ETL Project Documentation — Movies Data ETL using Python & Docker

Project Overview

This project involves setting up an ETL (Extract, Transform, Load) pipeline for movie data using Docker, Google Cloud Pub/Sub emulator, and Python scripts. The primary goal is to process and analyze movie rating data, leveraging containerization and Pub/Sub for efficient data handling and processing.

Detailed Explanation

1. Environment Setup

setup_env.bat

```
@echo off
py -m venv venv
call venv\Scripts\activate
pip install numpy pandas google-cloud-pubsub docker
echo Environment setup completed.
```

- py -m venv venv: Creates a virtual environment named venv to manage dependencies.
- call venv\Scripts\activate: Activates the virtual environment.
- pip install numpy pandas google-cloud-pubsub docker: Installs the required Python libraries:
  - numpy and pandas for data manipulation.
  - google-cloud-pubsub for interacting with Google Pub/Sub.
  - docker for Docker client interactions.
- echo Environment setup completed.: Provides feedback that the setup is complete.

Why use a virtual environment?

- To isolate project dependencies from the global Python environment and avoid conflicts.

2. Data Download

download_data.bat

```
@echo off
REM Create the data directory if it doesn't exist
if not exist "..\data" mkdir ..\data

echo Downloading MovieLens dataset...
curl -o ..\data\ml-100k.zip https://files.grouplens.org/datasets/movielens/ml-100k.zip
echo Unzipping dataset...
powershell -command "Expand-Archive -Path '..\data\ml-100k.zip' -DestinationPath '..\data\'"
echo Data download and extraction complete.
```

- If not exist "..\data" mkdir ..\data: Checks if the data directory exists; if not, it creates it.
- curl -o ..\data\ml-100k.zip https://files.grouplens.org/datasets/movielens/ml-100k.zip: Downloads the MovieLens dataset zip file.
- powershell -command "Expand-Archive -Path '..\data\ml-100k.zip' -DestinationPath '..\data/'": Unzips the dataset using PowerShell.

Why download and unzip the dataset?

- To prepare the raw data for further processing.

3. Dockerfile

```
Use the official Python image from the Docker Hub
 FROM python:3.12.5-slim

 # Set the working directory in the container
 WORKDIR /app
  # Copy the current directory contents into the container at /app
COPY . /app

# Install the required libraries
RUN pip install --no-cache-dir numpy pandas google-cloud-pubsub scipy matplotlib seaborn subprocess

# Command to start the Pub/Sub emulator (if needed)
CMD ["gcloud", "beta", "emulators", "pubsub", "start", "--host-port=0.0.0.0:8085"]
```

- FROM python: 3.12.5-slim: Uses a lightweight Python image to build the Docker container.
- WORKDIR /app: Sets the working directory inside the container.
- COPY . /app: Copies all files from the host to the container.
- RUN pip install --no-cache-dir ...: Installs required libraries.
- CMD ["gcloud", "beta", "emulators", "pubsub", "start", "--host-port=0.0.0.0:8085"]: Starts the Pub/Sub emulator within the container.

Why use Docker?

- Ensures a consistent environment for running and testing your application.

4. Start Pub/Sub Emulator

start_emulator.py

```
import os
# Start the Pub/Sub emulator in a Docker container
os.system("docker run -d --name pubsub-emulator -p 8085:8085 google/cloud-sdk gcloud beta emulators pubsub
start --host-port=0.0.0.0:8085")
print("Pub/Sub emulator started.")
```

- os.system(...): Executes a command to start the Pub/Sub emulator in a Docker container.

Why use an emulator?

- Allows for local development and testing of Pub/Sub functionality without incurring cloud costs.

5. Build Docker Image

```
docker build -t movies-etl-image .
```

- Builds a Docker image named movies-etl-image based on the Dockerfile.

Why build an image?

- To package the application and its dependencies into a single unit for easy deployment.

6. Run Docker Container

```
docker run -d --name movies-etl-container -p 8085:8085 movies-etl-image
```

- Runs a Docker container named movies-etl-container based on the built image.

Why run a container?

- To execute the application in a controlled environment.

7. Create Topic and Subscription

create_topic_subscription.py

```
import os
from google.cloud import pubsub_v1

# Set up the environment variable to point to the local Pub/Sub emulator
os.environ["PUBSUB_EMULATOR_HOST"] = "localhost:8085"

# Set the project ID and other parameters
project_id = "test-project"
topic_id = "movies-topic"
subscription_id = "movies-subscription"

# Set up Pub/Sub client
publisher = pubsub_v1.PublisherClient()
subscriber = pubsub_v1.SubscriberClient()

# Create a topic
project_path = f"projects/{project_id}"
topic_path = publisher.topic_path(project_id, topic_id)
publisher.create_topic(name=topic_path)
print(f"Topic '{topic_id}' created.")

# Create a subscription
subscription_path = subscriber.subscription_path(project_id, subscription_id)
subscriber.create_subscription(name=subscription_path, topic=topic_path)
print(f"Subscription '{subscription_id}' created.")
```

- os.environ["PUBSUB_EMULATOR_HOST"] = "localhost:8085": Sets the environment variable to use the local Pub/Sub emulator.
- publisher.create_topic(...): Creates a Pub/Sub topic.
- subscriber.create_subscription(...): Creates a subscription for the topic.

Why create topics and subscriptions?

- To organize and manage data flows within the Pub/Sub system.

8. Extract CSV Files

process_data.py

```
import pandas as pd

# Define the path to the data files
ratings_file = 'D:/DEPI Data Engineer/0_projects/ETL_MOVIES/venv/data/ml-100k/u.data'
movies_file = 'D:/DEPI Data Engineer/0_projects/ETL_MOVIES/venv/data/ml-100k/u.item'

# Create a DataFrame for Movie Ratings
ratings = pd.read_csv(ratings_file, delimiter='\t', header=None, names=['user_id', 'item_id', 'rating', 
'timestamp'])

# Create a DataFrame for Movie Names
with open(movies_file, 'r', encoding="ISO-8859-1") as read_file:
    counter = 0
    movies_df = pd.DataFrame(columns=['item_id', 'movie_name', 'release_timestamp'])

    # Iterate through the lines in the file
    for line in read_file:
        fields = line.split('|')
        item_id, movie_name, release_timestamp = fields[0], fields[1], fields[2]
        movie_name = movie_name[0:len(movie_name) - len(' (1234)')]

```

```python
        line_data = [int(item_id), str(movie_name), release_timestamp]

        temp_df = pd.DataFrame(data=[line_data], columns=['item_id', 'movie_name', 'release_timestamp'])
        movies_df = pd.concat([temp_df, movies_df], ignore_index=True)

        counter += 1

    # Sort Values by item id
    movies_df.sort_values(by='item_id', ascending=True, inplace=True)

# Close file
read_file.close()

# Export to CSV
ratings.to_csv('D:/DEPI Data Engineer/0_projects/ETL_MOVIES/venv/data/ratings.csv', index=False)
movies_df.to_csv('D:/DEPI Data Engineer/0_projects/ETL_MOVIES/venv/data/movies.csv', index=False)
```

- pd.read_csv(...): Reads the movie ratings and movie names data into DataFrames.
- with open(...) as read_file: Reads movie names file and processes each line.
- pd.concat(...): Aggregates movie data into a single DataFrame.
- ratings.to_csv(...) and movies_df.to_csv(...): Saves the DataFrames as CSV files.

Why extract and save data?

- To prepare the raw data for preprocessing and analysis.

9. Preprocess Data

preprocessing_data.py

```python
import pandas as pd
import os
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Define paths for the existing files
ratings_csv_path = r'D:\DEPI Data Engineer\0_projects\ETL_MOVIES\venv\data\ratings.csv'
movies_csv_path = r'D:\DEPI Data Engineer\0_projects\ETL_MOVIES\venv\data\movies.csv'
merged_csv_path = r'D:\DEPI Data Engineer\0_projects\ETL_MOVIES\venv\data\full_data.csv'

# Load the datasets into DataFrames
ratings_df = pd.read_csv(ratings_csv_path)
movies_df = pd.read_csv(movies_csv_path)

full_data = pd.merge(ratings_df, movies_df, on='item_id')

# Save the merged DataFrame to a new CSV file
full_data.head(10)
full_data = full_data.sort_values(by='item_id', ascending=True)
full_data

# data preprocessing
### 1st step: check data:
full_data.info()
missing_percentage = full_data['release_timestamp'].isnull().mean() * 100
print(f"Missing percentage: {missing_percentage:.2f}%")
mode_timestamp = full_data['release_timestamp'].mode()[0]
full_data['release_timestamp'].fillna(mode_timestamp, inplace=True)
print(full_data['release_timestamp'].isnull().sum())

# Check outliers
def detect_outliers(data, column):
    z_scores = np.abs(stats.zscore(data[column]))
    return np.where(z_scores > 3)

sns.boxplot(x=full_data['rating'])

full_data['timestamp'] = pd.to_datetime(full_data['timestamp'], unit='s')
full_data['release_timestamp'] = pd.to_datetime(full_data['release_timestamp'], format='%d-%b-%Y')
```

```
full_data['datestamp'] = full_data['timestamp'].dt.date
full_data['time'] = full_data['timestamp'].dt.time
full_data = full_data.drop(columns=['timestamp'])

invalid_rows = full_data[full_data['datestamp'] < full_data['release_timestamp']]
unique_invalid_movies = invalid_rows['movie_name'].unique()
plt.figure(figsize=(15, 5))
sns.countplot(x=invalid_rows['movie_name'])
plt.xticks(rotation=90, fontsize=8)
plt.show()

len(full_data[full_data.duplicated()])
full_data.to_csv(merged_csv_path, index=False)
```

