Data Engineering use case

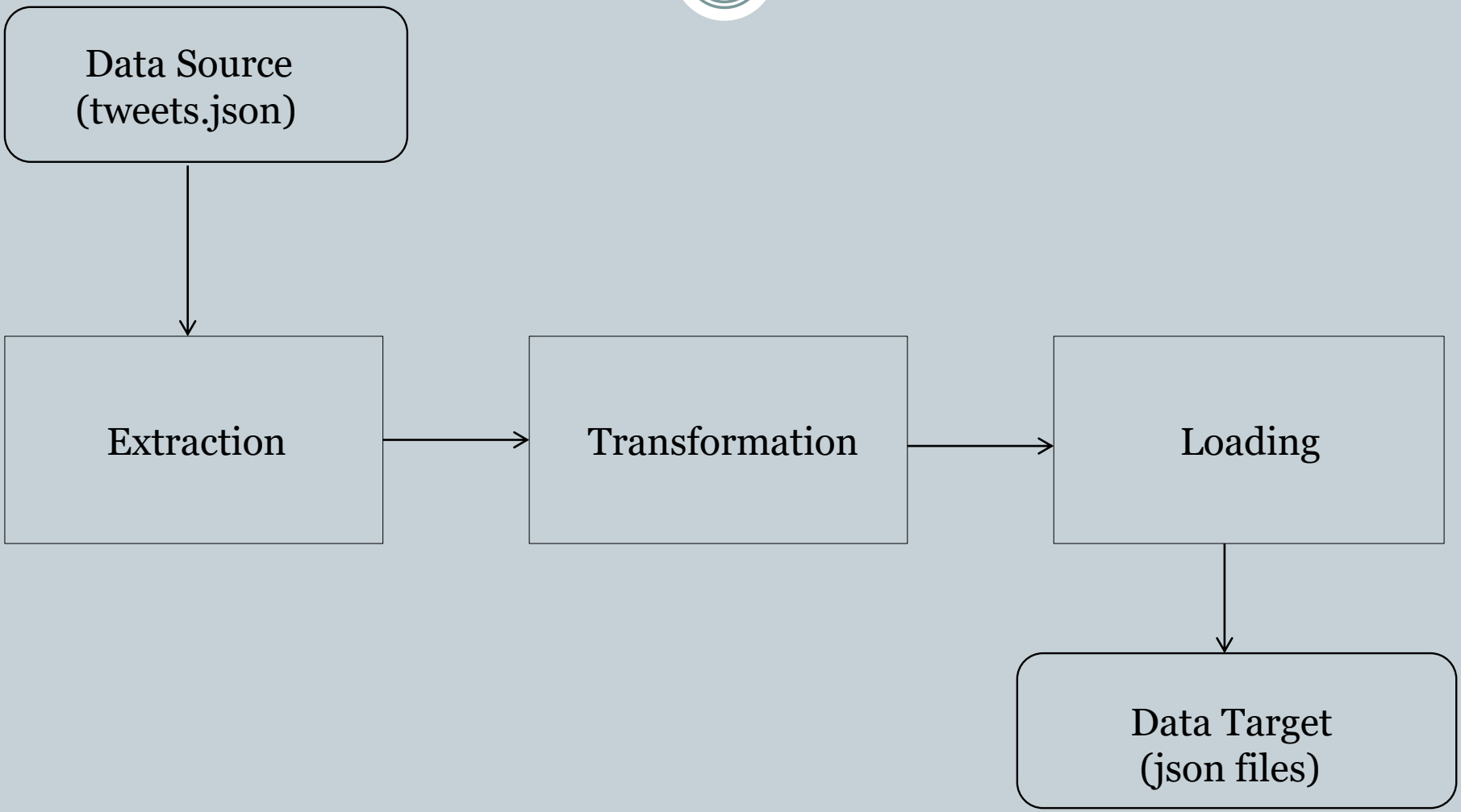
Munich, February 2023

FLiX

Topics Covered

1. High Level Block Diagram – ETL Process
2. High Level Approach
3. Step by Step Code Walkthrough
4. Sample Data
5. Task Automation
6. How to improve the solution
7. Dream Infrastructure
8. Steps to Run the Project

High Level Block Diagram – ETL Process



ETL Process – Extraction

- Data Source: This is where the raw data is stored, such as a tweets.json file containing tweets data.

- Extraction: In this stage, the raw data is extracted from the data source, and the relevant tweets are identified based on their hashtag(#FlixBus) and the timeframe of interest (last available week)

ETL Process – Transformation

- Transformation: In this stage, the tweets are parsed to extract relevant data, such as the date and time of the tweet, the hashtags included in the tweet, and whether it is a retweet or not.
- The data is also transformed to calculate various metrics for each user.

ETL Process - Loading

- Loading: In this stage, the extracted and transformed data is loaded into a target data store in a json format.

- Overall, this ETL process is designed to search for tweets with hashtag #flixbus, extract relevant data from those tweets, transform the data into useful metrics for each user, and store the data in a target data store for further analysis or use.

