Build a Financial Data Pipeline using AWS and PySpark

Business Overview

In the application of remittance and currency conversion, the normalization of values plays a crucial role in various industries, especially the banking sector. Most FinTech and bank sector companies often extend loans to individuals and businesses only after evaluating their creditworthiness based on rigorous analysis of past transactions and remittances.

Customers who are frequent travelers or engage in international transactions often perform banking activities such as deposits and withdrawals in different currencies like INR, USD, and SGD. This needs to be more consistent when assessing their financial health for services like lending. A comprehensive data pipeline integrates raw bank transaction data with real-time API exchange rates, and then maptrans actions to standardized currency values by date. This solution streamlines workflows which empowers organizations to focus on lending decisions rather than currency normalization challenges. By deploying the infrastructure of these data pipelines using AWS services into their platform, these organizations can drive insights rapidly and effectively for their business growth.

Key features:

1. **Data Extraction**: The implementation of API calls in a Docker container on EC2 ensures isolated, scalable, and consistent execution across environments.
2. **Central Location**: The Raw data, API data, and processed data are stored in a single location, an S3 bucket, which simplifies data monitoring and updates from one central point.
3. **Complex Workflow Handling**: The cloud-based pipeline efficiently manages complex data workflows, providing scalability, robustness, and seamless execution across multiple stages.
4. **Data Analysis**: The final transformed data is loaded into the S3 bucket and analyzed using SQL queries in Athena, which enables analysis within the cloud workspace for various outputs.

Aim:

The project aims to normalize raw bank transaction data into a single base currency, specifically Singapore Dollars (SGD), using AWS services. The primary goal is to transform customers' transaction values with precise reference to real-time exchange rates fetched from the Open Exchange API at the corresponding transaction dates. Once Docker is installed on an EC2 instance, it fetches real-time exchange rates from the Open Exchange API and uploads them to an S3 bucket alongside the bank transaction data. The detailed Spark transformations are run on an EMR cluster to calculate normalized transaction values for deposits and withdrawals. Afterward, the final processed data is stored in S3 as partitioned parquet files. A Glue Crawler is used to catalog this data, making it available for analysis using Athena queries. This pipeline ensures scalability and flexibility, allowing for the easy addition of new AWS services to enhance the process.

Tech Stack:

- **Programming:** SQL, Python
- **Library:** PySpark
- **Services:** AWS S3, AWS EC2, AWS EMR, AWS Glue, AWS Athena, Docker

AWS S3:

AWS S3, or Simple Storage Service, S3 Bucket is a versatile object storage solution provided by Amazon Web Services for securely storing and retrieving a wide range of data types, including semi-structured, unstructured, and structured data. It serves as a fundamental component for creating data lakes, which are repositories that store vast amounts of raw data in their native format for various analytical purposes.

AWS EC2:

Amazon EC2 (Elastic Compute Cloud) provides scalable virtual servers in the cloud, allowing users to rent computing resources without the need for physical hardware. It offers flexibility to choose specific configurations of power, memory, and storage based on project requirements. With EC2, applications, websites, and workloads can be deployed quickly and efficiently, with the ability to scale as needed. The service operates on a pay-as-you-go model, ensuring cost-effectiveness by charging only for the resources used.

AWS EMR:

AWS EMR (Elastic MapReduce) is a fully managed cloud service that simplifies running big data frameworks like Apache Hadoop, Apache Spark, and Apache Hive on scalable clusters of virtual machines. Integrated with the AWS ecosystem, EMR works seamlessly with services such as Amazon S3, EC2, and Redshift, making it suitable for big data analytics, ETL pipelines, and machine learning applications. EMR processes data in parallel across multiple EC2 instances, splitting datasets into chunks for efficient processing.

AWS Glue:

AWS Glue is a fully managed ETL service that automates infrastructure setup, orchestration, and optimization, making data integration seamless and efficient. It uses a serverless architecture that eliminates the need for managing servers, ensuring automatic scaling based on data size and processing complexity. The Glue Data Catalog acts as a central metadata repository, storing information about data sources, schemas, and transformations, which simplifies data discovery and organization. This centralized catalog integrates with tools like Athena and Redshift for easy querying and analysis. AWS Glue streamlines the ETL process, allowing organizations to focus on analytics without worrying about operational challenges.

AWS Athena:

Amazon Athena is a serverless interactive query service that allows users to analyze data stored in Amazon S3 using SQL, without the need to load it into a database. This makes it a cost-effective and efficient solution for querying large datasets. Athena integrates seamlessly with other AWS services, such as AWS Glue for metadata management, Amazon QuickSight for visualization, and AWS Lambda for automation, enabling a comprehensive data processing and analytics pipeline.

Docker:

Docker is a containerization platform that solves the **“it works on my machine”** problem. that packages applications and their dependencies into containers, ensuring seamless functionality across various environments such as development, testing, and production. It simplifies the process of creating, deploying, and running applications by encapsulating everything needed to execute the application within a container. For example, a web application developed with specific versions of libraries and

dependencies can be packaged into a Docker container. This container can then be run on any system, regardless of the underlying infrastructure, ensuring consistency.

Data Description:

In this project, there are two distinct data sources, each serving a different purpose. The first data source involves an API from Openexchangerates, which provides exchange rate data for various currencies. This API offers historical exchange rates with time-specific timestamps, allowing for the conversion of transaction amounts into a standard base currency, such as SGD. The second data source is a static file containing raw transaction data, which includes detailed information about each transaction made by the customer. The columns in this dataset include:

- **Account_ID**: A Unique identifier for every bank account.
- **Date**: The date when the transaction occurred.
- **Transaction_details**: Describes the nature of the transaction.
- **Chq_no**: The cheque number used, if applicable (often empty).
- **Value_date**: The date when the transaction is considered to have value in the account.
- **Withdrawal_amt**: The amount withdrawn in a given transaction.
- **Withdrawal_currency**: The currency in which the withdrawal was made.
- **Deposit_amt**: The amount deposited into the account.
- **Deposit_currency**: The currency in which the deposit was made.
- **Balance_amt**: The account balance after the transaction is processed.

Approach:

1. The Raw data was initially loaded into an S3 bucket, acting as a central location for all data used in the project.
2. An EC2 instance was created, and Docker was installed on it. The Dockerfile was executed within the container to call an API and retrieve currency exchange data from the Open Exchange Rates platform.
3. An EMR cluster was created to handle big data tasks, and SSH was executed to log into the EMR Cluster.
4. A Spark job was run on top of the EMR cluster to convert raw transaction data into Parquet format for optimized processing.

5. The PySpark transformations were performed on the raw data and API data to normalize the currency data to a single base currency (e.g., Singapore Dollar), and the final data was stored in the same S3 bucket.
6. A Glue crawler was created and ran on top of the final transformed S3 Data which populated the Glue Data Catalog. A set of Athena queries were executed to analyze different parameters based on the processed data and use case.

Key Takeaways:

- Understanding of Use Case and dataset
- Understanding of AWS Services and their applications
- Visualizing the complete Data Pipeline
- Connecting to an AWS EC2 instance via SSH
- Introduction and Installation of Docker on EC2 Instance
- Creation of a Docker Container from a Docker Image
- Processing API currency data within the container
- Creating a cluster on AWS EMR
- Data Processing to convert CSV data to Parquet format
- Perform transformations on the dataset using Pyspark
- Execution of AWS Glue Crawler on S3 data
- Querying of processed data using Athena

Note: We recommend monitoring AWS Services and deleting those that are not in use, as they can result in high costs.

Architecture Diagram: