two performance metrics that TPCDI Benchmark should report.

- Performance Metric
- Price/Performance Metric

As we are using a serverless Data Warehouse along with serverless etc processes on cloud, we do not have the maintenance cost for them. Hence second metric will not be a subject to this report.

Calculation of *TPC-DI* Performance Metric: The performance metric for the benchmark is a combined metric using the three throughput calculations from the timed phases, and is computed as follows:

$$TPC\_DI_{RPS} = Trunc(GeoMean(TH, Min(TI1, TI2)))$$

Where:

- TH is the throughput of the Historical Load
- T_{I1} is the throughput of Incremental Update 1
- T_{I2} is the throughput of Incremental Update 2
- GeoMean() is the geometric mean of the arguments.
- Min() is the argument with the least value.

- Trunc() is the whole number portion of the argument.

5.1.1 Historical Load Throughput

Let us define $CT_{BatchID}$ as the value in the MessageDateAndTime field of the record in the DImessages table where the MessageType field is 'PCR' and the BatchID field is equal to the BatchID for the phase.[TPC14]

In simple terms, $CT_{BatchID}$ represents when the batch started. Then the elapsed time, expressed in units of seconds, for the Historical Load is: $E_H = CT_1 - CT_0$.

Also, let R_H be the number of rows of source data for the **Historical Load**.

We can then calculate the **Throughput for the Historical Load** as follows: $T_H = R_H / E_H$

5.1.2 Incremental Load Throughput

Chapter 7.5.3 of *TPC-DI* specification describes how the second important performance metric, incremental load throughput, should be calculated.

First we need to calculate the total time passed during both incremental updates. We will refer to them as E_{I1} and E_{I2} for incremental update one and two respectively. The elapsed times, expressed in units of seconds, for the Incremental Updates are:

$$E_{I1} = CT_2 - CT_1$$

and

$$E_{I2} = CT_3 - CT_2$$

Where, The completion timestamp (CT) of each phase identified by BatchID in the format $CT_{BatchID}$

CT represents the value in the MessageDateAndTime field of the record in the DImessages table where the MessageType field is 'PCR' and the BatchID field is equal to the BatchID for the phase. [TPC14]

In simple terms, $CT_{BatchID}$ represents when the batch started.

Following SQL code is an example to receive this timestamp:

Listing 5.1: Example SQL code for CT

```sql
SELECT MessageDateAndTime
FROM DImessages
WHERE BatchID = 1 AND MessageType = 'PCR';
```

The total number of rows of Source Data in the Incremental Update data sets are R_{I1} and R_{I2}, and are reported by DIGen as the row count for Batch2 and Batch3, respectively.

The throughputs of the Incremental Updates are:

$$T_{I1} = R_{I1}/Max(E_{I1}, 1800)$$

and

$$T_{I2} = R_{I2}/Max(E_{I2}, 1800)$$

6 Challenges Faced

6.1 Problems with AWS Glue

6.1.1 What is AWS Glue?

AWS Glue is a serverless Data Integration service that makes data preparation simpler, faster, and cheaper. It enables discovering various data sources and managing them in a centralized data catalog as well as visually creating, running, and monitoring ETL pipelines to load data into data lakes. [Sera]

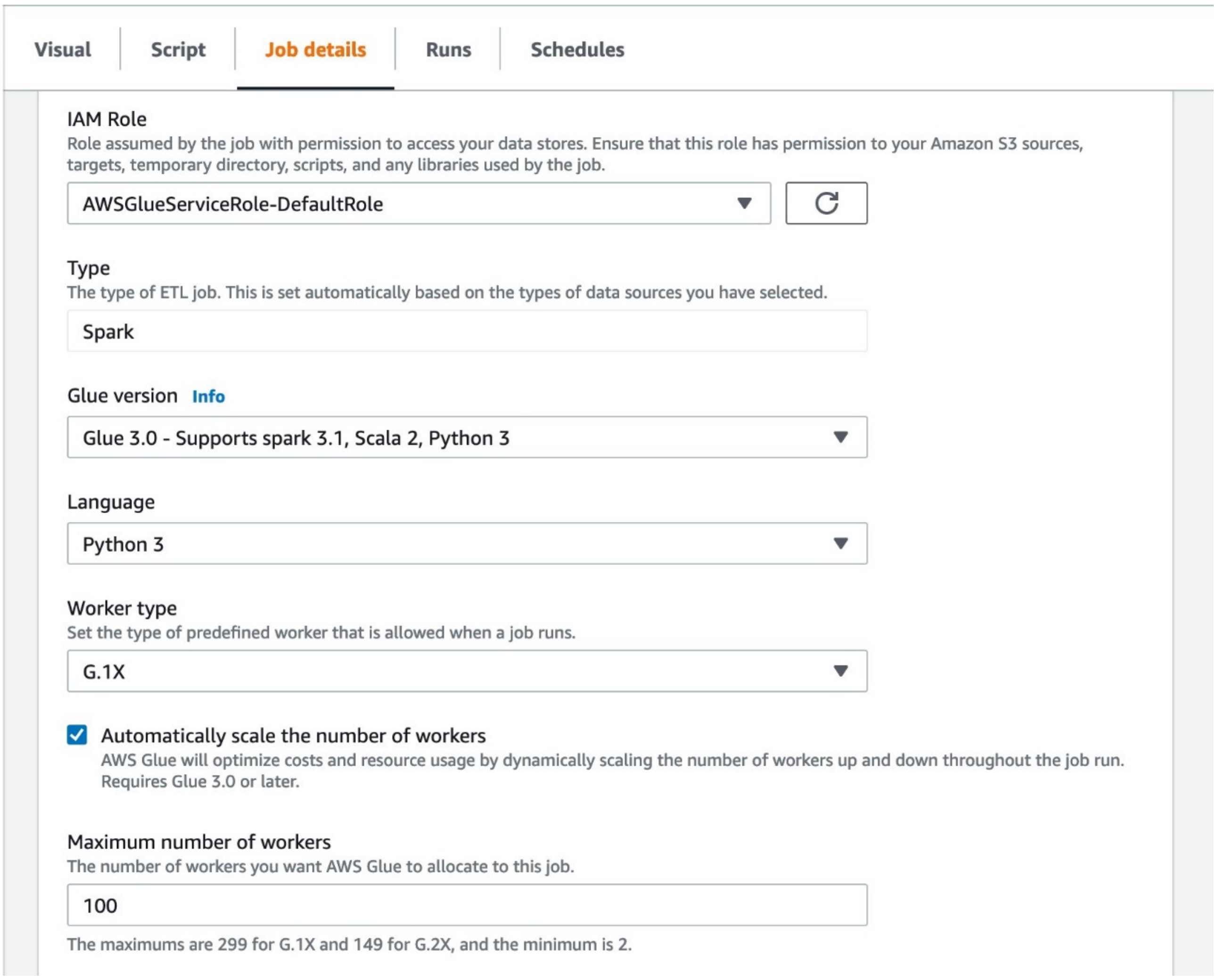
Below we will touch into four main advantages of AWS Glue that attracted us to give it a try for this project's ETL processes.

