Real-Time FX Normalization Pipeline using AWS EMR and Apache Spark

Author: Apoorv Mishra
Course: Cloud Computing
Date: 21 December 2025

Abstract

This paper presents a cloud-based solution for normalizing multi-currency financial transactions to a single base currency (USD) using AWS services. The system processes transaction data through an ETL pipeline built on AWS EMR with Apache Spark, stores results in Amazon S3, and visualizes insights through an interactive Streamlit dashboard. The project demonstrates how organizations can leverage cloud computing to handle currency conversion at scale, enabling better financial analysis and reporting. Key results show successful processing of 5,000+ transactions across 10 currencies with real-time dashboard updates and anomaly detection capabilities.

1. Introduction

1.1 Problem Statement

Global businesses deal with transactions in multiple currencies daily. A company operating in 10 countries might receive payments in EUR, GBP, INR, JPY, and other currencies. Without normalization, it's impossible to answer simple questions like "What was our total revenue this month?" or "Which product category generates the most income?"

Manual currency conversion is error-prone and doesn't scale. Exchange rates fluctuate daily, and historical transactions need to be converted using the rate that was valid on that specific date.

1.2 Why This Matters

- **Financial Reporting:** Companies need consolidated reports in a single currency
- **Decision Making:** Comparing performance across regions requires normalized data
- **Compliance:** Regulatory requirements often mandate reporting in local currency equivalents
- **Analytics:** Machine learning models and dashboards need consistent numerical scales

1.3 Our Solution

We built an automated pipeline that:

1. Ingests raw transaction data and daily FX rates
2. Joins transactions with the correct exchange rate based on date
3. Calculates USD-equivalent amounts
4. Outputs clean, normalized data for analysis
5. Displays results in an interactive dashboard

2. Architecture

2.1 System Overview

```
┌─────────────────┐      ┌─────────────────┐      ┌─────────────────┐
│   Data Sources  │      │   Processing    │      │    Output       │
│                 │      │                 │      │                 │
│  transactions   │─────▶│   AWS EMR       │─────▶│   Normalized    │
│  .csv           │      │  (Spark)        │      │  .parquet       │
│                 │      │                 │      │                 │
│  fx_rates       │─────▶│   PySpark       │─────▶│   Dashboard     │
│  .csv           │      │  Script         │      │  (Streamlit)    │
└─────────────────┘      └─────────────────┘      └─────────────────┘
        │                        │                        │
        └────────────────────────┴────────────────────────┘
                                 
                              Amazon S3
                     (apoorv-financial-pipeline-2025)
```

2.2 Components Explained

Amazon S3 (Storage)

- Acts as our data lake
- Stores raw CSV files in `/bronze/` folder
- Stores processed Parquet files in `/output/normalized/`
- Stores the PySpark script in `/scripts/`
- Why S3? It's cheap, scalable, and integrates well with EMR

AWS EMR (Processing)

- Managed Hadoop/Spark cluster
- We used EMR 7.2.0 with Spark 3.5
- Configuration: 1 master node + 2 core nodes (m5.xlarge instances)
- Handles the heavy lifting of data processing

- Auto-scales based on workload

PySpark Script (fx_normalization.py)

- Reads transaction CSV and FX rates CSV from S3
- Validates schemas (checks required columns exist)
- Extracts date from timestamp for joining
- Left-joins transactions with FX rates on currency + date
- Calculates `amount_usd = amount * rate_to_usd`
- Handles missing FX rates gracefully (flags them)
- Writes output as Parquet, partitioned by currency

Streamlit Dashboard

- Python-based web app
- Connects to S3 to read normalized data
- Provides filters for date, currency, product, channel
- Shows KPIs: total transactions, volume, averages
- Displays 7 interactive charts using Plotly
- Includes executive summary and anomaly detection

2.3 Data Flow

1. **Bronze Layer (Raw Data)**
    - `s3://apoorv-financial-pipeline-2025/bronze/transactions/transactions.csv`
    - `s3://apoorv-financial-pipeline-2025/bronze/fx/fx_rates.csv`
2. **Processing**
    - EMR cluster runs the Spark job
    - Spark reads both CSVs, joins them, calculates USD amounts
3. **Output Layer**
    - `s3://apoorv-financial-pipeline-2025/output/normalized/`
    - Data partitioned by currency for efficient querying
4. **Visualization**
    - Streamlit reads Parquet files from S3
    - Renders interactive dashboard

3. Implementation Details

3.1 Dataset Description

Transactions Dataset (5,000 records)

Column	Description
txn_id	Unique transaction identifier
customer_id	Customer reference
txn_ts	Transaction timestamp
amount	Original amount in local currency
currency	3-letter currency code (USD, EUR, etc.)
merchant_country	Country where transaction occurred
channel	ONLINE, POS, MOBILE, ATM, WIRE
product_type	ECOM, RETAIL, SUBSCRIPTION, etc.

