Final Report
CS 5604: Information Storage and Retrieval

Webpages (WP) Team:
Jostein Barry-Straume, Cristian Vives,
Wentao Fan, Peng Tan, Shuaicheng Zhang,
Yang Hu, Tishauna Wilson

December 18, 2020

Instructed by Professor Edward A. Fox

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

The first major goal of this project is to build a state-of-the-art information retrieval engine for searching webpages and for opening up access to existing and new webpage collections resulting from Digital Library Research Laboratory (DLRL) projects relating to eventsarchive.org.

The task of the Webpage (WP) team was to provide the functionality of making any archived webpage accessible and indexed. The webpages can be obtained either through event focused crawlers or collections of data, such as WARC files containing webpages, or sets of tweets which contains embedded URLs. Toward completion of the project, the WP team worked on four major tasks: 1.) Contents of WARC files searchable through ElasticSearch. 2.) Contents of WARC files cleaned and searchable through ElasticSearch. 3.) Event focused crawler running and producing WARC files. 4.) Additional extracted/derived information (e.g., dates, classes) made searchable.

The foundation of the software is a Docker container cluster employing Airflow, a Reasoner, and Kubernetes. The raw data of the information content of the given webpage collections is stored using the Network File System (NFS), while Ceph is used for persistent storage for the Docker containers. Retrieval, analysis, and visualization of the webpage collection is carried out with ElasticSearch and Kibana, respectively. These two technologies form an Elastic Stack application which serves as the vehicle with which the WP team indexes, maps, and stores the processed data and model outputs with regards to webpage collections.

The software is co-designed by 7 team members of Virginia Tech graduate students, all members of the same computer science class, CS 5604: Information Storage and Retrieval. The course is taught by Professor Edward A. Fox. Dr. Fox structures the class in a way for his students to perform in a "mock" business development setting. In other words, the academic project submitted by the WP team for all intents and purposes can be viewed as a microcosm of software development within a corporate structure.

This submission focuses on the work of the WP team, which creates and administers Docker containers such that various services are tested and deployed in whole. Said services pertain solely to the ingestion, cleansing, analysis, extraction, classification, and indexing of webpages and their respective content.

Contents

List of Tables **5**

List of Figures **6**

1 Overview **7**
1.1 Project Management . . . 7
1.2 Problems and Challenges . . . 7
1.3 Solutions Developed . . . 8
1.4 Future Work . . . 9

2 Literature Review **12**
2.1 Intelligent Event Focused Crawling . . . 12
2.2 Chapters 19-21 in Introduction to Information Retrieval Course Book . . . 12
2.3 Representational State Transfers (REST) . . . 12
2.4 RESTful API . . . 13
2.5 Text classification . . . 14
2.6 Text summarization . . . 14

3 Requirements **16**
3.1 Overall Project Requirements . . . 16
3.2 WP Team Requirements . . . 16

4 Design **18**
4.1 Approach . . . 18
4.2 Background Information and Methodology . . . 21
4.3 Tools . . . 22
4.3.1 Docker . . . 22
4.3.2 ElasticSearch . . . 22
4.3.3 JSON . . . 23
4.3.4 WARCIO . . . 23
4.3.5 BeautifulSoup . . . 23
4.3.6 WARC files . . . 23
4.4 Deliverables . . . 24

5 Implementation **26**
5.1 Timeline . . . 26
5.2 Milestones and Deliverables . . . 28
5.3 Methods Selection . . . 30

6 User Manual **32**
6.1 Service Utilization . . . 32

7 Developer Manual **34**
7.1 Input . . . 34
7.2 Extract URLs . . . 34
7.3 Archive Webpages . . . 38
7.4 Extract Data . . . 40
7.5 Text Classification . . . 41
7.6 Text Summarization . . . 43
7.7 Indexing . . . 45
7.8 Unit testing . . . 47
7.9 Accessing the Virtual Machine . . . 49
7.10 Service Registration . . . 50

Bibliography **55**

List of Tables

1 Tasks and Timeline 26
2 Milestones 28
3 Deliverables 29
4 Containers and their function in the System 29

List of Figures

1 Rancher Errors . . . 9
2 Deployed Services on Rancher . . . 10
3 YAML File for CI/CD Pipeline . . . 11
4 WP Task Approach . . . 18
5 Extract URL Flowchart . . . 19
6 Archive Webpage Flowchart . . . 19
7 Extract Webpage Data Flowchart . . . 20
8 Index Flowchart . . . 20
9 Example of a WARC File . . . 24
10 Example of index structure from ElasticSearch that the Index service creates . . . 31
11 WP Team Container Registry . . . 32
12 Part of EXTRACT URL Code . . . 34
13 Part of Extract URL Code . . . 35
14 Output of Generate URL . . . 37
15 Example JSON Data . . . 38
16 Archive URL Input Data . . . 39
17 Output of Extract Data . . . 41
18 Text Classification Training . . . 42
19 Text Classification Examples . . . 43
20 Before Summarizing . . . 45
21 After Summarizing . . . 45
22 Input of ElasticSearch Data . . . 46
23 Code of ElasticSearch . . . 46
24 ElasticSearch Index Output Data Sample . . . 47
25 Unit Test Passed Examples . . . 48
26 Unit Test Failed Examples . . . 49
27 Using SSH to Connect to VPN . . . 50
28 Service Table . . . 51
29 Goal Table . . . 53
30 Reasoner Table . . . 54

1 Overview

1.1 Project Management

The web team (WP) is responsible for collecting, processing and storing webpages from different sources. The main goal of our team is to ingest, clean, analyze, extract, classify, and index webpages. The sources include URLs obtained using "crawlers," tweets containing URL information from the TWT team, WARC files, and other archives collected by Mohamed Farag. For the first portion of the project, we spent much time on understanding related background and knowledge about this work. We strove to develop an understanding of our duties as they relate to the other teams. As such, we spent the first few weeks of this project familiarizing ourselves with the related technologies, and reading related literature.

The WP (webpage) team consists of seven members, so having proper team management was crucial to ensure success of the project. In order to achieve this, the team held weekly meetings and stand-up meetings before every class. The stand-up meetings were especially helpful, since it gave team members time to talk about what they had accomplished and how they would move on with work related to the project. Discord was also commonly used to establish proper communication amongst the team members. Furthermore, Discord facilitated communication with members of other teams, which was relatively helpful to the ElasticSearch members to understand how to proceed with the project and to understand how the data should be indexed.

Due to the nature of the project and the many people involved, the WP team used a divide-and-conquer strategy, resulting in the formation of four sub-teams:

- Extract URLs
- Archive webpages
- Extract data
- Indexing

Also, Mohamed was a great point of contact, and meetings were established biweekly to seek feedback and evaluation on the project's progress. Class meetings were also used to meet with the team and brainstorm ideas with other teams.

1.2 Problems and Challenges

We have faced a few challenges including:

A. **Collect raw data**: Databases of different media contain a large amount of data. Identifying and accessing a sample collection of documents of a given topic is crucial when starting the prototyping for this project.

B. **Extract embedded links from data**: Raw data cannot explain anything directly to the user. Specific conditions must be set to grab useful information from it, no matter whether they are links, keywords, etc.

C. **Extract relevant texts from available links**: Extracting raw webpage text will rarely result in proper indexed data. Instead, protocols must be set in place to accurately isolate key words and key text. Extracting those accurately from a large text corpus is crucial to the success of the project.

D. **Understanding nature of data**: Since the WP team is new, there are challenges understanding what the output data (i.e., what is being indexed) should look like.

E. **Port existing solution to deployment**: A lot of the code written right now works well locally. It will be necessary to port many of these solutions to their own Docker containers running on a server that will contain many of the services required by the overall project.

F. **Approach to Dockerization philosophy**: Understanding the difference between a task and a service has been an ongoing issue throughout the project. Thus, knowing what should be a service and what shouldn't be has affected what code and pipeline we Dockerize.

G. **Using an abstract indexing service**: There was a lot of debate on how the indexing should be done, whether it was an integrated service in the WP pipeline, an independent service developed by the WP team, or a service developed for the WP by another team (such as the INT team).

1.3 Solutions Developed

In order to deal with some of the challenges mentioned in §1.2, we often contacted Mohamed Magdy Gharib Farag, a Research Scholar with Virginia Tech (mmagdy@vt.edu). Mohamed's breadth of knowledge pertaining to our project's topic was invaluable to our success.

Mohamed was instrumental in aiding and guiding the WP team in solving the first three challenges addressed in §1.2. As the WP team's Subject Matter Expert (SME), Mohamed created a Virtual Machine (VM) to store the collection of tweets containing URLs. In the initial phases of development, a small sample of data was provided by the SME to establish a properly functioning workflow. With the given sample data, the problems and challenges with regards to extraction of embedded links and relevant text were able to be solved.

To refine our understanding of the overall goals, specific sub-tasks required, and final software deliverable of our team, meetings and discussions took place with various members in our "company's" organization. Prashant Chandrasekar was also of much help to further understand how the indexing team will work with the integration team on the data ingestion objective.

The solution to this challenge was accomplished by revising the WP team's System Workflow Artifacts Guidelines (SWAG). In particular, discerning overarching goals versus required subtasks for a goal was crucial for the WP team to ascertain a cognisant understanding of the nature of the data and final output. Moreover, further refinement of the SWAG was important in the steps of informing the design for the Front-End team, as well as generating a workflow based on the established goals.

To solve the problem of deploying services, a local ElasticSearch Stack was developed by the

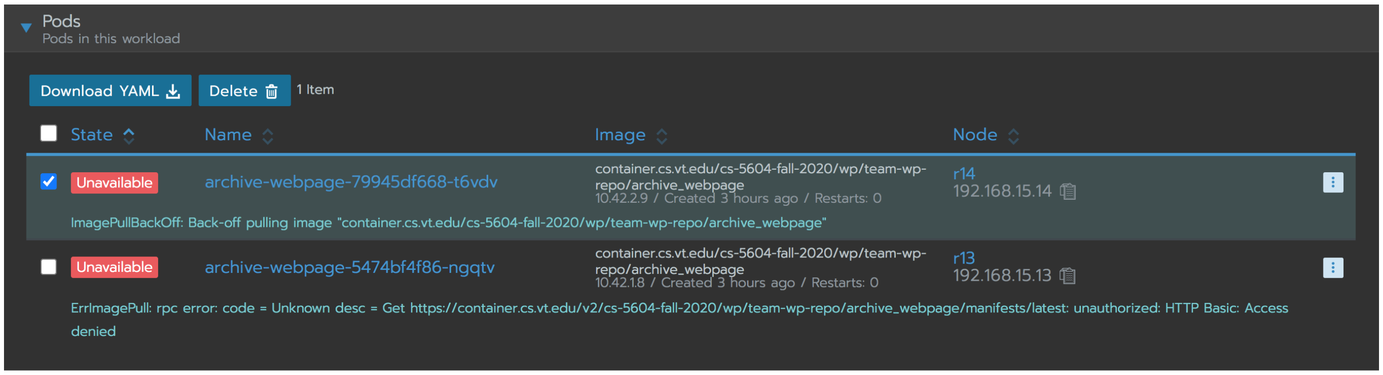
WP team. This served as an interim deployment site until a final end-point application could be established by the Integration team. This will be realized by taking advantage of the Container Registry hosted on GitLab. In other words, Gitlab has built-in Docker functionality. For more information on Docker containers, please see [1].

In order to solve the Dockerization philosophy issues, many meetings with the course TAs and professor were scheduled. After a lot of discussion, it was determined that each task in the pipeline would work as its own service and will thus be Dockerized.

Initially it was believed that the INT team would provide an abstract indexing service that any team could use. However, due to time constraints, it was decided that each team would provide their own indexing service for their own respective data.

1.4 Future Work

All of our services have been deployed to cloud.cs.vt.edu. However, they are not currently working on the cloud. For instance, all of the services related to data ingestion (extract_url,archive_webpage,extract_d
and index). Many hours were spent troubleshooting, with little to show for it from a production deployment standpoint. Please refer to Figure 1 for the two errors that are impeding proper functioning services. Figure 2 shows the six services that have been deployed to the cs5604-wp-db namespace on Rancher.



State	Name	Image	Node
Unavailable	archive-webpage-79945df668-t6vdv	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage 10.42.2.9 / Created 3 hours ago / Restarts: 0	r14 192.168.15.14

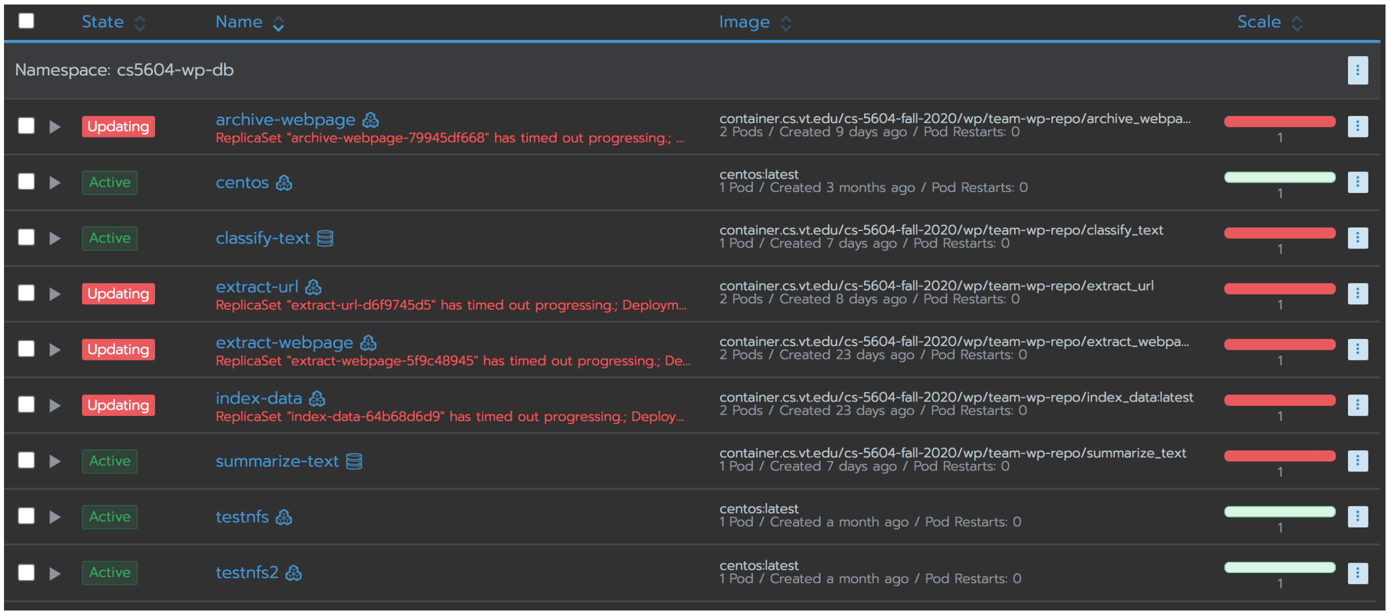
ImagePullBackOff: Back-off pulling image "container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage"

State	Name	Image	Node
Unavailable	archive-webpage-5474bf4f86-ngqtv	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage 10.42.1.8 / Created 3 hours ago / Restarts: 0	r13 192.168.15.13

ErrImagePull: rpc error: code = Unknown desc = Get https://container.cs.vt.edu/v2/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage/manifests/latest: unauthorized: HTTP Basic: Access denied

Figure 1: Rancher Errors

It is suspected that there is an issue with our service integration into Rancher. However, our services were only able to be registered via Airflow on the due date of this finalized report. Obviously, this does not leave much time for troubleshooting. Future work would entail resolving these errors so that user may leverage our services via the website that the Front-End team has developed. For what it is worth, these services work locally and their respective containers are registered on Gitlab.



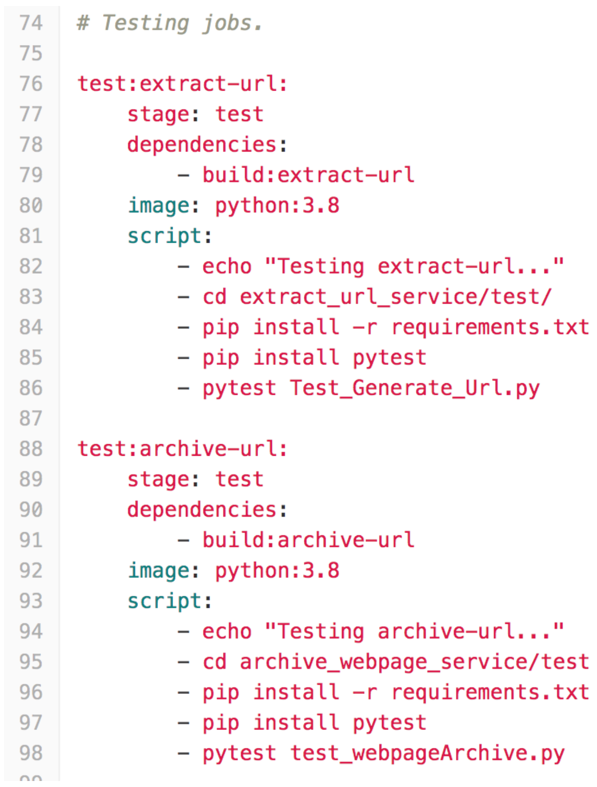
Namespace: cs5604-wp-db

State	Name	Image	Scale
Updating	archive-webpage ReplicaSet "archive-webpage-79945df668" has timed out progressing.; ...	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpa... 2 Pods / Created 9 days ago / Pod Restarts: 0	1
Active	centos	centos:latest 1 Pod / Created 3 months ago / Pod Restarts: 0	1
Active	classify-text	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/classify_text 1 Pod / Created 7 days ago / Pod Restarts: 0	1
Updating	extract-url ReplicaSet "extract-url-d6f9745d5" has timed out progressing.; Deploym...	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_url 2 Pods / Created 8 days ago / Pod Restarts: 0	1
Updating	extract-webpage ReplicaSet "extract-webpage-5f9c48945" has timed out progressing.; De...	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_webpa... 2 Pods / Created 23 days ago / Pod Restarts: 0	1
Updating	index-data ReplicaSet "index-data-64b68d6d9" has timed out progressing.; Deploy...	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/index_data:latest 2 Pods / Created 23 days ago / Pod Restarts: 0	1
Active	summarize-text	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/summarize_text 1 Pod / Created 7 days ago / Pod Restarts: 0	1
Active	testnfs	centos:latest 1 Pod / Created a month ago / Pod Restarts: 0	1
Active	testnfs2	centos:latest 1 Pod / Created a month ago / Pod Restarts: 0	1

Figure 2: Deployed Services on Rancher

All of our services have unit tests and a YAML file exists for automated pipeline testing. Figure

3 shows a portion of the YAML file.



```
# Testing jobs.

test:extract-url:
    stage: test
    dependencies:
        - build:extract-url
    image: python:3.8
    script:
        - echo "Testing extract-url..."
        - cd extract_url_service/test/
        - pip install -r requirements.txt
        - pip install pytest
        - pytest Test_Generate_Url.py

test:archive-url:
    stage: test
    dependencies:
        - build:archive-url
    image: python:3.8
    script:
        - echo "Testing archive-url..."
        - cd archive_webpage_service/test
        - pip install -r requirements.txt
        - pip install pytest
        - pytest test_webpageArchive.py
```

Figure 3: YAML File for CI/CD Pipeline

2 Literature Review

2.1 Intelligent Event Focused Crawling

During the course project, we have found Mohamed Farag's dissertation [2] to be useful to understand our problems at this time. Mohamed Farag's dissertation explains the event focused crawler which can obtain a collection of URLs related to some specific events. Also, the CodeOcean version of EFC and the VM `efc1.cs.vt.edu` were helpful.

The motivation for the event focused crawler is the fact that there is a lack of archiving of important global events. In order to make the crawler work efficiently, a database with high quality URLs is required at first. Though our team will generally use crawled URLs sourced from the Tweet Team which relates to a real world event, we plan to obtain URLs from some other channels.

2.2 Chapters 19-21 in Introduction to Information Retrieval Course Book

Chapters 19-21 of *Introduction to Information Retrieval* by Christopher D. Manning, et al. [3] explained the basic use and structure of a crawler of hypertext. This method was originally applied for searching and indexing, but all of the properties discussed in chapters 19-21 are able to be applied to small or large projects. At this time, we will introduce those methods and properties which are generally suitable to the scale of our current project.

Chapter 19 explained the principles and difficulties of constructing a web crawler. The web content is a combination of diversified elements including not only text, but also voice, pictures, and any other different types of media.