Simple ETL pipeline development with Autoscaling

AWS Glue removes infrastructure management by providing automatic provisioning and worker management as well as consolidating all the Data Integration needs into a single service. This is achieved by AWS Glue Auto Scaling which automatically resizes the computing resources of AWS Glue Spark job capacity in order to reduce cost.

How does it work?
According to the AWS Glue documentation, traditionally, AWS Glue launches a serverless Spark cluster of a fixed size. The computing resources are held for the whole job run until it is completed. With the new AWS Glue Auto Scaling feature, after you enable it for your AWS Glue Spark jobs, AWS Glue dynamically allocates compute resource considering the given maximum number of workers. [Sera]

This means that as Spark requests more executers, more AWS Glue workers are added to the cluster. When the executor has been idle without active computation tasks for a period of time and associated shuffle dependencies, the executor and corresponding worker are removed. [Sera]



IAM Role
Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.
AWSGlueServiceRole-DefaultRole

Type
The type of ETL job. This is set automatically based on the types of data sources you have selected.
Spark

Glue version Info
Glue 3.0 - Supports spark 3.1, Scala 2, Python 3

Language
Python 3

Worker type
Set the type of predefined worker that is allowed when a job runs.
G.1X

[✓] Automatically scale the number of workers
AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

Maximum number of workers
The number of workers you want AWS Glue to allocate to this job.
100
The maximums are 299 for G.1X and 149 for G.2X, and the minimum is 2.

Figure 6.1: Enabling autoscaling for an aws glue job

Which use cases can we benefit from it?

According to the AWS Glue documentation, the following are interesting use cases for our ETL processes:

- Batch jobs to process unpredictable amounts of data
- Jobs containing driver-heavy workloads (for example, processing many small files)
- Jobs containing multiple stages with uneven compute demands or due to data skews
- Jobs to **write large amounts of data into Data Warehouses such as Amazon Redshift** or read and write from databases

Automatically discovering data with crawlers

AWS Glue provides crawlers that automatically infer schema information and integrate them into AWS Glue Data Catalog. Figure 6.2 shows how this process works.

