Tweet Collection Management

CS 5604 — Information Storage and Retrieval
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Team TWT
Hitesh Baadkar
Pranav Chimote
Megan Hicks
Ikjot Juneja
Manisha Kusuma
Ujjval Mehta
Akash Patil
Irith Sharma

December 17, 2020

Abstract

The Tweet Collection Management (TWT) Team aims to ingest 5 billion tweets, clean this data, analyze the metadata present, extract key information, classify tweets into categories, and finally, index these tweets into Elasticsearch to browse and query. The main deliverable of this project is a running software application for searching tweets and for viewing Twitter collections from Digital Library Research Laboratory (DLRL) event archive projects.

As a starting point, we focused on two development goals: (1) hashtag-based and (2) username-based search for tweets. For IR1, we completed extraction of two fields within our sample collection: hashtags and username. Sample code for TwiRole, a user-classification program, was investigated for use in our project. We were able to sample from multiple collections of tweets, spanning topics like COVID-19 and hurricanes. Initial work encompassed using a sample collection, provided via Google Drive. An NFS-based persistent storage was later involved to allow access to larger collections. In total, we have developed 9 services to extract key information like username, hashtags, geo-location, and keywords from tweets. We have also developed services to allow for parsing and cleaning of raw API data, and backup of data in an Apache Parquet filestore. All services are Dockerized and added to the GitLab Container Registry. The services are deployed in the CS cloud cluster to integrate services into the full search engine workflow. A service is created to convert WARC files to JSON for reading archive files into the application. Unit testing of services is complete and end-to-end tests have been conducted to improve system robustness and avoid failure during deployment. The TWT team has indexed 3,200 tweets into the Elasticsearch index. Future work could involve parallelization of the extraction of metadata, an alternative feature-flag approach, advanced geo-location inference, and adoption of the DMI-TCAT format.

Key deliverables include a data body that allows for search, sort, filter, and visualization of raw tweet collections and metadata analysis; a running software application for searching tweets and for viewing Twitter collections from Digital Library Research Laboratory (DLRL) event archive projects; and a user guide to assist those using the system.

Acknowledgements

Special thanks to Dr. Edward A. Fox, Prashant Chandrasekar, Xinyue Wang, and to NSF for support through CMMI-1638207.

Contents

List of Tables

List of Figures

1 Overview

The Tweet Collection Management (TWT) Team aims to ingest 5 billion tweets, clean this data, analyze the metadata present, extract key information, classify tweets into categories, and index these tweets into Elasticsearch (ELS). The main deliverable of this project is a running software application for searching tweets and for viewing tweet collections from Digital Library Research Laboratory (DLRL) event archive projects. This report will refer to our implementation as a subsystem of the collective work of the CS 5604 class. The resulting system from this class will allow information storage and retrieval of tweets, along with electronic theses and dissertations (ETDs) and webpages.

As we began our project, we first read and studied more about what had been done in our area of development. We looked at previous work that was completed as part of the class (CS 5604) in 2019 by the Tobacco Settlement Team. We also researched key tools we needed such as Elasticsearch, Twitter-Based Knowledge Graph [13], and TwiRole. Finally, we researched methods for extracting unique tweet data such as geo-location.

In addition to reading background information, we consulted with a subject matter expert (SME). Our team was assigned Xinyue Wang, who had previously conducted research related to extraction and analysis of Twitter data. Our SME provided us with a list of milestones for the semester. These milestones were:

1. Downloading Twitter collections from the Social Feed Manager (SFM) server, open-source software that harvests social media data and web resources from Twitter, to index tweets and make them searchable.
2. Downloading Twitter collections from the DLRL event archive projects, a database of 5 billion tweets, to index.
3. Utilizing TwiRole software to determine if a tweet was sent by a male, female, or brand, to categorize the tweets in the collection.
4. Determining a chosen set of services to work on tweets that will run and manage Twitter data through the front-end (FE) interface.
5. Using a filestore in Parquet format for backup storage of the Twitter database.

Additionally, our SME gave us critical input which helped us develop our services.

Using the input from the SME along with knowledge gained from our research, we developed our requirements for the project. Ultimately, we developed nine services, beginning with extracting raw tweets from a WARC file, and ending with an indexed file containing key fields (such as hashtag, geolocation and others) that a user could query from the front-end.

We developed a data body that allows for search, sort, and visualization of tweets by the user. Our services currently run serially (sequentially), but future development could allow them to run in parallel to allow services to be utilized independently. This type of implementation is intended to allow for continuous integration and continuous development (CI/CD). The team, in addition to writing code focused on functionality, also wrote unit tests to ensure that current and future work operates as expected. GitLab has been chosen to house our repository, testing, and container registry.

Our team also contributed to the cross-functional Elasticsearch team, which validated that the various types of data (tweets, theses, dissertations, and webpages) were indexed properly. The Elasticsearch team made sure the front-end had consistent endpoints available to search and filter through this data.

Our design scheme stores Twitter data in two formats: JSON and Parquet. Parquet will be used for storing the raw tweet data, with all fields from the original collection schema. This is important since we have such a large collection of tweets, with fields in a nested data structure format. Parquet provides performance columnar storage, is made to handle such complex data in bulk, and features many efficient data compression and encoding techniques.

We also developed a user guide to help navigate our system, and a developer's manual for future changes and improvements to our work. Throughout the semester, the bulk of our work involved developing programs for our

services as well as coordination and administration needed for deployment. Once the services were completed and registered, we ran end-to-end tests using our Dockerized services.

Some possibilities for future work include implementing a parallel workflow and adding feature flags for querying. Our services currently operate in series, however are designed/implemented in such a modular way that they could be run in parallel on the base cleaned tweets.

2 Literature Review

To gain a clearer picture of the work for our team and the work we had available to build off, we consulted several papers and research efforts.

2.1 Elasticsearch (ELS) CS5604 Fall 2019

Li et al. [11] utilized Elasticsearch to support "searching, ranking, browsing, and presenting recommendations" for collection management tobacco (CMT) documents as well as Virginia Tech's collection of 30,000 ETDs. They studied:

- Ingesting the collections of data
- Deciding relevant and important fields
- Incorporating another team's machine learning (ML) and NLP models for sentiment analysis and clustering, among others
- Weighting and nesting queries
- Supporting Kibana

Figure 1 displays the overarching structure and connectivity of the different teams involved in this system.

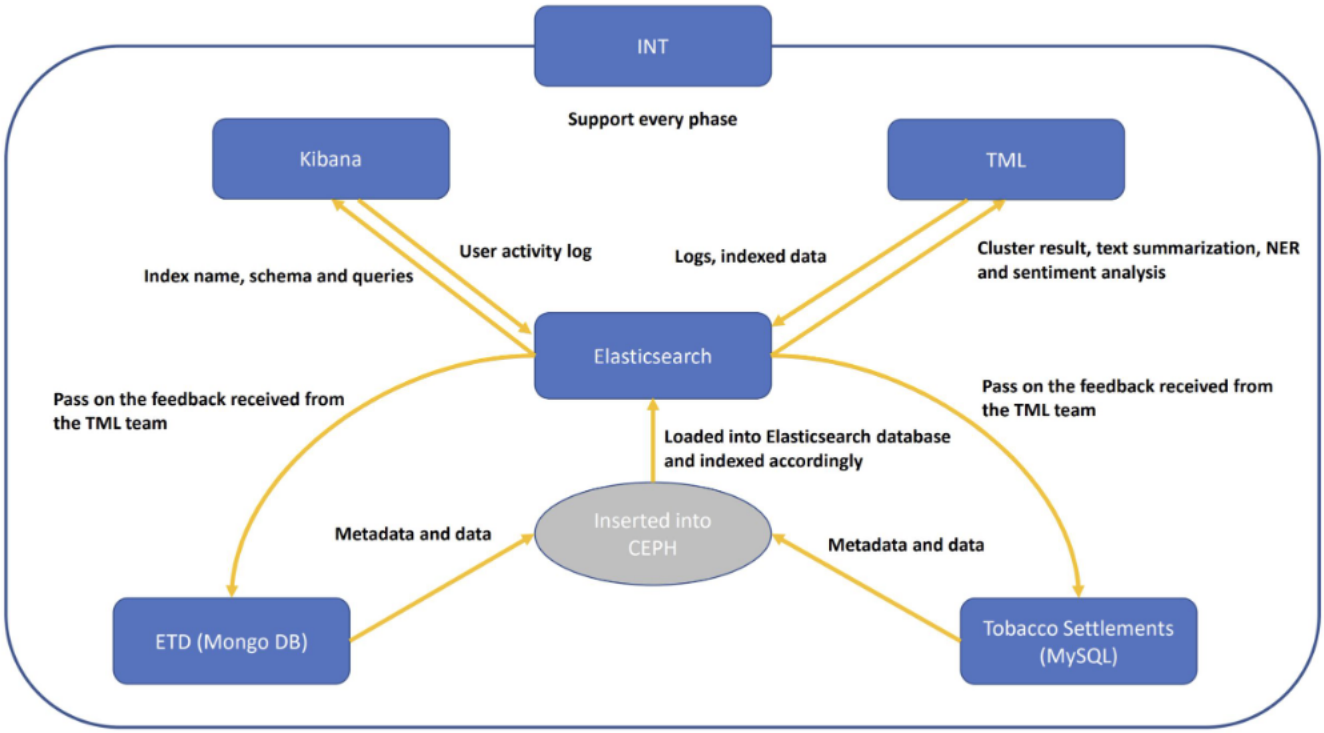

Figure 1: Elasticsearch-related data flows for Fall 2019 system [11]

They made use of Python (along with shell scripts) for ingesting and indexing data into ELS. They provide code snippets and reference their codebase for these tasks. Our Elasticsearch usage will be limited to indexing and querying, rather than using the previous approach of extending Kibana.

2.2 Twitter-Based Knowledge Graph for Researchers

Meno and Vincent [13] strove to build an ontology-relation graph between various Twitter entities (users, locations, dates, etc.). Their main functionality involved the ability to query against this "hypergraph" of data by an arbitrary user, having them provide what they know and what they wanted to find, and returning back a workflow path in this graph. Most of their work focused towards generating the comma-separated value (CSV) files for their graphs. They opted to store these graphs in a Grakn database on a VM, initially using Neo4j, but finding more functionality in the former. Their intent was to allow SMEs to more easily generate workflows for their projects involving Twitter data and advanced analysis. While the graph portion of their work is mostly not relevant to our own, they provide valuable insight in parsing and storing of their data.

We will not work directly with graphs or examine ontology relations. Instead, our work will purely focus on querying data/metadata of the tweets in ELS, rather than finding connections between a (sub)set. Primarily, we will be working with JSON data, not CSV, so our parsing schemes will be different and simpler.

2.3 Geo-Locating Tweets with Latent Location Information.

Dr. Lee's study [8] describes how to disambiguate the location of tweets. They collected over 1.4 billion tweets and used the following process to identify geo-location:

1. Extracted geonames (geospatial named entities such as "Greenville").
2. Predicted implicit state information using geonames.
3. Predicted implicit state information using geonames and location indicative words (LIWs).

Their work was successful, but time consuming and complicated. While we will strive to use the approach to help us geo-locate tweets for our work, it will be a stretch goal for our team and our implementation will be limited.

2.4 A Hybrid Model for Role-related User Classification on Twitter

TwiRole is a hybrid model for Twitter user classification. It is specifically designed to detect whether a given user is male, female, or brand-related [10]. Provided as both a CodeOcean container for portability and a Github repository [9], TwiRole can easily run on a collection of tweets. Li et al. note that this model outperforms existing work, and has up to 90% accuracy while utilizing features like username and profile image. More detail is available later in this report.

Using it is relatively simple and can easily be incorporated into our workflows to provide additional information for our subsystem's users. Note that our team incorporates the model and one classifier (the only one made available in the Github repository, named Classifier 1) as provided and made no changes to the existing design. The Python base requires no additional dependencies and can be used as a separate service. As seen in Figure 2, a user can provide TwiRole with a username or a CSV of multiple usernames. Then, for each task (username), TwiRole uses a percentage scale to indicate its prediction regarding if the user is a brand, male, or female.

```
+ python -u user_classifier.py -f ../data/user/test.csv
/opt/conda/lib/python3.6/site-
packages/sklearn/externals/joblib/externals/cloudpickle/cloudpickle.py:47:
DeprecationWarning: the imp module is deprecated in favour of importlib; see
the module's documentation for alternative uses
  import imp
Task    1: CNN              =>  •[30mBrand  •[0m•[30m[Brand: 100.00%,
Female: 0.00%, Male: 0.00%]•[0m
Classifier_1:   [Brand: 80.00%, Female: 20.00%, Male: 0.00%]
Classifier_2:   [Brand: 100.00%, Female: 0.00%, Male: 0.00%]
Classifier_3:   [Brand: 99.99%, Female: 0.00%, Male: 0.01%]
Task    2: ArianaGrande     =>  •[31mFemale •[0m•[31m[Brand: 0.10%, Female:
99.85%, Male: 0.05%]•[0m
Classifier_1:   [Brand: 50.00%, Female: 50.00%, Male: 0.00%]
Classifier_2:   [Brand: 20.00%, Female: 70.00%, Male: 10.00%]
Classifier_3:   [Brand: 0.00%, Female: 99.99%, Male: 0.01%]
Task    3: jimmyfallon      =>  •[34mMale   •[0m•[34m[Brand: 3.40%, Female:
1.85%, Male: 94.75%]•[0m
Classifier_1:   [Brand: 0.00%, Female: 0.00%, Male: 100.00%]
Classifier_2:   [Brand: 40.00%, Female: 20.00%, Male: 40.00%]
Classifier_3:   [Brand: 3.84%, Female: 5.64%, Male: 90.52%]
```

Figure 2: Sample TwiRole categorization [9]

3 Requirements

This section will discuss what the TWT team needs to accomplish in regards to functionality, level of quality, and user support.

