

SquareShift Task Documentation - Data Engineer Role

Stock Analysis - [Github Repo](#)

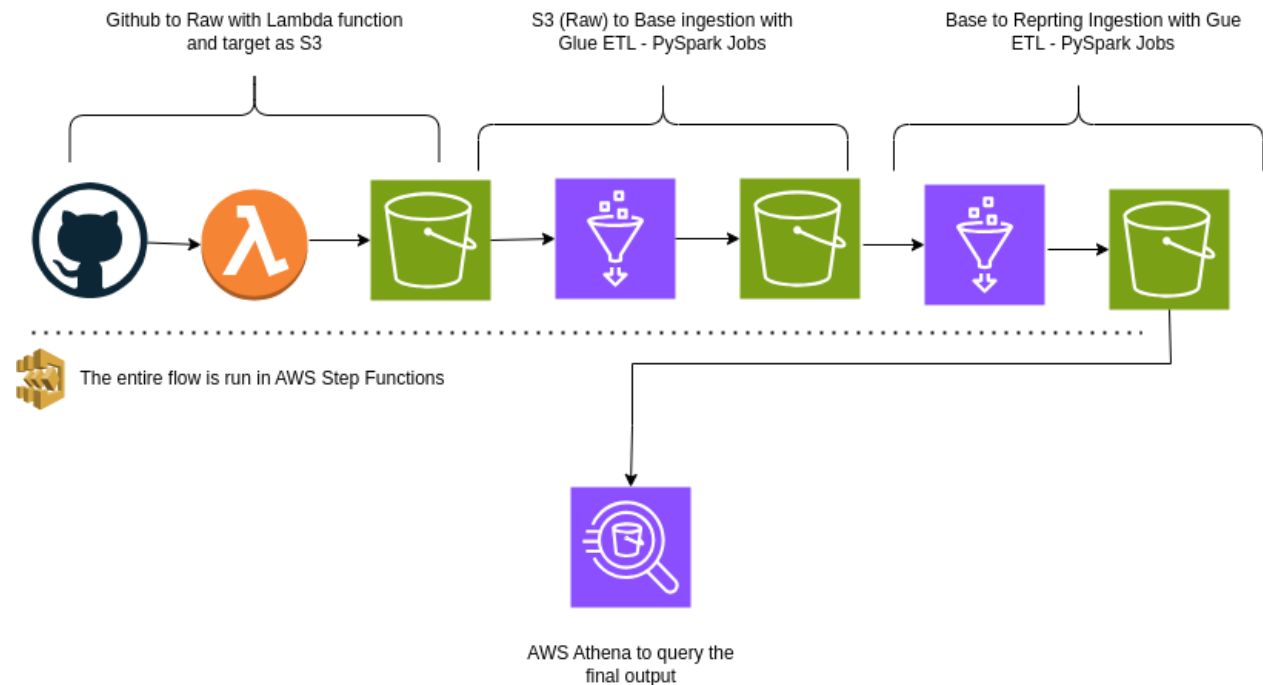
Author: Praghadeesh T K S

Date : 16th October 2023

Documentation Overview

- Overall Architecture
- Tech Stack
- Workflow Summary - Codes, Files, Process, Data Observation, Performance Optimization
- Miscellaneous

Overall Architecture



Tech Stack

- AWS Lambda
- AWS Glue ETL Jobs (PySpark)
- Simple Notification Service for sending pipeline alerts
- AWS S3 (Object Storage)
- AWS Athena (to query reporting tables)
- AWS Step Functions (to orchestrate the jobs)

Workflow Summary

Codes - Please refer the Github repo links for Code Comments and Explanation

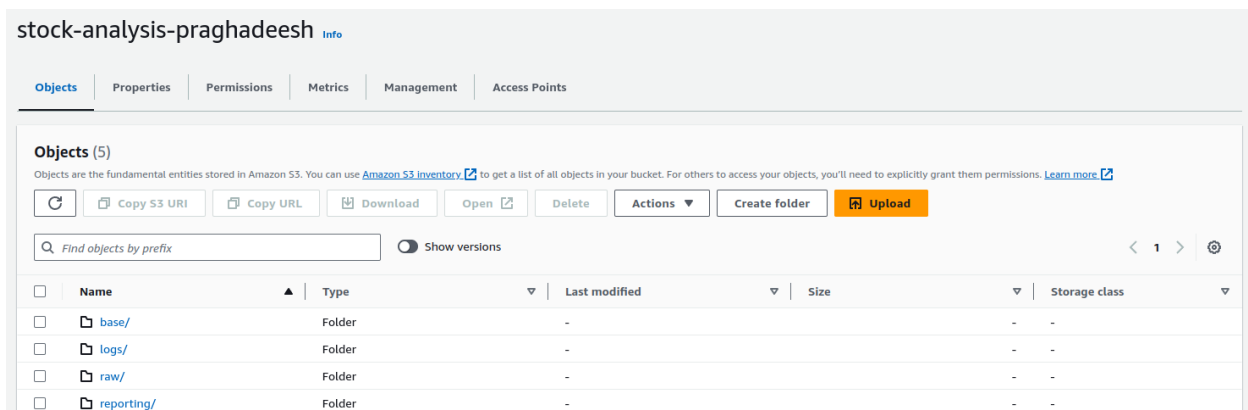
- [RawToBase.py - AWS Glue](#)
- [BaseToReporting.py - AWS Glue](#)
- [GithubToRaw.py - AWS Lambda](#)

