METCS777 – Term Paper Code Sample Documentation

Team members: Keerthi Uppalapati, Nidhi Nama

1. Environment Setup:

This project was developed using Apache Airflow, Docker, and VS Code. The following setup was used:

- Python 3 environment managed inside Airflow Docker containers.
- Airflow webserver, Scheduler, Worker and PostgreSQL services configured via docker-compose.yml.
- Data files mounted inside the Docker container at /opt/airflow/data.
- Development and testing conducted using VS Code with Docker integration enabled.

To setup the environment:

1. Install Docker and Docker Compose on your system.
2. Clone or create your Airflow project directory (airflow-docker/).
3. Place the provided DAG scripts inside dags.
4. Run docker compose up -d to start the Airflow environment.
5. Access the Airflow web UI at http://localhost:8081.

2. How to Run the Code

1. Ensure amazon.csv is located inside /opt/airflow/data.
2. Verify DAGs are listed in the Airflow UI: etl_amazon_reviews_dag and enhanced_xcom_demo_with_csv.
3. Trigger the DAG manually from the Airflow UI or via CLI: airflow dags trigger etl_amazon_reviews_dag.
4. Monitor progress in the Airflow web interface under Graph View or Logs.

3. Dataset Description

The dataset used is amazon.csv, containing customer review data for Amazon products. Typical columns include:

- product_id: Unique identifier for each product
- product_name: Name or title of the product
- category: Product category
- discounted_price: Price after any discounts applied
- actual_price: Original price before discount
- discount_percentage: Discount applied in percentage
- rating: Numeric rating given by the user
- rating_count: Total number of ratings the product has received
- about_product: Short description of the product
- user_id: Unique identifier of the reviewer
- user_name: Name of the reviewer
- review_id: Unique identifier for the review
- review_title: Title of the review
- review_content: Text content of the review

- img_link: URL to the product image

- product_link: URL to the product page on Amazon

This dataset enables analysis of average ratings per category and helps demonstrate ETL and data transformation tasks.

4. Code 1: ETL Amazon Reviews DAG

This DAG performs a traditional Extract-Transform-Load (ETL) process on the Amazon reviews dataset.
- Extract: Reads amazon.csv from the data directory.
- Transform: Calculates average ratings per product category.
- Load: Saves the transformed dataset to transformed_amazon_reviews.csv.

Code:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import pandas as pd

default_args = {
    'owner': 'Keerthi',
    'depends_on_past': False,
    'start_date': datetime(2025, 10, 28),
    'email_on_failure': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'etl_amazon_reviews_dag',
    default_args=default_args,
    description='ETL Workflow for Aggregating Amazon Product Reviews Data',
    schedule=timedelta(days=1),
    catchup=False,
)
```

```
DATA_DIR = '/opt/airflow/data'

def extract_data(**kwargs):
    file_path = f'{DATA_DIR}/amazon.csv'
    output_path = f'{DATA_DIR}/extracted_data.csv'
    df = pd.read_csv(file_path)
    df.to_csv(output_path, index=False)
    kwargs['ti'].xcom_push(key='extracted_data_path', value=output_path)

def transform_data(**kwargs):
    ti = kwargs['ti']
    input_path = ti.xcom_pull(task_ids='extract', key='extracted_data_path')
    df = pd.read_csv(input_path)
    df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
    avg_rating_by_category = df.groupby('category')['rating'].mean().reset_index()
    avg_rating_by_category.columns = ['Category', 'AverageRating']
    output_path = f'{DATA_DIR}/transformed_data.csv'
    avg_rating_by_category.to_csv(output_path, index=False)
    ti.xcom_push(key='transformed_data_path', value=output_path)

def load_data(**kwargs):
    ti = kwargs['ti']
    input_path = ti.xcom_pull(task_ids='transform', key='transformed_data_path')
    transformed_data = pd.read_csv(input_path)
    output_path = f'{DATA_DIR}/transformed_amazon_reviews.csv'
    transformed_data.to_csv(output_path, index=False)

extract = PythonOperator(task_id='extract', python_callable=extract_data, dag=dag)
transform = PythonOperator(task_id='transform', python_callable=transform_data,
dag=dag)
load = PythonOperator(task_id='load', python_callable=load_data, dag=dag)

extract >> transform >> load
```

5. Code 2: Enhanced XCom Demo DAG

This DAG demonstrates an advanced workflow where data is passed between tasks using Airflow's XCom mechanism.
It performs data cleaning, normalization, branching, and visualization based on data conditions.

Code:

```
from airflow import DAG

from airflow.operators.python import PythonOperator, BranchPythonOperator

from airflow.operators.empty import EmptyOperator

from datetime import datetime

import pendulum

import pandas as pd

import os

from airflow.utils.trigger_rule import TriggerRule



# Default arguments for the DAG

default_args = {

    'owner': 'airflow',

    'start_date': pendulum.now('UTC').subtract(days=1),

}


# Specify the correct file paths

csv_file_path = '/opt/airflow/data/amazon.csv'

processed_csv_path = '/opt/airflow/data/processed_amazon.csv'

visualization_path = '/opt/airflow/data/data_visualization.png'
```

```
# Define the DAG
with DAG(
    dag_id='enhanced_xcom_demo_with_csv',
    default_args=default_args,
    schedule=None,  # Replace schedule_interval with schedule
    catchup=False,
) as dag:

    # Task 1: Read data from CSV and push it to XCom
    def read_csv_data(**kwargs):
        from airflow.utils.log.logging_mixin import LoggingMixin
        logger = LoggingMixin().log

        if not os.path.exists(csv_file_path):
            logger.error(f"CSV file not found at {csv_file_path}")
            raise FileNotFoundError(f"CSV file not found at {csv_file_path}")

        df = pd.read_csv(csv_file_path)
        logger.info(f"Read {len(df)} rows from {csv_file_path}")

        # Push the DataFrame to XCom (converted to JSON)
        kwargs['ti'].xcom_push(key='raw_data', value=df.to_json())

    read_data = PythonOperator(
        task_id='read_csv_data',
        python_callable=read_csv_data,
```

```
)

# Task 2: Clean the data
def clean_data(**kwargs):
    from airflow.utils.log.logging_mixin import LoggingMixin
    logger = LoggingMixin().log

    ti = kwargs['ti']
    raw_data_json = ti.xcom_pull(key='raw_data', task_ids='read_csv_data')
    df = pd.read_json(raw_data_json)

    # Data cleaning steps (e.g., drop rows with missing values)
    df_cleaned = df.dropna()
    logger.info(f"Dropped {len(df) - len(df_cleaned)} rows with missing values")

    # Push the cleaned DataFrame to XCom
    ti.xcom_push(key='cleaned_data', value=df_cleaned.to_json())

clean_data = PythonOperator(
    task_id='clean_data',
    python_callable=clean_data,
)

# Task 3: Transform the data
def transform_data(**kwargs):
    from airflow.utils.log.logging_mixin import LoggingMixin
```

```python
    logger = LoggingMixin().log

    ti = kwargs['ti']
    cleaned_data_json = ti.xcom_pull(key='cleaned_data', task_ids='clean_data')
    df = pd.read_json(cleaned_data_json)

    # Data transformation steps (e.g., normalize numerical columns)
    numerical_cols = df.select_dtypes(include='number').columns
    if not numerical_cols.empty:
        df[numerical_cols] = df[numerical_cols].apply(lambda x: (x - x.mean()) / x.std())
        logger.info(f"Normalized numerical columns: {list(numerical_cols)}")
    else:
        logger.warning("No numerical columns found to normalize")

    # Push the transformed DataFrame to XCom
    ti.xcom_push(key='transformed_data', value=df.to_json())

transform_data = PythonOperator(
    task_id='transform_data',
    python_callable=transform_data,
)

# Task 4: Decide whether to visualize data based on a condition
def decide_next_step(**kwargs):
    from airflow.utils.log.logging_mixin import LoggingMixin
    logger = LoggingMixin().log
```

```
        ti = kwargs['ti']
        transformed_data_json = ti.xcom_pull(key='transformed_data',
task_ids='transform_data')
        df = pd.read_json(transformed_data_json)

        # If the dataset has more than 100 rows, proceed to visualization
        if len(df) > 100:
            logger.info("Data has more than 100 rows, proceeding to visualization")
            return 'visualize_data'
        else:
            logger.info("Data has 100 or fewer rows, skipping visualization")
            return 'skip_visualization'

    branching = BranchPythonOperator(
        task_id='branching',
        python_callable=decide_next_step,
    )

    # Task 5a: Visualize the data
    def visualize_data(**kwargs):
        from airflow.utils.log.logging_mixin import LoggingMixin
        logger = LoggingMixin().log

        ti = kwargs['ti']
        transformed_data_json = ti.xcom_pull(key='transformed_data',
task_ids='transform_data')
```

```
    df = pd.read_json(transformed_data_json)

    # Set Matplotlib backend to 'Agg' before importing pyplot
    import matplotlib
    matplotlib.use('Agg')
    import matplotlib.pyplot as plt

    # Create a simple plot (e.g., histogram of a numerical column)
    numerical_cols = df.select_dtypes(include='number').columns
    if not numerical_cols.empty:
        plt.figure(figsize=(10, 6))
        df[numerical_cols[0]].hist()
        plt.title(f'Histogram of {numerical_cols[0]}')
        plt.xlabel(numerical_cols[0])
        plt.ylabel('Frequency')
        plt.tight_layout()
        plt.savefig(visualization_path)
        plt.close()
        logger.info(f"Visualization saved at {visualization_path}")
    else:
        logger.warning("No numerical columns available for visualization")
        # Optionally, you can create a placeholder image or skip saving

visualize_data = PythonOperator(
    task_id='visualize_data',
    python_callable=visualize_data,
```

```
    )

    # Task 5b: Skip visualization
    skip_visualization = EmptyOperator(
        task_id='skip_visualization',
    )

    # Task 6: Save processed data to CSV
    def save_processed_data(**kwargs):
        from airflow.utils.log.logging_mixin import LoggingMixin
        logger = LoggingMixin().log
        ti = kwargs['ti']
        transformed_data_json = ti.xcom_pull(key='transformed_data',
task_ids='transform_data')
        df = pd.read_json(transformed_data_json)

        # Save the DataFrame to a CSV file
        df.to_csv(processed_csv_path, index=False)
        logger.info(f"Processed data saved at {processed_csv_path}")