Table 1: Design goals for TWT team (workflow artifacts)

Goal ID	Input(s)	Information goal/output	Performance criteria	Granularity of results
1	Raw collection of tweets (in .warc)	ELS index column on geo-location	Real-time output	Subset of tweets from the geo-location column based on the user's query for region/location
2	Raw collection of tweets (in .warc)	ELS index column on hashtags contained in tweets	Real-time output	Subset of tweets from the hashtag column based on the user's query for a particular hashtag
3	Raw collection of tweets (in .warc)	ELS index column on each tweet's originating username	Real-time output	Subset of tweets from the username column based on the user's query for a particular username
4	Raw collection of tweets (in .warc)	ELS index column on mentions in tweets	Real-time output	Subset of tweets from the username mentions column based on the user's query for a particular username
5	A sub-collection of tweets (from ELS query)	The filename for a sub-collection of tweets (in .json/.parquet) that the user can download	Batch output	.json/.parquet filename
6	A sub-collection of tweets (in .json/.parquet)	A DataFrame containing the tweets in this sub-collection	Batch input	Cleaned tweets in a DataFrame
7	Raw collection of tweets (in .warc)	ELS index column on tweet keywords	Real-time content	Subset of tweets from the keywords column based on the user's query for keyword(s)
8	Raw collection of tweets (in .warc)	ELS index column on TwiRole categorization for tweets	Batch input	Subset of tweets from the TwiRole column based on the user's query on a classification
9	Raw collection of tweets (in .warc)	ELS index column on each tweet's unique ID	Real-time output	No information shown, only used for internal operations/organization
10	Raw collection of tweets (in .warc)	ELS index column on each tweet's associated timestamp	Real-time output	Subset of tweets from the username index based on the user's query for a particular time range

Table 1 lists the design goals for the TWT team. The "Input(s)" column contains information that the user/collection will provide as input for mining/analysis. The input for all goals is either a raw collection of tweets (in Web ARChive format) or a sub-collection of tweets. The "Information goal/output" column describes the metadata values we will extract from the input collection to be used in an Elasticsearch index. Here, an index is an organizational structure used to store a set of records, typically all with the same metadata/columns. These metadata values are based on the final information goal that is of interest to the SME and to general users. The "Performance criteria"

column discusses whether real-time response is needed, or work can be done in batches. The "Granularity of results" column describes what information should be shown and how it should be organized. For most of our goals, a subset of tweets would be presented to the user based on a query.

3.1 Processing and extracting data

Before the first Interim Report (IR1), our team was able to work with a sample JSON collection provided by our SME. After establishing the first eight user goals in Table 1, we decided to focus on two we could accomplish as soon as possible: hashtags (goal 2) and username (goal 3). These goals shaped most of the data processing and extraction work for the rest of the project.

After IR2, we found that tweet collection data (collected by SFM and YTK) are in Web ARChive format (known as WARC format). Our services cannot process files in this format. Therefore, we added a service that will read a WARC file, extract the contained tweet data, and write to a .json file. Once this service is complete, the .json file can be processed through our other services: fields are parsed and added and the resulting data is indexed in Elasticsearch.

Before we could extract hashtags and usernames from tweet collections, we had to pre-process the tweets. This involved removing any unnecessary fields and cleaning the data provided by yourTwapperKeeper (YTK) and Social Feed Manager (SFM). A third format for tweet collections originates from the DMI-TCAT application [1], but our team does not yet have access to a collection in this format. Refer to Section 7.1 for details on how to generate collections in these formats. Refer to Appendix B for sample Twitter data from Social Feed Manager.


```
{
    "archivesource": "twitter-search",
    "text": "RT @thereaIbanksy: Never forget. \n\n#WalterScott http://t.co/EdEiU8ZwLk",
    "to_user_id": "",
    "from_user": "Jamal_Chatha",
   "id": "586220033648406528",
    "from_user_id": "2601261336",
    "iso_language_code": "en",
    "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Android</a>",
    "profile_image_url": "http://abs.twimg.com/images/themes/theme1/bg.png",
    "geo_type": "",
    "geo_coordinates_0": 0,
    "geo_coordinates_1": 0,
    "created_at": "Thu Apr 09 17:32:02 +0000 2015",
    "time": 1428600722
}
```


Figure 3: Sample Twitter data from yourTwapperKeeper [2]

As seen in Figures 3 a tweet (record) includes various information, most of which is unnecessary for our team and for the users. In order to parse such a large JSON and search the collection, it is important to pre-process these collections to only the tweets' high-level metadata. After all the tweet data is pre-processed, the textual tweet content can be scanned for the '#' symbol to find and process all hashtags. The 'from_user' or 'user/screen_name' metadata fields can be searched and used for extracting the username associated with the tweet.

3.2 Efficient data loading

Given that the size of collections is often in the 100's of gigabytes, loading such collections for pre-processing is time consuming. To address this, we explored some data loading options to optimize the input data loading process.

- Explored methods for batch-wise dataloading using Numpy and `pandas`.

- Explored data loading using Dask [3], a parallel processing library. Among many other features, Dask provides an API that emulates `pandas`, while implementing chunking and parallelization transparently.

We tested the performance of the above methods. Results show that Dask is the better method for data loading in case of bigger datasets. Through its parallel computing features, Dask allows for rapid and efficient scaling of computation. It provides an easy way to handle large and big data in Python with minimal extra effort beyond the regular `pandas` workflow.

3.3 Indexing tweets and creating services

We are able to work with both the YTK-formatted and SFM-formatted data to extract key fields, such as hashtags, username, mentions, geo-location, keywords, and timestamp. Once this content is extracted, it is indexed into Elasticsearch. This is crucial as it allows the user to search based on these fields.

Our team has created services to modularize extracting these data fields. More information about the specific services and their functionality is listed in Section 5. We have Dockerized each service so they can be used independently and in combination with each other. Dockerization of these services also allows them to be used on collections outside the initial set and allows the query results to be exported. The Docker containers have access to Ceph and network file service (NFS)-based storage, where raw tweet data will be stored.

We are using TwiRole for tweet categorization. We are using the code provided through the TwiRole GitHub repository [9] to categorize the tweets in a collection. Then, a metadata field with this categorization is added to each tweet. This field is then added to our Elasticsearch index.

An initial test of TwiRole on a set of 25 tweets took approximately 5 minutes to complete. Since running TwiRole is time intensive and takes a lot of storage space, we have decided to batch categorize our collections. Currently we have a TwiRole service that processes small sets of tweets. For larger collections, we plan to extract all usernames into a CSV file. Then, we can run large batches of unique Twitter usernames through the TwiRole service and create an index column with the categorization of each tweet. Batch processing the TwiRole categorization will avoid performance issues with doing such calculations repeatedly in real-time.

4 Design

4.1 Approach

Our plan started with characterizing the data. This included the amount of, source of, and key values that we wished to capture from the data (e.g., dates spanning, locations from, total amount of). We then identified a sample collection of tweets to work with (sfm-coronavirus-sample.json). Next, we developed services to clean and save the data into intermediate JSON files for passing between services. After IR2, we realized that our raw collections come in .warc file (rather than .json) format. We added a conversion service to the front of our workflow. From this point, key data is extracted from the tweets (further details can be seen in Section 4.3.2) and placed in intermediate files. Finally, the data is ingested into an index in Elasticsearch, to be accessed through the FE interface. More detail is given later and in Figure 4.

Our approach involves developing separate services that a user can (indirectly) access to analyze tweets. Code is written into as small services as possible, in order to increase modularization. These services are then loaded into Docker containers. This again allows for easier and modular updates in the future.

4.2 Tools

- Python will compose a majority of our project. Each service will be written in Python, utilizing both standard and external libraries (such as `gensim`).
- Docker will be used to containerize our services. It is intended to modularize our work and provide an easy interface for future work to build off of.
- TwiRole will be used for tweet categorization. As stated previously, it determines whether tweets from a given Twitter account are likely to be from a male, female, or brand. Figure 2 demonstrates use of the classifier with an example collection.
- `gensim` [15] (specifically the Text Summarization module) will determine the keywords within a tweet's textual content. This allows the user to search for topics such as "coronavirus" or the "Presidential Debate" and receive tweets likely to be relevant.
- Elasticsearch will be used for processing and querying collections of tweets, based on the different columns of the index created for our collections. Some example columns are: originating user, hashtags, and user mentions for/in the tweet. Our team will provide the data and the fields against which users will query.
- Apache Parquet is a storage format intended for more efficient queries on large datasets. It stores tables in columnar format, instead of row-based. Our team will be using Parquet format for backup of large collections of tweets, to complement existing collections in JSON format.
- Various social media management systems will provide our raw collections of tweets. As previously mentioned, we will utilize collections originating from YTK and SFM. The Twitter API is referenced throughout the duration of this project, since that format very closely aligns with the output format from SFM, which manages multiple social media types including Twitter.
- `warcio` [17] is a streaming library for reading and writing web pages in WARC format. `warcio` iterates over a stream of WARC records using the ArchiveIterator.

4.3 Methodology

Our team followed a workflow-based methodology to develop our subsystem. This process was recommended by another SME, Prashant Chandrasekar, and involved capturing user-requests and goals, then describing workflows that represent solution design and implementation specifics. Doing so enables our team to build a subsystem that the user wants and a well-described implementation.

The first step included learning who the user is and what goals they wish to accomplish through the system solution. This can be done through interviews, recordings of interactions, choice tests, etc. Key results from this step are a persona for each user "type" our subsystem must cater to, as well as the explicit list of goals they would like our subsystem to support. Our users (researchers at Virginia Tech) require software that will search existing and new tweet collections. In an effort to keep our product relevant, we have used a modular approach (i.e., each service defined and deployed separately). Additionally, a user manual will be provided to assist in using the system. Specific goals are listed in Table 1.

The second step involves breaking down these goals into tasks such that the final workflow design can support every task identified. Refer to Appendix A for tabulated detail on how the goals listed in Table 1 were broken down. Each goal has two associated tables: a data table and a tasks table. As an example, Table A.1 lists out the data values required to complete the first goal involving geo-location. The first row, with data ID 1, refers to our unmodified collection of tweets. Next, the same collection of tweets is read into a DataFrame object and preprocessed/cleaned for more efficient consumption. The last step involves the same tweets now with an additional metadata field containing the extracted geo-location. Table A.2 displays how tasks are used to progress from one data type to the next (as listed in the data table). The first task, with task ID 1, describes filtering the raw data (data ID 1) into a DataFrame of cleaned tweets (data ID 2). The second task involves adding the extracted geo-location as a metadata field to that DataFrame.

The other data/task table pairs can be read similarly. Note that all tables in Appendix A refer to DataFrames of tweets being passed around from task to task. However, in order to better suit the Docker containerization setup, we switch to (and later in this report, describe) passing around a filename for a file containing those tweets instead.

4.4 Conceptual background

The Architecture Design (Figure 4) created by the integration (INT) team visually depicts how the data will progress from raw form through to the user. The TWT team workflow diagram (Figure 5) shows how our portion of the process operates. We receive a collection of raw tweets in WARC format. We extract relevant fields from this collection into .json files. From there, the metadata extracted with these services is passed to Elasticsearch where the FE team can display to the user.

It is important to note that our services are currently run serially/sequentially. However, our design allows for parallelization of the extraction services (as depicted in Figure 6) with an additional merging service combining the results. It is outside the scope of our project, but we recommend this merger service for future work or a future version of this system.