Chapter 20 discussed the general properties of a typical web crawler. A qualified crawler is usually expected to be both robust and polite. Robustness means that the ability of a crawler is not going to be trapped in certain domains. This helps determine how efficiently the crawler could work. Politeness means that a crawler is not going to make the server suffer from a lot of requests.

Chapter 21 discussed how to use the anchor text and extended anchor text to judge whether a link has enough information for crawling. Text analysis and two related website scoring algorithms (hub score and authority score) are covered in this chapter.

2.3 Representational State Transfers (REST)

Representational state transfers (REST) is an architectural style for distributed hypersystems [4]. This architectural style is founded on 6 main principles:

- **Client Server**: The basis of REST is a separation between a client and server. The server offers a list of services, and waits to receive a request from a client. Once the server receives the request from a client, it can accept or deny the request, and respond to the client. The key idea involves separating the user from the server (data storage), so data access can be interfaced and accessible from many services. For instance, whilst the ElasticSearch server must be executed in a specific manner, requests to the ElasticSearch server can be done through Python, Lucene, Java, etc., as long as the interface is provided.

- **Stateless**: The following constraints modified the client-server constraint to now be a client-stateless-server constraint. What this means is that each request to the server must contain all of the required information. It should not use any stored context to further understand the request. This improves visibility, as the server can only interpret incoming requests in one manner. It also improves reliability, as it allows the server to respond to partial failures. Finally, it also improves scalability, as the server doesn't have to store any sort of state in between requests. The only downside is an increase in overhead, as there might be a lot of repeat data in the requests from the client to the server.

- **Cache**: Cache constraints can be added such that requests to the server can be marked as cached or not. If marked as cached, the request can be stored until a new fresh version is needed. This improves efficiency and can save bandwidth. The only downside is sometimes dealing with expired data.

- **Uniform Interface**: Decoupling components in the REST architecture makes the architecture portable, as it's not dependent on a specific system. This means that keeping the interface the same between any client and server helps with the portability of the architecture, as the system type of the client does not matter. For instance, ElasticSearch follows the REST architecture, and always uses JSON as a response, regardless of what system the client employs.

- **Layered System**: REST has the constraint of hiding server intermediaries. This means that a server can have multiple indices, and a client that connects to a server does not know (or cares) what index it connects to. ElasticSearch follows this principle by allowing multiple shards per node, as this information is hidden from a client that makes requests to an ElasticSearch server.

2.4 RESTful API

Web services that conform to the REST architecture are defined as RESTful APIs [5]. More specifically, HTTP RESTful APIs must:

- Have a base URI (Uniform Resource Identifier), which is a string of characters which unambiguously identify a source, usually adhering to a state of protocols and rules [6] (such as starting with http). An example would be http://localhost:9200.

- Utilize standard HTTP methods.

- The media type which defines how data is transferred from the client and server (defining the uniform interface), such as using JSON.

For instance, ElasticSearch is considered a RESTful API, and adhering to the HTTP REST architecture makes it very useful for building an information retrieval system due to all of the qualities and constraints mentioned above.

2.5 Text classification

Text classification has been a big part of retrieving relevant information, where filtering irrelevant documents/record becomes essential. In order to identify the documents/records with the correct topic that we want to archive, we need to perform text classification. There are three main approaches to solve the problem: Statistical Approach, traditional machine learning and deep learning.

- Statistical Approach: A method is employed such as TF-DF and BOW(bag of words) to perform the classification task. These methods perform classification by capturing certain data's probability distributions.
- Traditional machine learning approach: A method is employed such as decision tree, random forest, or SVM (support vector machine) to perform the classification. These methods either capture data's probability distribution or determine the information structure to form the basis of the classification task.
- Deep learning approach: Documents can go through an embedding procedure to convert into contextualized embeddings via techniques such as Doc2Vec. Documents also can be contextualized using language model methods such as BERT (Pre-training of Deep Bidirectional Transformers for Language Understanding) [7], and then use a fully connected neural net for final classification.

These three approaches are both powerful enough to achieve good F1 scores regarding the classification task but each of them has their own advantages and disadvantages: Statistical methods only learn shallow information given from the text which makes it easier to implement but not necessarily most accurate and robust to noisy data. Traditional machine learning does not depend on a significant amount of data to train, and performs better in such settings. A deep learning approach can better represent the documents when the number of training documents is enough. By utilizing text classification, we have enhanced the understanding of our documents/records.

2.6 Text summarization

Text Summarization is a technique used to produce precise and accurate summaries of large amounts of text, focusing on the portion that conveys useful information without losing its overall meaning. There are broadly two different approaches that are used for text summarization: extractive and generative (also called abstractive).

- **Extractive text summarization**: As the name suggests, this is to extract one sentence or a few sentences from a document or document set. The advantage of this scheme is that it is simple and practical and will not be completely separated from the document itself. In other words, the extractive summarization technique focuses on choosing how paragraphs, important sentences, etc. produce a semantically equivalent shorter version of the original documents. The implication of sentences is determined based on linguistic and statistical features.

- **Generative text summarization**: Although extractive text summaries have their advantages, they may also have shortcomings such as incoherent summary generation, poor control of the number of words, and unclear subject matter of the target sentence.The quality of the summary depends on the original text. There is no such problem with generative text summarization, because it is an end-to-end process. This kind of technical solution can benefit from translation tasks and dialogue tasks, so that it can absorb and learn from successful experience with those.

3 Requirements

3.1 Overall Project Requirements

The goal of this project is to build an information storage and retrieval system. In order to do so, the following requirements must be fulfilled:

- Build a state-of-the-art information retrieval and analysis system that can support 3 item collections: electronic theses and dissertations (ETD), webpages (WP), and tweets (TWT).
- Support at least: 30k ETDs, 5B tweets, and 3 million webpages.
- Actively cooperate amongst teams to ensure teams don't fall behind.
- Actively have each member track their progress each week using `Ally.io`.
- Actively have stand up each week to share the progress of each team member and help each other with any blockers.
- Test solutions with small and large batches of real data, following a continuous integration approach.
- Keep relevant code updated on `git.cs.vt.edu` in their proper branches in a well documented manner.
- Dockerize solutions such as that the integration team can utilize and deploy them when necessary.

3.2 WP Team Requirements

The WP team is in charge of creating a set of dockerized services. Then when piped together, they can act as a pipeline which will take some sort of webpage data as input and index it through ElasticSearch. In order to achieve this, the following requirements must be fulfilled:

- Unless otherwise stated, all textual information contained in the processed document must be saved and transferred to other groups. It needs to be extended to support other types of documents. The textual information is created by the groups of the WP team (Generate URLs, Archive URLs, Extract Data, and Indexing).
- Create a set of dockerized services, divvied up among the sub-teams outlined in §1.1 to help all other teams get used to the content.
- Create a working pipeline out of the dockerized services that can generate data from tweets or a set of URLs.
- Create a service that connects to the integration team ElasticSearch server and ingests any input data.

- Dockerize the pipeline so that it can be deployed anywhere. Deploy it to make the container more portable and use the server efficiently. The container isrepresenting a standardized unit of software.
- Provide an optimal ElasticSearch index with correct schema that is usable by different teams for the project. Since various intelligent text analysis algorithms require data to be extracted in different formats, we also provide other teams with full-text data in the required format.

4 Design

4.1 Approach

Before tackling the design of the project, the team met with Mohamed to understand how the team should be structured. It was discussed that the best approach would be to break into four sub-teams as outlined in §1.1, to identify a set of goals for the project, and to identify a set of tasks required for the project. Thus, the first goal was identified as **generate data**. That is, given a set of tweets, extract relevant URLs from the tweets. Of course, it would not read in tweets and outright generate the URLs, rather there would be a set of tasks that lead to the eventual generation of the URLs (§1.1). Furthermore, this goal would include functionality to read in data directly in URL format. A second goal, **ingest data**, will ingest data and generate an ElasticSearch database. Thus, when both of these goals are coupled together, the following workflow of tasks will be produced, as seen in Figure 4

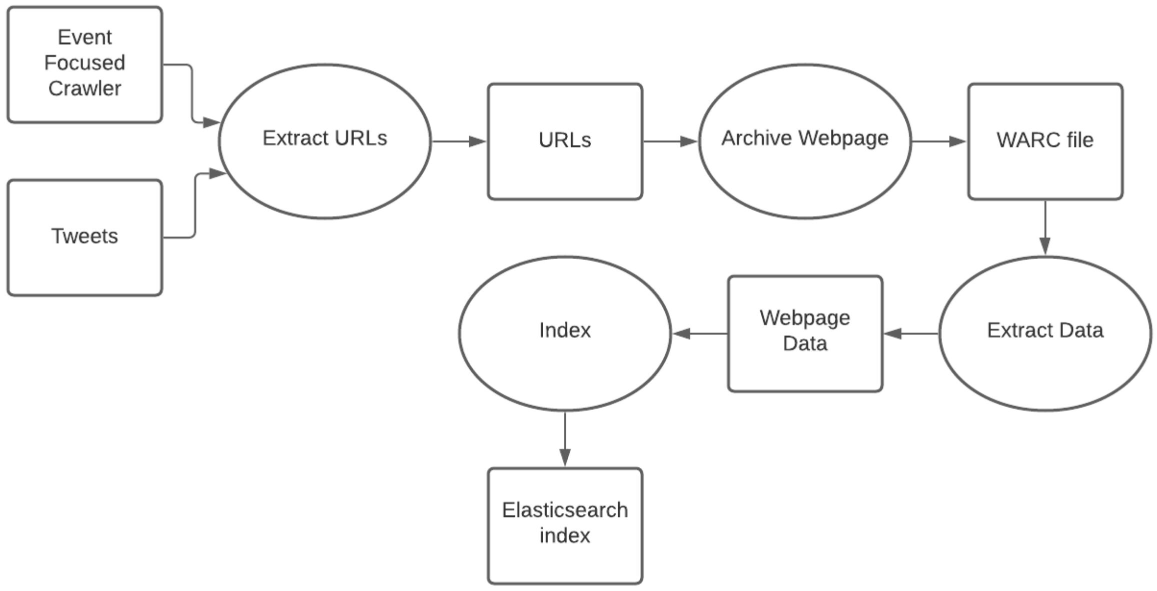


Figure 4: WP Task Approach

In this case, each sub-team outlined in §1.1 is in charge of creating a working, dockerized service. The first approach is to start with extracting URLs from tweet data in JSONL format using Python scripts. This is identified as Extract URL; the flowchart that explains it is in Figure 5, which outlines the overall heuristic of the Extract URL service.

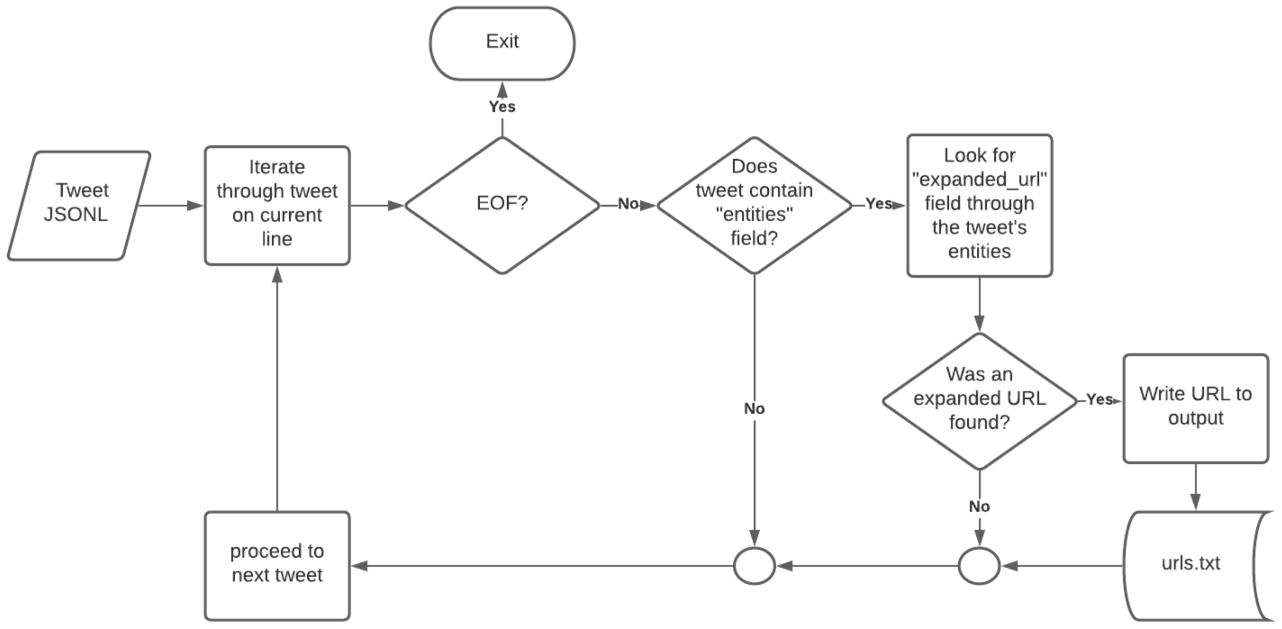

Exit
Yes
Tweet JSONL
Iterate through tweet on current line
EOF?
No
Does tweet contain "entities" field?
Yes
Look for "expanded_url" field through the tweet's entities
No
Was an expanded URL found?
Yes
Write URL to output
No
proceed to next tweet
urls.txt

Figure 5: Extract URL Flowchart

Next, the extracted URLs will need to be converted to webpage data and archived. To convert the extracted URLs to webpage data, the python requests library is used to get the HTML of the webpage Next, to archive the webpage data, WARCIO, the WARC Streaming Library, is used to archive the webpage data and metadata by filtering request and response records and generating a WARC file with the webpage data. A flowchart that explains this is shown in Figure 6.

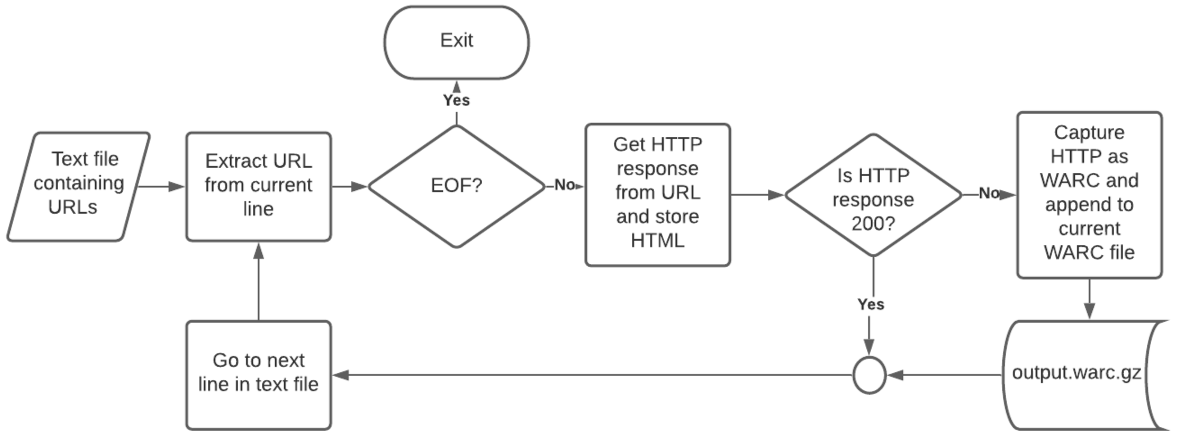
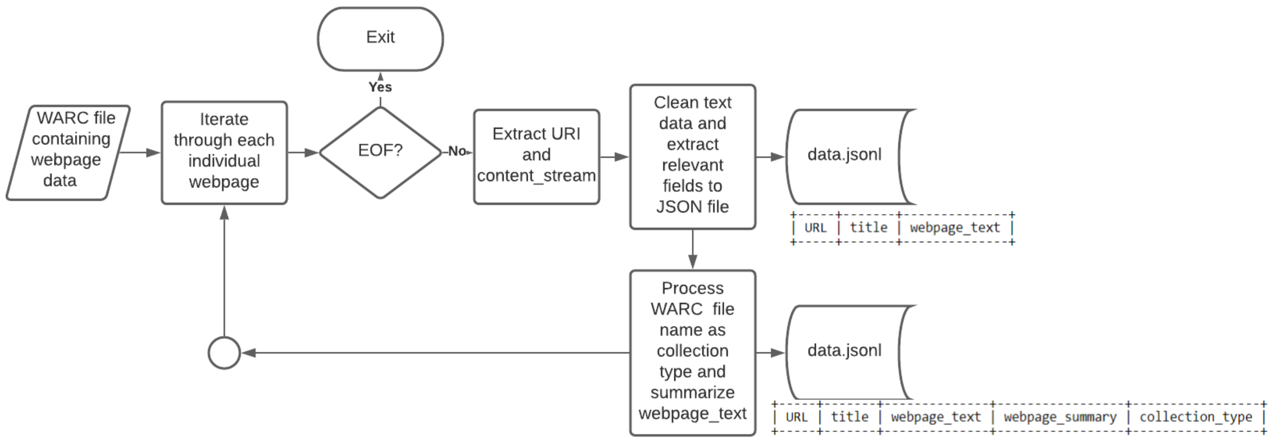

Exit
Yes
Text file containing URLs
Extract URL from current line
EOF?
No
Get HTTP response from URL and store HTML
Is HTTP response 200?
No
Capture HTTP as WARC and append to current WARC file
Yes
Go to next line in text file
output.warc.gz

Figure 6: Archive Webpage Flowchart

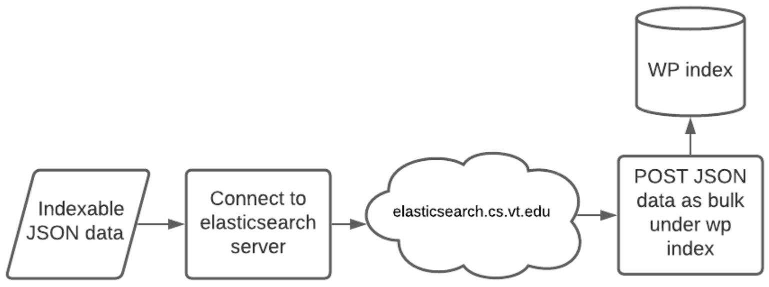
Next, webpage data will need to be extracted from the newly generated WARC file. At first, WARCIO will be used to grab the HTML, then BeautifulSoup will be used to extract text from the HTML. A flowchart that explains this is shown in Figure 7.



Exit
Yes
WARC file containing webpage data
Iterate through each individual webpage
EOF?
No
Extract URI and content_stream
Clean text data and extract relevant fields to JSON file
data.jsonl
URL | title | webpage_text
Process WARC file name as collection type and summarize webpage_text
data.jsonl
URL | title | webpage_text | webpage_summary | collection_type

Figure 7: Extract Webpage Data Flowchart

Lastly, ElasticSearch will be used to index all of the text and metadata (see Figure 8).



WP index
Indexable JSON data
Connect to elasticsearch server
elasticsearch.cs.vt.edu
POST JSON data as bulk under wp index

Figure 8: Index Flowchart

Notice that Figure 4 treats both goals as a set of continuous tasks, where the output of the former is the input of the latter. Thus, a user generates data (e.g., has a set of tweets and wants to extract the data associated with them), and then populates an ElasticSearch index with this extracted data.

Apart from the main tasks identified above, two other independent tasks have been developed. The first task is a text summary task. The idea is to take some raw text and provide a summary of the text. The key idea is to provide quick information for users who are querying a lot of webpage data from the ElasticSearch server. By doing so, users will get the general idea of a webpage by reading the summary rather than the raw text of a webpage, as some webpages contain a copious amount of text. The next task is a text classification task. Often, users want to populate the ElasticSearch index with a lot of data, for instance, Twitter data. Many of the webpages collected using Twitter data might be non-relevant. Thus, a text classification task that can filter out non-relevant data can be useful to users. Note that both these tasks would act independently and don't necessarily belong to the pipeline outlined in Figure 4.

Each of the tasks identified above will be treated as an independent service, each running in their own separate Docker container. The philosophy behind this decision comes from being able to create a flexible pipeline that can be reduced or expanded. For instance, if a user wants to ingest data, given webpage URLs and not tweets, the *Extract URLs* container would not be executed. The pipeline would be easily expanded, by adding new containers, and by redirecting the input and output of containers within the pipeline.

4.2 Background Information and Methodology

The aim of this project is to create an information retrieval system that the Virginia Tech library could use. More specifically, the WP team was entrusted with providing the functionality of ingesting webpage data. Original data was provided by Mohamed as tweet data. To further understand the nature of the project and data, the team often met with Mohamed to gain clarification.