    save_data = PythonOperator(
        task_id='save_processed_data',
        python_callable=save_processed_data,
    )
```

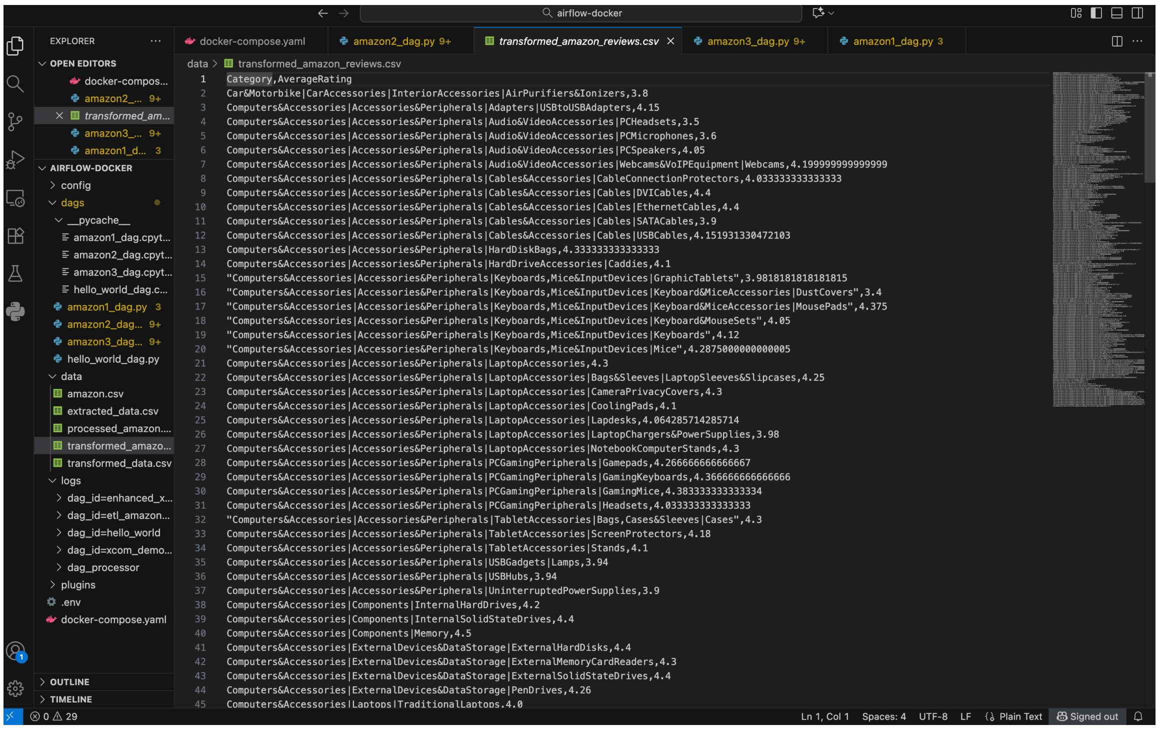
```
    # Task 7: End task
    end = EmptyOperator(
        task_id='end',
    )
    join_branches = EmptyOperator(
        task_id='join_branches',
        trigger_rule=TriggerRule.NONE_FAILED_MIN_ONE_SUCCESS  # Runs if at least one branch succeeded
    )