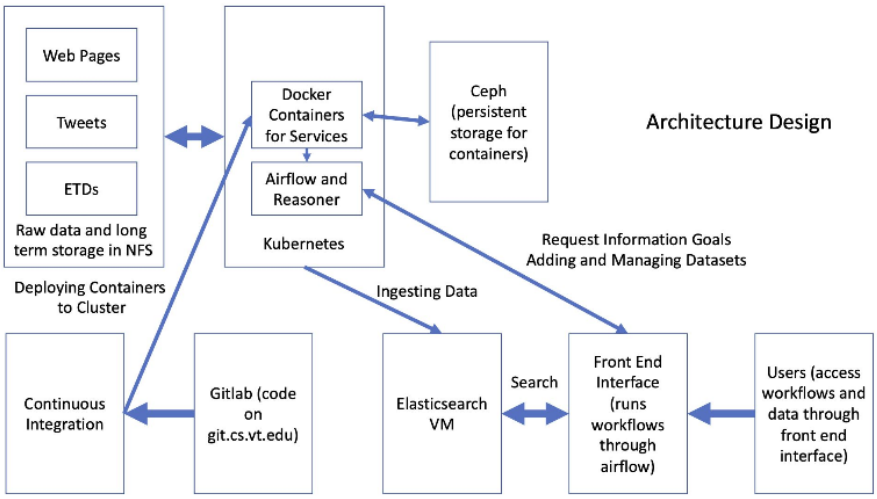

Figure 4: Updated overview of Fall 2020 system and team interactions [5]

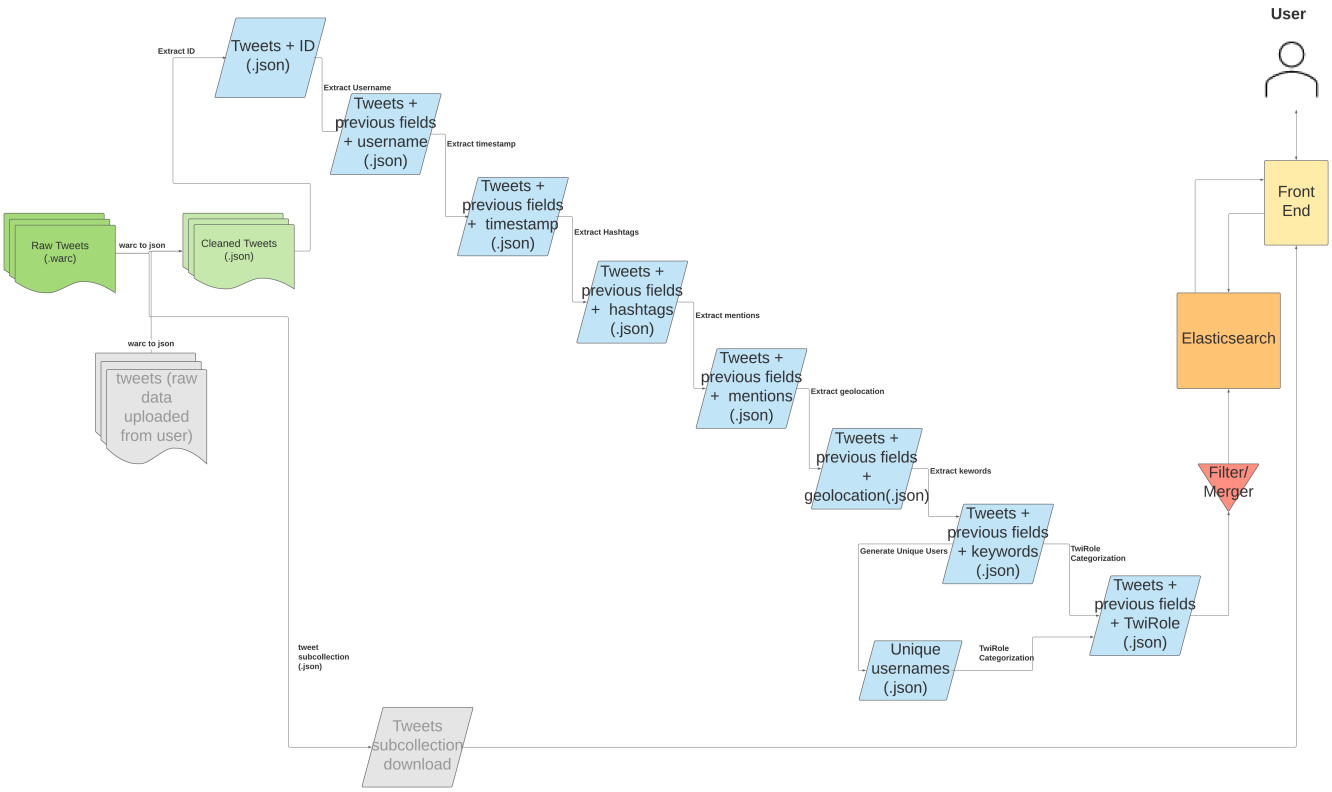

Figure 5: Workflow diagram for TWT team, depicting states of data and services to move between states

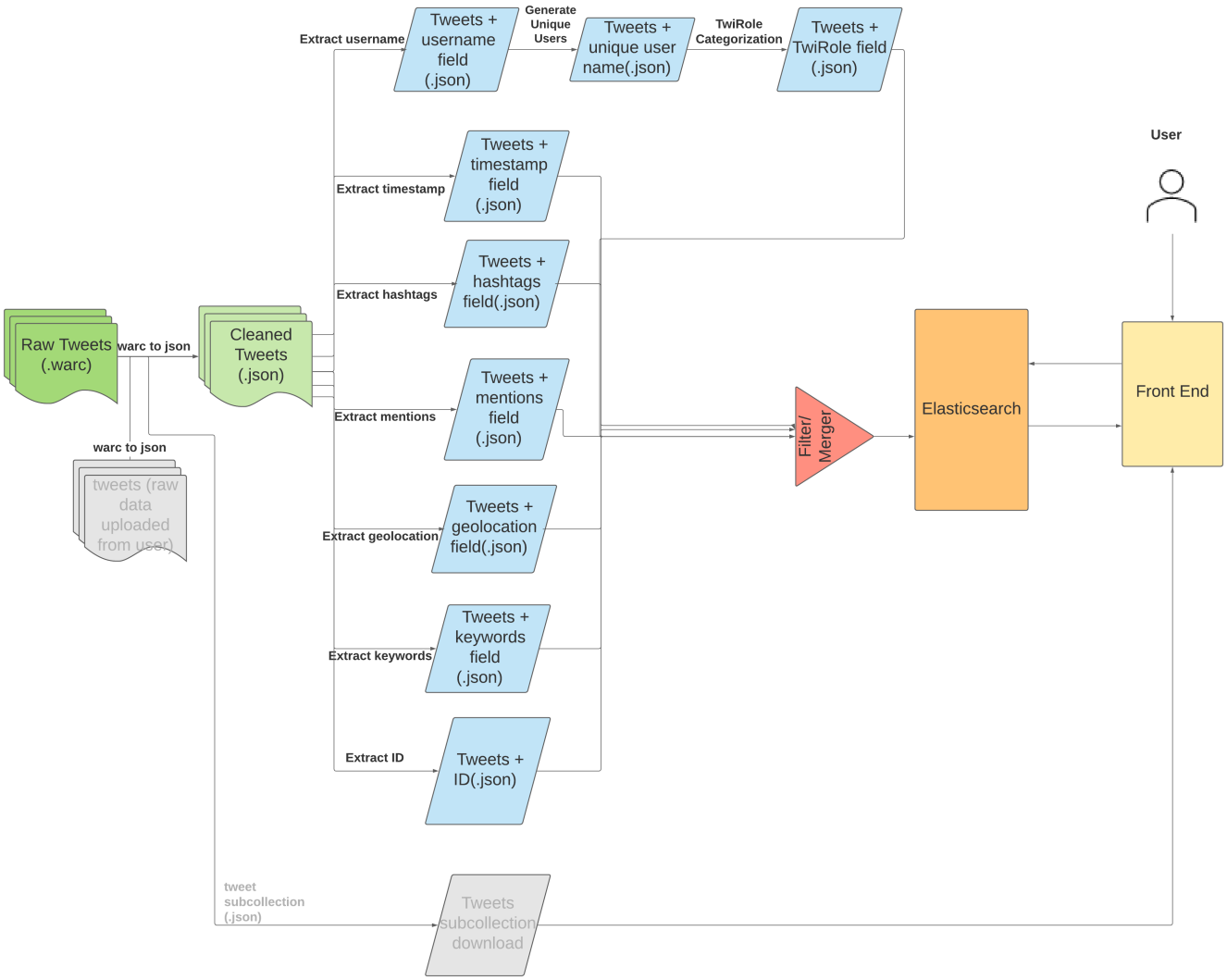

Figure 6: Alternative workflow diagram for TWT team, parallel configuration

4.5 Deliverables

- Data body that allows for search, sort, and visualization.
- Running software application for searching tweets and for viewing Twitter collections from DLRL event archive projects.
- A user guide for using the system.
- A developer's manual for future use.
- A final report documenting our process, events and changes, and final product.
- A final presentation to share our results.

4.6 Timeline

Table 2: Timeline details

Task	Target completion	Assignee	Status
Defined project roles	09/02/2020	Team	Complete
Reviewed key literature	09/07/2020	Team	Complete
Talked with SME about goals and requirements	09/07/2020	Team	Complete
Identified 2 goals to focus on first: Hashtag and Username	09/07/2020	Team	Complete
Metadata for two fields extracted from sample collection (username and hashtags)	09/17/2020	Ikjot, Akash, Pranav, Ujjval	Complete
TwiRole sample code provided and investigated	09/17/2020	Manisha	Complete
Completed IR1	09/17/2020	Team	Complete
Met with Dr. Li to learn more about available datasets and TwiRole	09/23/2020	Pranav, Megan	Complete
Revisit goals and services; clarify metadata to be extracted	09/28/2020	Team	Complete
Finalized services to be delivered	10/02/2020	Team	Complete
Successfully extract fields on sample collection	10/07/2020	Ikjot, Akash, Pranav, Ujjval, Megan, Hitesh	Complete
Initial test storing tweets in Parquet	10/2/2020	Irith	Complete
Completed IR2	10/08/2020	Team	Complete
WARC service (to JSON conversion)	10/18/2020	Megan	Complete
Unit testing	10/21/2020	Team	Complete
Tweet collections ingestion (full collection from NFS)	10/23/2020	Team	Complete
Content of tweets cleaned and metadata extracted - extract key data (Hashtag, Username, Geo-location, Mentions, Keywords, ID, and Timestamp)	10/23/2020	Team	Complete
Tweet categorization via TwiRole	10/23/2020	Manisha	Complete
Container registry setup	10/25/2020	Pranav	Complete
Collections accessible from container	10/25/2020	Megan, Irith	Complete
Chunk processing/optimization of approach	10/30/2020	Akash, Irith	In-progress
Completed IR3	10/28/2020	Team	Complete
Explore other format options (pickle, numpy, etc) for intermediate outputs	11/6/2020	Akash, Ujjval	Complete
Working CI/CD deployment in GitLab	11/18/2020	Irith	Complete
Refined unit-testing structure	11/18/2020	Akash, Ikjot, Ujjval	Complete
Completed initial service registration	11/19/2020	Megan	Complete
Automated services using Airflow and reasoner engine	11/20/2020	Megan, INT Team	Complete
Unique user service for TwiRole implemented	11/22/2020	Ujjval	Complete
Updated CI/CD deployment with new testing structure	11/29/2020	Irith	Complete
TwiRole service modified and Dockerized for use as a service	11/30/2020	Manisha	Complete
Final presentation	12/2/2020	Team	Complete
Project completion	12/9/2020	Team	Complete

In Table 2, we describe the tasks our team has/plans to accomplish, by what date, who specifically will work on that task, and a short status label.

5 Implementation

In this section, we cover implementation details for the set of goals which we have identified after discussion with our SMEs. All tasks require a set of services to complete the goal. Based on discussions with the INT team regarding container allocation for each team, we have selected a set of services which will complete our goals.

Table 3: Services to support tasks

ID	Service Name	Input(s)	Output
1	WARC to JSON converter	Name of raw tweets .warc, backup location for collection	Name of .json containing cleaned tweets, name of .parquet containing cleaned tweets
2	Add ID field	Name of .json containing cleaned data (in .json/.parquet)	Name of .json containing tweets with id field added
3	Add username field	Name of .json containing cleaned data	Name of .json containing tweets with username field added
4	Add timestamp field	Name of .json containing cleaned data	Name of .json containing tweets with timestamp field added
5	Add hashtags field	Name of .json containing cleaned data	Name of .json containing tweets with hashtags field added
6	Add username mentions field	Name of .json containing cleaned data	Name of .json containing tweets with username mentions field added
7	Add geo-location field	Name of .json containing cleaned data	Name of .json containing tweets with geo-location field added
8	Add keywords field	Name of .json containing cleaned data	Name of .json containing tweets with keywords field added
9	Generate list of unique users	Name of .json containing cleaned data	Name of .json containing unique users
10	Add TwiRole classification field	Name of .json containing cleaned data, name of .json containing unique users and previous categorizations	Name of .json containing tweets with TwiRole classification field added, name of .json containing updated unique users and their categorizations
11	Filter/merge fields	Name of .json containing cleaned data + fields of interest	Name of .json containing only fields of interest
12	Generalized indexing using Elasticsearch	Name of .json containing tweets with fields to be indexed	Name of text file containing indexing results

For all tasks, we have created Python-based modules, wherein we read a collection (.json format) into a `pandas` DataFrame, which makes large dataset processing easy and efficient. All modules are implemented to accommodate the requirements of containers and clusters provided by the INT team. We reference previous work by Bock in managing collections of tweets [4].

5.1 WARC-to-JSON tweet conversion

Our raw/original tweet collections (provided by SFM and YTK) are in a WARC (Web ARChive) format, so it is necessary to read the WARC file, extract the tweet data, and write it to a .json file in order to be utilized within our other services. Web ARChive (WARC) is an archive format used to preserve multiple (often many) digital resources in one archive file along with related metadata. For our team's purposes, this means that tweets are compiled (by SFM and YTK) in one file along with metadata; this is called a WARC record. Each WARC record has a header followed by record content. There are two types of WARC records; tweet data is found in "response" type records [6].

In order to allow use of the data in these files, we have implemented a service to read a WARC file, extract the tweet data within, and write it to a .json file. This service takes place first in the pipeline, since no other services can be conducted until the initial data is transferred from the WARC file. We utilized `Warcio` [17], a streaming library

for reading and writing web pages in WARC format. `warcio` iterates over a stream of WARC records using a method called ArchiveIterator. Our service determines whether the record contains tweet data, and if so, extracts it. Once the data has been read from the WARC file and transferred to a .json file, it can then be processed through our other services.

5.2 ID extraction

The ID field is not intended for user consumption, instead used for organizing tweet collections in Elasticsearch. By providing the ID of each tweet, Elasticsearch can prevent duplicate index operations from adding duplicate entries, instead overwriting with a new version of the tweet with the same ID. This service can also double as a dehydration service, to extract IDs from tweets for sharing collections compactly and effectively. The following set of services will be used to extract IDs from tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 2 - Adding an ID column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. For each tweet in our DataFrame, we find the ID for that tweet, unifying format to a string as needed. We extract this value and add this to a new column. We again write the DataFrame back into a .json file.

