Project Documentation

Development of a Data Lake for Incremental and Snapshot Data Management in AWS Environment

Use Cases

- Incremental Data Management
- Snapshot Data Management
- Error Management and Data Reprocessing

Technologies Used

The main technologies used in this project are the following:

AWS S3

Amazon Simple Storage Service or S3 provides object storage through a web service interface. S3 has been chosen for this project because of its high resilience, low cost per GB (between $0.021 and $0.023 per GB for "S3 Standard" storage), and its scalability for managing big data.

AWS Athena

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard SQL. Since the storage service in use for out project is AWS S3, Athena is the natural choice of query running service. It also scales automatically—executing queries in parallel—so results are fast, even with large datasets and complex queries.

AWS Glue

AWS Glue is a fully managed extract, transform, and load (ETL) service. Glue is a relatively new technology, lets us use PySpark language in its ETL jobs which makes it a very powerful tool. It also integrates well with AWS S3 and AWS Athena. 4 different AWS Glue services have been used in this project: Glue Data Catalog, Glue Crawlers, Glue ETL Jobs and Glue Workflows.

Glue Data Catalog

AWS Glue Data Catalog is used as a central repository that is used to store metadata for all the data assets of the user. Information in the Data Catalog is stored as metadata tables, where each table specifies a single data store, in our case on S3. Glue Data Catalog contains references to data that is used as sources and targets of extract, transform, and load (ETL) jobs in AWS Glue.

Glue Crawlers

AWS Glue Crawlers are the primary means to populating the Glue Data Catalog. When a crawler runs, it takes the following actions to interrogate a data store:

1. Classifies data to determine the format, schema, and associated properties of the raw data
2. Groups data into tables or partitions
3. Writes metadata to the Glue Data Catalog

Custom Classifiers

Crawlers allow the use of custom classifiers to help them detect the correct schema for the data store at hand. Custom classifiers can be created using Grok patterns, XML tags, JavaScript Object Notation (JSON), or comma-separated values (CSV). Multiple classifiers can be defined and associated to a single Glue Crawler, in a priority-based order. AWS Glue runs custom classifiers before built-in classifiers, and when a crawler finds a classifier that matches the data, the classification string and schema are used in the definition of tables that are written to your AWS Glue Data Catalog. If no classifier matches the data, the Glue Crawler uses its own built-in classifiers to classify the data.

Problems in the use of Glue Crawlers

- Problems detecting non-standard data: Glue Crawlers fail to identify data correctly if there are non standard structures in the data, such as when there are header rows, quoted fields, or null fields. This problem can be addressed in two ways: - Using a custom Grok classifier. - Using OpenCSV serialization/deserialization. Our preferred method in this project has been using OpenCSV serialization/deserialization as it is the simpler of the two methods.
- Problems detecting the correct data types: Glue Crawlers are not always successful at detecting the correct data types. The solutions mentioned for the previous problems don't work here, as both the Grok classifier and the OpenCSV serializer/deserializer see all data as strings. No solution for this problem has been found at the Crawler level; however, data types can easily be corrected at the ETL job level. Due to the aforementioned limits of Glue Crawlers, besides a first time use for inferring the schema of new structures of data (for which the Crawler has to be configured accordingly), Glue Crawlers are mainly used for the detection of new partitions in structured data.

Glue ETL Jobs

Jobs are the tool for performing extract, transform, and load (ETL) work in Glue. When a job is run, AWS Glue runs a script that extracts data from the source(s), transforms the data, and loads it into the target(s). Glue enables us to use PySpark script for our ETL jobs. This is the main feature that makes Glue ETL jobs a powerful tool. For typical use cases, Glue can also autogenerate scripts with AWS Glue extensions that can later be modified.

Job Bookmarks

Job Bookmarks are a feature of Glue ETL jobs that help maintain state information and prevent the reprocessing of already-processed data. In this way they prevent data duplication in the job target. Job bookmarks store the states for a job. Each instance of the state is keyed by a job name and a version number.