The WP team is new this year. Mohamed provided a collection of tweet data related to some event (e.g., Coronavirus tweet data). Much of the data did not contain webpage URLs, due to the way they were collected. The method employed to collect tweets was by using the Twarc Python API, which allows the user to query and store tweets based on a certain keyword without checking whether the tweet contains an embedded URL or not. Thus, preprocessing was required to filter out tweets that did not contain embedded URLs. Moreover, certain tweets in the data became unavailable. Whether the owner of the tweet deleted it, or the tweet was deleted for violating Twitter's terms of service, there were many cases where the tweet was not available. In this case, the tweet was ignored.

In the 2017 offering of CS5604, there was a team named Collection Management Webpages (CMW) that fetched webpages mentioned in given tweets, created WARC files covering them, processed them, and loaded them into HBase. The WP team utilized some of the CMW team functionality (fetching the webpages identified by the given tweets, creating WARC files, and processing the files).

The tweet data was provided as JSONL files, as specified by the Twitter API [8]. Often, the URL referenced by the tweet would be in a different field of the JSONL file, which depended on what type of tweet it was: a standard tweet with the URL found in the text of the tweet, an extended tweet with an embedded URL, a quoted tweet, or a quoted tweet of an extended tweet. These seemed to be all of the types of tweets to be accounted for, and the relevant URL was found in a different field within the JSONL file for each different type of tweet. Lastly, many of the provided URLs were in a shortened version, so getting their expanded representation was crucial to the success of the project.

4.3 Tools

The tools that will be used to develop this project will be Python, Rancher (Kubernetes), Kubectl, Docker Hub, JSON formatted tweets, WARCIO - a fast, standalone library to read and write WARC Format commonly used in web archives, BeautifulSoup - a Python package for parsing HTML and XML documents, and ElasticSearch - a distributed, open source search and analytics engine for all types of data.

4.3.1 Docker

Docker provides advantages similar to running a virtual machine. Some of those advantages include always being able to run an application in the same environment. As a result, inconsistencies are avoided with regards to the application's behavior. In other words, if the application works on a given computer, then it will work on another computer, as well as a live server [1].

Moreover, Docker enables a project to be delivered in a sandbox, which ensures both security and eliminates conflicts between projects. Along the same line of thought, Docker makes it easier to deploy another's code within the organization. Due to its sandbox nature, the requirement for installing tools and dependencies is removed as an initial step in deploying a given project.

It is important to note that Docker provides these advantages without the hassle and overhead of managing virtual machines. This is because the code and environment are all wrapped up in a Docker container. However, a container is not a full virtual machine. Virtual machines are resource heavy on the host machine. The overhead is lower with a container because the host machine's kernel is shared, but everything on top of that is separate.

In other words, a container is a compromise between the extreme separation and sandbox environment virtual machines offer. As a result, a container can be launched within seconds versus minutes with virtual machines. This is extremely beneficial when collaborating in an organization and performing code integration of various features that have been developed across teams.

More specifically, a container is a running instance of an image. An image is a template for creating the desired environment. Images can be thought of as a snapshot of the system at a particular point in time. Images are defined using a Docker file, which in turn is just a text file that contains a list of steps to perform in order to create the image. Writing and running a Docker file builds an image, which in turn runs and builds a container.

4.3.2 ElasticSearch

ElasticSearch is a search engine based on the Lucene library [9]. It provides a distributed, multi-tenant full-text search engine with HTTP Web interface and schema-less JSON documents. ElasticSearch is developed in Java and released as open source software under the Apache license. The official client is available in Java, .NET, PHP, Python, Apache Groovy, Ruby, and many other languages. In other words, ElasticSearch is a Lucene-based search server. It provides a full-text search engine with distributed multi-user capabilities, based on a RESTful web interface.

ElasticSearch can be used to search various documents. It provides scalable search, has near real-time search, and supports multi-tenancy. ElasticSearch is distributed, which means that the index can be divided into shards, and each shard can have 0 or more copies. Each node hosts one or more shards and acts as a coordinator to delegate operations to the correct shards. Re-balancing and routing are done automatically. By using ElasticSearch in cloud computing, the goal is to achieve a real-time, stable, reliable, and fast search.

4.3.3 JSON

JSON is a lightweight data interchange format based on JavaScript functionality [10]. Despite being derived from JavaScript, JSON has cemented itself as a standard data format across many languages and systems. JSON is founded on the idea of having key-value pairs. For each key, which is a string, there is a corresponding value, which may be of different types. If the value is encapsulated by curly braces, it's considered an object and may have more fields within it. If it isn't, it can be a Boolean, String, number, or NULL.

JSONL is a type of JSON format which stands for "json lines." This means that a JSONL file can have multiple objects per file, each object corresponding to a line. This file was used through the pipeline, as it makes more sense to bulk transfer data rather than pass one JSON file at a time.

4.3.4 WARCIO

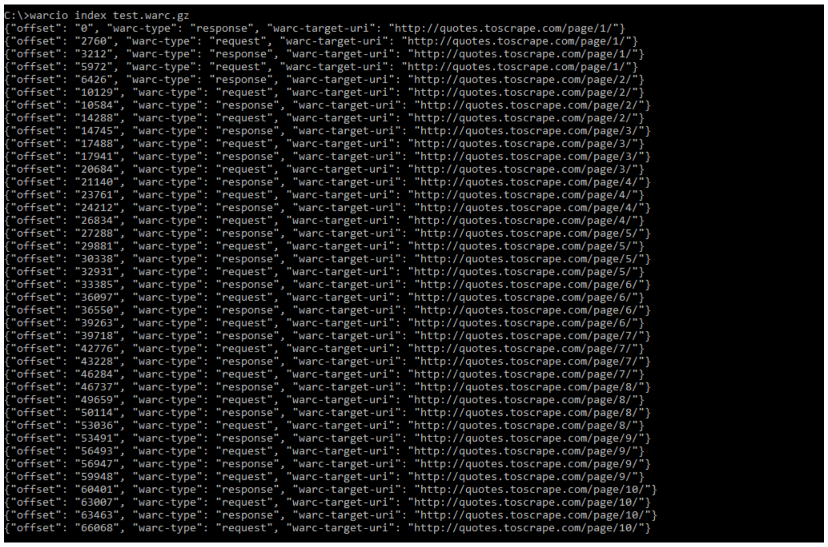
WARCIO: WARC Streaming Library [11] provides a fast, standalone way to read and write WARC Format commonly used in web archives. WARCIO supports reading and writing of WARC files compliant with both the WARC 1.0 and WARC 1.1 ISO standards. It can be installed with: pip install WARCIO. This library is a spin-off of the WARC reading and writing component of the pywb high-fidelity replay library, a key component of Webrecorder. The library is designed for fast, low-level access to web archival content, oriented around a stream of WARC records rather than files.

4.3.5 BeautifulSoup

BeautifulSoup [12] is a Python library which can grab data out from HTML and XML files. It works by providing idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work. The reference covers Beautiful Soup version 4.9.2. The examples in this reference should work the same way in Python 2.7 and Python 3.8.

4.3.6 WARC files

The WARC (Web ARChive) file format provides a convention for connecting multiple resource records (data objects) together [13]. Each resource record consists of a set of simple text titles and an arbitrary data block to form a long file. The WARC format is an extension of the ARC file format and is traditionally used to store "Web crawls" as a sequence of content blocks obtained from the World Wide Web. Each capture in the ARC file has a single-line title that describes the content and length of the harvest very briefly. Then comes the retrieval protocol response message and content. The WARC format is a standard method for structuring, managing, and storing billions of resources collected from the Web or elsewhere. It is used to build applications for collection (such as the open source Heritrix Web crawler), as well as to manage, access, and exchange content. The way the WARC file is created and the way the resources are stored and presented will depend on the implementation of the software and application.



```
C:\>warcio index test.warc.gz
{"offset": "0", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/1/"}
{"offset": "2760", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/1/"}
{"offset": "3212", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/1/"}
{"offset": "5972", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/1/"}
{"offset": "6426", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/2/"}
{"offset": "10129", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/2/"}
{"offset": "10584", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/2/"}
{"offset": "14288", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/2/"}
{"offset": "14745", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/3/"}
{"offset": "17488", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/3/"}
{"offset": "17941", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/3/"}
{"offset": "20684", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/3/"}
{"offset": "21140", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/4/"}
{"offset": "23761", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/4/"}
{"offset": "24212", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/4/"}
{"offset": "26834", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/4/"}
{"offset": "27288", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/5/"}
{"offset": "29881", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/5/"}
{"offset": "30338", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/5/"}
{"offset": "32931", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/5/"}
{"offset": "33385", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/6/"}
{"offset": "36097", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/6/"}
{"offset": "36550", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/6/"}
{"offset": "39263", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/6/"}
{"offset": "39718", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/7/"}
{"offset": "42776", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/7/"}
{"offset": "43228", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/7/"}
{"offset": "46284", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/7/"}
{"offset": "46737", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/8/"}
{"offset": "49659", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/8/"}
{"offset": "50114", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/8/"}
{"offset": "53036", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/8/"}
{"offset": "53491", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/9/"}
{"offset": "56493", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/9/"}
{"offset": "56947", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/9/"}
{"offset": "59948", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/9/"}
{"offset": "60401", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/10/"}
{"offset": "63007", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/10/"}
{"offset": "63463", "warc-type": "response", "warc-target-uri": "http://quotes.toscrape.com/page/10/"}
{"offset": "66068", "warc-type": "request", "warc-target-uri": "http://quotes.toscrape.com/page/10/"}
```

Figure 9: Example of a WARC File

4.4 Deliverables

In order to meet the overall project and team requirements, a set of deliverables was set in place. First of all, three interim reports that acted as a draft of the final report were required to be written.

- **Interim Report 1**: Interim report 1 contains a skeleton of the report and should contain any relevant background information.
- **Interim Report 2**: Interim report 2 expands on IR1 by fixing any errors, including more background information, and expanding on current progress. By IR2, each of the sub-teams should have a working demo. It's not necessary for the demo to work with relevant data (should work with dummy data), as to not hinder the teams' progress due to the nature of each subsequent team depending on the previous teams' output for their input.
- **Interim Report 3**: Interim report 3 expands on IR2 by fixing any grammatical errors and addresses any comments made by the professor and classmates. Whilst the bulk of the writing was done in IR2, IR3 should include any relevant information that was not present in IR2.

- **Final Report**:The final report expands on I3 by fixing any grammatical errors and addresses any comments made by the professor and classmates. The goal of the final report is to have a finalized submission for VTechWorks.

Apart from interim reports and the final report, there are a set of tasks that will be handled by their own Docker containers. These will be used by the integration team and webpage team to provide certain webpage-related services. These services are highlighted in §4.1.

- **Extract URLs**: Create a service that reads in Twitter data as JSONL files, and extracts any embedded URLs if possible.
- **Archive Webpage**: Create a service that reads in a list of URLs, filters webpages that cannot be accessed successfully, and generates a singular WARC file for that given list.
- **Extract Data**: Given a WARC file, parse through the stored webpages and extract webpage data into a JSON file, where each entry in the JSON file contains data for a single webpage. The extracted webpage data currently includes: webpage title, URL, and webpage text.
- **Index**: Create a service that takes data input as JSONL, connects to an ElasticSearch server (not spawned by the WP team), and ingests the given data to an ElasticSearch index. The data will be indexed under the wp index.
- **Summarize Text**: Given a JSONL file with a text entry, return the same JSONL with an extra field containing the text summary.
- **Classify Text**: Given a text file with each line in the file containing a given body of text, parse said body of text and classify as either relevant (to COVID-19) or not, returning a text file which records the classification results.

5 Implementation

5.1 Timeline

Table 1 shows our schedule. It contains the task description, our estimated timeline in weeks, team members and/or sub-teams responsible for accomplishing the tasks, and the current status. This schedule has been added to and changed over time.

Sub-teams and their respective members are as follows:

1. **Extract URLs**: Wentao Fan, Peng Tan, Yang Hu, Cristian Vives
2. **Archive Webpage**: Shuaicheng Zhang, Tishauna Wilson
3. **Extract Data**: Wentao Fan, Peng Tan, Yang Hu, Cristian Vives
4. **Index**: Cristian Vives, Jostein Barry-Straume
5. **Classify Text**: Shuaicheng Zhang
6. **Summarize Text**: Wentao Fan, Peng Tan, Yang Hu, Jostein Barry-Straume

Table 1: Tasks and Timeline

Task	Timeline (week)	Assignee	Status
Conceptual background research and system workflow artifact guidelines	1-4	ALL	DONE
Assign sub-team membership and establish project roles and responsibilities	1	ALL	DONE
Creation of separate sub-team git branches in GitLab WP team repository	2	Subteam 4	DONE
SSH into Virtual Machine created by Mohamed and change password	3	ALL	DONE
Initial meeting with Subject Matter Expert (SME)	3	ALL	DONE
Prototype script to output URLs from tweets	7	Subteam 1	DONE
Prototype script to output WARC files from URLs	7	Subteam 2	DONE
Prototype script to output Webpages text from WARC files	7	Subteam 3	DONE

Continued on next page

Table 1 – continued from previous page

Task	Timeline (week)	Assignee	Status
Prototype script to generate ElasticSearch Index from sample JSON API	7	Subteam 4	DONE
Create Local Elastic Stack application to test indexing of sample data	7	Subteam 4	DONE
Team-2-team meeting with Integration team regarding Docker containers for each service	8	ALL	DONE
Prototype script to generate ElasticSearch Index from WARC files	10	Subteam 4	DONE
Acquire a larger sample data from SME	10	ALL	DONE
Test and validate extract URLs script using a larger collection of sample data	10	Subteam 1	DONE
Test and validate archive webpage script using a larger collection of sample data	10	Subteam 2	DONE
Test and validate extract data script using a larger collection of sample data	10	Subteam 3	DONE
Test and validate index script using a larger collection of sample data	10	Subteam 4	DONE
Deploy services to Docker containers	10	ALL	DONE
Implement webpage title classification as an additional service		ALL	DONE
Container testing, evaluation, and integration into the CS cloud Kubernetes cluster	13	ALL	DONE
Finalize the services so they can be dockerized	11	ALL	DONE
Plan to conduct an end-to-end test	11	ALL	DONE
Edit and Add to IR3	10	Tishauna	DONE
Pipeline	10	Cristian and Jostein	DONE
Text Classification	12	Shuaicheng	DONE
Text Summarization	12	Wentao and Peng and Yang	DONE

5.2 Milestones and Deliverables

Our milestones over time are shown in Table 2. We will provide deliverables as listed in Table 3. Table 4 lists the container images per team along with their associated function within the system.

Table 2: Milestones

Task #	Completion Date	Milestone
1	09/17	Interim Report 1: Construct initial skeleton of the final report and include any relevant background information compiled to date
2	10/04	All prototype services for each sub-team implemented
3	10/05	Interim Presentation 2: Demo live service of goals 1 and 2
4	10/08	Interim Report 2: Expand on IR1 to include project progress to date
5	10/16	Extract URLs: Update service to output JSONL files containing relevant webpage hyperlink
6	10/16	Archive Webpage: Update service to generate a singular WARC file (many-to-one relationship)
7	10/16	Extract Data: Update service to clean and parse JSON data
8	10/16	Index: Update service to take in WARC file as input instead of pure JSON
9	10/23	Deploy each service to Docker containers
10	10/23	Index: Connect service to Integration team provided Elastic Stack end-point
11	10/23	Implement working pipeline for data generation
12	10/23	Implement working pipeline for data ingestion
13	10/25	Interim Presentation 3: Live demo of data generation and ingestion pipelines
14	10/30	Interim Report 3: Expand on report to include relevant progress to date
15	11/20	Test and evaluate data pipelines and services
16	12/02	Final Project Presentation: Live demo of fully tested and deployed software for data extraction and ingestion of webpages
17	12/09	Final Project Report: Deliver fully tested and deployed software for data extraction and ingestion of webpages

Table 3: Deliverables

Task #	Completion Date	Deliverables
1	09/08	Teams formed
2	09/17	Interim Report 1
3	09/21	Reviews for Interim Report 1
4	10/8	Interim Report 2
5	10/12	Reviews for Interim Report 2
6	10/29	Interim Report 3
7	11/2	Reviews for Interim Report 2
8	12/2	Final presentations
9	12/9	Final Project Report

Table 4: Containers and their function in the System

Image Name	Docker Image	Function in the System
Extract URLs	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_url	Take in as an input a collection of either tweets or event focused crawler results, and return a set of valid URLs for the given category of collection
Archive Webpage	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage	Take in as an input a set of URLs, and return a WARC file containing all Webpages as an archive
Extract Data	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_webpage	Take in as an input a WARC file, then extract and return the webpage data from the contained URLs in the form of JSONL
Text Summarization	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/summarize_text	Take in as a JSONL file, then perform text summarization on a given set of webpages' content, and return a JSONL file of said webpage summaries
Index	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/index_data	Take in as an input a JSONL file, perform indexing, and populate an ElasticSearch index

Continued on next page

Table 4 – continued from previous page

Team Name	Container Image	Function in the System
Classify Text	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/classify_text	Take in as an text file, perform text classification, and produce prediction results to a text file

5.3 Methods Selection

This section is dedicated to the evaluation process that resulted in the selection of the best methods for the WP team project.

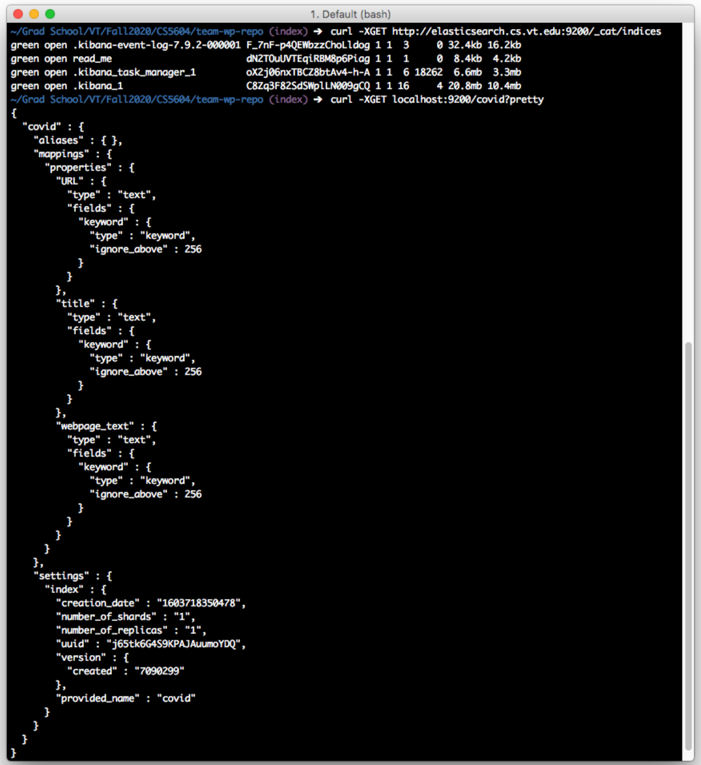
For URL extraction, basic Python APIs such as "json" and "re" are utilized. Future software development might involve implementation with the Tweepy API [14], due to its ease of use with query handling of retweet and quote attributes. However, at the current time the priority was to establish a baseline prototype that was up and running. The best method in the short term is to use familiar tried and true APIs. In the long term, the best method to extract URLs may change. Evaluation of the WP team's current URL generation service needs to be tested on a large subset of the data before any determination is made.

For archiving webpages, WARCIO is utilized. WARCIO is a Streaming Library that archives the webpage data and metadata by filtering request and response records and generating WARC files. This is the current best method for this service because of the wisdom imparted on the WP team by way of the Subject Matter Expert (SME). The SME has intimate knowledge of web archiving, so the WP team is following his suggestion to make use of WARCIO. Moreover, the initial code to get started was provided by the SME and involves the WARCIO API. Currently there are no plans to test other methods of web archiving, as the WP team feels this is not a good use of their time.

For extracting webpage data, WARCIO is used to read the input since it is of the form of a single WARC file for a given service process. From there, the BeautifulSoup and JSON APIs provide the functionality for extracting webpage HTML data and writing it to a line delimited JSON file. BeautifulSoup is the gold standard for webpage content extraction, so there really is no need to test alternative implementations.

For indexing webpage data, the ElasticSearch API is utilized. Use of ElasticSearch is required per the overall project, as well as the WP team, requirements. So by default, ElasticSearch is the best choice to index the data.