5.3 Username extraction

Usernames form the basis of many analyses, as it is the most basic unique identifier of an individual. It can also be used to categorize tweets and display them easily in a Twitter search. Usernames enable researchers to analyze activity patterns of an individual and, consequently, their friends and followers. It can be used to determine valuable information, such as the total number of tweets by a user and the user profile information (profile URL, user activity such as followed pages, likes, retweets, and user hashtags). It provides great insight into determining trending topics and in sentiment analysis, where it is useful in determining if proliferating or anti-social topics can cause changes in peer behavior. The following services will be used to extract the originating username (user who tweeted it) for a specific tweet:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 3 - Adding a username column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. For each tweet in our DataFrame, we find the originating username for that tweet (which would be the corresponding value given to the 'ScreenName' key in the 'users' column of our collection). We extract this value and add this to a new column, and write the DataFrame back into a .json file.

5.4 Timestamp extraction

The timestamp associated with each tweet is typically in reference to when the tweet was created/posted. This field can be used when querying datetime ranges and retrieving a list of tweets created during that time. The following set of services will be used to extract timestamps from tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 4 - Adding an timestamp column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. For each tweet in our DataFrame, we find the timestamp for that tweet. We extract this value and add this to a new column. We again write the DataFrame back into a .json file.

5.5 Hashtag extraction

People use the hashtag symbol (#) before a relevant keyword or phrase in their tweet to categorize those tweets and help them show more easily in Twitter search [16]. When using Twitter, clicking or tapping on a hashtagged word in any message shows the other tweets that include that hashtag. We are trying to emulate this feature seen in Twitter API [16] by finding tweets with a given query hashtag. The following set of services will be used to extract hashtags from tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets, which is a more scaleable format. We clean this data and write it back into a .json file.
- Service 5 - Adding a hashtags column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. We find hashtags in each tweet's content using regular expressions (regex), then add the search results to the new column. We again write back into a .json file.

5.6 Username mentions extraction

A username mention is when a username is present anywhere in the body of the tweet. It will be preceded by the symbol '@'. The following services will be used to extract mentions from the collection of tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 6 - Adding a field for username mentions: We take the resulting .json file from service 1 and convert its contents into a DataFrame. We find all mentions in each tweet's textual content using simple search that will find all words that start with '@', and then add these search results to a new column. We again write this DataFrame back into a .json file.

5.7 Geo-location extraction

One of the most important uses of geo-location on Twitter is to track breaking news and events. Some researchers/users are also interested in knowing what topics are trending in certain locations. For this, we are using the 'coordinates' key in the metadata to extract location information (latitude and longitude). The following set of services will be used to extract the geo-locations from tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (from a .json file) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 7 - Adding a geo-location column: We take the resulting .json file from service 1 and convert its contents to a DataFrame. For each tweet in our Dataframe, we extract explicitly mentioned location information from the 'coordinates' key and add it to a new column. We again write the DataFrame back into a .json file.

5.8 Keyword extraction

For general summarization, we could use the `gensim` text summarization module [15]. This module summarizes the given text, by extracting one or more important sentences from the text. We can adjust how much text the summarizer outputs via input parameters (ratio and word count). However, for this task, we use the keywords module, which is a part of the `gensim` text summarization module. Keyword extraction works in the same way as summary generation (i.e., sentence extraction), in that the algorithm tries to find words that are important or seem representative of the entire text. The keywords are not always single words; in the case of multi-word keywords, they are typically all nouns. The following set of services will be used to extract keywords from tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 8 - Adding a keyword column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. We then find keywords based on each tweet's textual content, then add those keywords to a new column. We again write it back into a .json file.

5.9 Unique users generation

From the purpose of running the TwiRole classifier, we need to extract a list of unique usernames. Hence, after we extract a complete list of usernames, we discard the duplicates and only keep the unique usernames. This is done in order to reduce the amount of data that will be passed to the TwiRole classifier, thereby making the classification process computationally more efficient. In addition, we update the same file/database over many runs of this pipeline, to cache previous categorizations from TwiRole and prevent unnecessary reruns on the same user. The following services will be used to extract the unique usernames:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 3 - Adding a username column: We take the resulting .json file from service 1 and convert its contents into a DataFrame. For each tweet in our DataFrame, we find the originating username for that tweet (which would be the corresponding value given to the 'ScreenName' key in the 'users' column of our collection). We extract this value and add this to a new column, and write the DataFrame back into a .json file.
- Service 9 - Dropping duplicate usernames: We take the resulting .json file from service 1 and convert its contents into a DataFrame. We remove all columns but username and drop the duplicates in order to get the unique usernames, then write the DataFrame back into a .json file.

5.10 Tweets categorization using TwiRole

The pre-trained version of TwiRole is published on Github for role-related user classification on Twitter. The model can automatically crawl a user's profile, profile image and recent tweets, and classify a Twitter user into a brand, female or male, which aids user-related research on Twitter.

Since running TwiRole is time intensive and takes a lot of storage space, we have decided to batch categorize our collections. For larger collections, we plan to extract all usernames into a CSV file. Then, we can run large batches of unique Twitter usernames through the TwiRole service and create an index column with the categorization of each tweet. Batch processing the TwiRole categorization will avoid performance issues with doing such calculations repeatedly in real-time.

Currently, we have a json file "*unique_users.json*" that serves as a database with columns for unique usernames and TwiRole categorization. This allows the TwiRole service to reference the "*unique_users.json*" to see if the categorization of a username was already calculated. This helps reduce the time for the service since we only run the categorization algorithm on new usernames and retrieve the previously calculated classification for memory if it exists.

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- Service 10 - Adding a TwiRole label column: For each tweet, we extract the username and check if it exists in "*unique_users.json*". If it does the TwiRole categorization stored in the file can be returned otherwise the username gets added to the file. The TwiRole classification algorithm reads the list of unique Twitter usernames and runs Classifier 1 to determine the user's category. Then, a metadata field will be created to store the TwiRole classification information. This classification is also stored in "*unique_users.json*" for future reference. We then write this DataFrame back into a .json file.

The unique usernames service can be used here to prevent redundant processing of usernames. For this, services 3 and 9 would be required before running the TwiRole service, which would allow for removal of the redundant username processing just before classification.

5.11 Field filtration/merge

We choose to filter the fields we index into Elasticsearch for two reasons:

1. Reduce the size of the index: By minimizing the amount of data in the index to only that which is needed by the front-end, we reduce the overhead and increase the performance from the perspective of the user.
2. Conflict with differing formats: YTK and SFM-formatted data have different fields that disagree with the metadata mapping once one format has been indexed/mapped.

The following set of services will be used to filter tweets:

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- One or more extraction services
- Service 11 - Filter fields: Provided a .json file and a list of fields to extract, write back only those fields to a .json file.

5.12 Elasticsearch indexing

Once we have extracted or generated fields to index on, we pass through the indexing service, which ingests all fields in the given .json file into a specified Elasticsearch index.

- Service 1 - Parsing the raw data: We take the raw collection of tweets (in .json) and convert its contents into a DataFrame of tweets. We clean this data and write it back into a .json file.
- One or more extraction services
- Service 11 - Provided a .json file and a list of fields to extract, write back only those fields to a .json file.
- Service 12 - Pushing the extracted columns into ELS: Create an index called 'twt' as needed, then add (or update) each tweet in the DataFrame in the index by the tweet's ID. Tracking the results of indexing, it outputs a text file containing information about the number of tweets indexed and how many were new tweets versus how many were updates to existing ones.

5.13 Unit Testing

Unit testing is a software testing method by which individual units of code are put through various tests to determine whether they are fit for use. It ascertains the quality of a piece of code. We have written unit tests for each of our services to check whether they function as intended. We have made use of the `pytest` framework to write/run the tests. We have developed the following test cases for each of our services:

- Null input
- Non-ASCII input
- Invalid file format (file that is not YTK or SFM .json)
- Non-existing file
- Correct output otherwise

5.14 End-to-end test

An end-to-end test has been conducted using our Dockerized services. While it required manually deploying containers, mounting volumes, and specifying filenames, the test was successful in starting from a raw collection of tweets in WARC format to indexing all the extracted fields in ELS. In Figure 7, we show a list of all containers utilized in this test, while Figure 8 displays an example test configuration. Figure 9 displays a list of intermediate files generated from each of our services running, as well as a response from ELS indicating these tweets having been indexed.

Figure 7: Containers used in manual end-to-end test

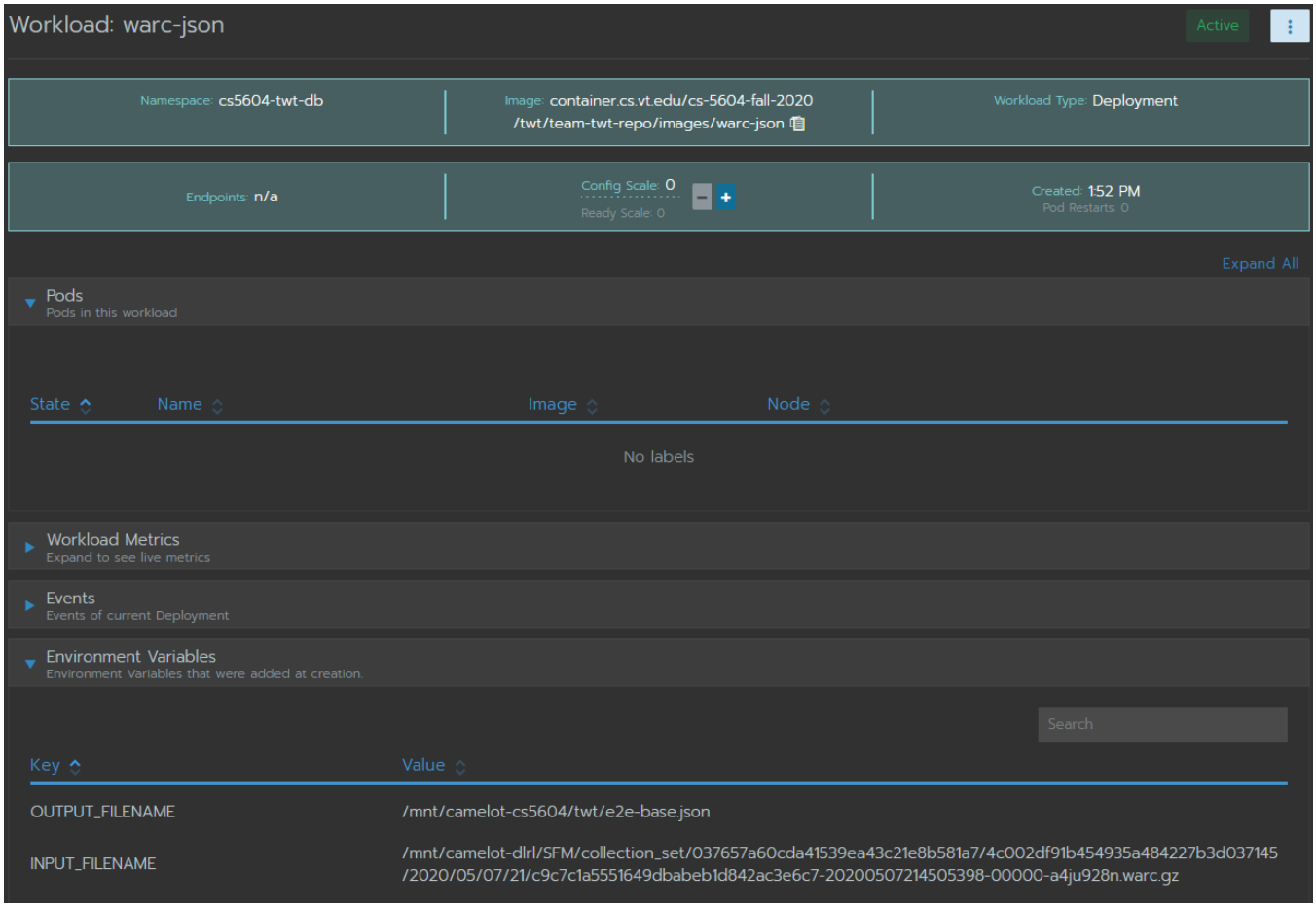

Figure 8: Environment variable configuration for WARC-to-JSON service

```
[root@centos-69bfd9bd87-zbs4w /]# cd /mnt/camelot-cs5604/twt/
[root@centos-69bfd9bd87-zbs4w twt]# ls -Ahl
total 100M
-rw-r--r-- 1 root root 14M Oct 29 17:56 e2e-base.json
-rw-r--r-- 1 root root 13M Oct 29 18:00 e2e-geolocation.json
-rw-r--r-- 1 root root 13M Oct 29 18:09 e2e-hashtag.json
-rw-r--r-- 1 root root 13M Oct 29 18:12 e2e-id.json
-rw-r--r-- 1 root root 13M Oct 29 18:20 e2e-keyword.json
-rw-r--r-- 1 root root 13M Oct 29 18:22 e2e-mentions.json
-rw-r--r-- 1 root root 13M Oct 29 18:24 e2e-timestamp.json
-rw-r--r-- 1 root root 13M Oct 29 18:26 e2e-username.json
[root@centos-69bfd9bd87-zbs4w twt]# curl -s elasticsearch.cs.vt.edu:9200/twt/_search?q=* | head -c 133 && echo ""
{"took":2,"timed_out":false,"_shards":{"total":1,"successful":1,"skipped":0,"failed":0},"hits":{"total":{"value":137,"relation":"eq"}
[root@centos-69bfd9bd87-zbs4w twt]#
```

Figure 9: Results from manual end-to-end test

A CI/CD pipeline has been set up by the INT team on GitLab to automate the testing, deployment, mounting, and configuration of our services. Our services have been registered in Airflow. This sets environment variables for automated and pipelined execution. During this test, our team measured statistics on each of our extraction services, such as size of the input/output files for each and average/median execution times. Table 4 lists this information for a sample collection of 3178 tweets, over 10 runs.

