Fintech Data Engineering ETL Pipeline

End-to-End Project Documentation

1. Project Overview

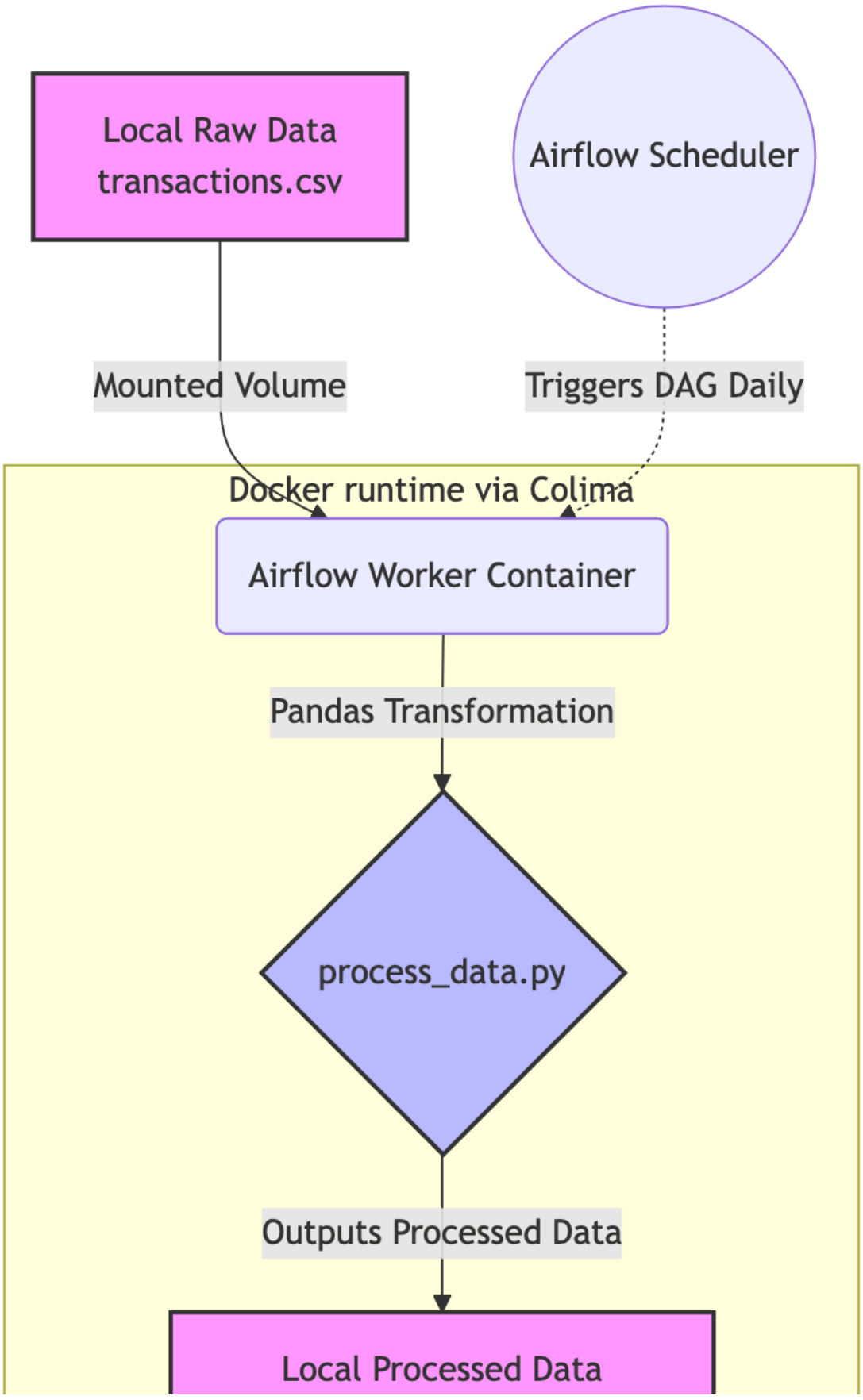
This project demonstrates a production-grade Data Engineering pipeline designed to extract, transform, and load (ETL) financial transaction data to detect potential fraud.