For classifying text data, genism API, nltk API and scikit-learn API are used. These APIs are used to form an end-to-end system. First genism API and nltk API are used to contextualize the input texts into Doc2Vec embeddings(the raw input is in the text format, to convert it to acceptable format for SVM, it's better using embedding techniques) to capture the overall meaning/semantic information of the documents, and, through scikit-learn API's one-class SVM classifier, these generated contextualized embeddings are being classified to either relevant to the COVID-19 or not. The reason for using one-class SVM is because we don't have a significant amount of training data and test data to utilize deep learning techniques (fully connected neural nets etc.) and one-class SVM is sufficient to perform classification to draw a decision boundary when there are only positive data.



```
~/Grad School/VT/Fall2020/CS5604/team-wp-repo (index) ➔ curl -XGET http://elasticsearch.cs.vt.edu:9200/_cat/indices
green open .kibana-event-log-7.9.2-000001 F_7nF-p4QEWbzzChoLldog 1 1  3     0 32.4kb 16.2kb
green open read_me                        dN2TOuUVTEqiRBM8p6Piag 1 1  1     0  8.4kb  4.2kb
green open .kibana_task_manager_1         oX2j06nxTBCZ8btAv4-h-A 1 1  6 18262  6.6mb  3.3mb
green open .kibana_1                      C8Zq3F82SdSWplLN009gCQ 1 1 16     4 20.8mb 10.4mb
~/Grad School/VT/Fall2020/CS5604/team-wp-repo (index) ➔ curl -XGET localhost:9200/covid?pretty
{
  "covid" : {
    "aliases" : { },
    "mappings" : {
      "properties" : {
        "URL" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "title" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "webpage_text" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        }
      }
    },
    "settings" : {
      "index" : {
        "creation_date" : "1603718350478",
        "number_of_shards" : "1",
        "number_of_replicas" : "1",
        "uuid" : "j65tk6G4S9KPAJAuumoYDQ",
        "version" : {
          "created" : "7090299"
        },
        "provided_name" : "covid"
      }
    }
  }
}
```

Figure 10: Example of index structure from ElasticSearch that the Index service creates

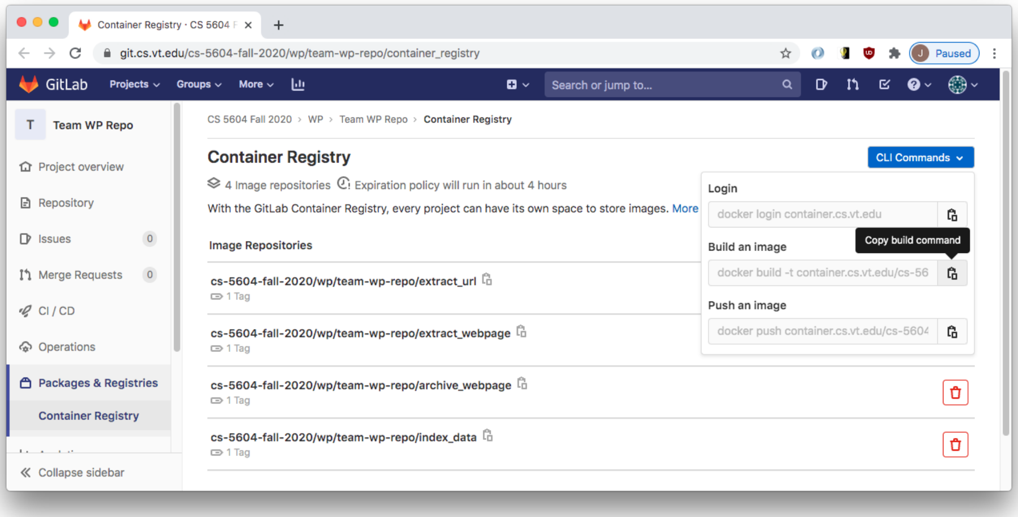
6 User Manual

6.1 Service Utilization

The WP Team has Dockerized its Minimal Viable Product into a pipeline of services. This pipeline is currently six Docker containers. Each container represents one service, and as such carries out one task when started. Right now input for each container is done through environment variables, which are appropriately set each time the container is spun.

As is shown in Figure 11, all Docker containers are registered in the Container Registry in Gitlab. Please refer to Figure 11 for how to copy the following commands:

- Login
- Build an Image
- Push an Image



Container Registry

4 Image repositories   Expiration policy will run in about 4 hours

With the GitLab Container Registry, every project can have its own space to store images. More

Image Repositories

cs-5604-fall-2020/wp/team-wp-repo/extract_url
cs-5604-fall-2020/wp/team-wp-repo/extract_webpage
cs-5604-fall-2020/wp/team-wp-repo/archive_webpage
cs-5604-fall-2020/wp/team-wp-repo/index_data

Login
```
docker login container.cs.vt.edu
```
Build an image (Copy build command)
```
docker build -t container.cs.vt.edu/cs-56
```
Push an image
```
docker push container.cs.vt.edu/cs-5604
```

Figure 11: WP Team Container Registry

For the convenience of future developers, please refer to the below snippet of code in which you can find all four commands to run each Docker container via the command line.

```
####################################
# Container 1: Extract URL Service #
####################################
docker run \
--env TWEET_INPUT_FILE=data/coronavirus0408_100.jsonl \
--env URL_OUTPUT_FILE=data/urls.txt \
--env VERBOSE=true \
-v /"$(pwd)"/data:/code/data extract_url

########################################
# Container 2: Archive Webpage Service #
########################################
docker run \
--env URL_INPUT_FILE=data/urls.txt \
--env WARC_OUTPUT_FILE=data/out.warc.gz \
--env VERBOSE=true \
-v /"$(pwd)"/data:/code/data archive_webpage \

########################################
# Container 3: Extract Webpage Service #
########################################
docker run \
--env WARC_INPUT_FILE=data/out.warc.gz \
--env JSONL_OUTPUT_FILE=data/output.jsonl \
--env VERBOSE=true \
-v /"$(pwd)"/data:/code/data extract_webpage \

###################################
# Container 4: Index Data Service #
###################################
docker run \
--env JSONL_INPUT_FILE=data/output.jsonl \
--env INDEX=wp \
--env ADDRESS=elasticsearch.cs.vt.edu:9200 \
--env VERBOSE=true \
-v /"$(pwd)"/data:/code/data index_data \

##########################
# Summarize Text Service #
##########################
docker run -v /"$(pwd)"/data:/code/data summarize_text

#########################
# Classify Text Service #
#########################
docker run -v /"$(pwd)"/data:/code/data classify_text
```

7 Developer Manual

7.1 Input

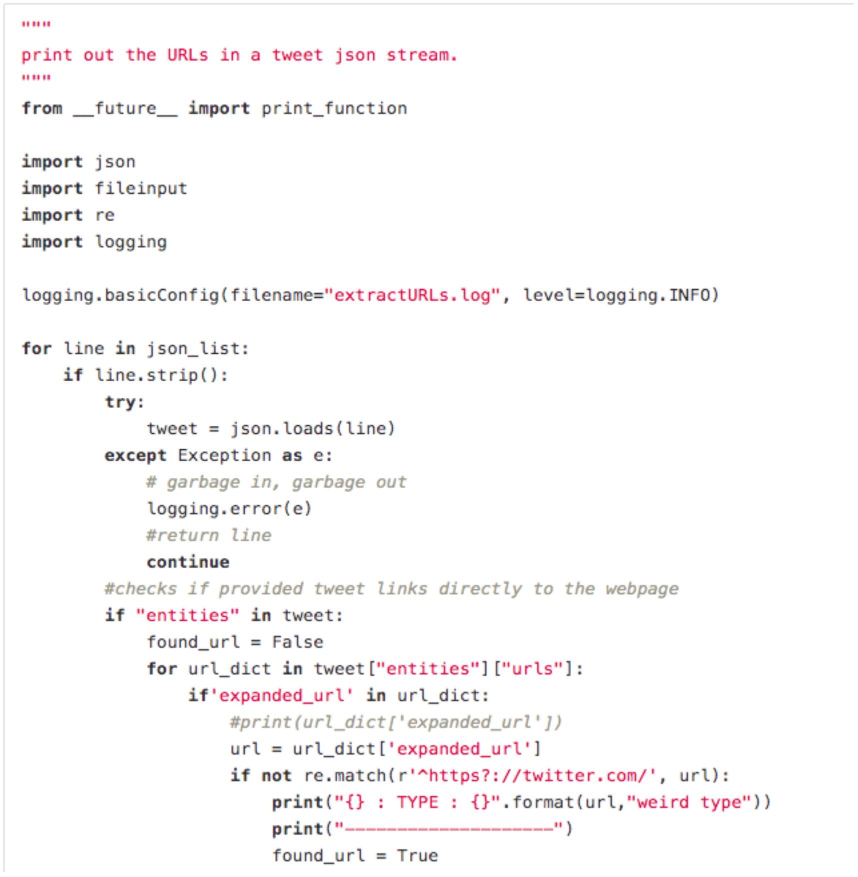
One starting point involves extracting URLs from tweets. The raw data of tweet text with embedded URLs is required. Raw data of tweets can be obtained from the Internet, other groups' support, or anywhere else. Store the tweet text with embedded URLs according to a specific formatting (e.g. JSON) so that "Extract URLs" can begin to work on this.

It is recommended to preprocess the raw data to filter out tweets without URLs before starting "Extract URLs". However, "Extract URLs" is still able to work with non-preprocessed data.

7.2 Extract URLs

This includes collecting raw data from Twitter and other media, and extracts specific URLs from the raw data. This is the start of this service.

With help from Mohamed, we use a Python script (extract_url.py) to preprocess JSON files that contain tweet data and extract any URLs present in any tweets. Once extracted, the URLs will be stored in a txt file.



```python
"""
print out the URLs in a tweet json stream.
"""
from __future__ import print_function

import json
import fileinput
import re
import logging

logging.basicConfig(filename="extractURLs.log", level=logging.INFO)

for line in json_list:
    if line.strip():
        try:
            tweet = json.loads(line)
        except Exception as e:
            # garbage in, garbage out
            logging.error(e)
            #return line
            continue
        #checks if provided tweet links directly to the webpage
        if "entities" in tweet:
            found_url = False
            for url_dict in tweet["entities"]["urls"]:
                if'expanded_url' in url_dict:
                    #print(url_dict['expanded_url'])
                    url = url_dict['expanded_url']
                    if not re.match(r'^https?://twitter.com/', url):
                        print("{} : TYPE : {}".format(url,"weird type"))
                        print("-------------------")
                        found_url = True
```

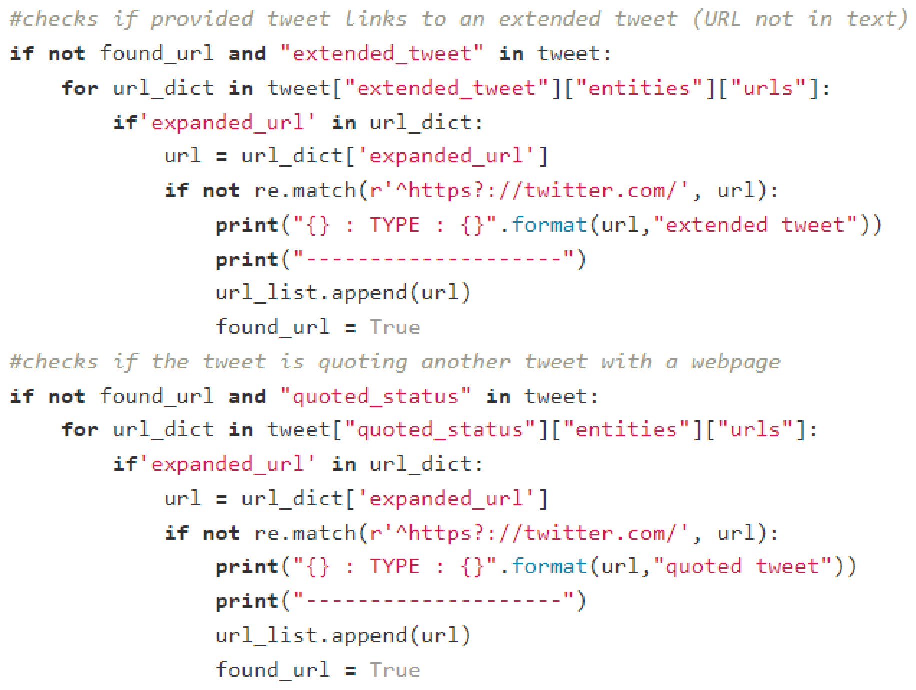
Figure 12: Part of EXTRACT URL Code

1. Import several important packages to process the JSONL file:

```
from __future__ import print_function
import JSON, fileinput, re, logging
```

2. Set configuration of log file:

```
logging.basicConfig
(filename="extractURLs.log", level=logging.INFO)
```



```
#checks if provided tweet links to an extended tweet (URL not in text)
if not found_url and "extended_tweet" in tweet:
    for url_dict in tweet["extended_tweet"]["entities"]["urls"]:
        if'expanded_url' in url_dict:
            url = url_dict['expanded_url']
            if not re.match(r'^https?://twitter.com/', url):
                print("{} : TYPE : {}".format(url,"extended tweet"))
                print("--------------------")
                url_list.append(url)
                found_url = True
#checks if the tweet is quoting another tweet with a webpage
if not found_url and "quoted_status" in tweet:
    for url_dict in tweet["quoted_status"]["entities"]["urls"]:
        if'expanded_url' in url_dict:
            url = url_dict['expanded_url']
            if not re.match(r'^https?://twitter.com/', url):
                print("{} : TYPE : {}".format(url,"quoted tweet"))
                print("--------------------")
                url_list.append(url)
                found_url = True
```

Figure 13: Part of Extract URL Code

3. Using package fileinput to get info from a tweet JSON stream line by line:

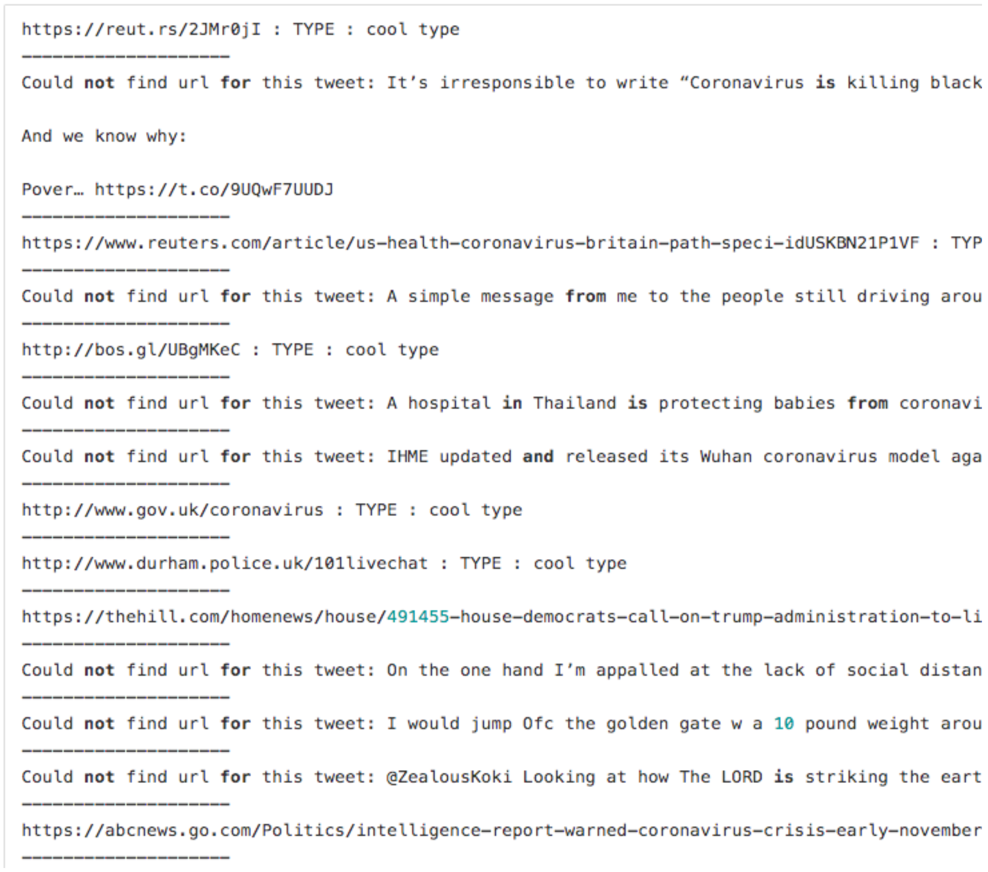
```
for line in fileinput.input('coronavirus0408_100.JSONl'):
```

4. Use the JSON package to parse the JSON data stream and print the "unshortened" URL:

```
try:
    tweet = JSON.loads(line)
except Exception as e:
    # garbage in, garbage out
    logging.error(e)
    #return line
    continue
# don't do the same work again
if 'unshortened_url' in tweet and tweet['unshortened_url']:
    #return line
        print(tweet['unshortened_url'])
        continue
    #tweet = JSON.loads(line)
```

5. If the tweet data stream is stored with entities as the key, then another data analysis method is used. The output URLs will be embedded with corresponding data type as defined: URLs are labeled as "cool type" if usable, otherwise they are labeled as "weird type."

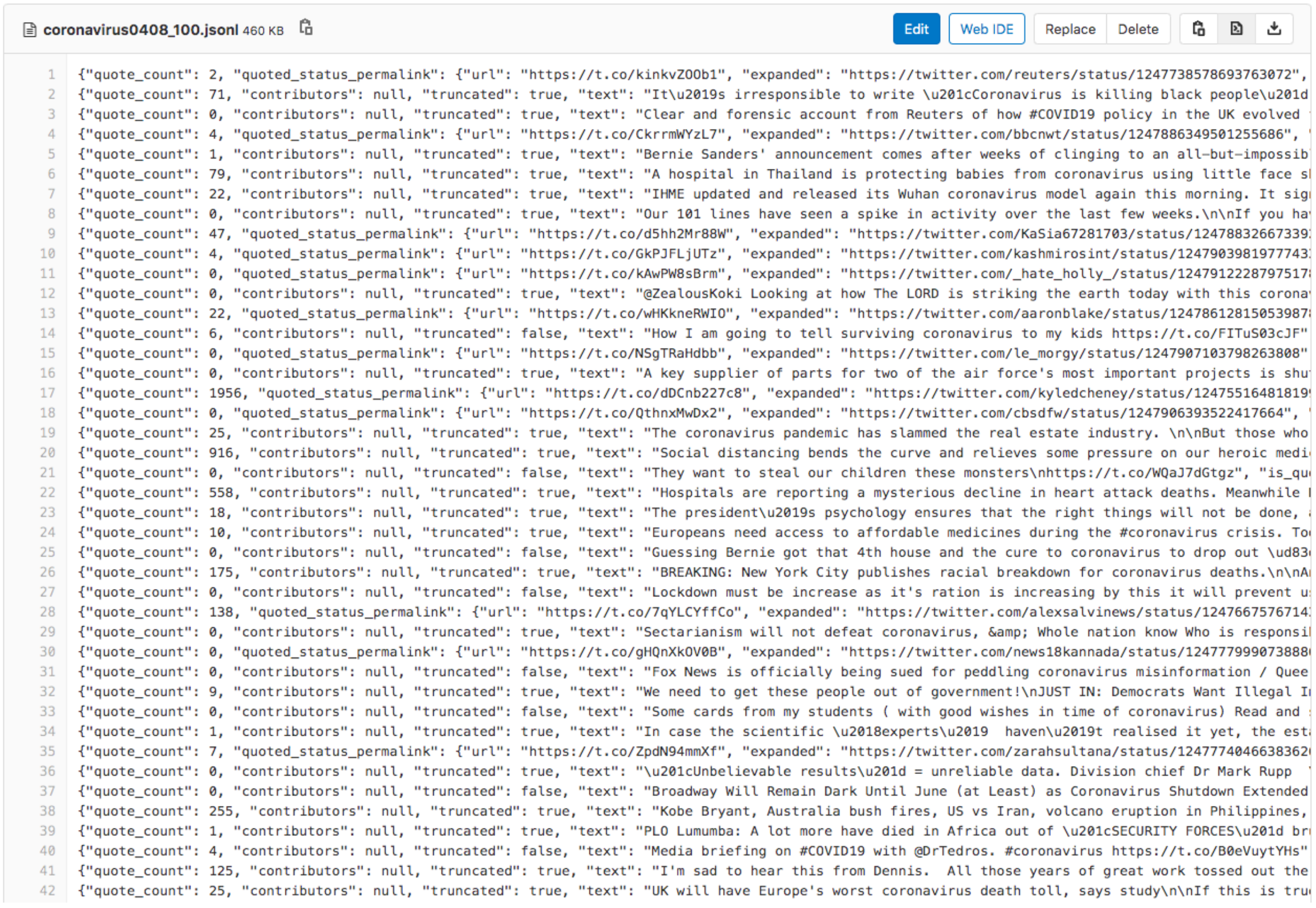
```
    if "entities" in tweet:
        #lng = tweet["lang"]
        #if lng != 'en':
            #print("Not English " + tweet["id_str"])
            #print("Not English " + tweet["lang"])
            #continue
        for url_dict in tweet["entities"]["urls"]:
            if 'unshortened_url' in url_dict:
                #print(url_dict['unshortened_url'])
                url = url_dict['unshortened_url']
            elif 'expanded_url' in url_dict:
                #print(url_dict['expanded_url'])
                url = url_dict['expanded_url']
            else:
                #print(url_dict['url'])
                url = url_dict['url']
            if url:
                if re.match(r'^https?://Twitter.com/', url):
                # don't hammer on Twitter.com urls that we know are not short
                #url_dict['unshortened_url'] = url
                    continue
                print(url)
```



```
https://reut.rs/2JMr0jI : TYPE : cool type
--------------------
Could not find url for this tweet: It’s irresponsible to write “Coronavirus is killing black

And we know why:

Pover… https://t.co/9UQwF7UUDJ
--------------------
https://www.reuters.com/article/us-health-coronavirus-britain-path-speci-idUSKBN21P1VF : TYP
--------------------
Could not find url for this tweet: A simple message from me to the people still driving arou
--------------------
http://bos.gl/UBgMKeC : TYPE : cool type
--------------------
Could not find url for this tweet: A hospital in Thailand is protecting babies from coronavi
--------------------
Could not find url for this tweet: IHME updated and released its Wuhan coronavirus model aga
--------------------
http://www.gov.uk/coronavirus : TYPE : cool type
--------------------
http://www.durham.police.uk/101livechat : TYPE : cool type
--------------------
https://thehill.com/homenews/house/491455-house-democrats-call-on-trump-administration-to-li
--------------------
Could not find url for this tweet: On the one hand I’m appalled at the lack of social distan
--------------------
Could not find url for this tweet: I would jump Ofc the golden gate w a 10 pound weight arou
--------------------
Could not find url for this tweet: @ZealousKoki Looking at how The LORD is striking the eart
--------------------
https://abcnews.go.com/Politics/intelligence-report-warned-coronavirus-crisis-early-november
--------------------
```