Table 4: Data sizes and execution times for extraction services

Service	Size before (bytes)	Size after (bytes)	Avg. time (s)	Median time (s)	Rate (tweets/s)
warc-json	1061442	13943466	1.5398	1.5612	2064
id	13943466	12804973	1.2687	1.2450	2505
username	12804973	12863213	1.7716	1.8040	1794
timestamp	12863213	12927700	1.4783	1.4815	2150
hashtags	12927700	12961397	2.3897	2.3975	1330
mentions	12961397	13026086	2.7703	2.8759	1147
geolocation	13026086	13062468	5.1614	5.2338	616
keywords	13062468	13100648	15.6159	15.6697	204

The sharp increase in size from the warc-json service is a result of decompressing the archive and extracting the tweet content. The slight decrease from the ID service is due to a reorganization of the intermediate .json, from record-oriented .json to column-oriented .json. Observing the rate at which individual services process tweets, the geolocation and keywords services appear to cause a bottleneck in the pipeline. The former examines nested structure in the metadata, while the latter must load the `gensim` library each time. Further analysis of these services may be needed, as well as optimization of the general approach such as vectorization and further parallelism.

6 Future Work

6.1 Indexing

Currently, 3.2k tweets are indexed in ELS. To index an existing collection (from the NFS storage), the team must update the environment variable for the warc-json service to point to the correct file and individually scale up each service along the pipeline. While the front-end allows for a user to upload their own collection, it does not allow a user to select an existing collection on the NFS. Either adding this functionality, or hooking directly into the INT team's service API to tweak/run the workflows requires additional work, but will automate the process of indexing collections.

6.2 Parallel workflows

Our services currently operate in series, however are designed/implemented in such a way that they could be run in parallel on the base cleaned tweets. Only the extract usernames, generate unique usernames, and the TwiRole services have any dependencies among other extraction services; all others are independent of each other. As previously discussed, a configuration for this would look something like Figure 6. Two changes would be required:

1. Environment variable configurations must be updated to instead use the base cleaned tweets from service 1 instead of chained from previous services.
2. The filter/merge service would need to be switched to the merge mode.

The latter has already been implemented and simply requires an environment variable update. In this mode, the filter/merge service takes file-field pairs, extracting a field per file into an single .json output to be indexed.

6.3 Feature-flag approach

The parallel option leads to another possible predicament: a user may only require a single field or a subset of fields from the full service set. In either our current configuration of our pipeline or in the parallel option, there isn't any mechanism for allowing a user to pick and choose which fields they want extracted. One possible extension is to have a single service take the base cleaned tweets .json file and a set of feature flags, then add a field for each record for each flag turned on, and finally output a result file with the desired fields only. This approach would require more extensive modifications to the codebase we are providing, along with some design reconsiderations for service/workload division.

6.4 Advanced geo-location inference

Our current implementation of geo-location extraction pulls from explicit location information stored in the metadata of each tweet. A more complex implementation could involve using implicit geo-location information, such as timezone, user profile location, locations mentioned in tweets, etc.

6.5 DMI-TCAT

The third format of tweets is unaccounted for in our services. The complexities of accessing a sample collection in this format exceeded the time available to our team during the semester. Detailed examination of the collection structure would be required (to find existing extracted fields as well as the format-specific keys for the fields in other services).

7 User's Manual

Our subsystem has been developed to be accessed completely from the front-end interface. This user manual outlines what information is available to a user and how they can filter the set of tweets according to their query. Finally, it gives examples of the data available and what data input and output will look like.

7.1 Front-end interface design and usage

See the FE team's report for further details on the tweet collection interface and how to utilize it.

7.2 Data origin

Currently accessible collections have been pulled from Social Feed Manager (SFM) and yourTwapperKeeper (YTK). If a user would like to upload their own data, the output format must follow one of these two applications.

7.3 Available metadata

The following is information available for users to view and query against, with a short description and how a query on that field will change the results they are shown:

- Username: The username associated with the tweet. Output will consist of tweets that originated from the queried username.
- Timestamp: The posted timestamp for each tweet. Output will consist of tweets in the given date/time range.
- Hashtags: Hashtags from the tweet content. Output will consist of tweets that contain the queried hashtag.
- Mentions: Usernames from the tweet content (i.e., tagged users). Output will consist of tweets that contain the queried username.
- Geo-location: Explicitly mentioned location information. Output will consist of tweets that contain the queried geo-location.
- Keywords: Relevant keywords based on each tweet's textual content. Output will consist of tweets that contain the queried keywords.
- TwiRole: A classification of (the user of) the tweet as a male, female, or brand. Output will consist of tweets that fall under the given categorization (male, female, brand).

7.4 Custom collections and running workflows

The front-end interface also allows curators to run workflows and our services with their own collections of tweets. In the future, the interface will also allow administrators to run workflows on existing collections of tweets, to more easily index the 5 billion tweets available to the team. Note: more detail about the front-end team's future work can be found in their report.

8 Developer's Manual

8.1 Prerequisites

For deploying and integrating new features in the services, the developer will require a running installation of Python 3, the required Python libraries, and Docker on a local system. To install Python, simply download the installer from `www.python.org` and use the recommended installation method. It will install all the required programs associated with Python for compiling code and installing new modules. Make sure to check the add Python to the path box to allow accessing commands like Python and pip from the terminal/command prompt. More tutorials and guides are found on their respective documentation pages [14].

For installing Docker, go to `www.docker.com`, create an account, and click on the link that says *Get Started with Docker Desktop*. This will guide you through installation of the Docker Client and a sample build to get you familiar with the application. If the need arises to explore different base images for the services, you can sign in at `hub.docker.com` to see the entire list of official images supported by Docker. More documentation can be found at [7].

8.2 Creating datasets

A developer will need to have a local installation of Docker and a Twitter Developer API key. Social Feed Manager and yourTwapperKeeper are the two applications that can be used to create your own collections. Social Feed Manager is developed by the George Washington University Libraries with the intention to empower institutions, students, and social media researchers to define and collect datasets from social media services. yourTwapperKeeper is a open source version of TwapperKeeper, created by John O'Reilly as a simple and easy way to archive data from Twitter directly on your server. Both the services can be used to collect data from the Twitter Streaming API as well as the Twitter Search API and export data in HTML, RSS, EXCEL, and JSON formats.

Installation documentation for the **Social Feed Manager** can be found on their documentation page [12]. The source code for **yourTwapperKeeper** can be found on their Github repository [2]. The documentation on procuring a Twitter Developer API key can be found on their respective documentation page [16].

8.3 Services

The source code for our services can be cloned from the TWT team's GitLab repository. All the services are located in the services/<service name> directory. To build the services, open the required directory in a terminal and use the *docker build -t <service name> .* command. See Figure 10 for reference. The Docker container can be run by using the container ID generated by the *docker build* command. Use the *docker run -it <service name>*. See Figure 11 for reference. To modify any service, you simply need to open the directory in a code editor and modify the *.py* source file to change the service according to your needs.

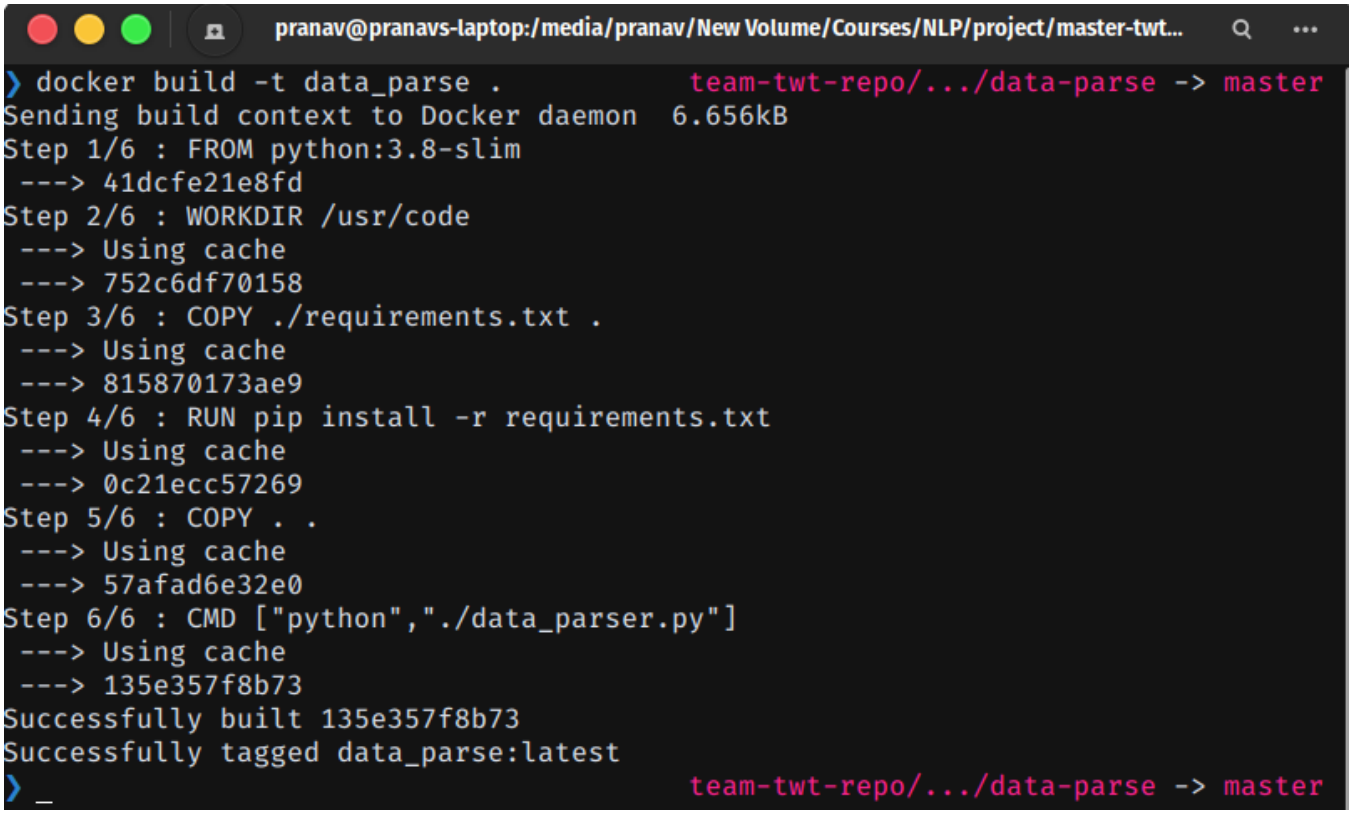

Figure 10: Building a Docker container for the data-parse service

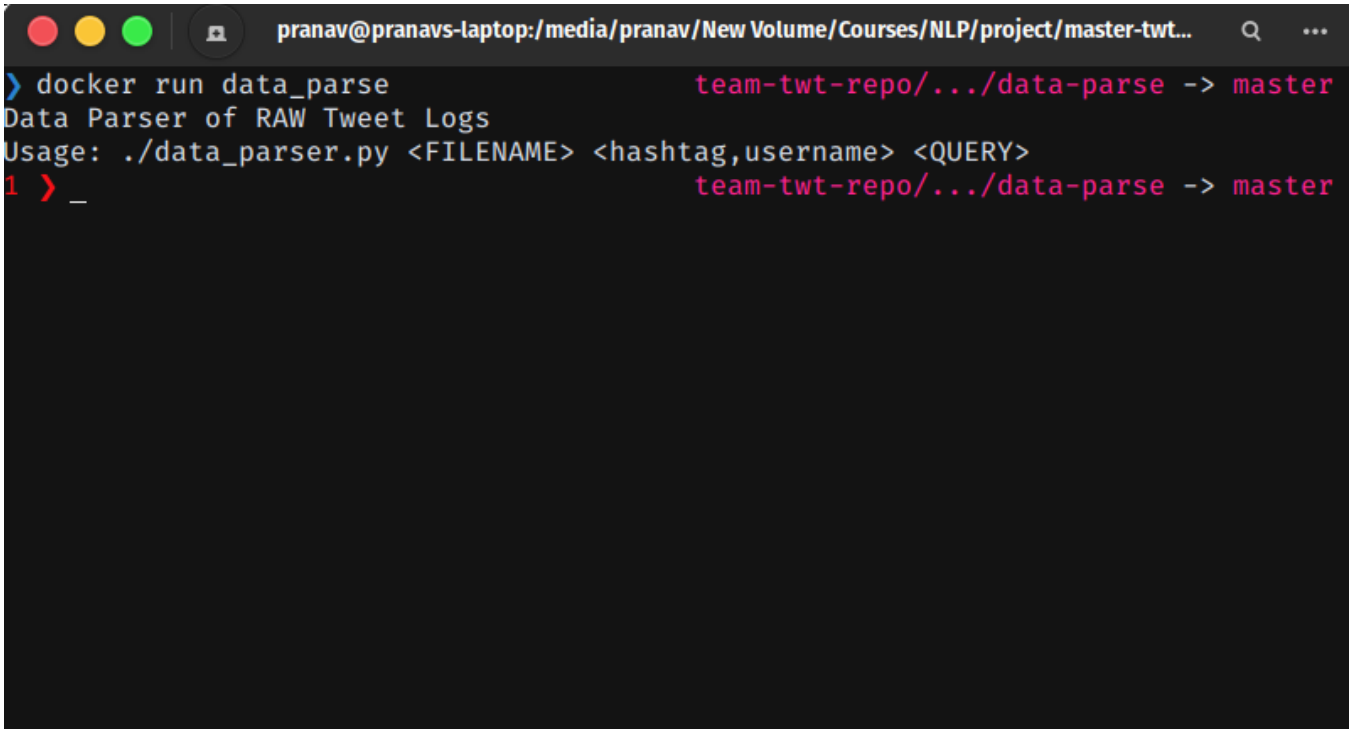

Figure 11: Running the Docker container for each service

To add any new service, go to the services directory and create a new folder for each service to be added. The folder would contain 3 files: the Dockerfile to build the Docker container, the source code for the service, and the

requirements file to install the prerequisite modules for the service. See Figure 12 for reference.