    # Define the task dependencies
    read_data >> clean_data >> transform_data >> branching
    branching >> visualize_data >> join_branches
    branching >> skip_visualization >> join_branches
    join_branches >> save_data >> end
```

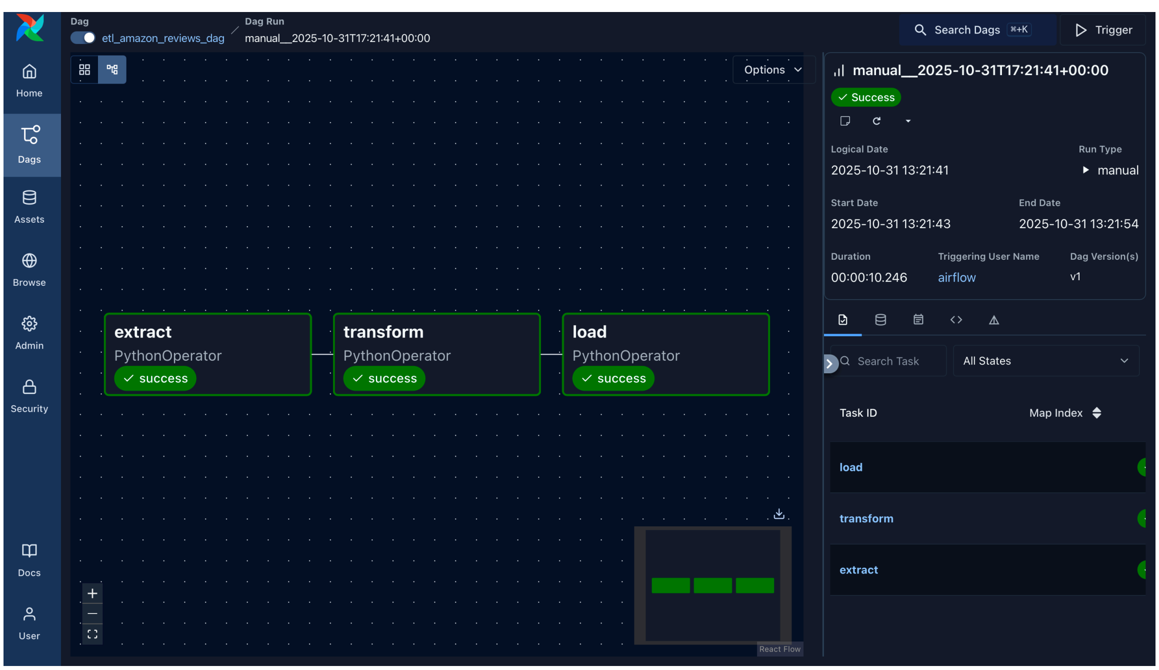
The DAG uses the amazon.csv file, performs transformations, and visualizes data if it contains more than 100 rows.

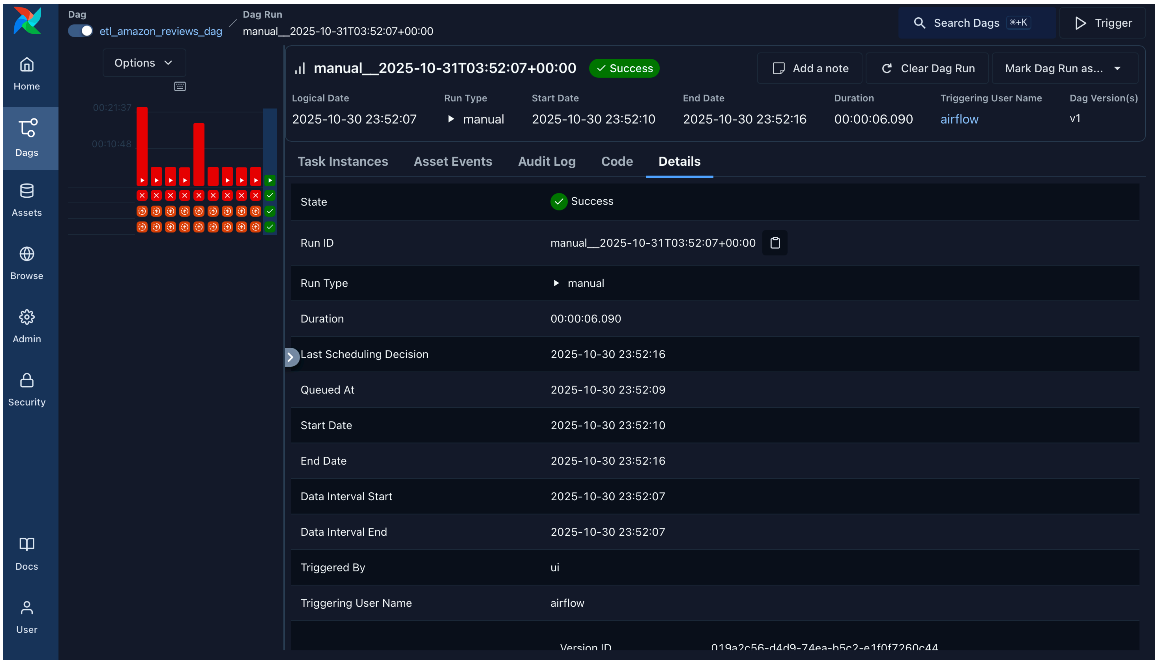
6. Results and Observations

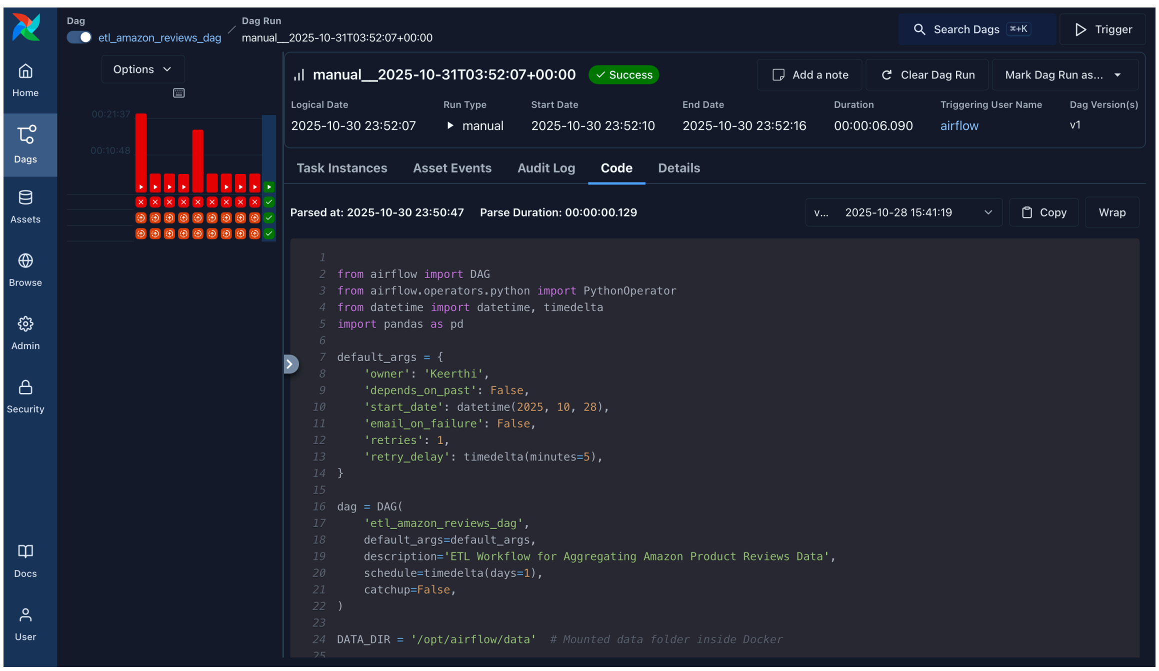
After executing both DAGs:

- The ETL DAG outputs transformed_amazon_reviews.csv containing the average rating per product category.

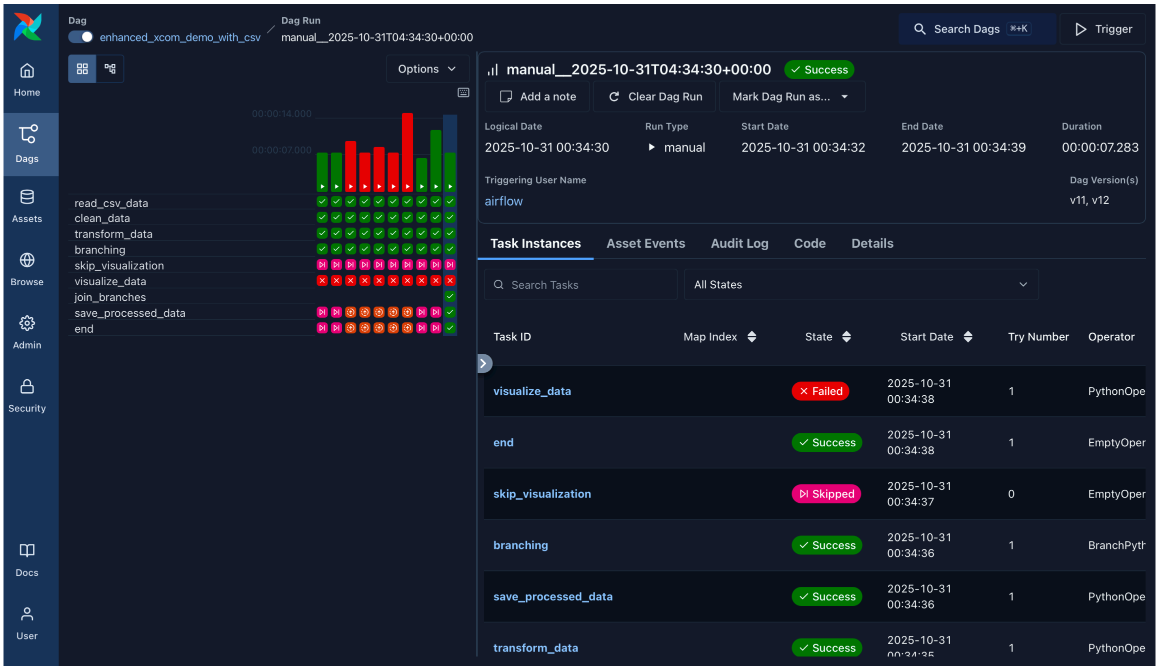