Figure 14: Output of Generate URL

6. Example Input Data: See Figure 21


coronavirus0408_100.jsonl 460 KB

```
{"quote_count": 2, "quoted_status_permalink": {"url": "https://t.co/kinkvZOOb1", "expanded": "https://twitter.com/reuters/status/1247738578693763072",
{"quote_count": 71, "contributors": null, "truncated": true, "text": "It\u2019s irresponsible to write \u201cCoronavirus is killing black people\u201d
{"quote_count": 0, "contributors": null, "truncated": true, "text": "Clear and forensic account from Reuters of how #COVID19 policy in the UK evolved
{"quote_count": 4, "quoted_status_permalink": {"url": "https://t.co/CkrrmWYzL7", "expanded": "https://twitter.com/bbcnwt/status/1247886349501255686",
{"quote_count": 1, "contributors": null, "truncated": true, "text": "Bernie Sanders' announcement comes after weeks of clinging to an all-but-impossib
{"quote_count": 79, "contributors": null, "truncated": true, "text": "A hospital in Thailand is protecting babies from coronavirus using little face s
{"quote_count": 22, "contributors": null, "truncated": true, "text": "IHME updated and released its Wuhan coronavirus model again this morning. It sig
{"quote_count": 0, "contributors": null, "truncated": true, "text": "Our 101 lines have seen a spike in activity over the last few weeks.\n\nIf you ha
{"quote_count": 47, "quoted_status_permalink": {"url": "https://t.co/d5hh2Mr88W", "expanded": "https://twitter.com/KaSia67281703/status/12478832667339
{"quote_count": 4, "quoted_status_permalink": {"url": "https://t.co/GkPJFLjUTz", "expanded": "https://twitter.com/kashmirosint/status/1247903981977743
{"quote_count": 0, "quoted_status_permalink": {"url": "https://t.co/kAwPW8sBrm", "expanded": "https://twitter.com/_hate_holly_/status/1247912228797517
{"quote_count": 0, "contributors": null, "truncated": true, "text": "@ZealousKoki Looking at how The LORD is striking the earth today with this corona
{"quote_count": 22, "quoted_status_permalink": {"url": "https://t.co/wHKkneRWIO", "expanded": "https://twitter.com/aaronblake/status/1247861281505398
{"quote_count": 6, "contributors": null, "truncated": false, "text": "How I am going to tell surviving coronavirus to my kids https://t.co/FITuS03cJF"
{"quote_count": 0, "quoted_status_permalink": {"url": "https://t.co/NSgTRaHdbb", "expanded": "https://twitter.com/le_morgy/status/1247907103798263808"
{"quote_count": 0, "contributors": null, "truncated": true, "text": "A key supplier of parts for two of the air force's most important projects is shu
{"quote_count": 1956, "quoted_status_permalink": {"url": "https://t.co/dDCnb227c8", "expanded": "https://twitter.com/kyledcheney/status/1247551648181
{"quote_count": 0, "quoted_status_permalink": {"url": "https://t.co/QthnxMwDx2", "expanded": "https://twitter.com/cbsdfw/status/1247906393522417664",
{"quote_count": 25, "contributors": null, "truncated": true, "text": "The coronavirus pandemic has slammed the real estate industry. \n\nBut those who
{"quote_count": 916, "contributors": null, "truncated": true, "text": "Social distancing bends the curve and relieves some pressure on our heroic medi
{"quote_count": 0, "contributors": null, "truncated": false, "text": "They want to steal our children these monsters\nhttps://t.co/WQaJ7dGtgz", "is_qu
{"quote_count": 558, "contributors": null, "truncated": true, "text": "Hospitals are reporting a mysterious decline in heart attack deaths. Meanwhile
{"quote_count": 18, "contributors": null, "truncated": true, "text": "The president\u2019s psychology ensures that the right things will not be done,
{"quote_count": 10, "contributors": null, "truncated": true, "text": "Europeans need access to affordable medicines during the #coronavirus crisis. To
{"quote_count": 0, "contributors": null, "truncated": false, "text": "Guessing Bernie got that 4th house and the cure to coronavirus to drop out \ud83
{"quote_count": 175, "contributors": null, "truncated": true, "text": "BREAKING: New York City publishes racial breakdown for coronavirus deaths.\n\nA
{"quote_count": 0, "contributors": null, "truncated": false, "text": "Lockdown must be increase as it's ration is increasing by this it will prevent u
{"quote_count": 138, "quoted_status_permalink": {"url": "https://t.co/7qYLCYffCo", "expanded": "https://twitter.com/alexsalvinews/status/124766757671
{"quote_count": 0, "contributors": null, "truncated": true, "text": "Sectarianism will not defeat coronavirus, &amp; Whole nation know Who is responsi
{"quote_count": 0, "quoted_status_permalink": {"url": "https://t.co/gHQnXkOV0B", "expanded": "https://twitter.com/news18kannada/status/124777999073888
{"quote_count": 0, "contributors": null, "truncated": false, "text": "Fox News is officially being sued for peddling coronavirus misinformation / Quee
{"quote_count": 9, "contributors": null, "truncated": true, "text": "We need to get these people out of government!\nJUST IN: Democrats Want Illegal I
{"quote_count": 0, "contributors": null, "truncated": false, "text": "Some cards from my students ( with good wishes in time of coronavirus) Read and
{"quote_count": 1, "contributors": null, "truncated": true, "text": "In case the scientific \u2018experts\u2019  haven\u2019t realised it yet, the est
{"quote_count": 7, "quoted_status_permalink": {"url": "https://t.co/ZpdN94mmXf", "expanded": "https://twitter.com/zarahsultana/status/124777404663836
{"quote_count": 0, "contributors": null, "truncated": true, "text": "\u201cUnbelievable results\u201d = unreliable data. Division chief Dr Mark Rupp
{"quote_count": 0, "contributors": null, "truncated": false, "text": "Broadway Will Remain Dark Until June (at Least) as Coronavirus Shutdown Extended
{"quote_count": 255, "contributors": null, "truncated": true, "text": "Kobe Bryant, Australia bush fires, US vs Iran, volcano eruption in Philippines,
{"quote_count": 1, "contributors": null, "truncated": true, "text": "PLO Lumumba: A lot more have died in Africa out of \u201cSECURITY FORCES\u201d br
{"quote_count": 4, "contributors": null, "truncated": false, "text": "Media briefing on #COVID19 with @DrTedros. #coronavirus https://t.co/B0eVuytYHs"
{"quote_count": 125, "contributors": null, "truncated": true, "text": "I'm sad to hear this from Dennis.  All those years of great work tossed out the
{"quote_count": 25, "contributors": null, "truncated": true, "text": "UK will have Europe's worst coronavirus death toll, says study\n\nIf this is tru
```

Figure 15: Example JSON Data

7.3 Archive Webpages

In this step, we want to extract webpages from the extracted URLs from the first step and archive them in the form of WARC files, which will facilitate the following steps. The Web Archive (WARC) format specifies a method for combining multiple digital resources into an aggregate archive file together with related information, which can easily be used to handle and process.

1. Import several important packages to process the URLs file including warcio, sys, and requests:

```
        from warcio.capture_http import capture_http
        import requests # requests must be imported after capture_http
        import sys
```

2. Get data in URLs file from command line argument

```
        urlsFile = sys.argv[1]
```

3. Read lines from the file to extract URL lists

```
with open(urlsFile) as f:
    urls = f.readlines()
urls = [u.strip() for u in urls]
```

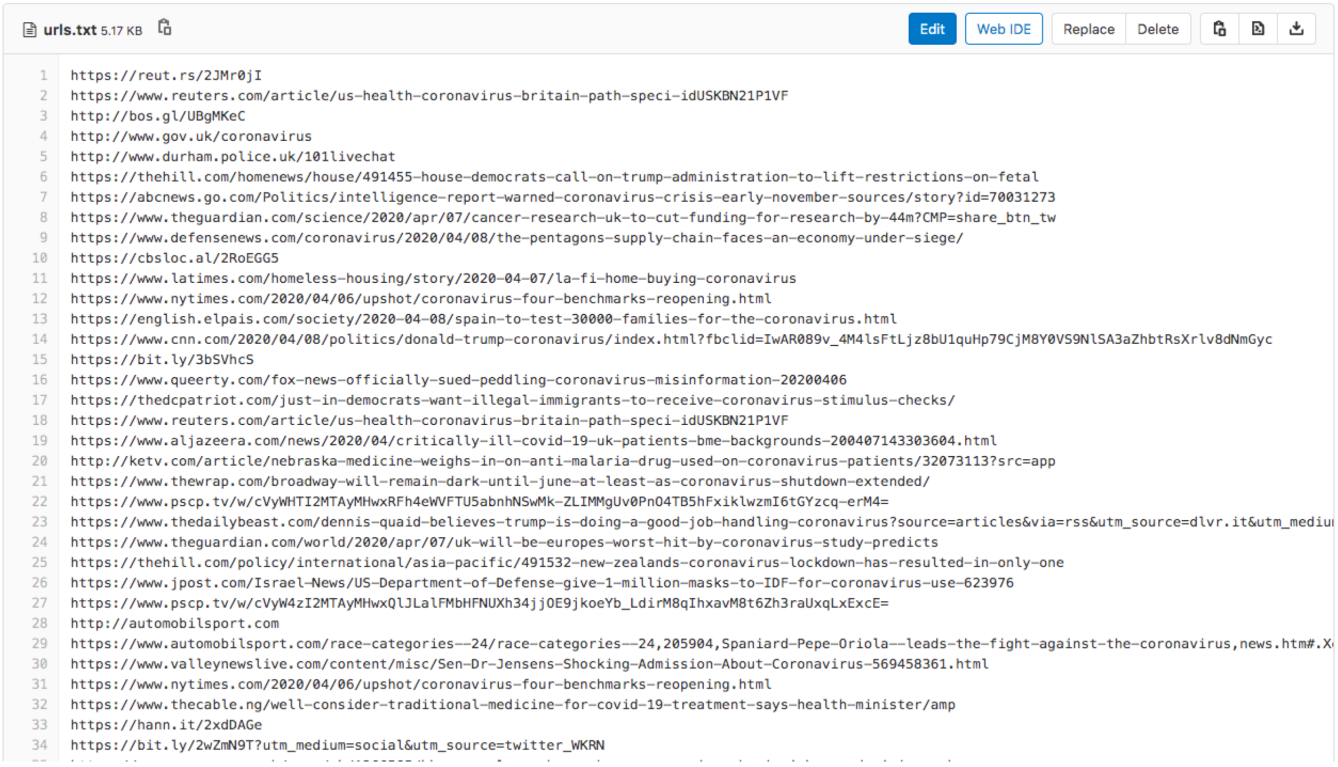
4. Filter Request and Response Records. 200 is a success code. If the request doesn't return 200, it means the webpage is not successfully retrieved therefore we will filter it out.

```
def filter_records(request, response, request_recorder):
    # return None, None to indicate records should be skipped
    if response.http_headers.get_statuscode() != '200':
    return None, None

    return request, response
```

5. Archive Webpages into WARC file

```
with capture_http('test.warc.gz'):
    for url in urls:
        requests.get(url)
```

6. Example Input Data (Figure 22)



```
https://reut.rs/2JMr0jI
https://www.reuters.com/article/us-health-coronavirus-britain-path-speci-idUSKBN21P1VF
http://bos.gl/UBgMKeC
http://www.gov.uk/coronavirus
http://www.durham.police.uk/101livechat
https://thehill.com/homenews/house/491455-house-democrats-call-on-trump-administration-to-lift-restrictions-on-fetal
https://abcnews.go.com/Politics/intelligence-report-warned-coronavirus-crisis-early-november-sources/story?id=70031273
https://www.theguardian.com/science/2020/apr/07/cancer-research-uk-to-cut-funding-for-research-by-44m?CMP=share_btn_tw
https://www.defensenews.com/coronavirus/2020/04/08/the-pentagons-supply-chain-faces-an-economy-under-siege/
https://cbsloc.al/2RoEGG5
https://www.latimes.com/homeless-housing/story/2020-04-07/la-fi-home-buying-coronavirus
https://www.nytimes.com/2020/04/06/upshot/coronavirus-four-benchmarks-reopening.html
https://english.elpais.com/society/2020-04-08/spain-to-test-30000-families-for-the-coronavirus.html
https://www.cnn.com/2020/04/08/politics/donald-trump-coronavirus/index.html?fbclid=IwAR089v_4M4lsFtLjz8bU1quHp79CjM8Y0VS9NlSA3aZhbtRsXrlv8dNmGyc
https://bit.ly/3bSVhcS
https://www.queerty.com/fox-news-officially-sued-peddling-coronavirus-misinformation-20200406
https://thedcpatriot.com/just-in-democrats-want-illegal-immigrants-to-receive-coronavirus-stimulus-checks/
https://www.reuters.com/article/us-health-coronavirus-britain-path-speci-idUSKBN21P1VF
https://www.aljazeera.com/news/2020/04/critically-ill-covid-19-uk-patients-bme-backgrounds-200407143303604.html
http://ketv.com/article/nebraska-medicine-weighs-in-on-anti-malaria-drug-used-on-coronavirus-patients/32073113?src=app
https://www.thewrap.com/broadway-will-remain-dark-until-june-at-least-as-coronavirus-shutdown-extended/
https://www.pscp.tv/w/cVyWHTI2MTAyMHwxRFh4eWVFTU5abnhNSwMk-ZLIMMgUv0PnO4TB5hFxiklwzmI6tGYzcq-erM4=
https://www.thedailybeast.com/dennis-quaid-believes-trump-is-doing-a-good-job-handling-coronavirus?source=articles&via=rss&utm_source=dlvr.it&utm_mediu
https://www.theguardian.com/world/2020/apr/07/uk-will-be-europes-worst-hit-by-coronavirus-study-predicts
https://thehill.com/policy/international/asia-pacific/491532-new-zealands-coronavirus-lockdown-has-resulted-in-only-one
https://www.jpost.com/Israel-News/US-Department-of-Defense-give-1-million-masks-to-IDF-for-coronavirus-use-623976
https://www.pscp.tv/w/cVyW4zI2MTAyMHwxQlJLalFMbHFNUXh34jjOE9jkoeYb_LdirM8qIhxavM8t6Zh3raUxqLxExcE=
http://automobilsport.com
https://www.automobilsport.com/race-categories--24/race-categories--24,205904,Spaniard-Pepe-Oriola--leads-the-fight-against-the-coronavirus,news.htm#.X
https://www.valleynewslive.com/content/misc/Sen-Dr-Jensens-Shocking-Admission-About-Coronavirus-569458361.html
https://www.nytimes.com/2020/04/06/upshot/coronavirus-four-benchmarks-reopening.html
https://www.thecable.ng/well-consider-traditional-medicine-for-covid-19-treatment-says-health-minister/amp
https://hann.it/2xdDAGe
https://bit.ly/2wZmN9T?utm_medium=social&utm_source=twitter_WKRN
```

Figure 16: Archive URL Input Data

7.4 Extract Data

After WARC files are generated from the Archive URLs group, the warcio Python module is used to grab the HTML from the WARC files. Once the HTML is obtained, BeautifulSoup is used to extract text from HTML files. Then, the HTML's content is transferred to a JSON file using JSON and Jsonlines. After that, these JSON files will be transferred to ElasticSearch to generate indexes for further usage.

1. Import important packages to process the WARC files

```
from warcio.archiveiterator import ArchiveIterator
from bs4 import BeautifulSoup
import JSON
import re
```

2. Using BeautifulSoup API to get text content from HTML and store the URL, title, text content into dict

```
def parse_content(url, content, JSON_file):
    soup = BeautifulSoup(content, 'html.parser')
    content = soup.get_text()
    content = re.sub('\n\r+', '\n\r',re.sub('\n+', '\n', content))
    title = soup.find('title').string
    JSON_list.append({"URL":url, "title":title, "webpage_text":
       ↪ content})
```

3. Using JSONlines and JSON API to process HTML information.

```
def transfer_JSONl(content):
    with open('output.JSONl', 'w') as outfile:
        for entry in JSON_list:
            JSON.dump(entry, outfile)
            outfile.write('\n')
```

4. Reading WARC file in the way of file stream

```
JSON_list = []
i = 0
url_list = []
with open('new1.warc', 'rb') as stream:
```

5. Using Warcio API to load file stream and get URL of HTML and the content of HTML

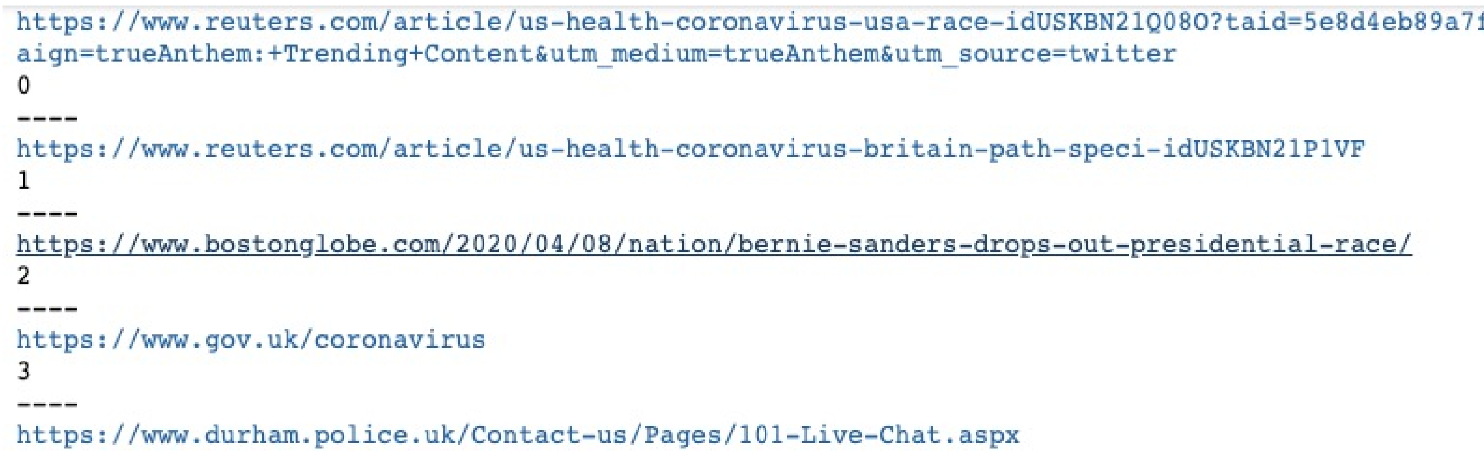
```
for record in ArchiveIterator(stream, arc2warc=True):
if record.rec_type == 'response':
    if 'text/html'in record.http_headers.get_header('Content-Type'):
        url = record.rec_headers.get_header('WARC-Target-URI')
```

```
                #check for duplicate!
                if url in url_list:
                    continue
                url_list.append(url)
                print(record.rec_headers.get_header('WARC-Target-URI'))
                print(i)
                print("----")
                i+=1
                parse_content(url, record.content_stream().read(), JSON_list)
```

6. Generate a JSON file in the current folder

```
        transfer_JSON1(JSON_list)
```

7. Example Data:



```
https://www.reuters.com/article/us-health-coronavirus-usa-race-idUSKBN21Q08O?taid=5e8d4eb89a7f
aign=trueAnthem:+Trending+Content&utm_medium=trueAnthem&utm_source=twitter
0
----
https://www.reuters.com/article/us-health-coronavirus-britain-path-speci-idUSKBN21P1VF
1
----
https://www.bostonglobe.com/2020/04/08/nation/bernie-sanders-drops-out-presidential-race/
2
----
https://www.gov.uk/coronavirus
3
----
https://www.durham.police.uk/Contact-us/Pages/101-Live-Chat.aspx
```

Figure 17: Output of Extract Data

7.5 Text Classification

Text Classification is a function that classifies text according to a certain category or categories. The category of text we are classifying in this task is COVID-19 information. The overall system used is an embedding algorithm that captures the semantic information from the texts and then an SVM classifier uses these contextualized embeddings for final classification. In order to make the system work, training is required when we first want to capture the category we want to classify. This is done by feeding in relevant text data to train a set of parameters that convert input texts to embeddings(parameters of DOC2Vec is saved), and then use these embeddings to train the parameters in the one class SVM(parameters of SVM is also saved). Eventually the pipeline is ready to use for future classification by the predict function. The input of the model is just raw text and the output is a file containing the classification results. The category can be easily changed to a different subject by simply retraining the model with text data from a desired category. The train function will save the parameters trained on the given category and use that as the basis for text classification on selected category.