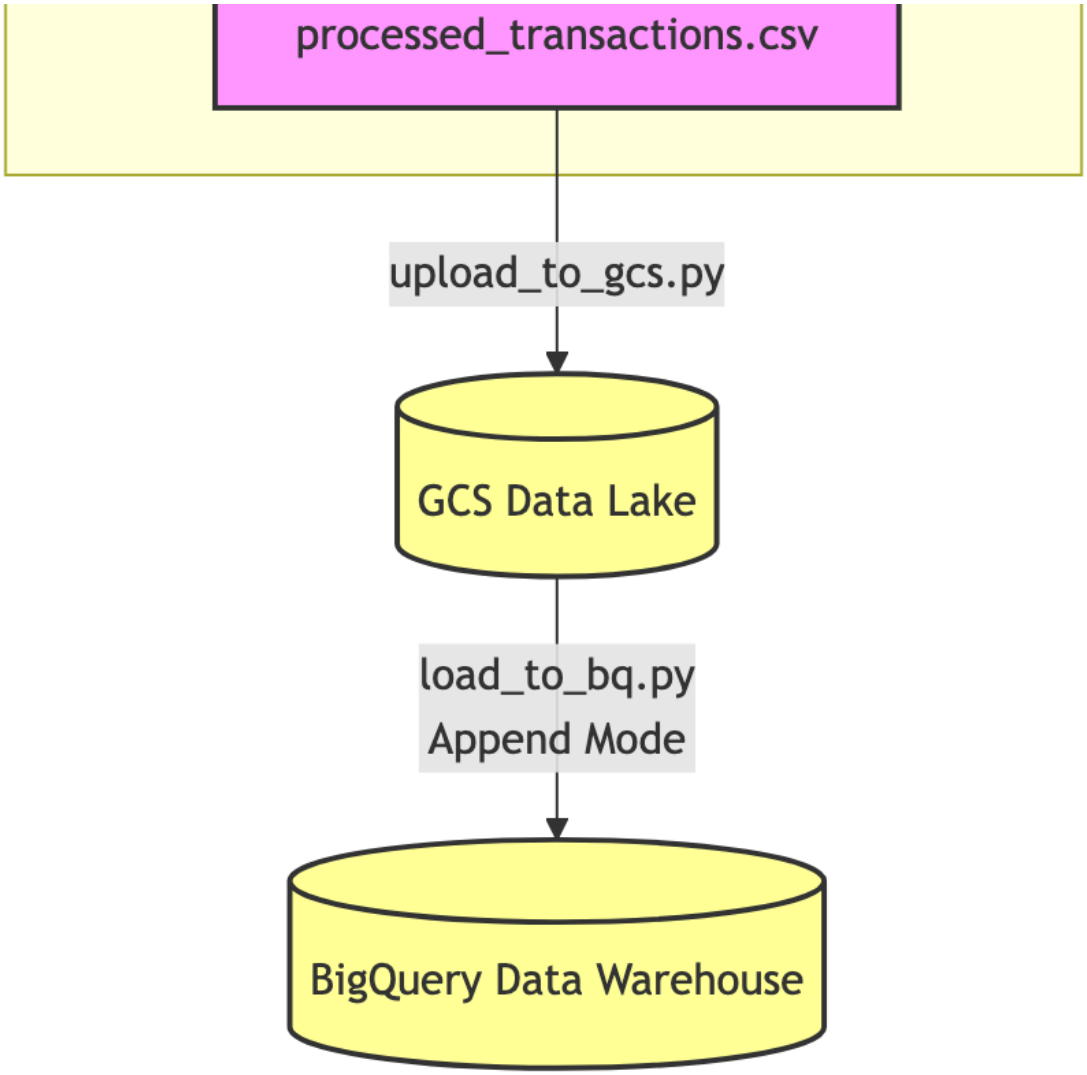
The entire environment is strictly containerized using Docker (via Colima on macOS) and orchestrated by Apache Airflow, ensuring cross-platform reproducibility and deployment readiness.

