**Data Processing Pipeline: Architecture & Implementation Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No** | **Author** | **Date** | **Version** |
| 1 | Hitesh Sonawane | 12 Mar 2025 | v.1.1 |
|  |  |  |  |

**Contents**

Introduction……………………………………………………………………………………………………………………………………2

Project Resources …………………………………………………………………………………………………………………………..2

Architecture Components ………………………………………………………………………………………………………………3

Best Practices & Considerations ...………………………………………………………………………………………………….4

**Security Best Practices …………………………………………………………………………………………………………………..4**

Future Enhancements (Can be) ………………………………………………………………………………………………….…..4

Conclusion ……………………………………………………………………………………………………………………………………..4

**Introduction**

This document provides a concise summary of the overall architecture and implementation of our data processing pipeline, which ingests, validates, and transforms financial data from raw CSV files to partitioned Parquet files in an S3-based data lake.

This pipeline ingests raw CSV files from S3, processes and validates the data via AWS Glue (using PySpark), writes the processed, partitioned data back to S3 in Parquet format, and creates an external table over the processed data using an AWS Glue Crawler. In addition, the pipeline sends email notifications with key job metrics via Amazon SNS. The infrastructure is deployed using Terraform (organized via modules) and automated via GitHub Actions.

Few features of this data pipeline are:

* Python, pyspark based data pipeline
* Automated test case validation using python pytest module
* AWS based data lake architecture
* Uses terraform for AWS resource creation
* GitHub as repository system with branching strategy and approval process.
* CICD using GitHub action for automated test case validations, terraform resource creation.
* AWS Glue as execution engine with spark capabilities.

Project Resources:

* **GitHub Repository**: <https://github.com/hiteshsonawane007/ohpen-etl-dataengineering>
* **AWS Account**
* **Tech stack Used:** Python, Pytest, GitHub, GitHub actions, YAML, Terraform, Gitignore
* **AWS Services Used:** IAM, AWSS Glue, Athena, S3, Glue Crawler.

Architecture Components:

1. **Data Storage (S3):**
   * Raw Data Bucket: ohpen-etl-raw-financial-data with a designated folder transactions/raw/ used for uploading CSV files.
   * Processed Data Bucket: ohpen-etl-processed-financial-data with a folder transactions/processed/ for storing processed Parquet files.
2. **ETL Processing (AWS Glue):**
   * Glue Job: Executes the ETL code packaged as a ZIP file (containing \_\_main\_\_.py, glue\_job.py, config.py, and utils.py).
   * The job reads data from the raw folder, applies data validation (e.g., filtering out rows with missing required fields or invalid currency codes), partitions the data by year and month, and writes the output to the processed folder.
   * Upon completion, the job sends an SNS alert summarizing metrics (total records processed, errors encountered).
3. **Data Catalog (Glue Crawler):**
   * Glue Crawler: Automatically crawls the processed data folder to create/update an external table (e.g., in the database etl\_database) that supports schema evolution.
   * Schema Evolution: The crawler is configured to update the table schema automatically if, for example, a new column (TransactionType) is added.
4. **Notification (SNS):**
   * SNS Topic: Created via Terraform with an email subscription (email address passed as a variable) to notify business users of job outcomes.
5. **Infrastructure & CI/CD (Terraform & GitHub Actions):**
   * Terraform Modules: The project is organized with modules for S3 buckets, SNS, Glue (Glue job and IAM roles), and Glue Crawler.
   * GitHub Actions: Automates testing, packaging (zips the ETL code), and deployment using Terraform.

**Implementation Details:**

* **Glue Job Code:**
  + Uses absolute imports (e.g., from etl import config) so that it functions correctly when packaged as a ZIP file.
  + The entry point is defined in \_\_main\_\_.py, which calls glue\_job.main().
  + Job parameters (input\_s3\_folder, output\_s3\_folder, sns\_topic\_arn) provided by the Glue job's default arguments enable flexible runtime configuration.
* **Terraform:**
  + S3 module creates buckets and optionally “dummy” folder objects (raw data folder is created; processed folder is left for Spark to manage).
  + SNS module provisions a topic and subscribes an email address (configured as a variable).
  + Glue module provisions the Glue job with appropriate IAM permissions (including S3 access, SNS publish, and CloudWatch Logs permissions).
  + A separate Glue Crawler module is used to create/update the external table over processed data, supporting schema evolution via an appropriate configuration.
* **CI/CD Packaging:**
  + The GitHub Actions workflow changes directory to the etl folder and packages the contents into a ZIP file containing a top‑level \_\_main\_\_.py and an etl/ subfolder if desired.
  + This ZIP file is uploaded to S3 and referenced by the Glue job.

**Best Practices & Considerations:**

* **Module Structure:** Organize Terraform code into modules to enhance reusability and maintainability.
* **IAM Permissions:** Follow the principle of least privilege in IAM policies for Glue jobs and crawlers.
* **Logging & Monitoring:** Enable CloudWatch Logs for Glue jobs, ensuring log groups can be created and written to.
* **Schema Evolution:** Use Glue Crawlers with a configuration (e.g., AddOrUpdateBehavior: "InheritFromTable") to automatically manage schema changes.
* **CI/CD:** Automate testing, packaging, and deployment via GitHub Actions; ensure the test environment (PYTHONPATH) matches production.
* **Alerting:** Use SNS to notify business users with clear, concise job run summaries.
* **Security Best Practices:**
  1. Maintain AWS Credentials in GitHub secrets.
  2. Deploy AWS Components in VPC
  3. Design IAM policies with least privileges.
  4. GitHub Repository best practices for branch and code commits.
  5. Follow appropriate Approval process once code committed to feature branch.

**Future Enhancements (Can be)**

* + Schedule Glue jobs for particular business requirement
  + Auto trigger AWS Glue jobs as per each file upload using S3 events and lambda
  + Data monitoring Setup
  + Data Quality Setup

**Conclusion**

This architecture provides a robust, automated solution for managing ETL pipelines in AWS, leveraging AWS Glue, S3, SNS, and Glue Crawlers. With Terraform-based infrastructure-as-code and integrated CI/CD, the system supports schema evolution, effective error handling, and real-time notifications—ensuring that business users and technical teams have up-to-date visibility of data processing outcomes.