**Documentation for Ubuntu (Pyspark)**

This documentation provides a step-by-step guide to setting up and running the data generation pipeline on an Ubuntu machine using PySpark. The pipeline generates synthetic data, processes it through a Medallion Architecture, and schedules execution using Cron or Airflow.

1. **Prerequisites:**

Python 3.x installed.

PySpark installed.

Airflow installed (optional, for orchestration).

1. **Folder Structure:**

DE (data pipeline)

├── config.json

├── scripts

├ ├── generate\_data\_with\_config.py

├ ├── bronze\_layer.py

├ ├── silver\_layer.py

├ ├── gold\_layer.py

├ └── run\_data\_pipeline.sh

├── logs

├ └── pipeline\_log.txt

├── DAGs

├ └── data\_pipeline.py

├── output

├ └── raw\_files\_{yyyymmdd}\_{timestamp}

├ ├── raw\_customers\_{yyyymmdd}\_{timestamp}.csv

├ ├── raw\_orders\_{yyyymmdd}\_{timestamp}.csv

├ ├── raw\_order\_{yyyymmdd}\_items\_{timestamp}.csv

├ └── raw\_products\_{yyyymmdd}\_{timestamp}.csv

├── medallion

├ ├── bronze

├ ├ └── bronze\_files\_{yyyymmdd}\_{timestamp}

├ ├ ├── bronze\_customers\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── bronze\_orders\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── bronze\_order\_items\_{yyyymmdd}\_{timestamp}.csv

├ ├ └── bronze\_products\_{yyyymmdd}\_{timestamp}.csv

├ ├── silver

├ ├ └── silver\_files\_{yyyymmdd}\_{timestamp}

├ ├ ├──silver\_customers\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── silver\_orders\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── silver\_order\_items\_{yyyymmdd}\_{timestamp}.csv

├ ├ └── silver\_products\_{yyyymmdd}\_{timestamp}.csv

├ └── gold

├ ├ └── gold\_files\_{yyyymmdd}\_{timestamp}

├ ├ ├── gold\_customers\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── gold\_orders\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── gold\_order\_items\_{yyyymmdd}\_{timestamp}.csv

├ ├ └── gold\_products\_{yyyymmdd}\_{timestamp}.csv

└── tracker

└── id\_tracker.json

1. **Configuration File (config.json):**

{

"paths": {

"output\_folder": "C:/data\_pipeline/output",

"tracker\_file": "C:/data\_pipeline/tracker/id\_tracker.json"

},

"data\_settings": {

"num\_rows\_per\_table": 500

}

}

output\_folder: Directory where CSV files will be saved.

tracker\_file: Path to the ID tracker JSON file.

num\_rows\_per\_table: Number of rows to generate per table.

1. **Scripts:**

* **Data Generation (generate\_data\_with\_config.py):**

1. Read config.json for settings.
2. Generates synthetic data for customers, products, orders, and order\_items.
3. Updates id\_tracker.json with the latest IDs.
4. Outputs CSV files with a timestamped naming convention (e.g., raw\_customers\_20231025\_120000.csv).

* **Bronze Layer (bronze\_layer.py):**

1. Ingests raw CSV files from the output folder.
2. Adds metadata (e.g., ingest\_timestamp, source system).
3. Performs basic schema validation.

* **Silver Layer (silver\_layer.py):**

1. Deduplicates rows.
2. Enforces referential integrity.
3. Quarantines invalid rows.
4. Standardizes data types.

* **Gold Layer (gold\_layer.py):**

1. Creates curated tables for analytics.
2. Combines orders, order\_items, and products into a fact\_sales table.
3. Joins with customers for demographics.
4. Applies business rules (e.g., excludes CANCELLED orders).
5. **Scheduling:**

**Option 1: Using Cron job (run\_data\_pipeline.sh):**

* Create a shell script run\_data\_pipeline.sh.
* Add the script to the crontab.
* Add the date and timestamp to run the pipeline daily at midnight and in morning as per the requirement.

**Option 2: Using Airflow (run\_pipeline.py):**

* Create an Airflow DAG data\_pipeline.py:
* Place the DAG in the Airflow DAGs folder and trigger it.
* Check the Airflow flow to monitor.

1. **Logging:**

Logs are captured in pipeline\_log.txt for each script.

**Example log entry:**

[2023-10-25 12:00:00] INFO: Data generation completed. 500 rows added to customers.

1. **Data Validation Checks:**

* **Schema Integrity:** Ensure CSV files have the correct columns. **– Bronze layer.**
* **Non-Null & Positive:** Validate customer\_id, product\_id, order\_id, quantity, and price. **- Silver layer.**
* **Duplicate Handling:** Check for duplicate IDs. **- Silver layer.**
* **Referential Integrity:** Ensure order\_items.order\_id exists in orders, etc. **- Silver layer.**
* **Business Rule 1:** Exclude CANCELLED orders **- Gold layer.**
* **Business Rule 2:** Date Validation **- Gold layer.**
* **Business Rule 3:**

1. Create a fact\_sales table with product details (if order\_items exists) **- Gold layer.**
2. Create a fact\_sales simplified version (without product details) **- Gold layer.**

* **Sum Checks:** Ensures data integrity by confirming that the total amount in orders correctly reflects the sum of its corresponding order\_items. (This varies if partial or cancelled) **- Gold layer.**

1. **Next Steps**

Integrate with cloud storage (e.g., S3, GCS).

Add monitoring and alerting for pipeline failures.