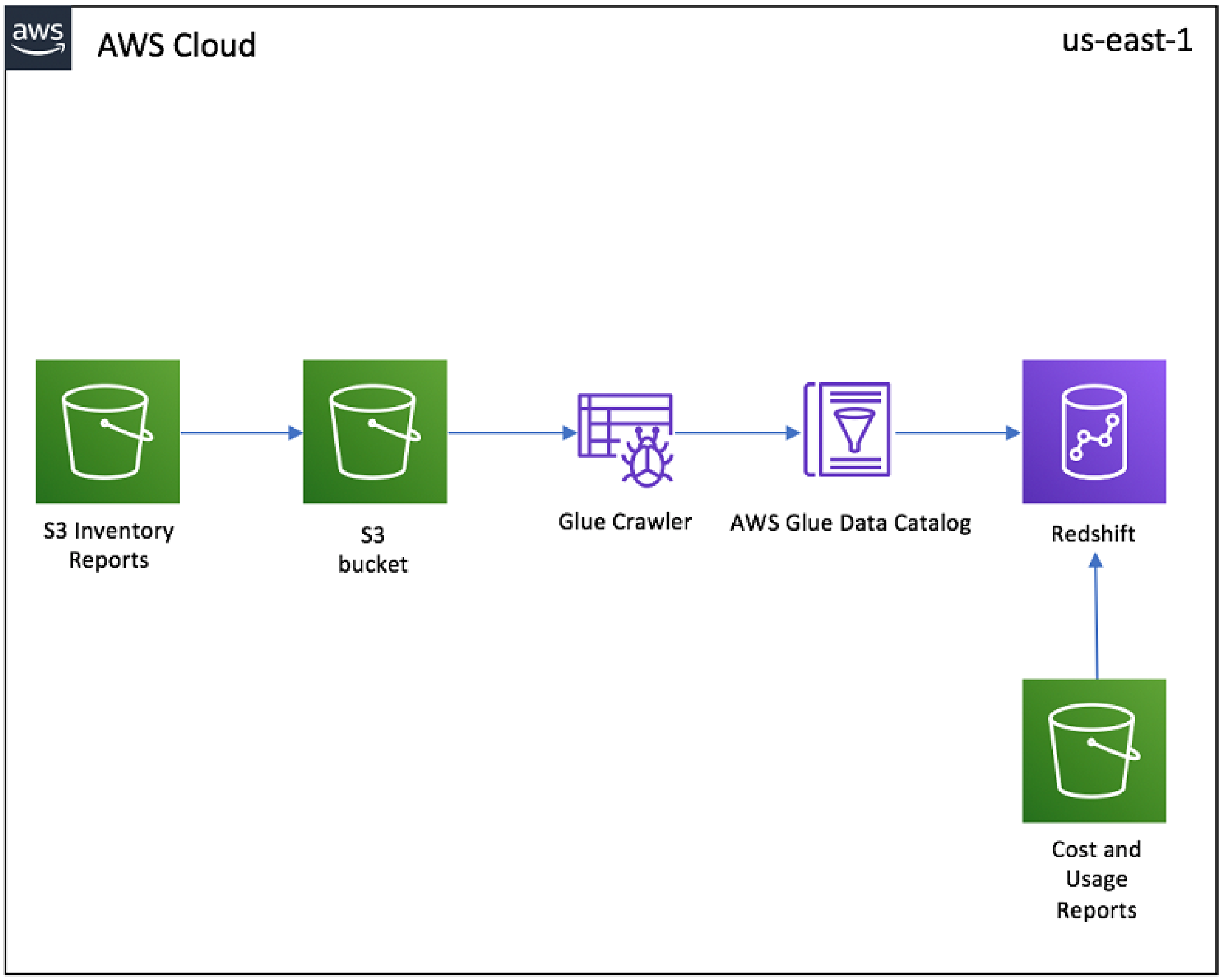


Figure 6.2: AWS Glue Crawler

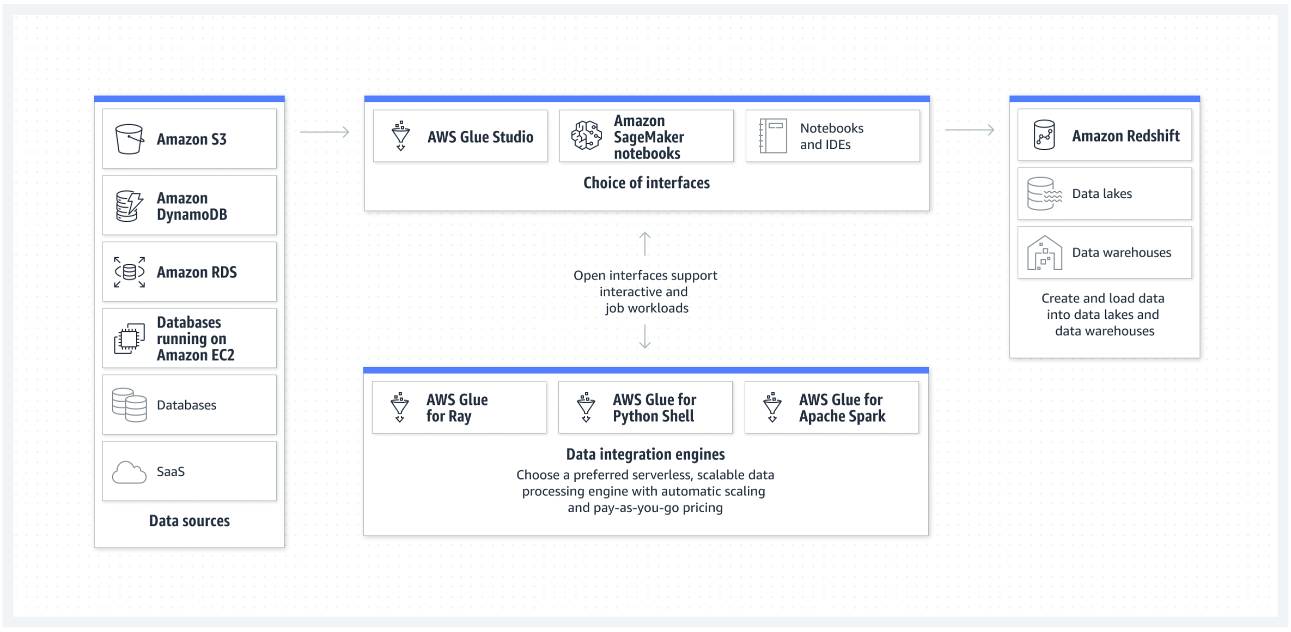

Amazon S3
Amazon DynamoDB
Amazon RDS
Databases running on Amazon EC2
Databases
SaaS
Data sources
AWS Glue Studio
Amazon SageMaker notebooks
Notebooks and IDEs
Choice of interfaces
Open interfaces support interactive and job workloads
AWS Glue for Ray
AWS Glue for Python Shell
AWS Glue for Apache Spark
Data integration engines
Choose a preferred serverless, scalable data processing engine with automatic scaling and pay-as-you-go pricing
Amazon Redshift
Data lakes
Data warehouses
Create and load data into data lakes and data warehouses

Figure 6.3: AWS Glue Interactions

Interactively exploring, experimenting on, and processing data

AWS Glue interactive sessions enables us to interactivley explore and prepare data using a notebook. This results in rapidly building, testing and running data preperation and analytics operations.

Interactive sessions also provide visual interface for ETL scripts.

Event driven ETL

AWS Glue enables running extract, transform, and load (ETL) jobs as new data arrives. For example, one can configure AWS Glue to initiate their ETL jobs to run as soon as new data becomes available in Amazon Simple Storage Service (S3). [Sera]

Figure 6.3 represents how this process works.

No code ETL jobs

AWS Glue Studio makes it easier to visually create, run, and monitor AWS Glue ETL jobs. One can build ETL jobs that move and transform data using a drag-and-drop editor, and AWS Glue automatically generates the code[Sera]. With this usage, we will have a faster ETL job creation that transform raw benchmark files into specific schema formats to be ready to load into Redshift.