High Level Approach

- From the given sample data (tweets.json), filter the tweets to only include those posted during the last available week with the hashtag #flixbus.

- Stored the above filtered tweets in a json file (flixbus_tweets_lastweek.json)

High Level Approach

- For each tweet, extracted the below data from the filtered tweets:
  - Date time of the tweet
  - All hashtags included in the tweet
  - Check if it is a retweet
- Stored the above extracted data in a json file (extracted_tweet_data.json)

High Level Approach

Performed below calculations per user,

- For per user:
  - Most recent number of followers
  - Most recent location
  - Average tweet length
  - Top five hashtags used

High Level Approach

- Also, performed below calculations based on the overall tweets data
  - Most active day during the last week
  - Total number of tweets with at least 3 hashtags
  - Maximum number of tweets per user
- Stored the above user metrics in per_user_statistics.json file

Step By Step Code Walkthrough

Steps taken to perform the task

1. Extracting twitter data from the tweets.json file
2. Extracting data from the last available week
3. Search for tweets with #flixbus, during the last available week
4. Extract tweet information such as Date time, Hashtags, if it is a retweet.
5. Calculate per user statistics
6. Calculate most active day during the last week, total number of tweets with at least 3 hashtags, maximum number of tweets per user

1. Extracting twitter data from the tweets.json file

- Function used get_tweets_from_sample_data()
- This function takes tweets.json as an argument and checks if the file is present or not.
- If it's present, the data would be loaded to a input_list else it returns a file not found error.

2. Extracting data from the last available week

- Assumption – Since the data stored in tweets.json is a static data and not real time, we are not sure what date range can be considered for the last week.
- Hence, have calculated last week date range from the available data by first checking the latest date which would be the start date and subtracting 7 days from the latest date would give us the end date.
- Function used get_last_week_start_end_date()

Extracting data for custom date ranges(single day, date range)

- Code is written in a way that it is possible to reprocess data for specific date ranges using a variable called custom_date.
- If custom_date value is set to True then start and end date values needs to be provided.
- If it's set to False then by default start and end date values will be set from the last available week using the function get_last_week_start_end_date()

Store the given Hashtag

- Assumption - considered the #FlixBus hashtag as case-insensitive (please refer slide no. 12 for detailed explanation)
- Functions used getHashTags()
- Returns the hashtag that needs to be used as a filter, in this case it returns flixbus

3. Search for tweets with #FlixBus, during the last available week

- Function used search_flixbus_tweets_last_week()
- In this function, we will first make a call to another function convert_createdtime_to_date() that will parse the date and time string of the tweet into a ‘datetime’ object.

- Post that we will compare and check if the tweet date is within the range of start date and end date.

- If yes, we will call another function extract_hash_tags() to extract all the hashtags used in that particular tweet.

Function - extract_hash_tags()

- Assumption - As a business, it would be valuable to search for tweets containing the hashtag #FlixBus, regardless of the letter case used, including variations like #flixbus, #Flixbus, and #FLIXBUS.
- Therefore, have treated the #FlixBus hashtag as case-insensitive and have searched for tweets containing any possible variation of the hashtag during the last week.
- The check for considering all the variations is done using a variable consider_all_combination

Function - search_hashtag( )

- search_flixbus_tweets_last_week() calls another function search_hashtag() to check if the given hashtag is present in the hashtags list
- Returns true if it's present, false otherwise.
- If the hashtag is present then we will store this tweet info in the Flixbus_tweets_lastweek.json file using the store_tweets() function

Function – store_tweets()

- store_tweets() takes two arguments – filename and the results to be stored
- Stores the given results in a json file under the Results_Data directory

4. Extract tweet information such as Date time, Hashtags, is it a retweet?

- Function used extract_tweet_data()
- Input to this function is all the tweets with #flixbus from last available week
- All the extracted information is stored in a processed_tweets dictionary with key as tweet id and values are Date time, Hashtags, is it a retweet.
- Extracted final results are then stored in the extracted_tweet_data.json file.

5. Calculate per user statistics

- Function used analyze_user_tweets()
- Input to this function is all tweets data with #flixbus, during the last available week
- Below metrics were calculated per user,
  1. Most recent followers - For each user, find their most recent follower count.
  2. Most recent location - For each user, find their most recent location.
  3. Average tweet length: For each user, calculate the average length of their tweets.

Top five hashtags calculation

- Top five hashtags used: For each user, count the number of times each hashtag is used and find the top five most frequently used hashtags.
- This is done using a function merge_counter() that returns a dictionary with hashtags as key and their respective count as value.
- Sorting the results based on the value gives the top 5 hashtags used.
- Results are stored in per_user_statistics.json file

6. Calculate most active day during the last week

- Function used get_most_activeday_and_three_hashtags()
- ‘extracted_tweets’ from step 4 are passed as an argument to this function
- Group the tweets by day and count the number of tweets in each group. Then, find the day with the highest count of tweets.

Calculate total number of tweets with at least 3 hashtags

- Function used get_most_activeday_and_three_hashtags()
- Filtered the tweets to only include those with at least three hashtags, and count the number of tweets that meet this criteria.