Three scripts have been developed as listed above for the given task, the scripts have been deployed in AWS Lambda where distributed compute is not required and AWS Glue ETL - where there was a need to run the PySpark jobs to leverage distributed compute.

Files Organization - S3 Bucket

A S3 bucket named *stock-analysis-praghadeesh* was created and the following directory structure was established

- Raw - the raw data from github is ingested
- Base - the schema is standardized and ingested as parquet
- Reporting - the tables where the business aggregations are done and stored
- Logs - where the logs are stored



The flow of the pipeline is as below

- Raw Data Ingestion (Ingesting from Github to S3 - Raw directory)
- Base Data Ingestion (Ingesting and doing minor transformations into S3 - Base directory)
- Reporting Ingestions (Applying business aggregations and writing into S3 - Reporting)

Raw Data Ingestion - AWS Lambda

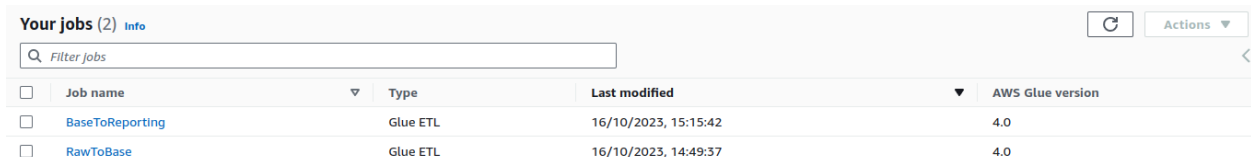
The AWS Lambda function with [GithubToRaw.py - AWS Lambda](#) script was used to ingest the Raw Data from Github and store the files in the S3 Storage. Lambda was chosen here as there wasn't a need for a distributed computer as there are no aggregations that are required to be performed.

Base Data Ingestion - AWS Glue

To ingest the Raw data and to do some basic transformation, we leveraged the use of PySpark on Top of the AWS Glue ETL Jobs with the script [RawToBase.py - AWS Glue](#). This writes the data as a parquet file into the base directory of the S3 bucket, at this stage the data still is not aggregated as per the business requirements.

Reporting Data Ingestion - AWS Glue

The business aggregations as mentioned in the requirements are applied on top of the base data, and performed efficiently by applying some of the Spark Concepts and optimization techniques and the same is stored as three different tables in parquet format within the reporting directory using the script [BaseToReporting.py - AWS Glue](#).

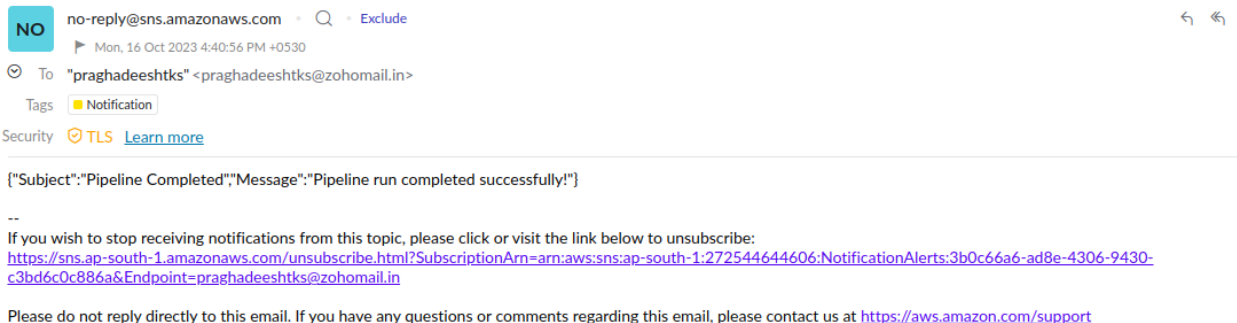


<input type="checkbox"/>	Job name	Type	Last modified	AWS Glue version
<input type="checkbox"/>	BaseToReporting	Glue ETL	16/10/2023, 15:15:42	4.0
<input type="checkbox"/>	RawToBase	Glue ETL	16/10/2023, 14:49:37	4.0

Notification - AWS SNS

At each step getting data from github to raw, raw to base and base to reporting - all of these stages are checked if they are successfully completed and notified to the subscription list of a certain topic using AWS SNS.