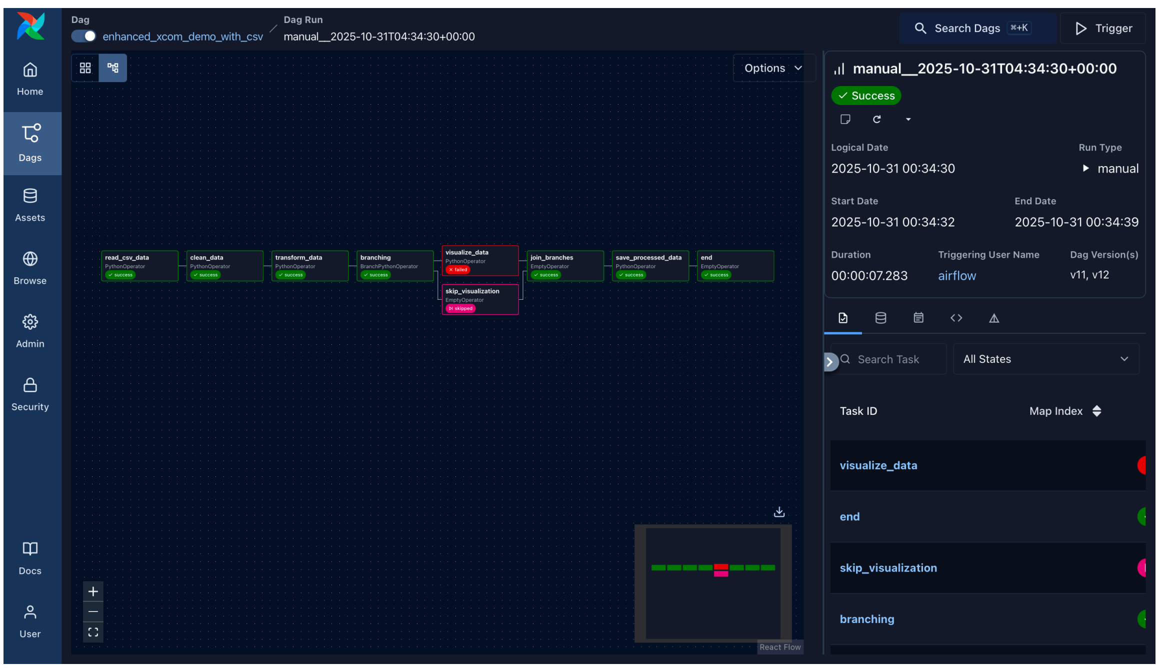




- The XCom DAG outputs processed_amazon.csv and data_visualization.png (if applicable).

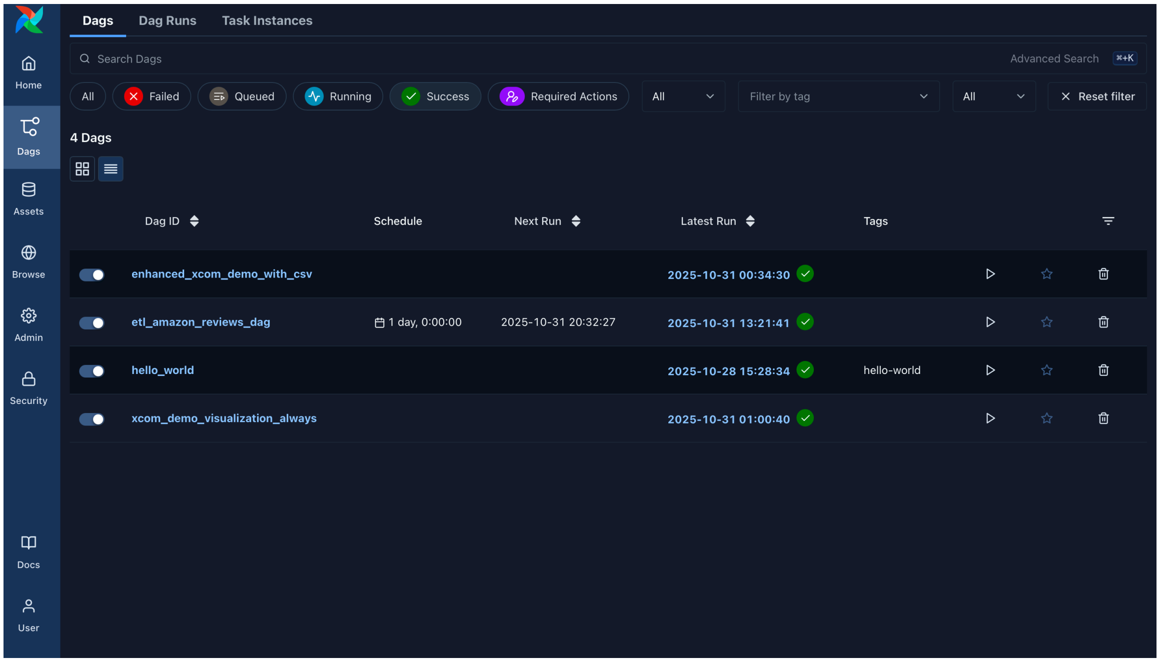


Dag
enhanced_xcom_demo_with_csv
Dag Run
manual__2025-10-31T04:34:30+00:00
Search Dags
Trigger
Home
Dags
Assets
Browse
Admin
Security
Docs
User
Options
read_csv_data
clean_data
transform_data
branching
skip_visualization
visualize_data
join_branches
save_processed_data
end
manual__2025-10-31T04:34:30+00:00
Success
Add a note
Clear Dag Run
Mark Dag Run as...
Logical Date
2025-10-31 00:34:30
Run Type
manual
Start Date
2025-10-31 00:34:32
End Date
2025-10-31 00:34:39
Duration