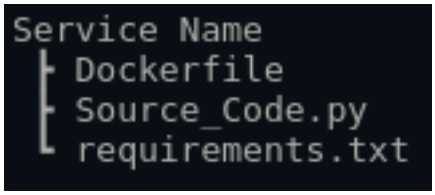

Figure 12: The file tree of a service

The Dockerfile should consist of the "python" base image and should follow the same format as the other services. See Figure 13 for reference.

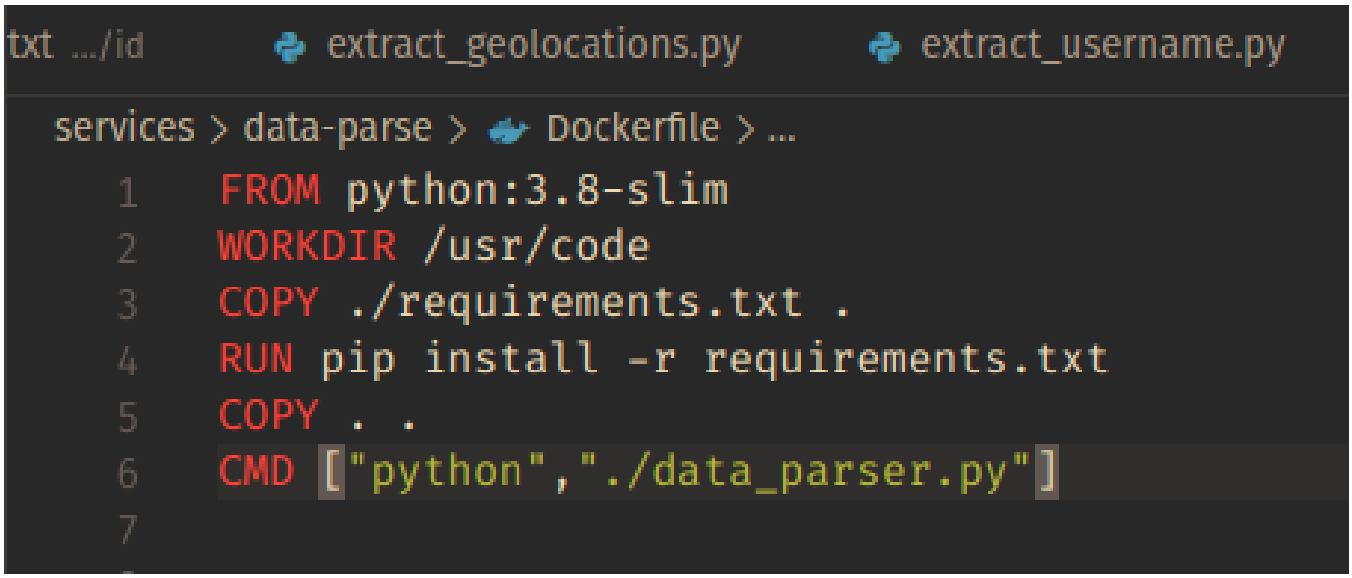

Figure 13: Dockerfile of the ElasticSearch export service

8.4 Service deployment

These services must be uploaded to the GitLab Container Registry to be deployed; see Figure 14 for reference. This can be done by first logging into the Docker container registry by typing the command *docker login container.cs.vt.edu*. This will prompt you to insert your CS CAS login credentials. Now, you need to make sure that you have the container built with a specific alias. This can be done by using the command *docker build -t container.cs.vt.edu/cs-5604-fall-2020/twt/team-twt-repo/<service name>* ., where the string following *-t* argument parameter is the alias for the container. Once the container image has been created, you need to push it to the registry using the command *docker push <service name>*, see Figure 15 for reference.

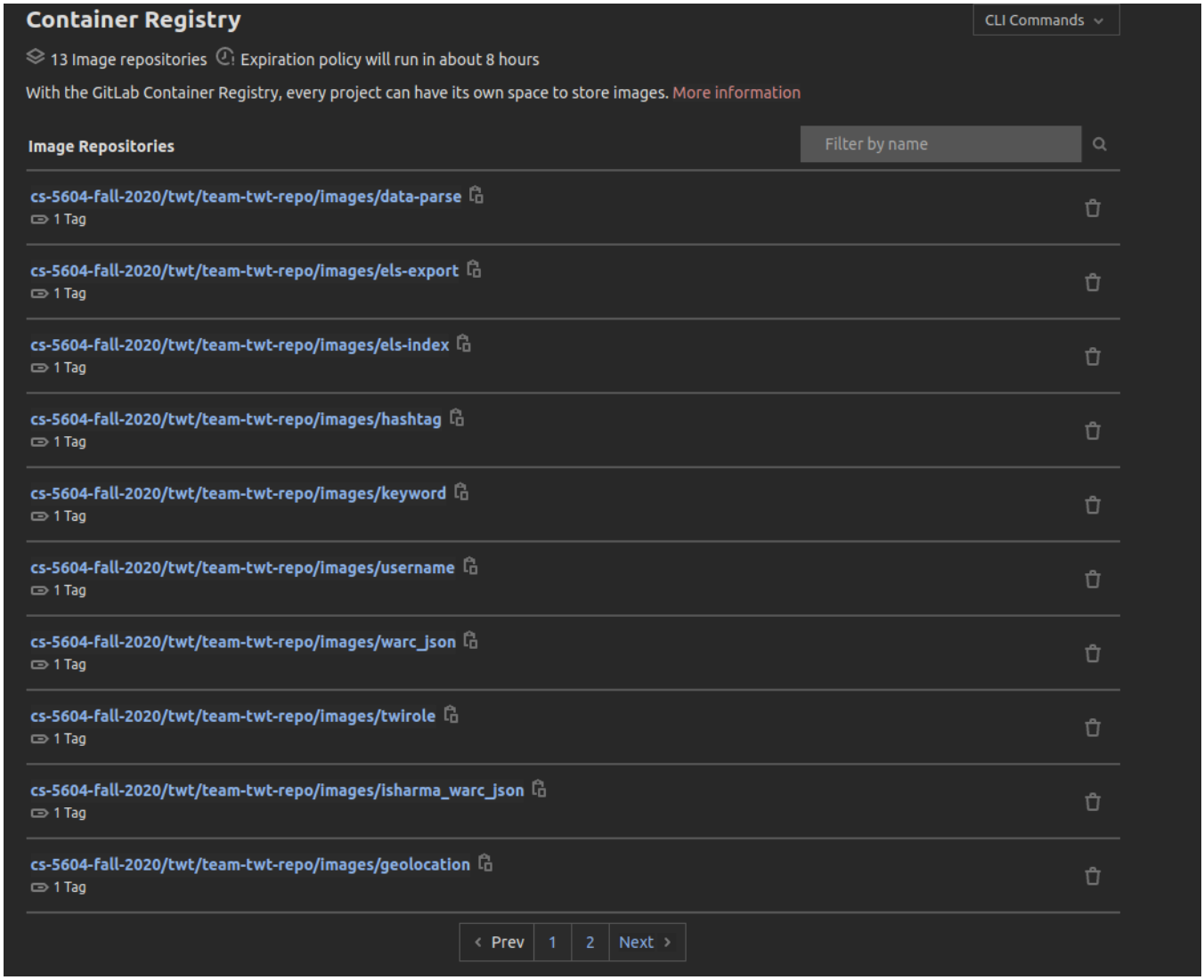

Figure 14: GitLab container registry

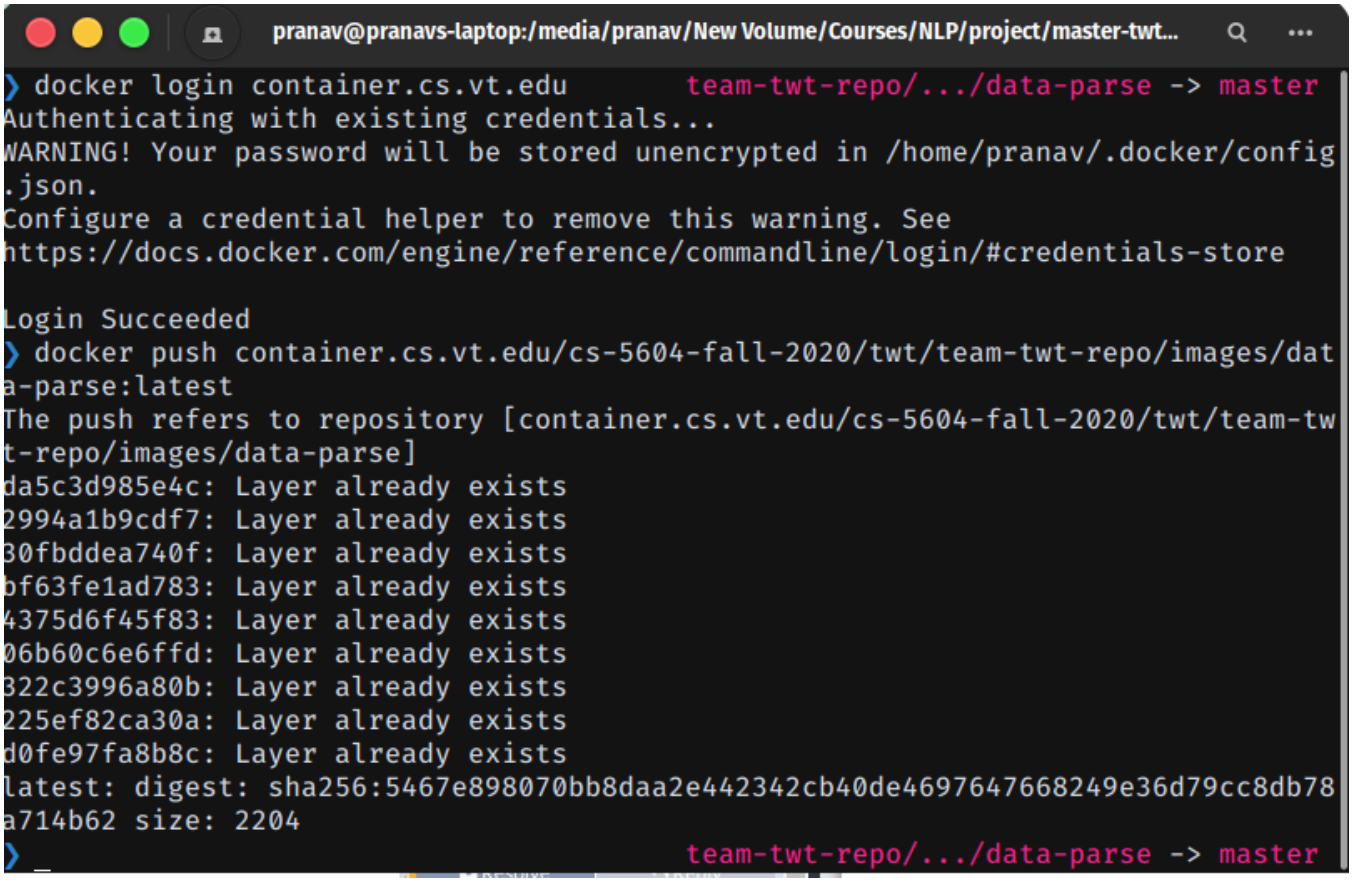

Figure 15: Uploading a Docker image to the GitLab container registry

To deploy a service, you need to generate a deploy token so that any deployment through GitLab is authenticated. You do so by going to *settings>repositories* and expanding on the *deploy tokens* tab. Fill in details like the name, expiry date (can be left blank for never), and scope of the token; see Figure 16 for reference. Once this has been created, a username and password will be generated. This username and password is visible only once, hence if lost will require a creation of a new token.

Go to your namespace on cloud.cs.vt.edu and navigate to *Resources>Secrets>Registry Credentials* and click on *Add Registry*. Fill in the details of the deploy token that you generated in the previous step; see Figure 17 for reference. Once the registry has been added, the service can be deployed by going to your *Resources>Workloads* and clicking on deploy. Fill in details as shown in Figure 18. Go to the *Volumes* tab and select *Use existing persistent volume* and go to Claim persistent volume and choose *camelot-cs5604* and give mount location as */mnt/camelot-cs5604*. Repeat the step to add *camelot-dlrl* as another volume with mount point */mnt/camelot-dlrl*; see Figure 19 for reference.

