SOFTWARE DESIGN

AWS Pipeline to ingest, process, and update the company's data from the third-party data sources

PREPARED FOR
XYZ as a part of the test task

PREPARED BY
Yurri Yurchenko, the candidate for the Staff Data Engineering position

FEB 04, 2023

Content

1. Description

For this part of the project, you will need to design a system to ingest data from a third party, expose it in the data lake and data warehouse, and populate a high-performance database for use in a micro-service. The existing components of the system are shown in the following diagram. It will be up to you to fill in the missing pieces to connect them all together.

We require data to be downloaded from a third-party SFTP site monthly. The provider makes new files available to us every month around the same day each month. This data needs to be retained in the raw form and exposed via Athena for us to research if needed. The data must also be processed and transformed into an internally defined data structure and exposed through Athena for querying.

Once the data is available in the Data Lake, we will also need it exposed from the Data Warehouse. The data in the warehouse does not need to include all the columns. There will be further processing to normalize enumerated fields and adjust the types to be easier to use from Redshift and the queries people will use.

We also require this data to be updated into a high-performance database (DynamoDB) so a microservice can expose this data through an API. The data format that is put into DynamoDB will be the same format as Redshift.

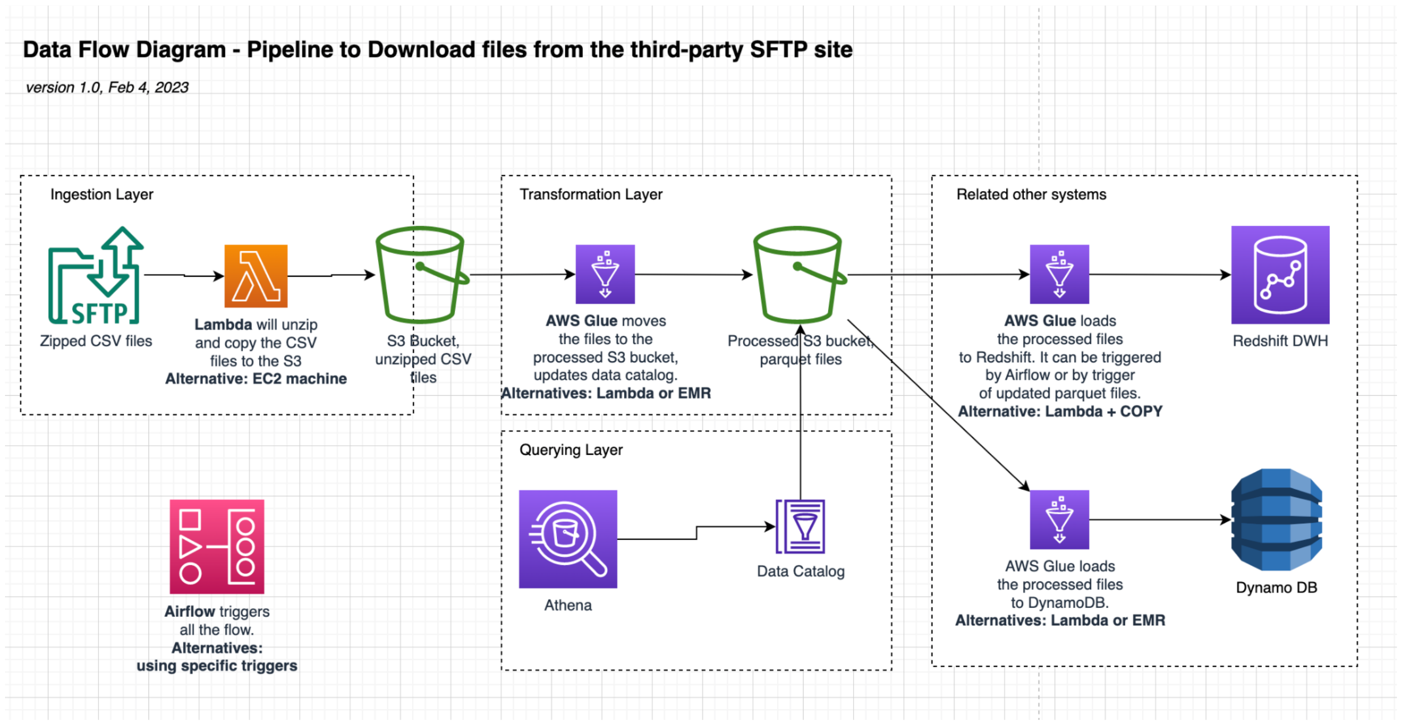
The system's requirements are:

- Download files from the third-party SFTP site.
- Store the raw files in S3.
- Process the raw files and make the data available to be queried using Athena.
- Process the raw files transforming the data into an internal format and making the data available through Athena.
- Make the data available in the data warehouse. Only a subset of the columns needs to be exposed in Redshift, and there will be some further transformations made based on business rules.
- Make the data available in DynamoDB for the Microservice.