1. Import necessary library

```
    from gensim.test.utils import common_texts
    from gensim.models.doc2vec import Doc2Vec, TaggedDocument
    from nltk.tokenize import word_tokenize
    from sklearn.svm import OneClassSVM
    from joblib import dump, load
    import numpy as np
    import os
    import nltk
```

2. Initialize the classifier which loads pretrained models

```
    model = CoronaClassifier()
```

3. Read the text file

```
    text_input = open(input_file, 'r')
    lines = text_input.readlines()
    print(text_input)
    print(lines)
```

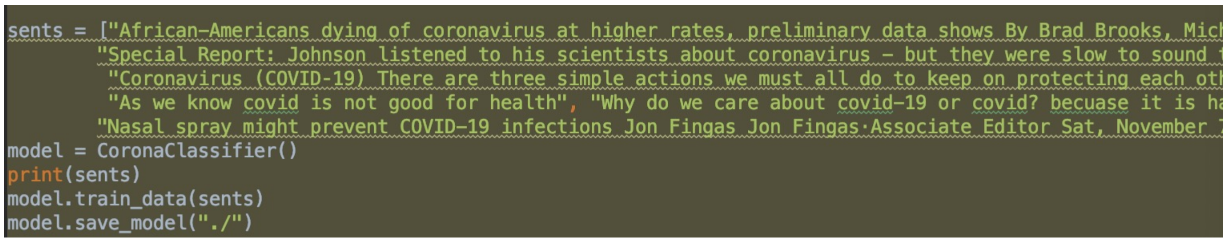
4. Make Predictions

```
    predictions = model.predict_multiple(lines)
    print(predictions)
```

5. Write results to a text file

```
    with open(output_name, 'w') as outfile:
        for predict in predictions:
            outfile.write("{}\n".format(predict))
```

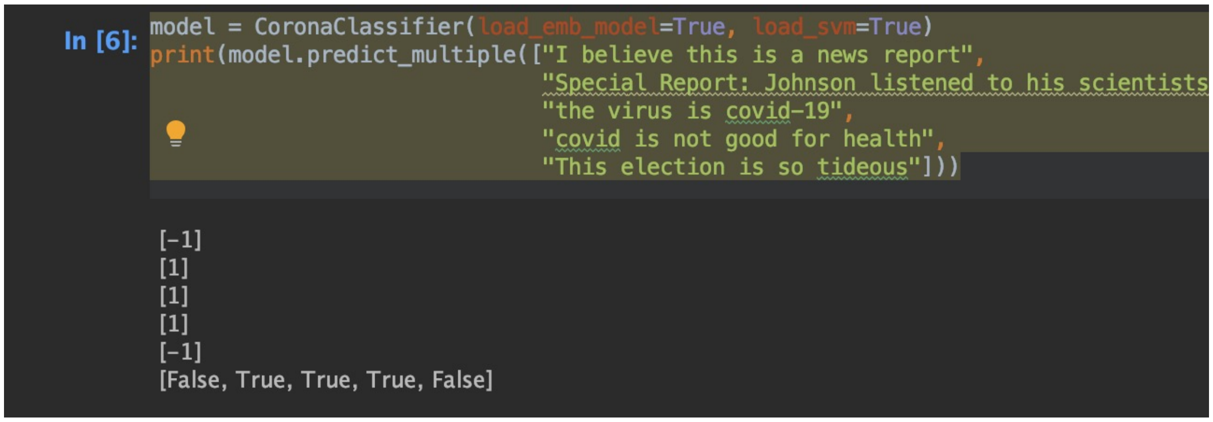
6. Example of training and saving parameters of Doc2Vec and SVM:



```
sents = ["African-Americans dying of coronavirus at higher rates, preliminary data shows By Brad Brooks, Mic
        "Special Report: Johnson listened to his scientists about coronavirus - but they were slow to sound
         "Coronavirus (COVID-19) There are three simple actions we must all do to keep on protecting each oth
         "As we know covid is not good for health", "Why do we care about covid-19 or covid? becuase it is ha
        "Nasal spray might prevent COVID-19 infections Jon Fingas Jon Fingas·Associate Editor Sat, November
model = CoronaClassifier()
print(sents)
model.train_data(sents)
model.save_model("./")
```

Figure 18: Text Classification Training

7. Example of how text classifcation results:



```
In [6]: model = CoronaClassifier(load_emb_model=True, load_svm=True)
        print(model.predict_multiple(["I believe this is a news report",
                                      "Special Report: Johnson listened to his scientists
                                      "the virus is covid-19",
                                      "covid is not good for health",
                                      "This election is so tideous"]))

        [-1]
        [1]
        [1]
        [1]
        [-1]
        [False, True, True, True, False]
```

Figure 19: Text Classification Examples

7.6 Text Summarization

Text summarization is a function of summarizing text, grabbing key words or information from the input text and removing unwanted words, punctuation and links. The format of the input and output are set to be the same. We use JSON files as input and output in this case.

1. Import important packages to process the JSON files

```
import os
import nltk
import string
import jsonlines
from bs4 import BeautifulSoup as bs
from nltk.corpus import stopwords
from gensim.summarization import summarize
#pip install rouge
from rouge import Rouge
import json
```

2. Methods for removing HTML tagging, punctuation, and stopwords, and for generating rough scores.

```
nltk.download('stopwords')
def remove_html(text):
    soup = bs(text, 'lxml')
    return soup.get_text().strip()
def remove_punctuation(text):
    return "".join([c for c in text if c not in string.punctuation])
def remove_stopwords(text):
    return [w for w in text if w not in stopwords.words('english')]
```

```
def rouge_score(summary, reference):
    summary = [summary]
    reference = [reference]
    rouge = Rouge()
    rouge_score = rouge.get_scores(summary, reference)
    print()
    print(rouge_score[0]["rouge-1"])
    print(rouge_score[0]["rouge-2"])
    print(rouge_score[0]["rouge-l"])
    print('--------------------------------------------------')
```

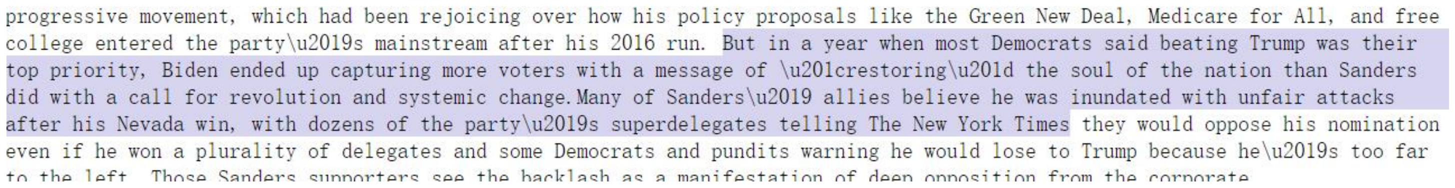
3. Summarization method to the input JSON files.

```
def sum1(file):
    with open("output.jsonl", "r+", encoding="utf8") as f:
        for item in jsonlines.Reader(f):
            content = remove_html(item['webpage_text'])
            summary = summarize(content)
            summary = summary.replace('/\r?\n|\r/g', '').strip()
            print(summary)
            rouge_score(summary, content)
```

4. Check type and condition of the input file

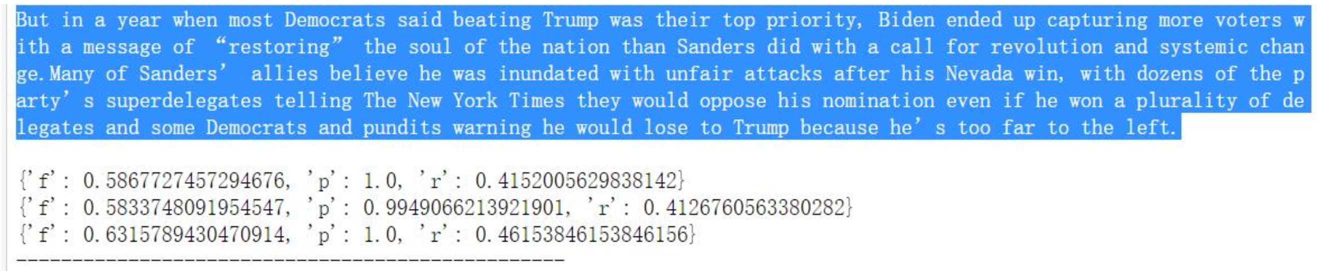
```
def notempty(file):
    if os.path.getsize('output.jsonl'):
        print('The file is NOT empty!!')
    else:
        print('The file is EMPTY!!')


def is_json(file):
    try:
        json.loads(file)

    except Exception as e:

        return e
    return True
```

5. Example of how summarization works:



```
progressive movement, which had been rejoicing over how his policy proposals like the Green New Deal, Medicare for All, and free
college entered the party\u2019s mainstream after his 2016 run. But in a year when most Democrats said beating Trump was their
top priority, Biden ended up capturing more voters with a message of \u201crestoring\u201d the soul of the nation than Sanders
did with a call for revolution and systemic change.Many of Sanders\u2019 allies believe he was inundated with unfair attacks
after his Nevada win, with dozens of the party\u2019s superdelegates telling The New York Times they would oppose his nomination
even if he won a plurality of delegates and some Democrats and pundits warning he would lose to Trump because he\u2019s too far
to the left. Those Sanders supporters see the backlash as a manifestation of deep opposition from the corporate
```

Figure 20: Before Summarizing



```
But in a year when most Democrats said beating Trump was their top priority, Biden ended up capturing more voters w
ith a message of “restoring” the soul of the nation than Sanders did with a call for revolution and systemic chan
ge.Many of Sanders’ allies believe he was inundated with unfair attacks after his Nevada win, with dozens of the p
arty’ s superdelegates telling The New York Times they would oppose his nomination even if he won a plurality of de
legates and some Democrats and pundits warning he would lose to Trump because he’ s too far to the left.

{'f': 0.5867727457294676, 'p': 1.0, 'r': 0.4152005629838142}
{'f': 0.5833748091954547, 'p': 0.9949066213921901, 'r': 0.4126760563380282}
{'f': 0.6315789430470914, 'p': 1.0, 'r': 0.46153846153846156}
--------------------------------------------------
```

Figure 21: After Summarizing

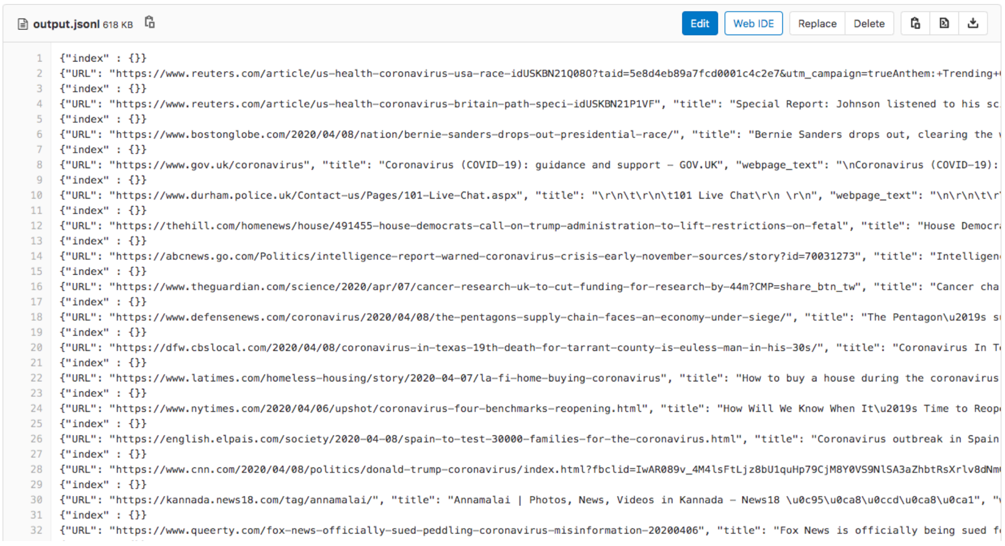
7.7 Indexing

Given a collection of indexable JSON data, the Indexing service conducts the following tasks:

- Connect to the ElasticSearch server end-point found at elasticsearch.cs.vt.edu. This is done via HTTPS on port 9200 with a private authentication token.
- Invoke ElasticSearch's bulk call to iterate through the input file, adding each document to the 'WP' index. If the 'WP' index does not exist on elasticsearch.cs.vt.edu, it is created.

The below itemized numerical list shows the format of the input file, the code for the indexing data service, and subsequently the format of the output file of the service:

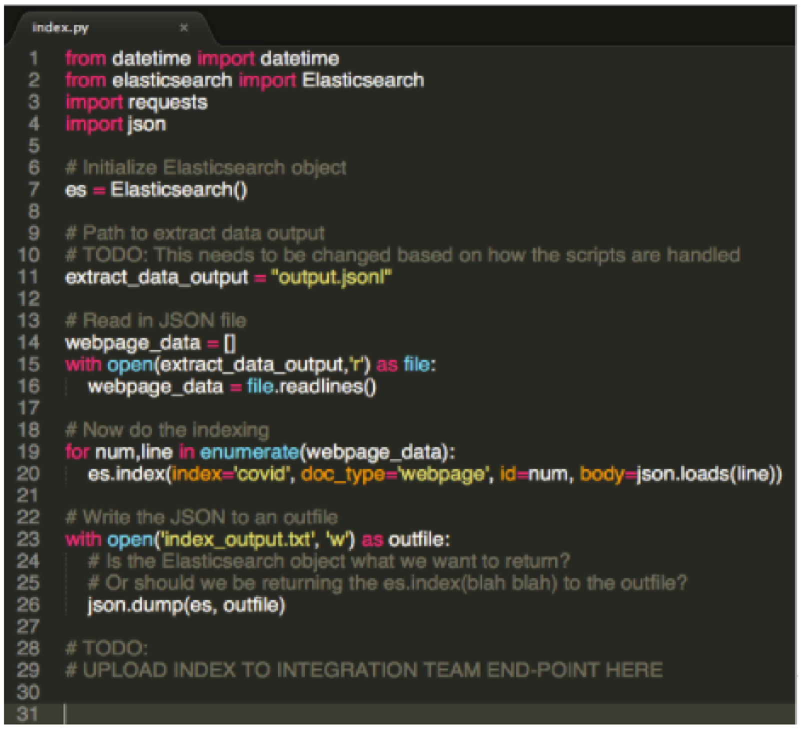
1. ElasticSearch Input Data



```
{"index" : {}}
{"URL": "https://www.reuters.com/article/us-health-coronavirus-usa-race-idUSKBN21Q080?taid=5e8d4eb89a7fcd0001c4c2e7&utm_campaign=trueAnthem:+Trending+
{"index" : {}}
{"URL": "https://www.reuters.com/article/us-health-coronavirus-britain-path-speci-idUSKBN21P1VF", "title": "Special Report: Johnson listened to his sc
{"index" : {}}
{"URL": "https://www.bostonglobe.com/2020/04/08/nation/bernie-sanders-drops-out-presidential-race/", "title": "Bernie Sanders drops out, clearing the
{"index" : {}}
{"URL": "https://www.gov.uk/coronavirus", "title": "Coronavirus (COVID-19): guidance and support - GOV.UK", "webpage_text": "\nCoronavirus (COVID-19):
{"index" : {}}
{"URL": "https://www.durham.police.uk/Contact-us/Pages/101-Live-Chat.aspx", "title": "\r\n\t\r\n\t101 Live Chat\r\n \r\n", "webpage_text": "\n\r\n\t\r
{"index" : {}}
{"URL": "https://thehill.com/homenews/house/491455-house-democrats-call-on-trump-administration-to-lift-restrictions-on-fetal", "title": "House Democr
{"index" : {}}
{"URL": "https://abcnews.go.com/Politics/intelligence-report-warned-coronavirus-crisis-early-november-sources/story?id=70031273", "title": "Intelligen
{"index" : {}}
{"URL": "https://www.theguardian.com/science/2020/apr/07/cancer-research-uk-to-cut-funding-for-research-by-44m?CMP=share_btn_tw", "title": "Cancer cha
{"index" : {}}
{"URL": "https://www.defensenews.com/coronavirus/2020/04/08/the-pentagons-supply-chain-faces-an-economy-under-siege/", "title": "The Pentagon\u2019s s
{"index" : {}}
{"URL": "https://dfw.cbslocal.com/2020/04/08/coronavirus-in-texas-19th-death-for-tarrant-county-is-euless-man-in-his-30s/", "title": "Coronavirus In T
{"index" : {}}
{"URL": "https://www.latimes.com/homeless-housing/story/2020-04-07/la-fi-home-buying-coronavirus", "title": "How to buy a house during the coronavirus
{"index" : {}}
{"URL": "https://www.nytimes.com/2020/04/06/upshot/coronavirus-four-benchmarks-reopening.html", "title": "How Will We Know When It\u2019s Time to Reop
{"index" : {}}
{"URL": "https://english.elpais.com/society/2020-04-08/spain-to-test-30000-families-for-the-coronavirus.html", "title": "Coronavirus outbreak in Spain
{"index" : {}}
{"URL": "https://www.cnn.com/2020/04/08/politics/donald-trump-coronavirus/index.html?fbclid=IwAR089v_4M4lsFtLjz8bU1quHp79CjM8Y0VS9NlSA3aZhbtRsXrlv8dNm
{"index" : {}}
{"URL": "https://kannada.news18.com/tag/annamalai/", "title": "Annamalai | Photos, News, Videos in Kannada - News18 \u0c95\u0ca8\u0ccd\u0ca8\u0ca1", "
{"index" : {}}
{"URL": "https://www.queerty.com/fox-news-officially-sued-peddling-coronavirus-misinformation-20200406", "title": "Fox News is officially being sued f
```