00:00:07.283
Triggering User Name
airflow
Dag Version(s)
v11, v12
Task Instances
Asset Events
Audit Log
Code
Details
Search Tasks
All States
Task ID
Map Index
State
Start Date
Try Number
Operator
visualize_data
Failed
2025-10-31 00:34:38
1
PythonOpe
end
Success
2025-10-31 00:34:38
1
EmptyOper
skip_visualization
Skipped
2025-10-31 00:34:37
0
EmptyOper
branching
Success
2025-10-31 00:34:36
1
BranchPyth
save_processed_data
Success
2025-10-31 00:34:36
1
PythonOpe
transform_data
Success
2025-10-31
1
PythonOpe



Dag
enhanced_xcom_demo_with_csv
Dag Run
manual__2025-10-31T04:34:30+00:00
Search Dags
Trigger
Home
Dags
Assets
Browse
Admin
Security
Docs
User
Options
manual__2025-10-31T04:34:30+00:00
Success
Logical Date
2025-10-31 00:34:30
Run Type
manual
Start Date
2025-10-31 00:34:32
End Date
2025-10-31 00:34:39
Duration
00:00:07.283
Triggering User Name
airflow
Dag Version(s)
v11, v12
Search Task
All States
Task ID
Map Index
visualize_data
end
skip_visualization
branching
React Flow

- Logs confirm successful task execution with details of data cleaning and transformation steps.



7. Conclusion

This project demonstrates a complete data pipeline setup using Apache Airflow in Docker.
The ETL and XCom DAGs show how Airflow can automate data workflows, manage dependencies, and enable reproducible data processing at scale.