When a script invokes job.init, it retrieves its state and always gets the latest version. Within a state, there are multiple state elements, which are specific to each source, transformation, and sink instance in the script. These state elements are identified by a transformation context that is attached to the corresponding element (source, transformation, or sink) in the script. The state elements are saved atomically when job.commit is invoked from the script. For Amazon S3 input sources, Glue job bookmarks check the last modified time of the files to verify which objects need to be reprocessed.

- **Problem in the use of Job Bookmarks**: Glue Job Bookmarks work perfectly for preventing data duplication; however, they cause problems when an error occurs and data needs to be reprocessed. The solution implemented for preventing this problem is a change in the architecture. We implemented an architecture inspired by the Snowplow platform, that gives us error control flexibility and in the mean time prevents data duplication.

DPUs

A single Data Processing Unit (DPU) provides 4 vCPU and 16 GB of memory, which is equivalent to a m4.xLarge EC2 instance. It can be set at the "Maximum capacity" job parameter. An AWS Glue job of type Apache Spark requires a minimum of 2 DPUs. By default, AWS Glue allocates 10 DPUs to each Apache Spark job. Every DPU hosts 2 executors. Out of the total 20 executors, 1 executor is reserved for Driver program and 1 DPU for the application master. The actual workload is carried out by 2 * 10–2 (Master) — 1(Driver) = 17 executors.

Worker types

The following worker types are available:

- **Standard** : With this worker type, a value for "**Maximum capacity**" (i.e. the number of DPU that can be allocated when this job runs) has to be set. The **Standard** worker type has a 50 GB disk and 2 executors.
- **G.1X**: With this worker type, a value for "**Number of workers**" has to be set. Each worker maps to 1 DPU (4 vCPU, 16 GB of memory, 64 GB disk), and provides 1 executor per worker. This worker type is ususally used for memory-intensive jobs.
- **G.2X**: With this worker type, just like with the G1X worker type, a value for "**Number of workers**" has to be set. Each worker maps to 2 DPU (8 vCPU, 32 GB of memory, 128 GB disk), and provides 1 executor per worker. This worker type is ususally used for jobs that run ML transforms.

The worker type for ETL jobs used in this project are of type "**Standard**".

Glue Workflows

- Glue Triggers: A trigger controls when an ETL job runs in AWS Glue. The triggering condition can be based on a schedule (as defined by a cron expression) or on an event. Triggers can also run on demand.

Using Glue Workflows, We can author directed acyclic graphs (DAGs) made of Glue Triggers, Crawlers and ETL Jobs. Workflows create dependencies between Glue entities and in this way, help us orchestrate automated ETL workloads.

AWS Lake Formation

AWS Lake Formation is a fully managed service that simplifies and automates many of the complicated manual steps usually required to create a data lake, including collecting, cleaning, cataloging, and preparing data for analytics. Lake Formation first identifies existing data stores in S3 or relational and NoSQL databases, and moves the data into the data lake. Then it crawls, catalogs, and prepares the data for analytics. The main technology used by Lake Formation under the hood is, however, AWS Glue. The problem with Lake Formation is that in order to adapt the automated processes like ETL Jobs and Crawlers to our specific use case, most of them had to be retouched and edited manually. At this point te choice was made to create the data lake using Glue in a way that suits our use cases instead of using AWS Lake Formation.

Parquet file format

Parquet is an open source file format for the Hadoop ecosystem that stores nested data structures in a flat columnar format. Parquet was chosen as the file format for structured data in our data lakes because compared to a traditional row-oriented approach, it is more efficient in terms of storage and performance.

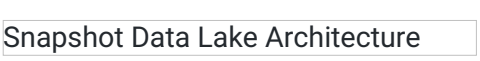
Architecture

The data lake architectures have been inspired by the Snowplow platform. The reason for this choice is the problems with the functioning of Glue ETL job Bookmarks for reprocessing data when an error occurs; i.e. a job fails or there is a bug in the code.