---

2. Architecture & Tech Stack

Architecture Flow


Local Raw Data
transactions.csv
Airflow Scheduler
Mounted Volume
Triggers DAG Daily
Docker runtime via Colima
Airflow Worker Container
Pandas Transformation
process_data.py
Outputs Processed Data
Local Processed Data



1. **Extract**: A local CSV financial dataset is mounted into the Airflow Worker container via Docker volumes.
2. **Transform**: A Python/Pandas script cleans the data, filters for high-value transactions, and applies fraud-flagging logic.
3. **Load (Data Lake)**: The processed data is pushed to a Google Cloud Storage (GCS) bucket, versioned by timestamp.
4. **Load (Data Warehouse)**: The new data is appended to a Google BigQuery table for downstream analytics and dashboarding.

Tech Stack

- **Containerization**: Docker CLI, Docker Compose, Colima (macOS runtime).
- **Orchestration**: Apache Airflow 2.9 (Dockerized, CeleryExecutor).
- **Processing**: Python 3.9+, Pandas.
- **Cloud Infrastructure**: Google Cloud Platform (GCS, BigQuery).

---

3. Pipeline Design (Airflow DAG)

The orchestration is defined in `dags/fintech_etl.py`. The DAG runs daily and consists of three sequential tasks:

1. `process_data`: Executes `scripts/process_data.py`. Reads raw data, applies Pandas transformations, outputs to `data/processed/`.
2. `upload_to_gcs`: Executes `scripts/upload_to_gcs.py`. Reads the processed CSV, appends a timestamp to the filename (e.g., `processed_transactions_20231025_1430.csv`), uploads it to GCS, and saves the generated filename to a local tracking text file.
3. `load_to_bq`: Executes `scripts/load_to_bq.py`. Reads the tracking text file to identify the latest GCS upload, and instructs BigQuery to append that specific file into the `transactions` table.

---

4. Setup & Execution Commands

Prerequisites

- Homebrew, Colima, Docker CLI.
- A GCP Project with a GCS Bucket and a BigQuery Dataset (`fintech_dataset`).
- A GCP Service Account JSON key saved at `config/gcp_key.json`.

Environment Configuration (`.env`)

```
AIRFLOW_UID=50000
GCS_BUCKET=your-bucket-name
GCP_PROJECT_ID=your-gcp-project-id
_AIRFLOW_WWW_USER_USERNAME=admin
_AIRFLOW_WWW_USER_PASSWORD=admin
```

Execution Steps

```
# 1. Start Docker runtime (macOS specific)
colima start --cpu 4 --memory 8

# 2. Add custom Python dependencies to Airflow
# (Requires Dockerfile with: RUN pip install pandas google-cloud-storage google-
```

```
cloud-bigquery)

# 3. Initialize Airflow Database (first run only)
docker-compose up airflow-init

# 4. Build custom image and start all services in detached mode
docker-compose up -d --build

# 5. Monitor logs (optional)
docker-compose logs -f airflow-worker
```

Accessing the UI

- Navigate to `http://localhost:8080`.
- Login with `admin` / `admin`.
- Unpause the `fintech_etl_pipeline` DAG and trigger it manually.

---

5. Directory Structure

```
Fintech_ETL/
├── dags/
│   └── fintech_etl.py         # Airflow DAG definition
├── data/
│   ├── raw/                   # Input Kaggle dataset
│   └── processed/             # Pandas output & tracking file
├── scripts/
│   ├── process_data.py        # Transformation logic
│   ├── upload_to_gcs.py       # Data Lake ingestion
│   └── load_to_bq.py          # Data Warehouse ingestion
├── config/
│   └── gcp_key.json           # Service Account Credentials (ignored in git)
├── logs/                      # Airflow task logs
├── plugins/                   # Custom Airflow plugins
├── Dockerfile                 # Custom Airflow image definition
├── docker-compose.yaml        # Multi-container orchestration
├── requirements.txt           # Python dependencies
├── .env                       # Environment variables
└── .gitignore                 # Git ignore rules
```