Calculate maximum number of tweets per user

- Function used - get_max_tweet_per_user()
- ‘per_user_statistics’ data is passed as an argument to this function.
- For each user, count the number of tweets they have posted and find the maximum count.
- In per_user_statistics, check for the no_of_tweets posted by each user and return the maximum number of tweets per user.

Sample Data from Results_Data directory

- Results will be stored in the below 3 different files under the Results_Data directory:

1. flixbus_tweets_lastweek.json – All the tweets data with #flixbus, during the last available week are stored in this file.
2. extracted_tweet_data.json – From the above search results, extracted information such as date time, all hashtags, is it a retweet or no, all such information are stored in this file.
3. per_user_statistics.json – per user metrics such as most recent number of followers, most recent location, average tweet length, top five hashtags used are stored.

Sample data in extracted_tweet_data.json


```
{
 "1531001390977409025": {

      "date_time": "Sun May 29 19:55:50 +0000 2022",
      "hashtags": [
         "flixbus",
         "flixtrain",
         "flix",
         "verkehrswende",
         "zugfahren"
      ],
      "isRetweet": true

   }
}
```


Note – Key in the above sample data is tweet id

Sample data in per_user_statistics.json


```
{
  "21217711": {
      "created_at": "Tue May 31 04:16:09 +0000 2022",
      "most_recent_followers": 1342,
      "most_recent_location": "Cambridge, MA",
      "average_tweet_length": 140,
      "top_five_hashtags_used": [
         "flixbus"
      ],
      "hashtagcounter": {
         "flixbus": 2
      },
      "no_of_tweets": 2
   }
}
```


Note – Key in the above sample data is user id

- Note - In the per_user_statistics.json, have captured ‘created_at’, ‘hashtagcounter’, ‘no_of_tweets’ in addition to the metrics mentioned in the problem statement.

- These values are helpful in calculating few other metrics. Hence captured it here for reference.

Automate task to run periodically

- To automate this ETL process to run periodically, we can use a cron scheduling tool in Python.
- We can write a Python script to perform the ETL process and call it etl_process.py.
- We can then set up a cron job to run this script periodically.
- ETL tool such as Apache Airflow also could be used to schedule and monitor the task automatically.

Sample cron expression to set up the cron job

- In the below example, we have set up the cron job to run the etl_process.py script every day at 12:00 PM. The output of the script is redirected to a log file (etl_process.log) for debugging purposes

```
# open crontab
$ crontab –e

# add the following line to the crontab file to schedule the job to run every day at 12:00 PM
0 12 * * * /usr/bin/python /path/to/etl_process.py >> /path/to/etl_process.log 2>&1
```

How to improve the existing solution?

- Improve data cleaning and preprocessing: The current solution does some basic cleaning and preprocessing of the data.
- However, more advanced techniques can be used to further clean and preprocess the data.
- For example, removing stop words, sentiment analysis can be performed to get more insights.

How to improve the existing solution?

- Use a scheduler to automate the task: The current solution requires manual intervention to run the task.
- A scheduler such as cron can be used to automate the task to run periodically.

Dream Infrastructure

My Dream Infrastructure would be the one that provides a highly scalable, efficient, and reliable solution for processing and analyzing large volumes of Twitter data in real-time.

Components for setting up Dream Infrastructure

1. Data Storage: Using a scalable data storage solution such as Amazon S3 or Google Cloud Storage to store the raw data.
2. Data Ingestion: Using a real-time data ingestion system such as Apache Kafka to ingest the data from the Twitter API streaming endpoint.
3. Data Processing: Using a distributed computing system such as Apache Spark to process the data in parallel.

Components for setting up Dream Infrastructure

4. Data Visualization: Use a dashboarding and visualization tool such as Tableau or PowerBI to create interactive dashboards and visualizations to display the insights from the data.
5. Monitoring and Alerting: Using a monitoring and alerting system such as Prometheus to monitor the pipeline's health, set up alerts for any failures.
6. Container Orchestration: Using a containerization platform such as Docker and Kubernetes to run the processing jobs and ensure scalability and resilience.

Steps to run the Project

1. Clone this repository to your local machine - https://github.com/ritugm/FlixBus_Data_Engineering_Usecase
2. Run the program by executing the index.py file from the version1 directory (cd version1, python index.py)

Note - In this Python project, the main file is called index.py. This file calls functions that are defined in another file called util.py.

Results Captured in Results_Data directory

Below files will be generated post running the code,

- logs.txt – all the logs are stored in this file, this file will be generated under the version1 directory

- Below 3 files will be generated under the version1/Results_Data directory
  1.flixbus_tweets_lastweek.json
  2. extracted_tweet_data.json
  3. per_user_statistics.json

Thank You!!

Overall, it was an amazing experience to work on this use case. Below hashtags describe my experience ☺

#DataEngineer, #LovedProblemSolving, #CreativeThinking #AwesomeUseCase

#CriticalThinking #LearningNeverStops