Since we use Amazon Redshift as our Data Warehouse solution, choosing an AWS ETL service would be the perfect solution for us considering AWS's seamless service integration. Combined with the powerful features above, we decided to use AWS Glue as our ETL tool at the beginning of

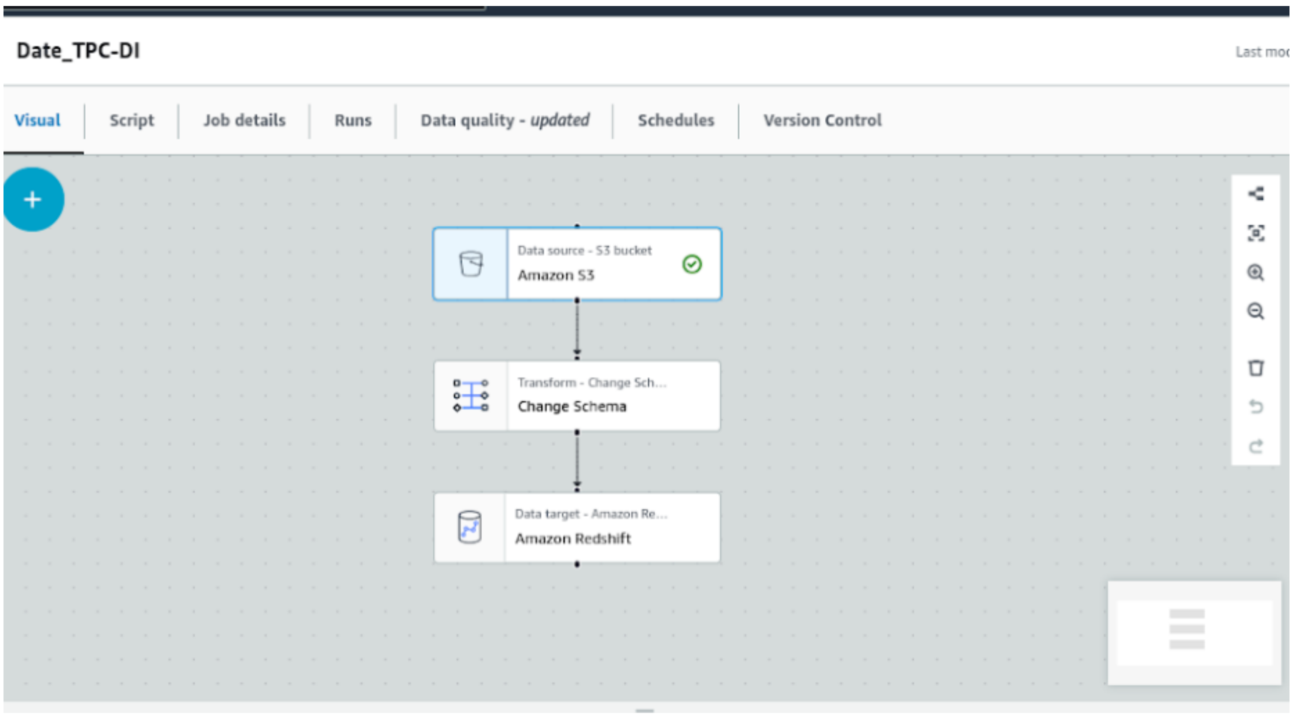


Figure 6.4: AWS Glue Studio drag and drop ETL processes

this project.

6.1.2 Expected pricing

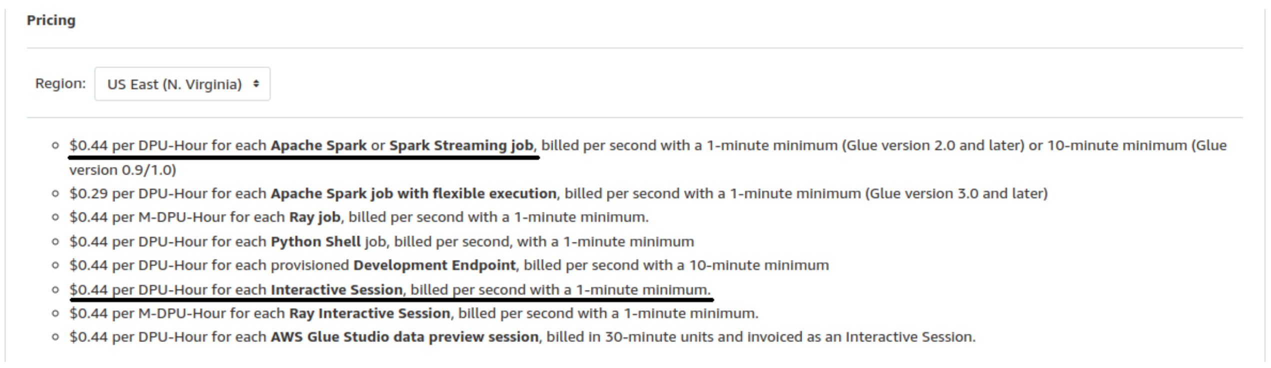
Pricing is a very important factor while using cloud services especially for teams that are on low budget. In that sense, we dived deep into pricing documentation of AWS Glue to calculate our expected expenses and figure out whether we can manage using it with our limited budget.

According to AWS Glue pricing documentation, you only pay for the time that your ETL job takes to run. There are no resources to manage and no upfront costs, and you are not charged for startup or shutdown time. AWS charges an hourly rate based on the number of data processing units (DPUs) used to run your ETL job. A single standard DPU provides 4 vCPU and 16 GB of memory, whereas a high-memory DPU (M-DPU) provides 4 vCPU and 32 GB of memory.[Serb]

There are four types of AWS Glue jobs: Apache Spark, Apache Spark Streaming, Ray (Preview), and Python Shell. Spark and Spark Streaming job runs require a minimum of 2 DPUs.

Interactive Sessions are optional, and billing applies only if you use it for interactive ETL code development. AWS charges for Interactive Sessions based on the time the session is active and the number of DPUs. Interactive sessions have configurable idle timeouts. AWS Glue Interactive Sessions require a minimum of 2 DPUs and have a default of 5 DPU.