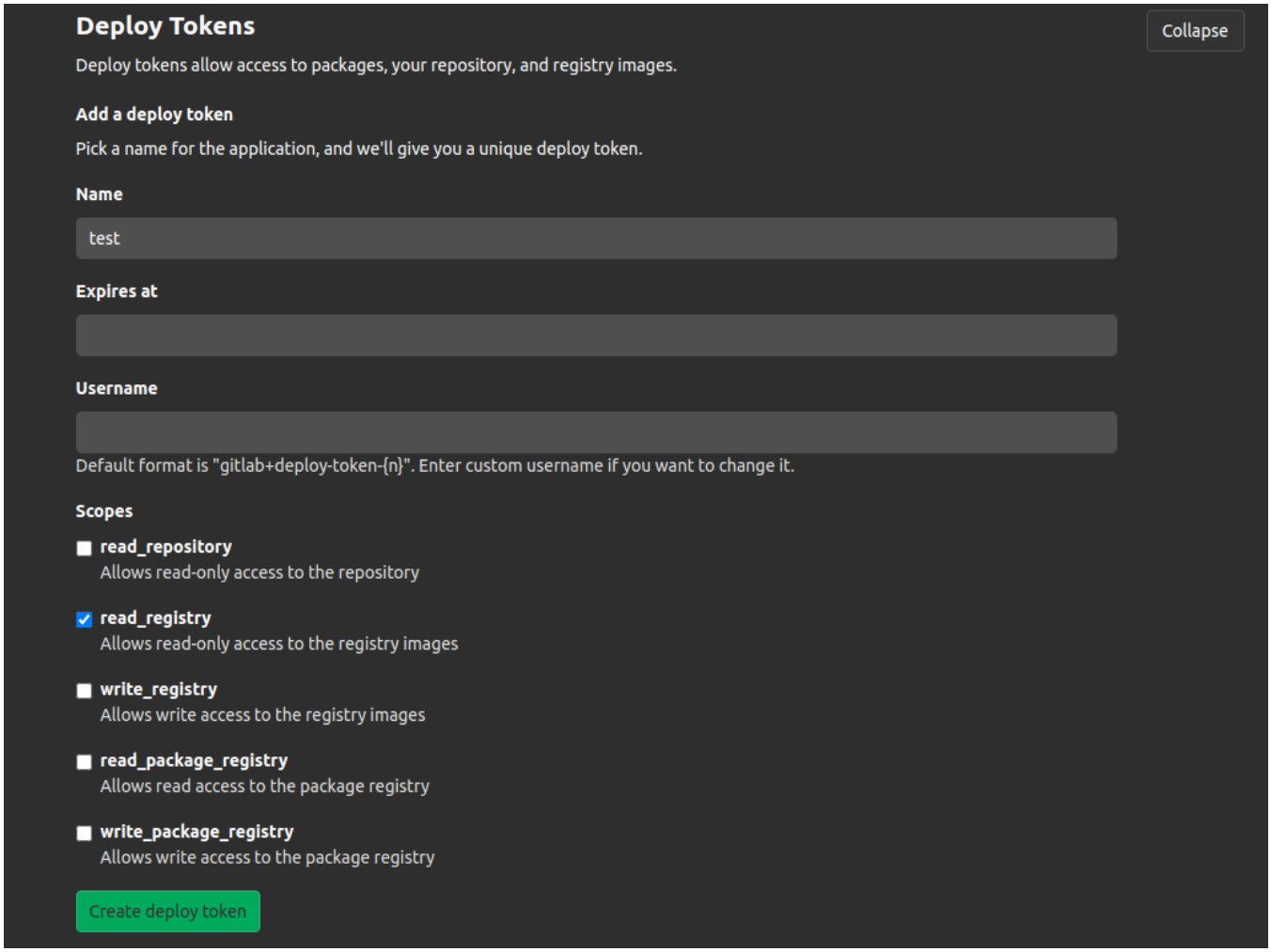

Figure 16: Generating a deploy token

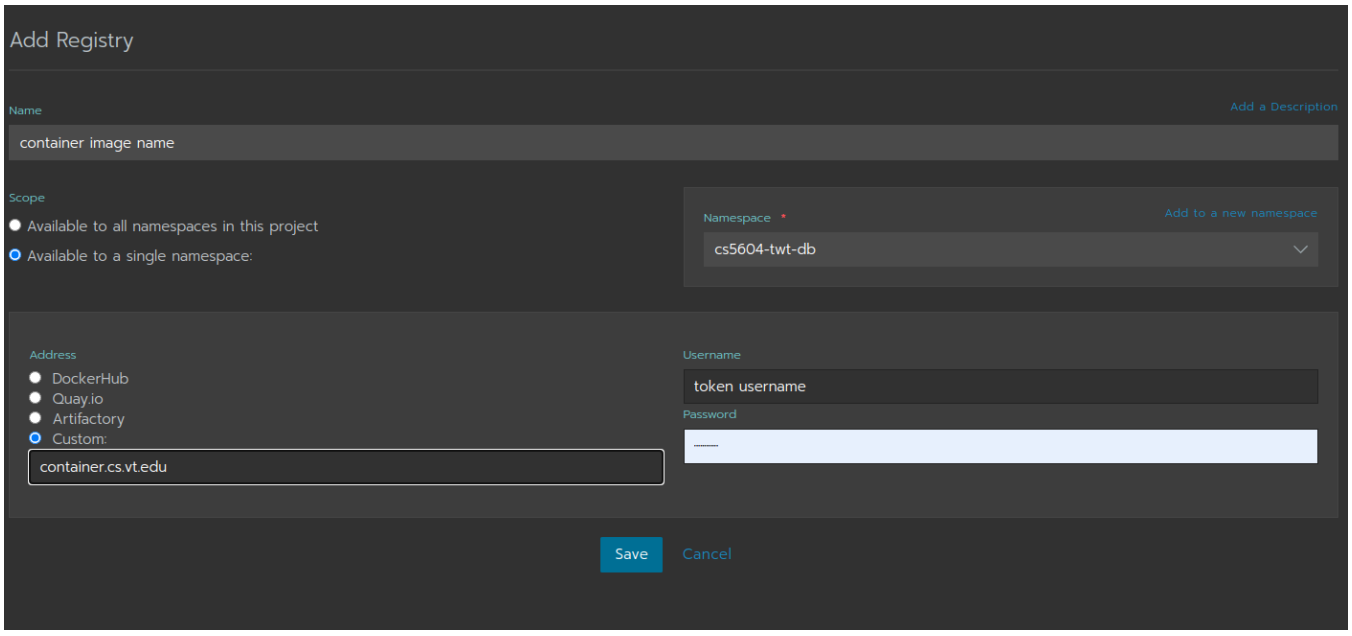

Figure 17: Adding the GitLab registry to the CS cluster

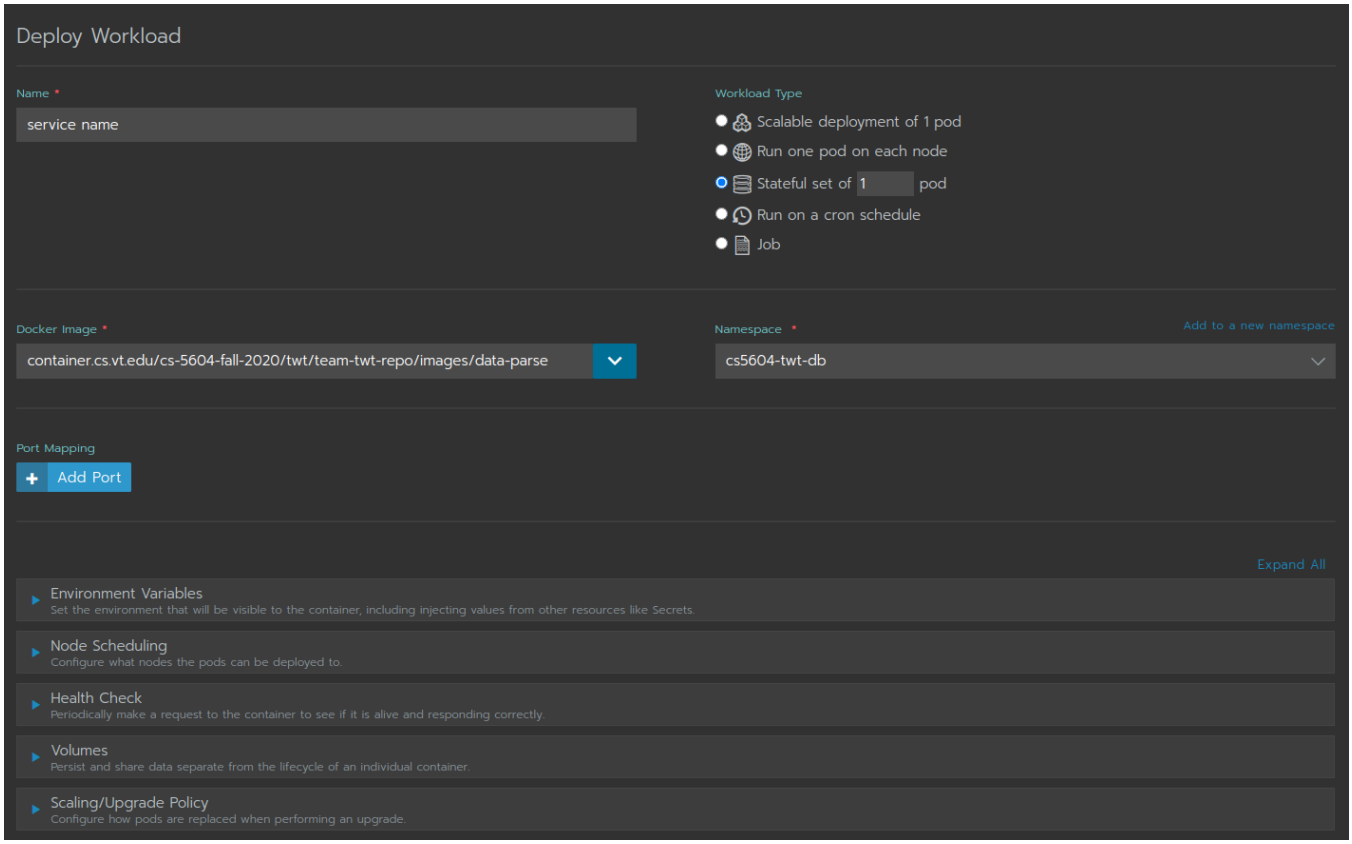

Figure 18: Deploying a image of a service

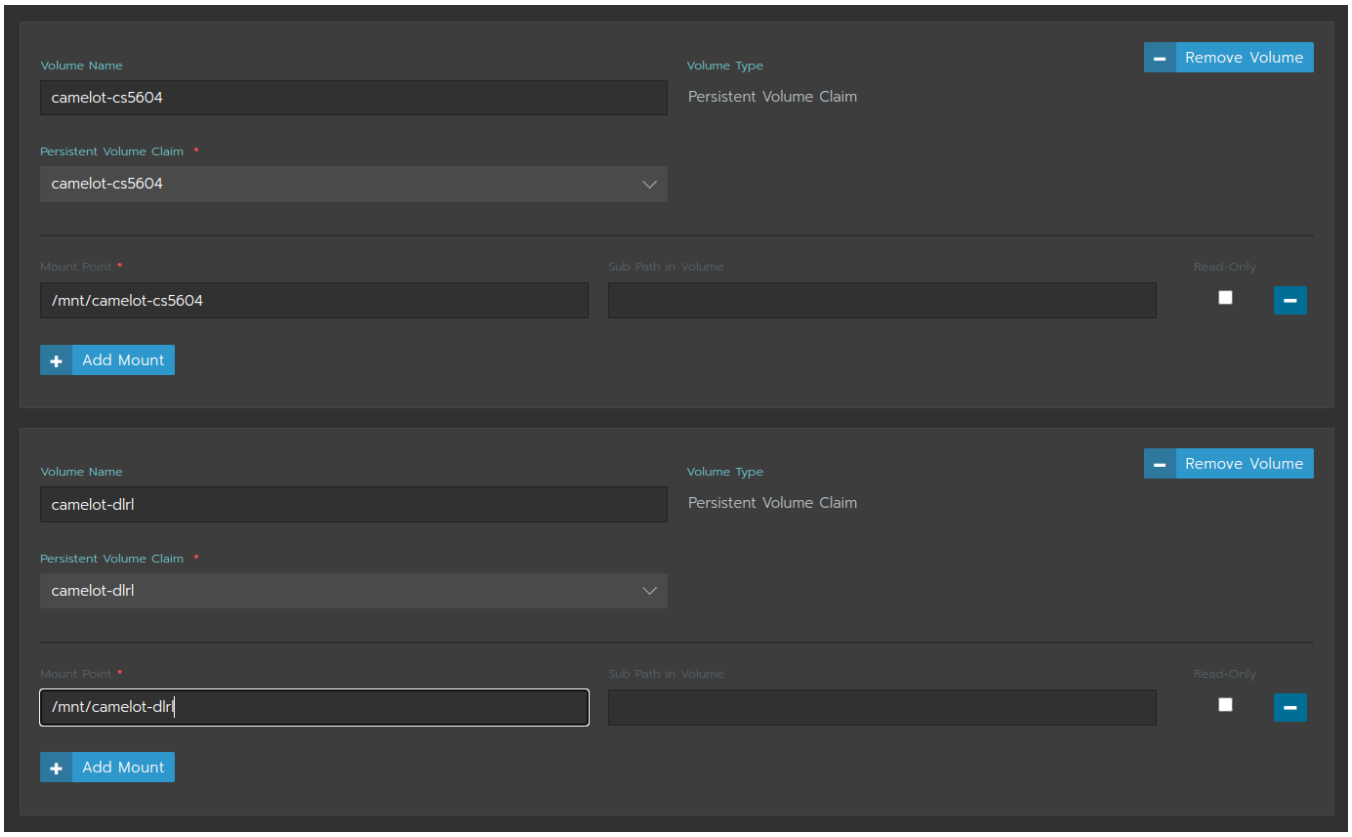

Figure 19: Mounting volumes to a service

References

[1] Erik Borra (https://github.com/ErikBorra). *GitHub - Digital Methods Initiative - Twitter Capture and Analysis Toolset*. Aug. 2011. URL: https://github.com/digitalmethodsinitiative/dmi-tcat (visited on 10/19/2020).