2. My assumptions

- The total size of uncompressed files per month is less than 1 GB
- If the total size of uncompressed files per month is more significant than 2-3 GB then see the decision records

3. The general architecture of the pipeline


Data Flow Diagram - Pipeline to Download files from the third-party SFTP site
version 1.0, Feb 4, 2023
Ingestion Layer
Zipped CSV files
Lambda will unzip and copy the CSV files to the S3
Alternative: EC2 machine
S3 Bucket, unzipped CSV files
Transformation Layer
AWS Glue moves the files to the processed S3 bucket, updates data catalog.
Alternatives: Lambda or EMR
Processed S3 bucket, parquet files
Related other systems
AWS Glue loads the processed files to Redshift. It can be triggered by Airflow or by trigger of updated parquet files.
Alternative: Lambda + COPY
Redshift DWH
Querying Layer
Athena
Data Catalog
AWS Glue loads the processed files to DynamoDB.
Alternatives: Lambda or EMR
Dynamo DB
Airflow triggers all the flow.
Alternatives: using specific triggers

4. Decision records

1. <u>To ingest zipped CSV files, the Lambda Function was selected.</u> It is easy to implement. It is cost-effective. It supports connection to SFTP (unlike Glue). But it is limited by 10 GB RAM, 15 mins of running. If the more powerful computation is needed, it can be replaced by EC2 machine (which is a bit harder to configure).
2. <u>The target S3 file type will be Parquet.</u> It is columnar that will help with the querying data from Athena (unlike Avro). It is better to work with Spark/EMR if it is needed in the future comparing with ORC. It is compressed (unlike CSV), which means less operational costs.
3. <u>To transform the CSV file to the internally queried file, it was selected AWS Glue Jobs/ETL.</u> It is easy to connect to the S3. It is easy to convert CSV to Parquet. According to the AWS best practices, it is effective on the relatively average datasets.
4. <u>To update Redshift DWH and Dynamo DB, AWS Glue was also selected.</u> The main reasons are consistency (3 parts of the pipeline use the same transformation engine), maintainability (code in one place), and easy access to the connectors for Redshift and DynamoDB.
5. <u>To orchestrate the running process, I would choose Apache Airflow</u>.
    a. Main benefits:

i. Better maintainability. All the stages, current and historical, of the pipeline, are visualized.
ii. Enough customizability. Excellent level of customization of the pipeline/DAG (Sensors, Branches, Scheduling, Alerting).
iii. Good changeability. There are easy-to-create reusable operators to trigger proper sub-systems (Glue, Lambda, etc.)
iv. It is possible to implement idempotent data pipelines in the future

b. Drawbacks:
   i. The extended learning curve for the team. It can be decreased by preparing a "Cook Book" for the most critical or typical cases.

5. Limitations and future changes

If in the future, the amount of data will be increased significantly, the possible changes are - instead of using AWS Glue, replace it with **AWS EMR**. It can be easy to scale. It supports different and very custom transformations.

Vice versa, when the amount of data will be decreased, it makes sense to use **Lambda Functions** instead of Glue. They are simple and cost-effective.

6. Technical requirements

6.1. Ingestion part of moving CSV files to the S3 bucket.

The main requirements:

- We need to create the new Lambda function.
- It should connect to the SFTP server, check if existed the new file, unzip and move this file to the S3 bucket using the most common Python libraries.
- The triggering of this function should be configured in Apache Airflow.
- If there is no file - this function should raise an error in the Airflow pipeline.

6.2. Transformation layer to process and convert CSV files to the Parquet

The main requirements are:

- Source-to-Target excel should be created and presented before the development steps. This file should be added to the KB for the project in Confluence.

- The glue ETL job is configured to transform the CSV dataset to the Parquet dataset using the previous mapping.
- The Airflow pipeline is updated to trigger this job after the ingestion step.
- This job should update the Data Catalog as well to provide the refreshed data for Athena.

6.3. Querying layer

The main requirements are:

- The new external table, related to the Parquet files from the previous step, should be created in Athena. Helpful documentation is here.
- Athena successfully queries the table.

6.4. Updating of Redshift DWH

The main requirements are:

- A new table in Redshift is created
- The glue ETL job is configured to upload data from the Parquet dataset to the Redshift
- The execution engine should be Apache Spark, glue job type should be Python Shell Job
- Most of the read/write operations should be done using Glue API

6.5. Updating of DynamoDB

- A new table in DynamoDB is created
- The glue ETL job is configured to upload data from the Parquet dataset to the DynamoDB
- The execution engine should be Apache Spark, glue job type should be Python Shell Job
- Most of the read/write operations should be done using Glue API

6.6. Alerting

We need to add alerts to cover different cases of failures. As an example:

- SFTP file is not loaded in time;
- The broken files were sent;
- Some of the services in the chain are not working correctly;

- The process was completed successfully, but the data is weird (as an example, we usually receive 1M rows, but the CSV file has only 100K) - the system should notify us about that.

6.7. Orchestration

The main requirements are:

- The DAG for triggering the entire pipeline is prepared.
- Retry = 4 for every loading/transformation in the pipeline with an interval of 2 hours.
- If the pipeline fails - an alert message to the Teams group is configured.
- If any step fails - an alert message to the notification e-mail is configured.

7. Deployment instructions

1. Initially, the appropriate new resources should be requested to create on the Dev environment.
2. The whole pipeline should be configured on Dev and validated.
3. The potential issues should be noticed and documented.
4. After that, we need to create deployment instructions for the Prod environment.
5. Then, the pipeline parts should be implemented and carefully validated on Prod.
6. The appropriate documentation should be added to the Corporate KB, and the related teams should be notified.

8. Testing

The manual tests should cover the following:

1. Firstly, tests should be done on Dev, then on Prod.
2. Need to be tested that during the loading, the same data should be on the S3, DynamoDB, and Redshift.
3. Basic checking of the further Redshift transformations should also be done.
4. Need to be tested edge cases (no data, broken data)