Incremental Data Lake

A "Mover" is a python script, saved as an ETL job on Glue, that transfers files from one datalake zone to another. 1. A Mover job moves data from the Landing Zone to the Processing Zone only if the Processing Zone is empty. 2. The Processor job reads data from the Processing Zone, cleans it up, converts it to Parquet format and writes the processed data to Query Processing Zone. 3. The Processor job also saves its own run ID together with the file names it has processed to a table in the Glue Data Catalog. 4. When the Processor job is done, a Mover moves the raw data from Processing Zone to the Archive, leaving the Processing Zone empty for new data to arrive. 5. The Query Runner job reads the data saved in the Query Processing Zone by the Processor job, runs a query and saves the results in the Query Results zone. 6. The Query Runner job also saves its own ID, together with the ID of the Processor job that has processed the data and the names of the files it has processed to a table in the Data Catalog. 7. When the Query Runner job is done, a Mover transfers data from the Query Processing Zone to the Processed Zone.

Snapshot Data Lake



1. A Mover job moves data from Landing Zone to Processing Zone if the Processing Zone is empty.
2. The Processor job reads data from the Processing Zone, cleans it up, converts it to Parquet format and writes the processed data to the Processed Zone.
3. When the Processor job is done, a Mover transfers data from the Processing Zone to the Archive.
4. The Joiner job reads data from the Processed Zone and joins the data with the Query Results from the incremental data. The output of this job is saved to the Join Results zone.

Error Management

Error Management

- The Deleter job is run in case of Errors in another job run.
- Ther erroneous job ID is passed to the Deleter as a parameter.
- The Deleter proceeds to remove data associated with the given job ID in the following order:
    - The Processed Zone
    - The Processor job run ID associations
    - The Query Results
    - Query Results job run ID associations
- Then it proceeds to move the file(s) associated with the given run ID from the Archive to the Landing Zone for being reprocessed.

Configurations

Some of the most important configuration settings used for the making of our data lakes are mentioned below.

Permissions

For the use of AWS Glue services like Crawlers and ETL jobs, an IAM role has been defined. This role must have the following IAM permissions to be able to work correctly:

- AmazonS3FullAccess
- AWSGlueServiceRole
- AWSGlueConsoleFullAccess

Table configuration for CSV log files [1]

- **Input format**: org.apache.hadoop.mapred.TextInputFormat
- **Output format**: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
- **Serde serialization lib** [2] : org.apache.hadoop.hive.serde2.OpenCSVSerde
- **Serde parameters** [3]:

Key	Value
quoteChar	"

Key	Value
separatorChar	\t

- **Table properties** [4]:

Key	Value
skip.header.line.count	2

Crawler configuration for CSV log files [5]

- **Grouping behavior for S3 data**: None
- **Schema updates in the data store**: Update the table definition in the data catalog for all data stores except S3. For tables that map to S3 data, add new columns only.
- **Inherit schema from table**: Update all new and existing partitions with metadata from the table.
- **Object deletion in the data store**: Mark the table as deprecated in the data catalog.

ETL job configurations

Spark ETL job configurations

- **Type**: Spark
- **Glue version**: Spark 2.4, Python 3 (Glue version 1.0)
- **ETL language**: Python
    - **Advanced Properties**:
    - **Job bookmark**: Disable
    - **Security configuration, script libraries, and job parameters**
    - **Worker type**: Standard
    - **Maximum capacity** [6]: 2
    - **Max concurrency**: 1
    - **Job timeout (minutes)** [7]: 2880

Mover job configurations

- **Type**: Python Shell
- **Python version**: Python 3 (Glue version 1.0)
- **Security configuration, script libraries, and job parameters**
    - **Maximum capacity** [8]: 0.0625
    - **Max concurrency**: 1
    - **Job timeout (minutes)** [7]: 2880

Workflow Diagrams

[To be completed]

1. Parquet tables can use the default Glue configurations.↩
2. To be able to use OpenCSVSerde, all column data types must be set to string.↩
3. Default serde parameters have not been included.↩
4. Default table properties have not been included.↩
5. Parquet Crawlers can use the default Glue configurations.↩
6. Default value is 10. Set to 2 for testing.↩
7. The default is 2,880 minutes (48 hours).↩ ↩
8. Default value has been used.↩