[2] John OBrien III (https://github.com/jobrieniii). *GitHub - 540co/yourTwapperKeeper: yourTwapperKeeper - Archive Your Social Media*. Oct. 2011. URL: https://github.com/540co/yourTwapperKeeper (visited on 09/15/2020).

[3] NumFOCUS (https://numfocus.org/). *Dask: Scalable analytics in Python*. 2019. URL: https://dask.org/ (visited on 11/01/2020).

[4] Matthew Bock. "A Framework for Hadoop Based Digital Libraries of Tweets". MS thesis. Department of Computer Science, Blacksburg, Virginia 24061: Virginia Polytechnic Institute and State University, July 2017. URL: http://hdl.handle.net/10919/78351 (visited on 09/15/2020).

[5] Prashant Chandrasekar. "Process to build the 'right' system". In: *Presentation for CS5604, Information Storage and Retrieval* (Oct. 2020). URL: https://canvas.vt.edu/courses/115585/files/folder/2020/Presentations?preview=14442772 (visited on 10/07/2020).

[6] Library of Congress. *Sustainability of Digital Formats: Planning for Library of Congress Collections*. Digital Preservation at the Library of Congress. Aug. 2009. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml (visited on 10/28/2020).

[7] Docker. *Documentation Home — Docs — Docker*. 2020. URL: https://docs.docker.com/ (visited on 12/04/2020).

[8] Sunshin Lee. "Geo-Locating Tweets with Latent Location Information". PhD thesis. Department of Computer Science, Blacksburg, Virginia 24061: Virginia Polytechnic Institute and State University, Feb. 2017. URL: http://hdl.handle.net/10919/75022 (visited on 10/07/2020).

[9] Liuqing Li. *GitHub - TwiRole: A Hybrid Model for Role-related User Classification on Twitter*. Apr. 2020. URL: https://github.com/liuqingli/TwiRole (visited on 10/07/2020).

[10] Liuqing Li et al. "A Hybrid Model for Role-related User Classification on Twitter". PhD thesis. Department of Computer Science, Blacksburg, Virginia 24061: Virginia Polytechnic Institute and State University, Nov. 2018. URL: http://hdl.handle.net/10919/86162 (visited on 10/07/2020).

[11] Yuan Li et al. *Final Report CS 5604: Information Storage and Retrieval*. Department of Computer Science, Blacksburg, Virginia 24061: Virginia Polytechnic Institute and State University, Dec. 2019. URL: http://hdl.handle.net/10919/96310 (visited on 09/14/2020).

[12] George Washington University Libraries. *Social Feed Manager (SFM)*. 2015. URL: https://gwu-libraries.github.io/sfm-ui/ (visited on 09/15/2020).

[13] Emma Meno and Kyle Vincent. *Twitter-Based Knowledge Graph for Researchers (CS4624 team project)*. Department of Computer Science, Blacksburg, Virginia 24061: Virginia Polytechnic Institute and State University, May 2020. URL: http://hdl.handle.net/10919/98239 (visited on 09/15/2020).

[14] Python.org. *Documentation Home — Docs — Python Developer*. 2020. URL: https://www.python.org/doc/ (visited on 12/04/2020).

[15] Radim Řehůřek. *Gensim - Text Summarization*. 2009. URL: https://radimrehurek.com/gensim/ (visited on 10/07/2020).

[16] Twitter. *Documentation Home — Docs — Twitter Developer*. 2020. URL: https://developer.twitter.com/en/docs (visited on 09/15/2020).

[17] Webrecorder. *WARCIO: WARC (and ARC) Streaming Library*. 2017. URL: https://github.com/webrecorder/warcio#warcio-warc-and-arc-streaming-library (visited on 10/21/2020).

Appendices

A Per-goal breakdown into data/tasks

Note that the DataFrames listed for input/output are used during processing, but Docker requires us to write to intermediary JSON files to pass data between services.

Table A.1: Goal 1 (extracting geo-location) data table

Data ID	Data name/description
1	Raw data
2	Cleaned data in DataFrame format
3	Geo-names (if found) for each tweet added to the new column in dataFrame

Table A.2: Goal 1 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json, .parquet, etc.)	2. DataFrame of tweets
2	Add location field	2. DataFrame of tweets	3. DataFrame of tweets + location field

Table A.3: Goal 2 (extracting hashtags) data table

Data ID	Data name/description
1	Raw data
2	Cleaned data in DataFrame format
3	Hashtags for each tweet added to the new column in dataFrame

Table A.4: Goal 2 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json, .parquet, etc.)	2. DataFrame of tweets
2	Add Hashtag field	2. DataFrame of tweets	3. DataFrame of tweets + hashtag field

Table A.5: Goal 3 (extracting username) data table

Data ID	Data name/description
1	Raw data
2	Data cleaned into DataFrame format
3	Additional column in dataframe containing originating username for corresponding tweets

Table A.6: Goal 3 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json, .parquet, etc.)	2. DataFrame of tweets
2	Add Username field	2. DataFrame of tweets	3. DataFrame of tweets + field of corresponding originating usernames

Table A.7: Goal 4 (extracting username mentions) data table

Data ID	Data name/description
1	Raw data
2	Data cleaned into DataFrame format
3	Additional column in dataframe containing username mentions for corresponding tweets

Table A.8: Goal 4 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json, .parquet, etc.)	2. DataFrame of tweets
2	Add Username mentions field	2. DataFrame of tweets	3. DataFrame of tweets + field of corresponding usernames mentions

Table A.9: Goal 5 (exporting a sub-collection) data table

Data ID	Data name/description
1	DataFrame of tweets
2	Filename for exported sub-collection of tweets

Table A.10: Goal 5 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Export tweets	1. DataFrame of tweets	2. Filename for exported sub-collection of tweets

Table A.11: Goal 6 (importing a sub-collection) data table

Data ID	Data name/description
1	Filename for imported sub-collection of tweets
2	Filename for raw tweets

Table A.12: Goal 6 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Import tweets	1.Filename for imported sub-collection of tweets	2. Filename for raw tweets

Table A.13: Goal 7 (extracting keywords) data table

Data ID	Data name/description
1	Raw data
2	Cleaned data in DataFrame format
3	Additional column in DataFrame containing keywords extracted from tweets

Table A.14: Goal 8 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json/.parquet)	2. DataFrame of tweets
2	Add keyword field	2. DataFrame of tweets	3. DataFrame of tweets + keyword field

Table A.15: Goal 8 (extracting user-classification information) data table

Data ID	Data name/description
1	Raw data
2	Cleaned data in DataFrame format
3	Additional column in DataFrame containing TwiRole info extracted from tweets

Table A.16: Goal 8 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json/.parquet)	2. DataFrame of tweets
2	Add TwiRole field	2. DataFrame of tweets	3. DataFrame of tweets + TwiRole field

Table A.17: Goal 9 (extracting timestamp information) data table

Data ID	Data name/description
1	Raw data
2	Cleaned data in DataFrame format
3	Additional column in DataFrame containing the timestamp for corresponding tweets

Table A.18: Goal 9 tasks table

Task ID	Task Description	Input data ID(s)	Output data ID
1	Filter raw data	1. Filename of file containing raw tweets (in .json/.parquet)	2. DataFrame of tweets
2	Add timestamp field	2. DataFrame of tweets	3. DataFrame + field of corresponding timestamps

B Sample SFM tweet metadata


```
{
    "quote_count": 0,
    "contributors": null,
    "truncated": true,
    "text": "The phoney quarantine is almost over. Bring on the real quarantine. #covid_19 #coronavirus
    ↪  #freebritney... https://t.co/BJTCp8fI6y",
    "is_quote_status": false,
    "in_reply_to_status_id": null,
    "reply_count": 0,
    "id": 1257271308062208000,
    "favorite_count": 0,
    "entities": {
        "user_mentions": [],
        "symbols": [],
        "hashtags": [
            {
                "indices": [
                    68,
                    77
                ],
                "text": "covid_19"
            },
            {
                "indices": [
                    78,
                    90
                ],
                "text": "coronavirus"
            },
            {
                "indices": [
                    91,
                    103
                ],
                "text": "freebritney"
            }
        ],
        "urls": [
            {
                "url": "https://t.co/BJTCp8fI6y",
                "indices": [
                    105,
                    128
                ],
                "expanded_url": "https://twitter.com/i/web/status/1257271308062208000",
                "display_url": "twitter.com/i/web/status/1..."
            }
        ]
    },
    "retweeted": false,
    "coordinates": {
        "type": "Point",
        "coordinates": [
            -80.25,
            43.55
        ]
    },
    "timestamp_ms": "1588591813470",
    "source": "<a href=\"http://instagram.com\" rel=\"nofollow\">Instagram</a>",
    "in_reply_to_screen_name": null,
    "id_str": "1257271308062208000",
    "retweet_count": 0,
    "in_reply_to_user_id": null,
    "favorited": false,
    "user": {
        "follow_request_sent": null,
        "profile_use_background_image": true,
        "default_profile_image": false,
```

```
        "id": 1096083799534960600,
        "default_profile": true,
        "verified": false,
        "profile_image_url_https": "https://pbs.twimg.com/profile_images/1251785729518288897/csccyZlo_normal.jpg",
        "profile_sidebar_fill_color": "DDEEF6",
        "profile_text_color": "333333",
        "followers_count": 125,
        "profile_sidebar_border_color": "C0DEED",
        "id_str": "1096083799534960640",
        "profile_background_color": "F5F8FA",
        "listed_count": 0,
        "profile_background_image_url_https": "",
        "utc_offset": null,
        "statuses_count": 1828,
        "description": "The subject who is truly loyal to the Chief Magistrate will neither advise nor submit
        ↪  to arbitrary measures.˜˜ Junius",
        "friends_count": 428,
        "location": "Dawn-Euphemia, Ontario",
        "profile_link_color": "1DA1F2",
        "profile_image_url": "http://pbs.twimg.com/profile_images/1251785729518288897/csccyZlo_normal.jpg",
        "following": null,
        "geo_enabled": true,
        "profile_banner_url": "https://pbs.twimg.com/profile_banners/1096083799534960640/1587283898",
        "profile_background_image_url": "",
        "name": "Mafun Ho",
        "lang": null,
        "profile_background_tile": false,
        "favourites_count": 15588,
        "screen_name": "Tumulus17",
        "notifications": null,
        "url": null,
        "created_at": "Thu Feb 14 16:28:36 +0000 2019",
        "contributors_enabled": false,
        "time_zone": null,
        "protected": false,
        "translator_type": "none",
        "is_translator": false
    },
    "geo": {
        "type": "Point",
        "coordinates": [
            43.55,
            -80.25
        ]
    },
    "in_reply_to_user_id_str": null,
    "possibly_sensitive": false,
    "lang": "en",
    "extended_tweet": {
        "display_text_range": [
            0,
            160
        ],
        "entities": {
            "user_mentions": [],
            "symbols": [],
            "hashtags": [
                {
                    "indices": [
                        68,
                        77
                    ],
                    "text": "covid_19"
                },
                {
                    "indices": [
                        78,
                        90
                    ],
                    "text": "coronavirus"
                },
```

```
                {
                    "indices": [
                        91,
                        103
                    ],
                    "text": "freebritney"
                },
                {
                    "indices": [
                        104,
                        118
                    ],
                    "text": "wrayandnephew"
                }
            ],
            "urls": [
                {
                    "url": "https://t.co/PF3a1rtMMG",
                    "indices": [
                        137,
                        160
                    ],
                    "expanded_url": "https://www.instagram.com/p/B_w6qFEnTYF/?igshid=1kb2euuf2aszb",
                    "display_url": "instagram.com/p/B_w6qFEnTYF/..."
                }
            ]
        },
        "full_text": "The phoney quarantine is almost over. Bring on the real quarantine. #covid_19 #coronavirus
        ↪  #freebritney #wrayandnephew @ Guelph, Ontario https://t.co/PF3a1rtMMG"
    },
    "created_at": "Mon May 04 11:30:13 +0000 2020",
    "filter_level": "low",
    "in_reply_to_status_id_str": null,
    "place": {
        "full_name": "Guelph, Ontario",
        "url": "https://api.twitter.com/1.1/geo/id/2740624a2d391c5c.json",
        "country": "Canada",
        "place_type": "city",
        "bounding_box": {
            "type": "Polygon",
            "coordinates": [
                [
                    [
                        -80.326879,
                        43.473802
                    ],
                    [
                        -80.326879,
                        43.594596
                    ],
                    [
                        -80.153377,
                        43.594596
                    ],
                    [
                        -80.153377,
                        43.473802
                    ]
                ]
            ]
        },
        "country_code": "CA",
        "attributes": {},
        "id": "2740624a2d391c5c",
        "name": "Guelph"
    }
}
```


Figure B. 1: Sample Twitter data from Social Feed Manager