AWS Glue Studio data previews enables one to test their transformations during the job-authoring process. Each AWS Glue Studio data preview session uses 2 DPUs, runs for 30 minutes, and stops

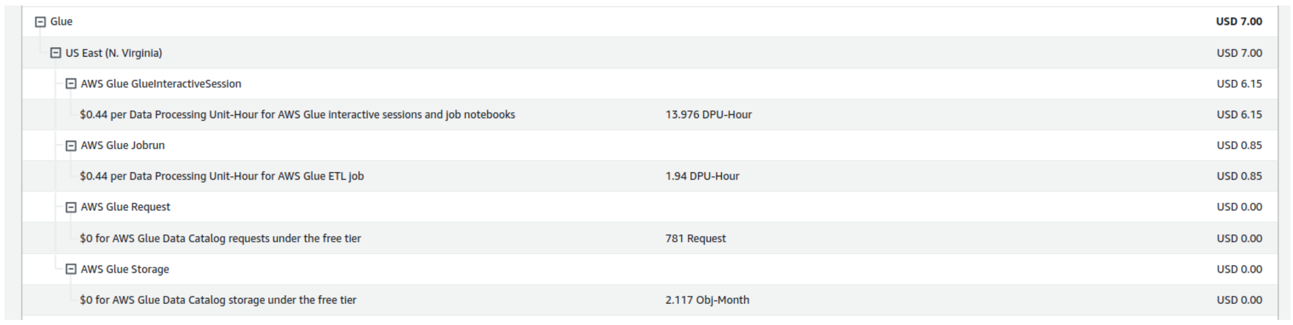


Pricing

Region: US East (N. Virginia)

- $0.44 per DPU-Hour for each **Apache Spark** or **Spark Streaming job**, billed per second with a 1-minute minimum (Glue version 2.0 and later) or 10-minute minimum (Glue version 0.9/1.0)
- $0.29 per DPU-Hour for each **Apache Spark job with flexible execution**, billed per second with a 1-minute minimum (Glue version 3.0 and later)
- $0.44 per M-DPU-Hour for each **Ray job**, billed per second with a 1-minute minimum.
- $0.44 per DPU-Hour for each **Python Shell** job, billed per second, with a 1-minute minimum
- $0.44 per DPU-Hour for each provisioned **Development Endpoint**, billed per second with a 10-minute minimum
- $0.44 per DPU-Hour for each **Interactive Session**, billed per second with a 1-minute minimum.
- $0.44 per M-DPU-Hour for each **Ray Interactive Session**, billed per second with a 1-minute minimum.
- $0.44 per DPU-Hour for each **AWS Glue Studio data preview session**, billed in 30-minute units and invoiced as an Interactive Session.

Figure 6.5: AWS Glue Pricing



Glue		**USD 7.00**
US East (N. Virginia)		USD 7.00
AWS Glue GlueInteractiveSession		USD 6.15
$0.44 per Data Processing Unit-Hour for AWS Glue interactive sessions and job notebooks	13.976 DPU-Hour	USD 6.15
AWS Glue Jobrun		USD 0.85
$0.44 per Data Processing Unit-Hour for AWS Glue ETL job	1.94 DPU-Hour	USD 0.85
AWS Glue Request		USD 0.00
$0 for AWS Glue Data Catalog requests under the free tier	781 Request	USD 0.00
AWS Glue Storage		USD 0.00
$0 for AWS Glue Data Catalog storage under the free tier	2.117 Obj-Month	USD 0.00

Figure 6.6: AWS Glue charge for our benchmark trials

automatically.

We decided to **not to utilize interactive notebook sessions** as we can easily replace them with available free notebook sessions on our local computers. We took this decision in order to reduce the costs.

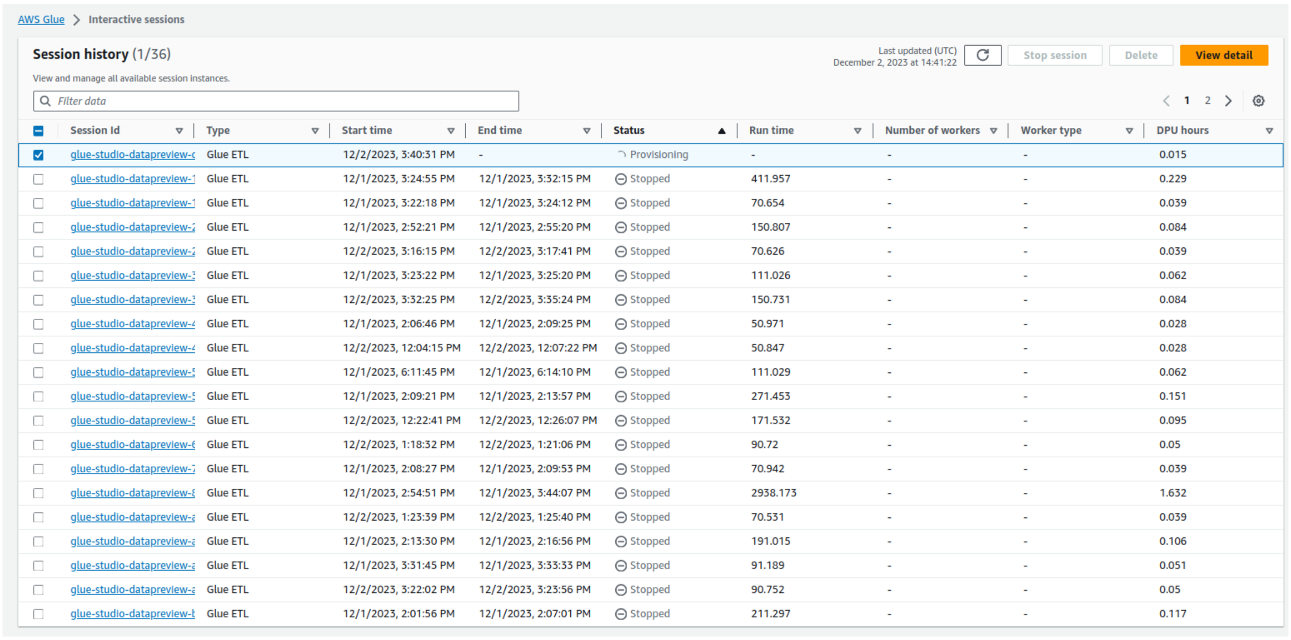
Underlined lines in Figure 6.5 shows that the only expected pricing for us should be the following:

- **$0.44 per DPU hour** for Apache Spark jobs, billed per second with a 1-minute minimum.
- **$0.44 per DPU hour** for AWS Glue crawlers with a 10-minute minimum per crawler run

6.1.3 Our pricing experience

While we were expecting to be charged either only for our Glue job runs or in the worst case scenario charged for Glue jobs plus crawler usage, we faced a strange pricing coming from AWS Glue interactive sessions.

Figure 6.6 shows that after running our AWS Glue jobs around 2 hours in total, we saw a usage of almost 14 DPU hours pricing for AWS Glue interactive sessions. This means that almost 90% of our pricing comes from a service we did not even know we used.


AWS Glue > Interactive sessions
Last updated (UTC) December 2, 2023 at 14:41:22
Stop session
Delete
View detail
Filter data
1 2

Session history (1/36)

View and manage all available session instances.

Session Id	Type	Start time	End time	Status	Run time	Number of workers	Worker type	DPU hours
glue-studio-datapreview-c	Glue ETL	12/2/2023, 3:40:31 PM	-	Provisioning	-	-	-	0.015
glue-studio-datapreview-1	Glue ETL	12/1/2023, 3:24:55 PM	12/1/2023, 3:32:15 PM	Stopped	411.957	-	-	0.229
glue-studio-datapreview-1	Glue ETL	12/1/2023, 3:22:18 PM	12/1/2023, 3:24:12 PM	Stopped	70.654	-	-	0.039
glue-studio-datapreview-2	Glue ETL	12/1/2023, 2:52:21 PM	12/1/2023, 2:55:20 PM	Stopped	150.807	-	-	0.084
glue-studio-datapreview-2	Glue ETL	12/2/2023, 3:16:15 PM	12/2/2023, 3:17:41 PM	Stopped	70.626	-	-	0.039
glue-studio-datapreview-3	Glue ETL	12/1/2023, 3:23:22 PM	12/1/2023, 3:25:20 PM	Stopped	111.026	-	-	0.062
glue-studio-datapreview-3	Glue ETL	12/2/2023, 3:32:25 PM	12/2/2023, 3:35:24 PM	Stopped	150.731	-	-	0.084
glue-studio-datapreview-4	Glue ETL	12/1/2023, 2:06:46 PM	12/1/2023, 2:09:25 PM	Stopped	50.971	-	-	0.028
glue-studio-datapreview-4	Glue ETL	12/2/2023, 12:04:15 PM	12/2/2023, 12:07:22 PM	Stopped	50.847	-	-	0.028
glue-studio-datapreview-5	Glue ETL	12/1/2023, 6:11:45 PM	12/1/2023, 6:14:10 PM	Stopped	111.029	-	-	0.062
glue-studio-datapreview-5	Glue ETL	12/1/2023, 2:09:21 PM	12/1/2023, 2:13:57 PM	Stopped	271.453	-	-	0.151
glue-studio-datapreview-5	Glue ETL	12/2/2023, 12:22:41 PM	12/2/2023, 12:26:07 PM	Stopped	171.532	-	-	0.095
glue-studio-datapreview-6	Glue ETL	12/2/2023, 1:18:32 PM	12/2/2023, 1:21:06 PM	Stopped	90.72	-	-	0.05
glue-studio-datapreview-7	Glue ETL	12/1/2023, 2:08:27 PM	12/1/2023, 2:09:53 PM	Stopped	70.942	-	-	0.039
glue-studio-datapreview-8	Glue ETL	12/1/2023, 2:54:51 PM	12/1/2023, 3:44:07 PM	Stopped	2938.173	-	-	1.632
glue-studio-datapreview-a	Glue ETL	12/2/2023, 1:23:39 PM	12/2/2023, 1:25:40 PM	Stopped	70.531	-	-	0.039
glue-studio-datapreview-a	Glue ETL	12/1/2023, 2:13:30 PM	12/1/2023, 2:16:56 PM	Stopped	191.015	-	-	0.106
glue-studio-datapreview-a	Glue ETL	12/1/2023, 3:31:45 PM	12/1/2023, 3:33:33 PM	Stopped	91.189	-	-	0.051
glue-studio-datapreview-a	Glue ETL	12/2/2023, 3:22:02 PM	12/2/2023, 3:23:56 PM	Stopped	90.752	-	-	0.05
glue-studio-datapreview-b	Glue ETL	12/1/2023, 2:01:56 PM	12/1/2023, 2:07:01 PM	Stopped	211.297	-	-	0.117

Figure 6.7: AWS Glue Job data preview continues even after disabling (provisioning)

When we dived deeper into this problem we have noticed that every time we open the AWS Glue Studio, the glue instance starts a **data preview session** automatically.

As we can see from the previous Figure 6.5, these sessions come with a pricing of " **$0.44 per DPU-Hour** for each AWS Glue Studio data preview session, billed in 30-minute units and invoiced as an Interactive Session."