AWS Notification Message



The Pipeline - AWS Step functions

The orchestration of Lambda and PySpark glue jobs are orchestrated with AWS Step functions and at each step a conditional and exception catching is provided to alert the subscribers in the SNS service for a specific topic.

Querying the Data - AWS Athena

The data is crawled with Glue Data Catalog Crawler over the base and reporting directories in the S3 Bucket, and is queried using AWS Athena. The AWS Athena query results are displayed as below

Query results		Query stats					
Completed		Time in queue: 108 ms Run time: 373 ms Data scanned: 0.59 KB					
Results (7)		<div> <div>Search rows</div> <div> <div>Copy</div> <div>Download results</div> </div> </div>					
#	sector	avg_open_price	avg_close_price	max_high_price	min_low_price	avg_volume	
1	LIFE SCIENCES	45.30218615437486	45.309679145671055	179.57	2.8	1772380.0983737975	
2	ENERGY & TRANSPORTATION	17.562800199046094	17.55603048785082	104.59	0.198	4669611.431967552	
3	TECHNOLOGY	72.61458867796536	72.61408919773959	705.07	0.09	1.149873897677966E7	
4	TRADE & SERVICES	58.672824213892305	58.68767543080501	104.9	24.87	2475920.9017587495	
5	FINANCE	19.604653790169134	19.610372925764285	41.45	7.2	10065.089239726794	
6	REAL ESTATE & CONSTRUCTION	10.176505082417574	10.168689285714283	18.42	5.2867	212976.1813186813	
7	MANUFACTURING	32.826966348273345	32.84054371785455	92.32	5.16	5396276.952240999	

The tables that can be queries are

- All_time_summary - Reporting
- Specifc_time_detailed - Reporting
- Speicifc_time_summary - Reporting
- Stock_data - Base
- Symbol_data - Base

Data source

AwsDataCatalog

Database

stock-db

Tables and views

Create

Filter tables and views

▼ Tables (5)

all_time_summary

specific_time_detailed

specific_time_summary

stock_data

symbol_data

Performance Optimization

- Persisted the dataframe with persist() API (by default MEMORY_AND_DISK_DESER is used as no StorageLevel is explicitly mentioned) to avoid the transformation being executed from start every time an action is triggered on the dataframe

```
# Persisting the dataframe so that the same transformations doesn't need to be re executed everytime an operation is performed on the DF
joined_data.persist()
```

- Broadcasted the smaller dataframe - the symbol data (smaller DF) is shared across all nodes to avoid shuffling of data across nodes with broadcast join thus reducing the

network I/O

```
# Broadcasting the Smaller DF - Symbol Data so that the data is stored in all the worker nodes and reduces shuffle operation across nodes reducing Network I/O
joined_data = (
    stock_data_cleaned.join(broadcast(symbol_data), on="stock_symbol", how="left")
)
```

- Used parquet file format as it is columnar to increase the performance of queries
- The data is partitioned and written as parquet file, thus joining the data should be faster and reduces the amount of data that needs to be shuffled across different nodes

```
# Stock Data and Symbol Data is written as Parquet
```

```
stock_data_cleaned.write.partitionBy("stock_symbol").format("parquet").mode("overwrite").save("s3://stock-analysis-praghadeesh/base/processed/stock_data")
symbol_data.write.format("parquet").mode("overwrite").save("s3://stock-analysis-praghadeesh/base/processed/symbol_data")
```

Data Observation

- The given data is in CSV format and both the Stock and Symbol informations are mixed up
- The stock data contains open, low, high, close, stock_symbol and timestamp
- The symbol data contains stock_symbol, Name, Country, Sector, Industry, Address
- The data doesn't have any nulls
- The timeline of the data available for each stock is varying and is not the same, providing us an assumption the stock is listed in a later point of time
- The stocks were found to have the same open, high, close and low prices but had some trading volumes, while this is rare there's still a possibility for Round-Lot Transactions or extremely illiquid stocks.