- pd.merge(...): Merges ratings and movie names DataFrames.
- full_data.info(): Displays info about the DataFrame to check for missing values and data types.
- full_data['release_timestamp'].fillna(mode_timestamp, inplace=True): Fills missing values in the release_timestamp column.
- detect_outliers(...): Function to detect outliers in data.
- sns.boxplot(...): Creates a boxplot to visualize outliers.
- pd.to_datetime(...): Converts timestamps to datetime objects.
- full_data.drop(columns=['timestamp']): Drops the original timestamp column.
- full_data[full_data['datestamp'] < full_data['release_timestamp']]: Identifies invalid rows where the release date is after the rating date.
- plt.figure(...): Creates a bar plot of invalid movies.
- full_data.to_csv(...): Saves the cleaned DataFrame to a CSV file.

Why preprocess data?

- To clean and prepare the data for analysis, ensuring consistency and quality.

10. Create Folder in Docker Container

```
mkdir -p /data
```

- Creates a /data directory in the running Docker container.
- Run this command using terminal of your container inside docker

Why create a directory in Docker?

- To store and access data files within the container.

11. Publish Data to Docker Container

publish_data.py

```
import subprocess

# Paths for the files and Docker container
file_path = r'D:\DEPI Data Engineer\0_projects\ETL_MOVIES\venv\data\full_data.csv'
container_id = '54fea1529a3c1360e7a9be6f5711b34b7cf6cb05955e99cba5aa59ffd1e6c95b'
container_path = '/data/full_data.csv'

def copy_to_container(file_path, container_id, container_path):
    try:
        result = subprocess.run(['docker', 'cp', file_path, f'{container_id}:{container_path}'],
                                stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

        if result.returncode == 0:
            print(f"Successfully copied {file_path} to {container_id}:{container_path}")
        else:
            print(f"Error copying file: {result.stderr}")
            return False
        return True
    except Exception as e:
        print(f"Exception occurred: {e}")
        return False

# Copy the CSV file to the Docker container
copy_to_container(file_path, container_id, container_path)
```

- subprocess.run(...): Executes a Docker command to copy the CSV file into the container.
- docker cp: Copies files between the host and Docker containers.

Why copy data to Docker?

- To make the processed data available inside the Docker container for further operations.

Tools and Alternatives

1. **Python & Libraries:**

- **pandas:** For data manipulation.
- **numpy:** For numerical operations.
- **google-cloud-pubsub:** For interacting with Pub/Sub.

2. **Docker:**

- For Containerization: Ensures environment consistency.

3. **Pub/Sub Emulator:**

- For Local Testing: Avoids cloud costs.

Sending Data: Messages vs. CSV

	Messages	**CSV Files**
Pros	Efficient for real-time processing; no need to handle large files.	Simpler to handle; compatible with many systems.
Cons	Requires parsing and structuring messages.	Simpler to handle; compatible with many systems.

Recommendation:

- For your current setup, sending data as CSV files might be simpler and more compatible with existing tools. Messages could be considered for real-time or streaming scenarios.
- prepare the data for analysis, ensuring consistency and quality.

References:

- Data Engineering Project — Movies Data ETL using Python & GCP
- Running GCP Pub/Sub Emulator on a Local Docker Environment

About The Author:

- **Name:** Nada Hamdy Fatehy
- **Email:** nadahamdy2172002@gmail.com
- **Phone:** +201207788890
- **Links:** LinkedIn | GitHub