We thought that this should be a feature that can be disabled. As a matter of fact, it can be disabled as one can see in Figure 6.8

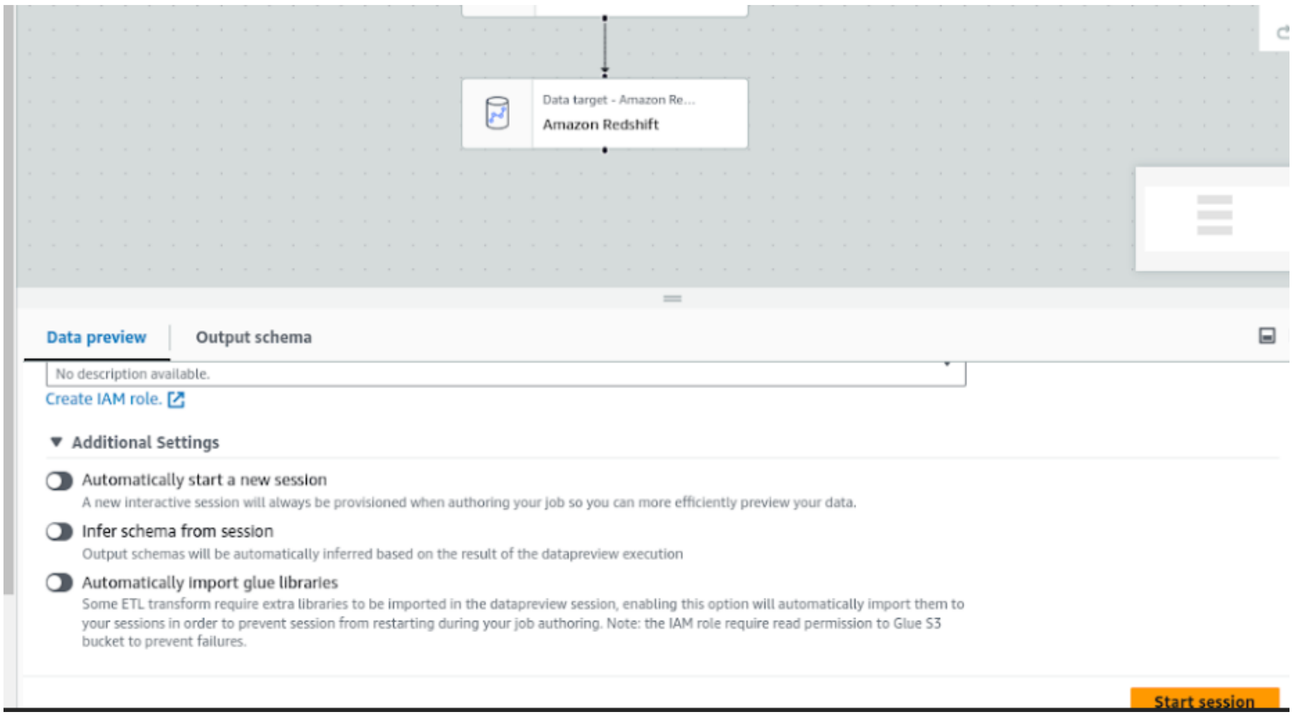
However, disabling this feature does not reflect neither on the other Glue jobs nor on the same job when it is started again. Which means, everytime we start a new Glue job, we need to manually disable this feature. However, the data preview job stays provisioned until it reflects our disabling choice, which in the mean time spends DPU hours.

Figure 6.7 shows the behavior mentioned above.

After noticing that **disabling the data preview functionality is not possible** and being charged $8 for only a total of 2 hours usage, **we decided to abandon AWS Glue for our ETL purposes**. Figure 6.6 shows our total pricing coming from each service.

6.2 AWS Networking & Security

Amazon Web Services is the biggest cloud in the world at this moment, and one of the reasons for it's success is it's secure defaults. *AWS* strives to be a secure cloud, where customers need to explicitly opt-in for more risky and costly features. For example, an *AWS Lambda*, by default doesn't



No description available.
Create IAM role.

Additional Settings

Automatically start a new session
A new interactive session will always be provisioned when authoring your job so you can more efficiently preview your data.

Infer schema from session
Output schemas will be automatically inferred based on the result of the datapreview execution

Automatically import glue libraries
Some ETL transform require extra libraries to be imported in the datapreview session, enabling this option will automatically import them to your sessions in order to prevent session from restarting during your job authoring. Note: the IAM role require read permission to Glue S3 bucket to prevent failures.

Figure 6.8:

have access to any other resource, this mean it cannot access any database, or *S3* bucket, etc. Everything needs to be explicitly allowed. This is amazing from the security point of view, but from the developer' side, it can be quite time consuming to figure out, specially in the beginning, all of the configuration to be changed in order to have all of the pieces all working together.

This made us to spent very precious time debugging specific AWS issues, such as for example the necessary permissions we needed to add to our Lambda Functions.

6.3 AWS Lambda dependencies

Another challenge we faced was one of the main limitations of *AWS Lambda*, which is the fact that *AWS Lambda* does not support external dependencies out of the box. This means that any library outside the *Python Standard Library* and *Boto3*, the Python AWS Client, are not possible to install out of the box.

This greatly limits the power of Python, as one of it's main benefits is it's vast ecosystem, arguably the biggest one of any programming language. To add a dependency to a *Lambda Function* you need to create a so-called *Lambda Layer*, which is a `.zip` file that contains additional code or data, since the code you can upload to a *Lambda Function* is rather limited, unzipped, it must be smaller than 250MB.

7 Conclusion

In conclusion, our project aimed to implement *TPC-DI benchmarking* on AWS using *AWS Lambda*, focusing on the efficiency of data integration workflows. The key objectives were to evaluate the feasibility and performance of utilizing AWS Lambda for TPC-DI benchmarking, evaluate scalability, efficiency, and cost-effectiveness, and provide insights into the advantages and challenges associated with using serverless computing for data integration.

