**Documentation**

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 processing can be carried out using a shell script in the sequential order bronze → silver → gold.

1. **Prerequisites:**

Python 3.x installed.

PySpark installed.

1. **Folder Structure:**

Data (data pipeline)

├── scripts

├ ├── generate\_data\_with\_config.py

├ ├── bronze\_layer.py

├ ├── silver\_layer.py

├ ├── gold\_layer.py

├ ├── run.sh

├ ├── config.json

├ └── config2.ini

├── logs

├ └── pipeline\_log.txt

├── output

├ ├── raw\_customers

├ ├ └── {yyyymmdd}

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

├ ├── raw\_orders

├ ├ └── {yyyymmdd}

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

├ ├── raw\_order\_items

├ ├ └── {yyyymmdd}

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

├ └── raw\_products

├ └── {yyyymmdd}

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

├── medallion

├ ├── bronze

├ ├ ├── bronze\_customers

├ ├ ├ └── {yyyymmdd}

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

├ ├ ├── bronze\_orders

├ ├ ├ └── {yyyymmdd}

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

├ ├ ├── bronze\_order\_items

├ ├ ├ └── {yyyymmdd}

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

├ ├ └── bronze\_products

├ ├ └── {yyyymmdd}

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

├ ├── silver

├ ├ ├── silver\_customers

├ ├ ├ └── {yyyymmdd}

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

├ ├ ├── silver\_orders

├ ├ ├ └── {yyyymmdd}

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

├ ├ ├ └── quarantine\_orders

├ ├ ├ └── quarantine\_orders\_{yyyymmdd}\_{timestamp}.csv

├ ├ ├── silver\_order\_items

├ ├ ├ └── {yyyymmdd}

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

├ ├ ├ └── quarantine\_order\_items

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

├ ├ └── silver\_products

├ ├ └── {yyyymmdd}

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

├ └── gold

├ ├── gold\_fact\_sales

├ ├ └── {yyyymmdd}

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

├ └── gold\_fact\_sales\_simplified

├ └── {yyyymmdd}

├ └── gold\_fact\_sales\_simplified\_{yyyymmdd}\_{timestamp}.csv

└── tracker

└── id\_tracker.json

1. **Configuration Files :**

**config.json:**

{

"paths": {

"output\_folder": "/home/bsk/PycharmProjects/Data/output",

"tracker\_file": "/home/bsk/PycharmProjects/Data/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.

**config2.ini:**

[PATHS]

# Bronze Layer Paths

input\_path\_bronze\_processing\_layer = /home/bsk/PycharmProjects/Data/output/

output\_path\_bronze\_processing\_layer = /home/bsk/PycharmProjects/Data/medallion/bronze/

# Silver Layer Paths

input\_path\_silver\_processing\_layer = /home/bsk/PycharmProjects/Data/medallion/bronze/

output\_path\_silver\_processing\_layer = /home/bsk/PycharmProjects/Data/medallion/silver/

# Gold Layer Paths

input\_path\_gold\_processing\_layer = /home/bsk/PycharmProjects/Data/medallion/silver/

output\_path\_gold\_processing\_layer = /home/bsk/PycharmProjects/Data/medallion/gold/

#Logs Folder Path

logs\_folder = /home/bsk/PycharmProjects/Data/logs/pipeline\_log.txt

This is an initialization file (ini file) to have all our input and output files to get and put the files while processing. With this hardcoding of the paths can be avoided.

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. **Shell script:**

**run.sh:**

This Bash script automates the execution of a data pipeline, sequentially running Bronze, Silver, and Gold jobs. It ensures a strict environment using set -euo pipefail and includes logging for better tracking. The script:

* Validates if Python 3 is installed.
* Captures an optional date argument (defaults to today).
* Executes bronze\_layer.py, silver\_layer.py, and gold\_layer.py in order.
* Checks exit codes after each stage, stopping execution if any job fails.
* Logs progress and errors, ensuring a structured pipeline run.

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 (in case of null values, product details or also in case of no order\_items data) **- 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.