FX Rates Dataset (900 records)

Column	Description
date	Rate effective date
currency	Currency code
rate_to_usd	Exchange rate to convert to USD

3.2 Key Code Snippet

The core join logic in PySpark:

```
# Extract date from transaction timestamp
txn_df = txn_df.withColumn("txn_date", to_date(col("txn_ts")))

# Join transactions with FX rates
df_joined = txn_df.join(
    fx_df,
    (txn_df.currency == fx_df.currency) & (txn_df.txn_date == fx_df.date),
    "left_outer"
)

# Calculate USD amount
df_final = df_joined.withColumn(
    "amount_usd",
    when(col("rate_to_usd").isNotNull(), col("amount") * col("rate_to_usd"))
    .otherwise(lit(None))
)
```

3.3 Challenges Faced

1. **EMR Instance Types:** m5.large wasn't available in our region, had to use m5.xlarge
2. **Schema Mismatch:** Initial script used `transaction_id` but actual data had `txn_id`
3. **Partitioned Data:** Dashboard had to handle Parquet files split by currency partition
4. **IAM Permissions:** Needed specific policies for EMR to access S3 and EC2

4. Results

4.1 Processing Performance

- **Dataset Size:** 5,000 transactions, 900 FX rate records
- **Processing Time:** ~2 minutes on EMR cluster
- **Output Size:** Parquet files totaling ~500KB (compressed)
- **Success Rate:** 100% transactions matched with FX rates

4.2 Dashboard Features

The dashboard successfully displays:

Feature	Description
KPI Cards	Total transactions, volume, average, unique customers
Bar Chart	Transaction volume by currency
Pie Chart	Transaction count distribution
Line Chart	Daily volume trends
Multi-line Chart	Currency-wise trends over time
Dual-axis Chart	Channel analysis (count vs volume)
Country Chart	Top merchant countries

4.3 Anomaly Detection

The system identifies:

- High-value transactions (>$50,000): flagged for review
- Very high-value (>$100,000): potential fraud indicators
- Anomaly rate calculated as percentage of total

4.4 Sample Output

```
Total Transactions: 5,000
Total Volume (USD): $2,847,293.45
Average Transaction: $569.46
Unique Customers: 499
Top Currency: USD (30% of volume)
Anomaly Rate: 2.3%
```

5. Related Work

This project builds upon several established concepts and technologies:

1. **Medallion Architecture (Databricks):** Our bronze/silver/gold layer approach is inspired by Databricks' medallion architecture for organizing data lakes.

2. **AWS EMR Best Practices:** We followed AWS documentation for EMR cluster sizing and S3 integration patterns.

3. **Streamlit Documentation:** Dashboard design patterns were adapted from Streamlit's gallery examples.

4. **Financial Data Processing:** Similar approaches are used by companies like Stripe and PayPal for multi-currency handling, though at much larger scale.

6. Conclusion and Future Work

6.1 Summary

We successfully built an end-to-end pipeline that:

- Ingests multi-currency transaction data
- Normalizes amounts to USD using date-appropriate exchange rates
- Stores results efficiently in Parquet format
- Visualizes insights through an interactive dashboard

The project demonstrates practical application of cloud computing concepts including distributed processing (Spark), managed services (EMR), object storage (S3), and modern visualization tools (Streamlit).

6.2 Future Improvements

1. **Real-time Processing:** Replace batch processing with Spark Streaming for live transactions
2. **More Currencies:** Expand from 10 to 50+ currencies
3. **ML Integration:** Add fraud detection models using transaction patterns
4. **Alerting:** Send email/Slack notifications for anomalies
5. **Cost Optimization:** Use Spot instances for EMR to reduce costs by 60-70%
6. **Historical Analysis:** Add year-over-year comparison features

7. References

[1] Amazon Web Services. "Amazon EMR Documentation." https://docs.aws.amazon.com/emr/

[2] Apache Spark. "PySpark Documentation." https://spark.apache.org/docs/latest/api/python/

[3] Streamlit. "Streamlit Documentation." https://docs.streamlit.io/

[4] Databricks. "Medallion Architecture." https://www.databricks.com/glossary/medallion-architecture

[5] AWS. "Best Practices for Amazon S3."
https://docs.aws.amazon.com/AmazonS3/latest/userguide/best-practices.html

Project Repository: Local development on Windows with AWS cloud backend
Dashboard URL: Hosted locally via Streamlit, accessible via Cloudflare tunnel for demo