Miscellaneous

S3 Bucket

Name: stock-analysis-praghadeesh

Location: ap-south-1

Access: Bucket and objects are not public

AWS Lambda

Name: IngestFromGithub

Runtime: Python 3.11

Timeout: 1 min

Memory: 128 MB

Ephemeral Storage: 512 MB

AWS Glue

Name: RawToBase and BaseToReporting

Runtime: Python 3.11, Spark 3.3

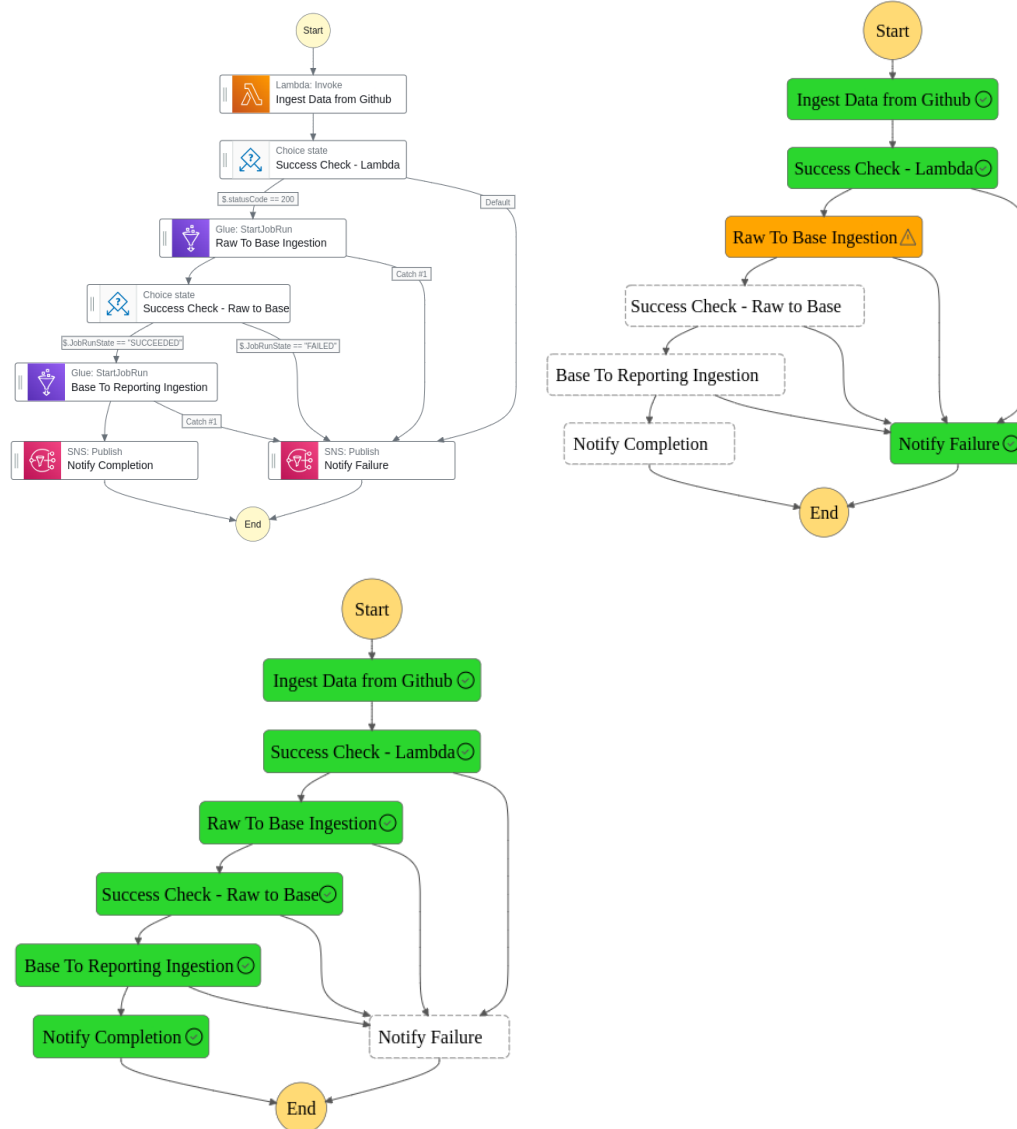
Worker Type: G1x (4 vCPU and 16 GB Memory)

No of Workers: 2

Memory: 128 MB

Ephemeral Storage: 512 MB

A Working flow of the fully functional pipeline - AWS Step Functions



I hope this document covers all the details, please do see the codes in GitHub Repo. In case of any questions, feel free to reach out to me.

As someone who has been working predominantly in Azure Ecosystem, the learning curve to get adapted to use AWS was pretty significant. Looking forward to expand my learning more 😊

Thanks,
Praghadeesh
praghadeeshtks@zohomail.in
+91-9688260611