Figure 22: Input of ElasticSearch Data

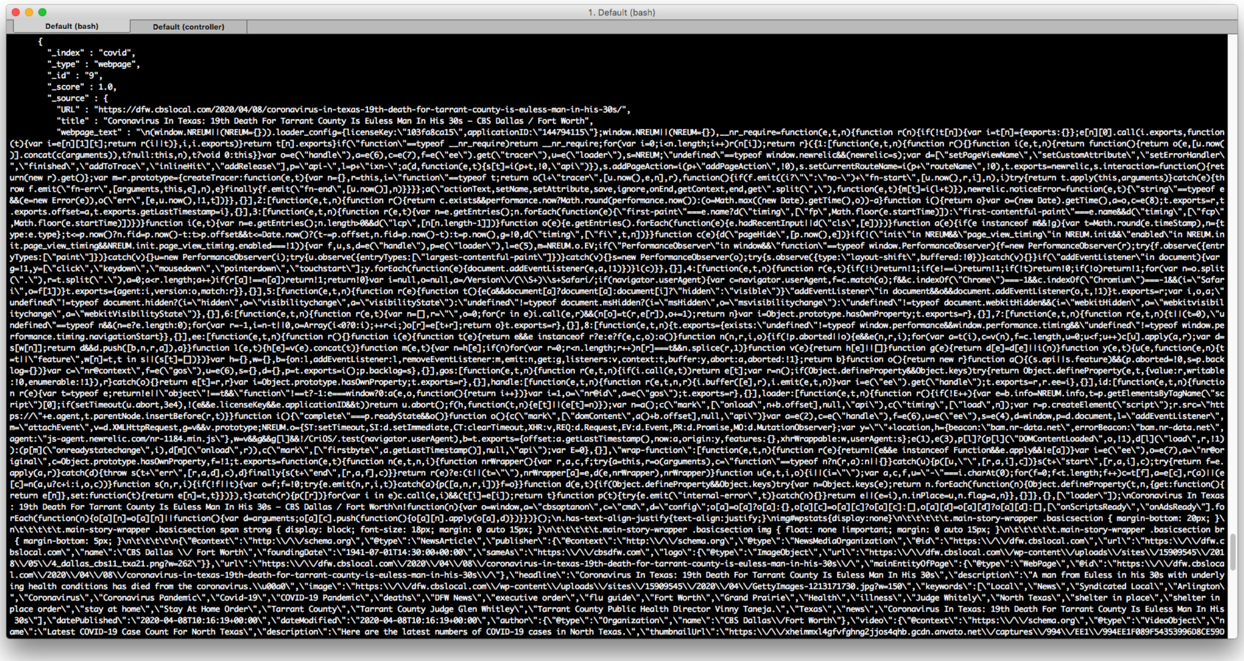
2. ElasticSearch Code



```python
from datetime import datetime
from elasticsearch import Elasticsearch
import requests
import json

# Initialize Elasticsearch object
es = Elasticsearch()

# Path to extract data output
# TODO: This needs to be changed based on how the scripts are handled
extract_data_output = "output.jsonl"

# Read in JSON file
webpage_data = []
with open(extract_data_output,'r') as file:
    webpage_data = file.readlines()

# Now do the indexing
for num,line in enumerate(webpage_data):
    es.index(index='covid', doc_type='webpage', id=num, body=json.loads(line))

# Write the JSON to an outfile
with open('index_output.txt', 'w') as outfile:
    # Is the Elasticsearch object what we want to return?
    # Or should we be returning the es.index(blah blah) to the outfile?
    json.dump(es, outfile)

# TODO:
# UPLOAD INDEX TO INTEGRATION TEAM END-POINT HERE

```

Figure 23: Code of ElasticSearch

3. ElasticSearch Indexed Output



```
      {
        "_index" : "covid",
        "_type" : "webpage",
        "_id" : "9",
        "_score" : 1.0,
        "_source" : {
          "URL" : "https://dfw.cbslocal.com/2020/04/08/coronavirus-in-texas-19th-death-for-tarrant-county-is-euless-man-in-his-30s/",
          "title" : "Coronavirus In Texas: 19th Death For Tarrant County Is Euless Man In His 30s - CBS Dallas / Fort Worth",
          "webpage_text" : "\n(window.NREUM||(NREUM={})).loader_config={licenseKey:\"103fa8ca15\",applicationID:\"144794115\"};window.NREUM||(NREUM={}),__nr_require=function(e,t,n){function r(n){if(!t[n]){var i=t[n]={exports:{}};e[n][0].call(i.exports,function
(t){var i=e[n][1][t];return r(i||t)},i,i.exports)}return t[n].exports}if(\"function\"==typeof __nr_require)return __nr_require;for(var i=0;i<n.length;i++)r(n[i]);return r}({1:[function(e,t,n){function r(){}function i(e,t,n){return function(){return o(e,[u.now(
)].concat(c(arguments)),t?null:this,n),t?void 0:this}}var o=e(\"handle\"),a=e(6),c=e(7),f=e(\"ee\").get(\"tracer\"),u=e(\"loader\"),s=NREUM;\"undefined\"==typeof window.newrelic&&(newrelic=s);var d=[\"setPageViewName\",\"setCustomAttribute\",\"setErrorHandler\
",\"finished\",\"addToTrace\",\"inlineHit\",\"addRelease\"],p=\"api-\",l=p+\"ixn-\";a(d,function(e,t){s[t]=i(p+t,!0,\"api\")}),s.addPageAction=i(p+\"addPageAction\",!0),s.setCurrentRouteName=i(p+\"routeName\",!0),t.exports=newrelic,s.interaction=function(){ret
urn(new r).get()};var m=r.prototype={createTracer:function(e,t){var n={},r=this,i=\"function\"==typeof t;return o(l+\"tracer\",[u.now(),e,n],r),function(){if(f.emit((i?\"\":\"no-\")+\"fn-start\",[u.now(),r,i],n),i)try{return t.apply(this,arguments)}catch(e){th
row f.emit(\"fn-err\",[arguments,this,e],n),e}finally{f.emit(\"fn-end\",[u.now()],n)}}}};a(\"actionText,setName,setAttribute,save,ignore,onEnd,getContext,end,get\".split(\",\"),function(e,t){m[t]=i(l+t)}),newrelic.noticeError=function(e,t){\"string\"==typeof e
&&(e=new Error(e)),o(\"err\",[e,u.now(),!1,t])}},{}],2:[function(e,t,n){function r(){return c.exists&&performance.now?Math.round(performance.now()):(o=Math.max((new Date).getTime(),o))-a}function i(){return o}var o=(new Date).getTime(),a=o,c=e(8);t.exports=r,t
.exports.offset=a,t.exports.getLastTimestamp=i},{}],3:[function(e,t,n){function r(e,t){var n=e.getEntries();n.forEach(function(e){\"first-paint\"===e.name?d(\"timing\",[\"fp\",Math.floor(e.startTime)]):\"first-contentful-paint\"===e.name&&d(\"timing\",[\"fcp\"
,Math.floor(e.startTime)])})}function i(e,t){var n=e.getEntries();n.length>0&&d(\"lcp\",[n[n.length-1]])}function o(e){e.getEntries().forEach(function(e){e.hadRecentInput||d(\"cls\",[e])})}function a(e){if(e instanceof m&&!g){var t=Math.round(e.timeStamp),n={t
ype:e.type};t<=p.now()?n.fid=p.now()-t:t>p.offset&&t<=Date.now()?(t-=p.offset,n.fid=p.now()-t):t=p.now(),g=!0,d(\"timing\",[\"fi\",t,n])}}function c(e){d(\"pageHide\",[p.now(),e])}if(!(\"init\"in NREUM&&\"page_view_timing\"in NREUM.init&&\"enabled\"in NREUM.in
it.page_view_timing&&NREUM.init.page_view_timing.enabled===!1)){var f,u,s,d=e(\"handle\"),p=e(\"loader\"),l=e(5),m=NREUM.o.EV;if(\"PerformanceObserver\"in window&&\"function\"==typeof window.PerformanceObserver){f=new PerformanceObserver(r);try{f.observe({entr
yTypes:[\"paint\"]})}catch(v){}u=new PerformanceObserver(i);try{u.observe({entryTypes:[\"largest-contentful-paint\"]})}catch(v){}s=new PerformanceObserver(o);try{s.observe({type:\"layout-shift\",buffered:!0})}catch(v){}}if(\"addEventListener\"in document){var
g=!1,y=[\"click\",\"keydown\",\"mousedown\",\"pointerdown\",\"touchstart\"];y.forEach(function(e){document.addEventListener(e,a,!1)})}l(c)}},{}],4:[function(e,t,n){function r(e,t){if(!i)return!1;if(e!==i)return!1;if(!t)return!0;if(!o)return!1;for(var n=o.split
(\".\"),r=t.split(\".\"),a=0;a<r.length;a++)if(r[a]!==n[a])return!1;return!0}var i=null,o=null,a=/Version\/(\S+)\s+Safari/;if(navigator.userAgent){var c=navigator.userAgent,f=c.match(a);f&&c.indexOf(\"Chrome\")===-1&&c.indexOf(\"Chromium\")===-1&&(i=\"Safar
i\",o=f[1])}t.exports={agent:i,version:o,match:r}},{}],5:[function(e,t,n){function r(e){function t(){e(a&&document[a]?document[a]:document[i]?\"hidden\":\"visible\")}\"addEventListener\"in document&&o&&document.addEventListener(o,t,!1)}t.exports=r;var i,o,a;\"
undefined\"!=typeof document.hidden?(i=\"hidden\",o=\"visibilitychange\",a=\"visibilityState\"):\"undefined\"!=typeof document.msHidden?(i=\"msHidden\",o=\"msvisibilitychange\"):\"undefined\"!=typeof document.webkitHidden&&(i=\"webkitHidden\",o=\"webkitvisibil
itychange\",a=\"webkitVisibilityState\")},{}],6:[function(e,t,n){function r(e,t){var n=[],r=\"\",o=0;for(r in e)i.call(e,r)&&(n[o]=t(r,e[r]),o+=1);return n}var i=Object.prototype.hasOwnProperty;t.exports=r},{}],7:[function(e,t,n){function r(e,t,n){t||(t=0),\"u
ndefined\"==typeof n&&(n=e?e.length:0);for(var r=-1,i=n-t||0,o=Array(i<0?0:i);++r<i;)o[r]=e[t+r];return o}t.exports=r},{}],8:[function(e,t,n){t.exports={exists:\"undefined\"!=typeof window.performance&&window.performance.timing&&\"undefined\"!=typeof window.pe
rformance.timing.navigationStart}},{}],ee:[function(e,t,n){function r(){}function i(e){function t(e){return e&&e instanceof r?e:e?f(e,c,o):o()}function n(n,r,i,o){if(!p.aborted||o){e&&e(n,r,i);for(var a=t(i),c=v(n),f=c.length,u=0;u<f;u++)c[u].apply(a,r);var d=
s[w[n]];return d&&d.push([b,n,r,a]),a}}function l(e,t){h[e]=v(e).concat(t)}function m(e,t){var n=h[e];if(n)for(var r=0;r<n.length;r++)n[r]===t&&n.splice(r,1)}function v(e){return h[e]||[]}function g(e){return d[e]=d[e]||i(n)}function y(e,t){u(e,function(e,n){t
=t||\"feature\",w[n]=t,t in s||(s[t]=[])})}var h={},w={},b={on:l,addEventListener:l,removeEventListener:m,emit:n,get:g,listeners:v,context:t,buffer:y,abort:a,aborted:!1};return b}function o(){return new r}function a(){(s.api||s.feature)&&(p.aborted=!0,s=p.back
log={})}var c=\"nr@context\",f=e(\"gos\"),u=e(6),s={},d={},p=t.exports=i();p.backlog=s},{}],gos:[function(e,t,n){function r(e,t,n){if(i.call(e,t))return e[t];var r=n();if(Object.defineProperty&&Object.keys)try{return Object.defineProperty(e,t,{value:r,writable
:!0,enumerable:!1}),r}catch(o){}return e[t]=r,r}var i=Object.prototype.hasOwnProperty;t.exports=r},{}],handle:[function(e,t,n){function r(e,t,n,r){i.buffer([e],r),i.emit(e,t,n)}var i=e(\"ee\").get(\"handle\");t.exports=r,r.ee=i},{}],id:[function(e,t,n){functio
n r(e){var t=typeof e;return!e||\"object\"!==t&&\"function\"!==t?-1:e===window?0:a(e,o,function(){return i++})}var i=1,o=\"nr@id\",a=e(\"gos\");t.exports=r},{}],loader:[function(e,t,n){function r(){if(!E++){var e=b.info=NREUM.info,t=p.getElementsByTagName(\"sc
ript\")[0];if(setTimeout(u.abort,3e4),!(e&&e.licenseKey&&e.applicationID&&t))return u.abort();f(h,function(t,n){e[t]||(e[t]=n)});var n=a();c(\"mark\",[\"onload\",n+b.offset],null,\"api\"),c(\"timing\",[\"load\",n]);var r=p.createElement(\"script\");r.src=\"htt
ps://\"+e.agent,t.parentNode.insertBefore(r,t)}}function i(){\"complete\"===p.readyState&&o()}function o(){c(\"mark\",[\"domContent\",a()+b.offset],null,\"api\")}var a=e(2),c=e(\"handle\"),f=e(6),u=e(\"ee\"),s=e(4),d=window,p=d.document,l=\"addEventListener\",
m=\"attachEvent\",v=d.XMLHttpRequest,g=v&&v.prototype;NREUM.o={ST:setTimeout,SI:d.setImmediate,CT:clearTimeout,XHR:v,REQ:d.Request,EV:d.Event,PR:d.Promise,MO:d.MutationObserver};var y=\"\"+location,h={beacon:\"bam.nr-data.net\",errorBeacon:\"bam.nr-data.net\",
agent:\"js-agent.newrelic.com/nr-1184.min.js\"},w=v&&g&&g[l]&&!/CriOS/.test(navigator.userAgent),b=t.exports={offset:a.getLastTimestamp(),now:a,origin:y,features:{},xhrWrappable:w,userAgent:s};e(1),e(3),p[l]?(p[l](\"DOMContentLoaded\",o,!1),d[l](\"load\",r,!1)
):(p[m](\"onreadystatechange\",i),d[m](\"onload\",r)),c(\"mark\",[\"firstbyte\",a.getLastTimestamp()],null,\"api\");var E=0},{}],\"wrap-function\":[function(e,t,n){function r(e){return!(e&&e instanceof Function&&e.apply&&!e[a])}var i=e(\"ee\"),o=e(7),a=\"nr@or
iginal\",c=Object.prototype.hasOwnProperty,f=!1;t.exports=function(e,t){function n(e,t,n,i){function nrWrapper(){var r,a,c,f;try{a=this,r=o(arguments),c=\"function\"==typeof n?n(r,a):n||{}}catch(u){p([u,\"\",[r,a,i],c])}s(t+\"start\",[r,a,i],c);try{return f=e.
apply(a,r)}catch(d){throw s(t+\"err\",[r,a,d],c),d}finally{s(t+\"end\",[r,a,f],c)}}return r(e)?e:(t||(t=\"\"),nrWrapper[a]=e,d(e,nrWrapper),nrWrapper)}function u(e,t,i,o){i||(i=\"\");var a,c,f,u=\"-\"===i.charAt(0);for(f=0;f<t.length;f++)c=t[f],a=e[c],r(a)||(e
[c]=n(a,u?c+i:i,o,c))}function s(n,r,i){if(!f||t){var o=f;f=!0;try{e.emit(n,r,i,t)}catch(a){p([a,n,r,i])}f=o}}function d(e,t){if(Object.defineProperty&&Object.keys)try{var n=Object.keys(e);return n.forEach(function(n){Object.defineProperty(t,n,{get:function(){
return e[n]},set:function(t){return e[n]=t,t}})}),t}catch(r){p([r])}for(var i in e)c.call(e,i)&&(t[i]=e[i]);return t}function p(t){try{e.emit(\"internal-error\",t)}catch(n){}}return e||(e=i),n.inPlace=u,n.flag=a,n}},{}]},{},[\"loader\"]);\nCoronavirus In Texas
: 19th Death For Tarrant County Is Euless Man In His 30s - CBS Dallas / Fort Worth\n!function(n){var o=window,a=\"cbsoptanon\",c=\"cmd\",d=\"config\";o[a]=o[a]?o[a]:{},o[a][c]=o[a][c]?o[a][c]:[],o[a][d]=o[a][d]?o[a][d]:[],[\"onScriptsReady\",\"onAdsReady\"].fo
rEach(function(n){o[a][n]=o[a][n]||function(){var d=arguments;o[a][c].push(function(){o[a][n].apply(o[a],d)})}})}();\n.has-text-align-justify{text-align:justify;}\nimg#wpstats{display:none}\n\t\t\t\t\t.main-story-wrapper .basicsection { margin-bottom: 20px; }\
n\t\t\t\t\t.main-story-wrapper .basicsection span strong { display: block; font-size: 18px; margin: 0 auto 15px; }\n\t\t\t\t\t.main-story-wrapper .basicsection img { float: none !important; margin: 0 auto 15px; }\n\t\t\t\t\t.main-story-wrapper .basicsection br
 { margin-bottom: 5px; }\n\t\t\t\t\n{\"@context\":\"http:\/\/schema.org\",\"@type\":\"NewsArticle\",\"publisher\":{\"@context\":\"http:\/\/schema.org\",\"@type\":\"NewsMediaOrganization\",\"@id\":\"https:\/\/dfw.cbslocal.com\",\"url\":\"https:\/\/dfw.c
bslocal.com\",\"name\":\"CBS Dallas \/ Fort Worth\",\"foundingDate\":\"1941-07-01T14:30:00+00:00\",\"sameAs\":\"https:\/\/cbsdfw.com\",\"logo\":{\"@type\":\"ImageObject\",\"url\":\"https:\/\/dfw.cbslocal.com\/wp-content\/uploads\/sites\/15909545\/201
8\/05\/4_dallas_cbs11_txa21.png?w=262\"}},\"url\":\"https:\/\/dfw.cbslocal.com\/2020\/04\/08\/coronavirus-in-texas-19th-death-for-tarrant-county-is-euless-man-in-his-30s\/\",\"mainEntityOfPage\":{\"@type\":\"WebPage\",\"@id\":\"https:\/\/dfw.cbsloca
l.com\/2020\/04\/08\/coronavirus-in-texas-19th-death-for-tarrant-county-is-euless-man-in-his-30s\/\"},\"headline\":\"Coronavirus In Texas: 19th Death For Tarrant County Is Euless Man In His 30s\",\"description\":\"A man from Euless in his 30s with underly
ing health conditions has died from the coronavirus.\u00a0\",\"image\":\"https:\/\/dfw.cbslocal.com\/wp-content\/uploads\/sites\/15909545\/2020\/04\/GettyImages-1213171730.jpg?w=150\",\"keywords\":[\"Local\",\"News\",\"Syndicated Local\",\"Arlington\
",\"Coronavirus\",\"Coronavirus Pandemic\",\"Covid-19\",\"COVID-19 Pandemic\",\"deaths\",\"DFW News\",\"executive order\",\"flu guide\",\"Fort Worth\",\"Grand Prairie\",\"Health\",\"illness\",\"Judge Whitely\",\"North Texas\",\"shelter in place\",\"shelter in
place order\",\"stay at home\",\"Stay At Home Order\",\"Tarrant County\",\"Tarrant County Judge Glen Whitley\",\"Tarrant County Public Health Director Vinny Taneja.\",\"Texas\",\"news\",\"Coronavirus In Texas: 19th Death For Tarrant County Is Euless Man In His
 30s\"],\"datePublished\":\"2020-04-08T10:16:19+00:00\",\"dateModified\":\"2020-04-08T10:16:19+00:00\",\"author\":{\"@type\":\"Organization\",\"name\":\"CBS Dallas\/Fort Worth\"},\"video\":{\"@context\":\"https:\/\/schema.org\",\"@type\":\"VideoObject\",\"n
ame\":\"Latest COVID-19 Case Count For North Texas\",\"description\":\"Here are the latest numbers of COVID-19 cases in North Texas.\",\"thumbnailUrl\":\"https:\/\/xheimmxl4gfvfghng2jjos4qhb.gcdn.anvato.net\/captures\/994\/EE1\/994EE1F089F54353996D8CE59D
```

Figure 24: ElasticSearch Index Output Data Sample

7.8 Unit testing

Unit tests are usually automated tests written and executed by software developers to ensure that a part of an application (called a "unit") conforms to its design and performs as expected. In process programming, a unit can be an entire module, but more generally a single function or process. In normal programming, a unit test is generally a class that inherits the previous class, we use this method for unit testing.