7.1 Benchmark Preparation

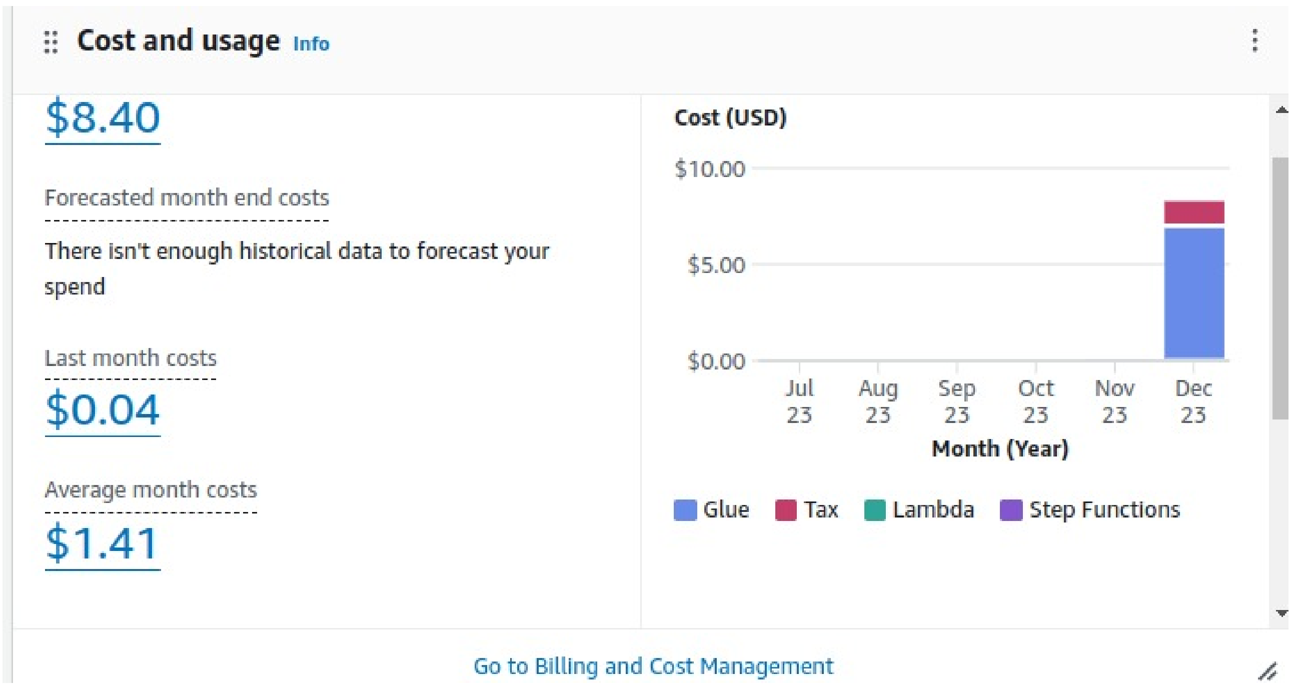
We detailed the data population process, including data definition, generation using TPC's *DIGen tool*, and loading into *Amazon Redshift*. The data warehouse initialization was performed based on TPC-DI specifications.

7.2 Execution

We elaborated on the execution steps of the TPC-DI benchmark, covering historical load and incremental updates. The historical load involved creating a dependency graph, development on local machines, and migration to AWS Lambda using *Step Functions*. Incremental updates utilized *CDC extracts*, and an *ETL process* was implemented for data extraction, transformation, and loading.

7.3 AWS for ETL

Now we will cover our conclusion regarding the usage of the *AWS* cloud to perform ETL workflows.



Cost and usage Info

$8.40

Forecasted month end costs

There isn't enough historical data to forecast your spend

Last month costs

$0.04

Average month costs

$1.41

Go to Billing and Cost Management

Figure 7.1: Pricing for AWS Glue

7.3.1 AWS Glue

Glue, as we've already seen is the go-to service when it comes to perform ETL workloads in AWS. It provides a very easy to use User Interface to architect your workloads, with a neat *Data Preview* feature that allows you to seamlessly know at every step, the status of the data.

Unfortunately, as we've seen, it doesn't provide a generous Free Tier, and some of those great features come at a cost. So even when we acknowledge the flexibility and power of *Glue*, we believe it's demanding pricing model makes it only suitable for bigger companies with a more accommodate budget.

Figure 7.1 shows the pricing of AWS Glue that mentioned above.

7.3.2 AWS Step Functions

We were greatly surprised by the flexibility and easy onboarding of *AWS Step Functions*, as getting started with it was as simple as you could expect from a workflow architecting tool. As simple as drag-and-drop the different operations we wanted to perform. Also the fact that we were able to use it for free was a really

7.4 Results

We discussed the performance metrics, focusing on historical load and incremental load throughput. The TPC-DI benchmark calculates two main performance metrics: *Performance Metric* and *Price/Performance Metric*. Challenges faced during the project, particularly related to *unexpected AWS Glue pricing* and networking/security configurations, were outlined.

7.5 Challenges Faced

Issues with AWS Glue, specifically unexpected pricing due to interactive sessions, led to reconsidering its use in the project. Challenges in AWS networking and security, as well as limitations in AWS Lambda regarding external dependencies, required careful consideration and additional effort to overcome.

7.6 Lessons Learned

The project underscored the importance of understanding and managing AWS services' configurations, security settings, and potential hidden costs. The limitations of AWS Lambda in handling external dependencies highlighted the need for effective workarounds, such as Lambda Layers.

In conclusion, while challenges were encountered, the project provided valuable insights into the practical aspects of implementing a TPC-DI benchmark on AWS Lambda, contributing to a better understanding of the complexities and considerations involved in serverless data integration workflows.

Bibliography

[TPC14] (TPC), Transaction Processing Performance Council (Nov. 2014). *TPC BENCHMARK DI (Data Integration) Standard Specification*. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DI_v1.1.0.pdf. Accessed: December 19, 2023.

[Sera] Services, Amazon Web (n.d.[a]). *AWS Glue Documentation*. https://aws.amazon.com/glue/. Accessed: December 19, 2023.

[Serb] — (n.d.[b]). *AWS Glue Documentation*. https://aws.amazon.com/glue/pricing/. Accessed: December 19, 2023.