Using TestExtractData as example:

1. Import important packages to process the Unit Test for Extract Data

```
import unittest
from warcio.archiveiterator import ArchiveIterator
from bs4 import BeautifulSoup
import json
import re
#Import the method we want to test from ExtractData
from ExtractData import extract_from_warc
```

2. Use the unittest package that comes with Python to build the Unit test class.

```
class Test_method(unittest.TestCase):

@classmethod
def setUpClass(cls):
        print("Before test case============")

    @classmethod
    def tearDownClass(cls):
        print("after test case============")
```

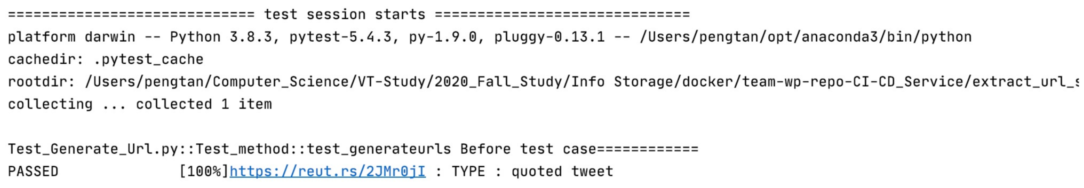
3. Build the corresponding test method according to the method to be tested

```
    def test_extractData(self):
    str_ = 'https://reut.rs/2JMr0jI'
    Flag = 0

        with open('covid.warc.gz', 'rb') as file:
            extract_from_warc(file)
        with open("output.jsonl",'w') as output_file:

            for line in output_file.readlines():
                print(line)
                if str_ in line:
                    Flag=1
                    break

    self.assertEqual(Flag, 1)
```

4. Run this test

```
    if (__name__ == '__main__'):
        unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

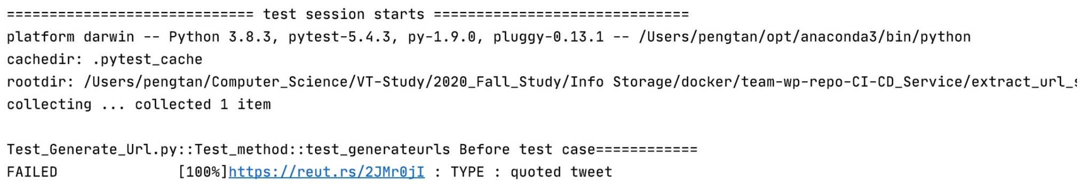
5. The Output of The Unit Test Passed:



```
============================= test session starts ==============================
platform darwin -- Python 3.8.3, pytest-5.4.3, py-1.9.0, pluggy-0.13.1 -- /Users/pengtan/opt/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /Users/pengtan/Computer_Science/VT-Study/2020_Fall_Study/Info Storage/docker/team-wp-repo-CI-CD_Service/extract_url_
collecting ... collected 1 item

Test_Generate_Url.py::Test_method::test_generateurls Before test case============
PASSED                [100%]https://reut.rs/2JMr0jI : TYPE : quoted tweet
```

Figure 25: Unit Test Passed Examples

6. The Output of The Unit Test Failed:



```
============================ test session starts ==============================
platform darwin -- Python 3.8.3, pytest-5.4.3, py-1.9.0, pluggy-0.13.1 -- /Users/pengtan/opt/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /Users/pengtan/Computer_Science/VT-Study/2020_Fall_Study/Info Storage/docker/team-wp-repo-CI-CD_Service/extract_url_
collecting ... collected 1 item

Test_Generate_Url.py::Test_method::test_generateurls Before test case============
FAILED             [100%]https://reut.rs/2JMr0jI : TYPE : quoted tweet
```

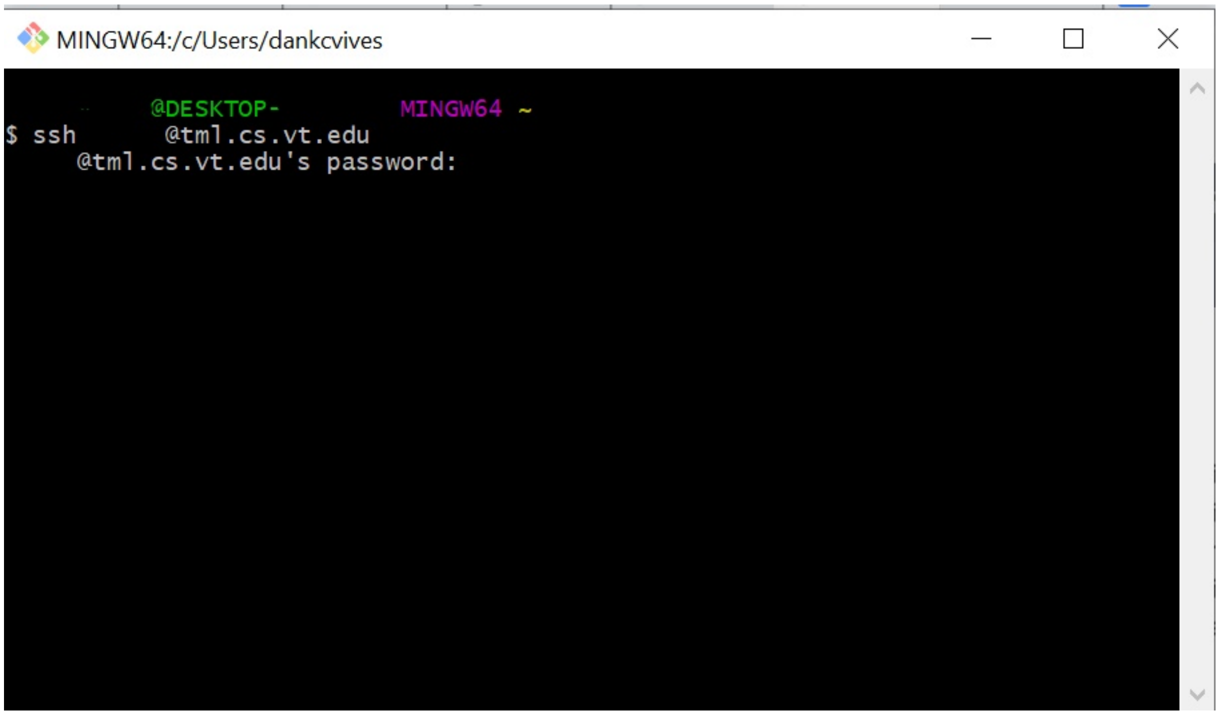
Figure 26: Unit Test Failed Examples

In this Unit Test, we will import the method needed to test. Create a test case by subclassing unittest.TestCase. This separate test is defined using methods whose names begin with the letters test. In test-extractData, it will use to test the imported method. When the test passes, the output window will display Passed. If the test fails, the output window will display the corresponding error message.

7.9 Accessing the Virtual Machine

Accessing the virtual machine is a necessity when interacting with the project, as it contains all the sample Twitter data that Mohamed provided. In order to do so, an account on the virtual machine, SSH, and a connection to the Virginia Tech VPN will be required. Accessing the virtual machine is done by following these steps:

1. Request an account for the virtual machine from Mohamed.
2. Install SSH in a shell that can operate SSH such as gitlab for Windows. This step is not required for Mac and Linux users.
3. Connect to the Virginia Tech VPN, as outlined §**??**
4. SSH into the virtual machine through *tml.cs.vt.edu* using the provided password



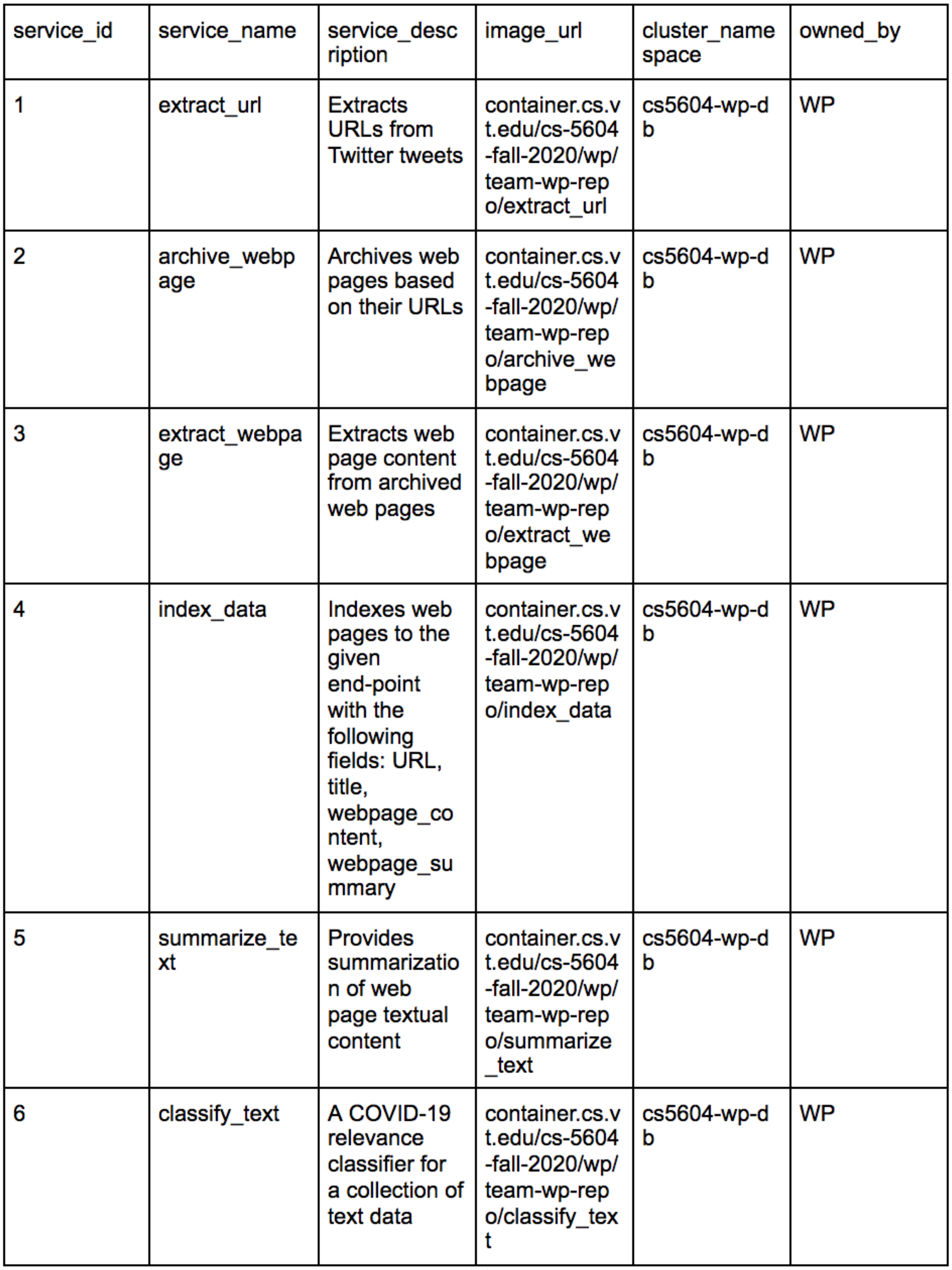
```
[redacted]@DESKTOP-[redacted] MINGW64 ~
$ ssh [redacted]@tml.cs.vt.edu
[redacted]@tml.cs.vt.edu's password:
```

Figure 27: Using SSH to Connect to VPN

5. Success! You have access to the VPN with all the tweet data.

7.10 Service Registration

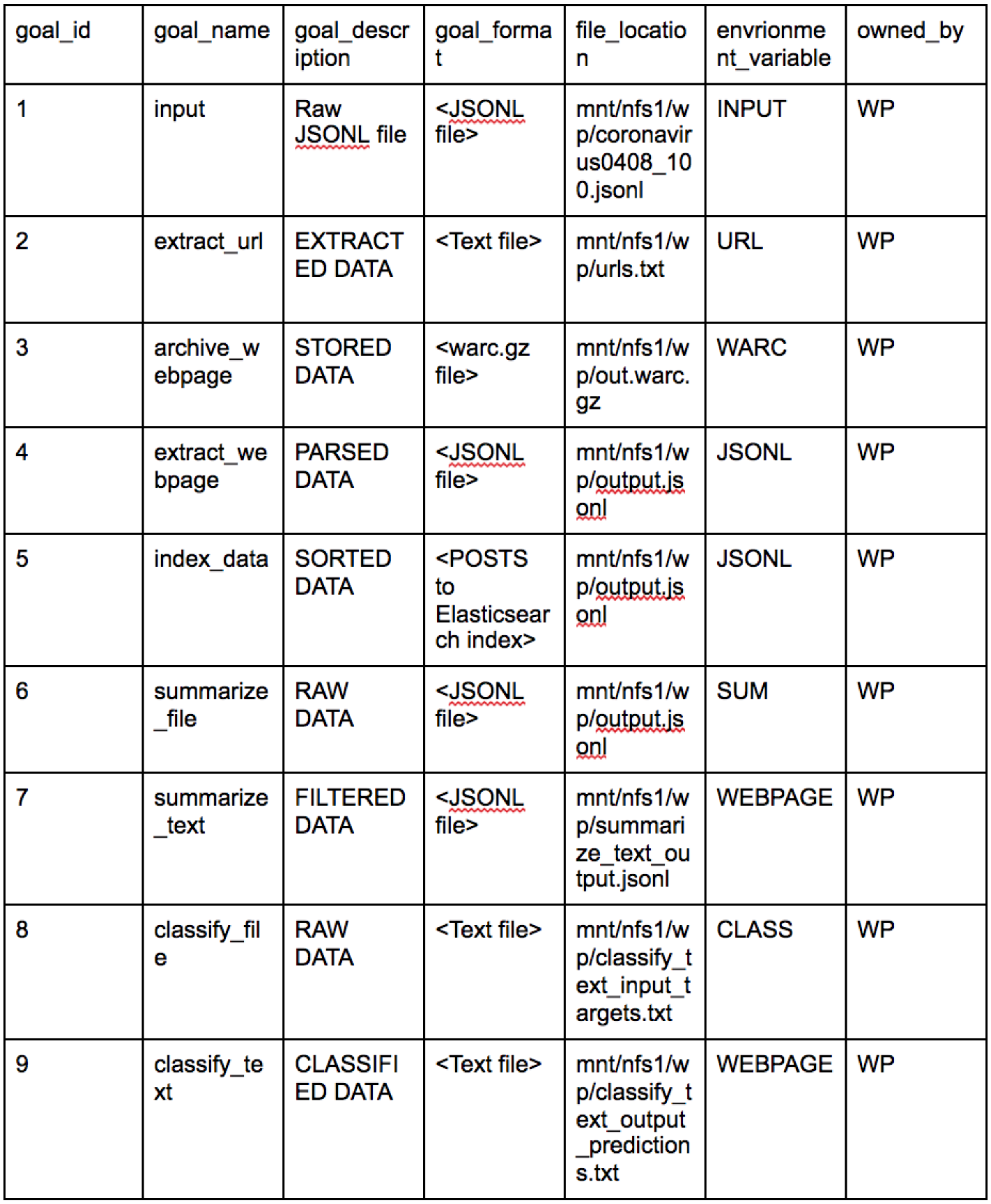
To deploy services to cloud.cs.vt.edu, a series of tables were created and provided to the Integration team. These tables follow Prashant Chandrasekar's (peecee@vt.edu) Artifact Guidelines, which initially served to both inform the design for the front-end team and generate workflows based on the goals.



service_id	service_name	service_description	image_url	cluster_namespace	owned_by
1	extract_url	Extracts URLs from Twitter tweets	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_url	cs5604-wp-db	WP
2	archive_webpage	Archives web pages based on their URLs	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/archive_webpage	cs5604-wp-db	WP
3	extract_webpage	Extracts web page content from archived web pages	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/extract_webpage	cs5604-wp-db	WP
4	index_data	Indexes web pages to the given end-point with the following fields: URL, title, webpage_content, webpage_summary	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/index_data	cs5604-wp-db	WP
5	summarize_text	Provides summarization of web page textual content	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/summarize_text	cs5604-wp-db	WP
6	classify_text	A COVID-19 relevance classifier for a collection of text data	container.cs.vt.edu/cs-5604-fall-2020/wp/team-wp-repo/classify_text	cs5604-wp-db	WP

Figure 28: Service Table

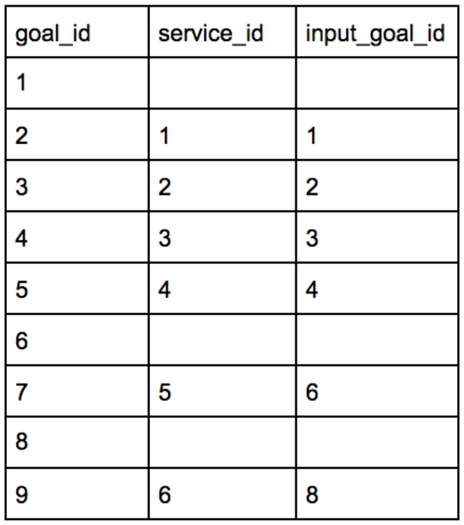
The three tables (Service, Goal, and Reasoner) allow the Integration team to register our deployed services on Rancher. Table 28 shows the services the WP team has registered on Airflow. Table 29 shows the goals the WP team has registered on Airflow. Table 30 shows how the services and goals work together based on file inputs and outputs.



goal_id	goal_name	goal_description	goal_format	file_location	envrionment_variable	owned_by
1	input	Raw JSONL file	<JSONL file>	mnt/nfs1/wp/coronavirus0408_100.jsonl	INPUT	WP
2	extract_url	EXTRACTED DATA	<Text file>	mnt/nfs1/wp/urls.txt	URL	WP
3	archive_webpage	STORED DATA	<warc.gz file>	mnt/nfs1/wp/out.warc.gz	WARC	WP
4	extract_webpage	PARSED DATA	<JSONL file>	mnt/nfs1/wp/output.jsonl	JSONL	WP
5	index_data	SORTED DATA	<POSTS to Elasticsearch index>	mnt/nfs1/wp/output.jsonl	JSONL	WP
6	summarize_file	RAW DATA	<JSONL file>	mnt/nfs1/wp/output.jsonl	SUM	WP
7	summarize_text	FILTERED DATA	<JSONL file>	mnt/nfs1/wp/summarize_text_output.jsonl	WEBPAGE	WP
8	classify_file	RAW DATA	<Text file>	mnt/nfs1/wp/classify_text_input_targets.txt	CLASS	WP
9	classify_text	CLASSIFIED DATA	<Text file>	mnt/nfs1/wp/classify_text_output_predictions.txt	WEBPAGE	WP

Figure 29: Goal Table

All service related data and corresponding files for goals are listed in these three tables. In other words, all goals contain a corresponding file. The environment variables listed in table 29 are passed into the registered Docker containers automatically through Airflow. The code for each corresponding service is modified to read data from environment variables. Additionally, each Dockerfile defines their respective environment variables and sets their values to the NFS file path for the given input variable.



goal_id	service_id	input_goal_id
1		
2	1	1
3	2	2
4	3	3
5	4	4
6		
7	5	6
8		
9	6	8

Figure 30: Reasoner Table

References

[1] Docker Documentation | Docker Documentation. [Online]. Available from: https://docs.docker.com/, 2020.

[2] Mohamed Magdy and Gharib Farag. *Intelligent Event Focused Crawling*. PhD thesis, Blacksburg, VA.

[3] Schütze Hinrich Manning D. Christopher, Raghavan Prabhakar. Introduction to Information Retrieval.

[4] Roy Thomas Fielding. Fielding Dissertation: Chapter 5: Representational State Transfer (REST). In *Architectural Styles and the Design of Network-based Software Architectures*. UNIVERSITY OF CALIFORNIA, IRVINE, University of California, Irvine, 2000.

[5] Leonard Richardson and Mike Amundsen. *RESTful Web APIs*. O'Reilly Media.

[6] URIs, URLs, and URNs: Clarifications and Recommendations 1.0. [Online]. Available from: https://www.w3.org/TR/uri-clarification/, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[8] Twitter. Twitter API Documentation | Docs | Twitter Developer. [Online]. Available from: https://developer.twitter.com/en/docs/twitter-api, 2020.

[9] GitHub - elastic/elasticsearch: Open Source, Distributed, RESTful Search Engine. [Online]. Available from: https://github.com/elastic/elasticsearch, 2020.

[10] Douglas Crockford. JSON. [Online]. Available from: https://www.json.org/json-en.html, 2020.

[11] WARCIO: WARC (and ARC) Streaming Library. [Online]. Available from: https://github.com/webrecorder/warcio#:~:text=Install%20with%3A%20pip%20install%20warcio,WARC%20records%20rather%20than%20files., 2020.

[12] Beautiful Soup Documentation. [Online], Available from: https://www.crummy.com/software/BeautifulSoup/bs4/doc/, 2020.

[13] Github - WARC Specifications. [Online]. Available from: https://iipc.github.io/warc-specifications, 2020.

[14] Tweepy Documentation. [Online], Available from: http://docs.tweepy.org/en/latest/, 2020.