Virginia Tech
Department of Computer Science
Blacksburg, VA 24061

CS 4624
Multimedia, Hypertext, and Information Access Capstone
Spring 2020

Text Data Mining Studio ETDs

Hannah Brinkley
Robert Rielage
Jason Vaanjilnorov
Christian Dy

Client: Dr. William Ingram
Professor: Edward A. Fox
May 13, 2020

Table of Contents

Table of Figures

1. Executive Summary

The goal of the Text Data Mining Studio ETD project was to develop a software system that allows less technically-minded researchers to be able to easily access a vast amount of data to be used for text data analytics. Specifically, the problem that our team addressed was the lack of a centralized tool for analysis of a large amount of text files that would be valuable data to be used for machine learning and other forms of analytics for long form text documents.

Our team created a centralized tool using ElasticSearch, JupyterHub, and Jupyter Notebooks, in the Docker Compose architecture, to provide this service to researchers. We envision this tool being used by researchers whose work involves the ingestion and analysis of large bodies of text. This work could involve producing deep learning based systems for textual analysis and search, for automatic abstract production, or perhaps for trend tracking across highly temporally spaced documents. The tool is intended to be a flexible foundation for these and other research tasks.

During the process of producing this tool, our team learned a great deal about Docker systems, compartmentalization, and JupyterHub, as well as working in a widely distributed team using virtual meeting analogs.

Unfortunately the tool is as of yet, incomplete, as the ElasticSearch systems are not linked to the JupyterHub frontend. We are hopeful that given the information outlined in the report that completing the tool will be possible with future work.

The approach our team took was to build the overall software system piece-by-piece. We started with installing Docker. We used this software as the cornerstone to install JupyterHub and the subsequent types of software that we configured to work together and deploy as one unit.

This report includes an explanation of the requirements given to us by our client, the design of our project, how we implemented the project, a User's Manual explaining how to run the software system, a Developer's Manual detailing the different parts of our software as well as how to continue work on the system in the future, and the lessons we learned in the process.

2. Introduction

As machine learning, data mining, and word processing have continued to increase in popularity and usefulness, having a library of electronic documents to be used for text and data analytics is incredibly valuable. At Virginia Tech, researchers attempting to use electronic text documents for data analytics research have run into a major issue relating to finding suitable documents for this purpose. Many such documents cannot be used due to copyright restrictions. To begin to remedy this problem, in Professor Fox's Fall 2019 CS 5604 class, a team of graduate students, the Collection Management of Electronic Theses and Dissertations (CME) team [1], turned to the corpus of Electronic Theses and Dissertations (ETDs) maintained by Virginia Tech University Libraries as a possible electronic text document to be used by researchers. The team chose this corpus because it could be used for research of particular interest. However, because these ETDs were in PDF format (which complicates data mining), the team had to write software that extracts well-formatted text files from these PDFs. These text files were then ingested into ElasticSearch, to be accessed for use in analytics research.

This is where our team comes in. We were asked by William Ingram, Assistant Dean and IT Director of the University Libraries, to create a software system that allows researchers - who may not necessarily have a technical background or much technical knowledge - to easily access this library of ETDs and use them in a data analytics IDE for their research. In order to deliver the most effective software system that meets the requirements specified by our client, our team combined several components. First, we built our software system to include ElasticSearch and the ETDs that have been processed by the CME team. Next, for the data analytics IDE that connects to ElasticSearch, we used Jupyter Notebooks. These Jupyter Notebooks are able to be deployed by JupyterHub. To containerize these components of the project, Docker is used to make it easy for researchers to deploy the software.

This report discusses the requirements outlined by our client, the project design, implementation, and evaluation, a user and developer manual, as well as the lessons we have learned through working on this project.

3. Requirements

The overall goal of the Text Data Mining Studio ETD project is to build an online tool to be used for data analytics of a corpus of electronic theses and dissertations (ETDs) that are stored with ElasticSearch. JupyterHub is to be installed and configured to allow multiple researchers to run Jupyter Notebooks all off the same system. ElasticSearch Python libraries allow users to connect to the ETD corpus as a backend for the Jupyter Notebooks. Docker will be used to containerize these parts of the project. Kibana will be used for visualization of the ElasticSearch data. All of the installation information should be documented and available for the future installation and usage of the software system.

Deliverables:

1. Full-text from the corpus of ETDs, indexed in ElasticSearch.
2. Working JupyterHub environment that allows users to query and work with the data stored with ElasticSearch, within the environment of Jupyter Notebooks.
3. Documentation, installation scripts, and a Docker container for the components of the project.

4. Design

The software system constructed for this project consists of five main components: an ElasticSearch data library backend, a Jupyter Notebook frontend IDE to be used with the ElasticSearch data, JupyterHub acting as the launch point for the software system, Docker that containerizes all of the parts of the system for easy deployment across systems, and Kibana that provides visualization of the data with ElasticSearch.

5. Plans

So far, our team has succeeded in using the Information Retrieval (IR) server, which we were granted access to by our client, to download, install, and begin the setup of Docker and JupyterHub. As specified by our client, we are using Docker Compose, a feature of Docker, to simplify containerization and service connection of our overall software service. For the setup of JupyterHub we have started working on installing ElasticSearch as the backend of the Jupyter Notebooks that will be deployed by JupyterHub.

In order to work towards completing our project, we still need to focus on document ingestion into ElasticSearch, implement the Kibana frontend to streamline ElasticSearch interaction, and design and implement deployment testing across platforms.

6. Implementation

The implementation of our project was carried out in several steps and had several components that came together to create our overall software system. This section will go into detail about the process of implementing our project as we went through these steps.

6.1. Initial Research and Exploration

When our team started this project the first step of implementing the solution to the problem given to us was to understand the specifications outlined by our client and to understand how we would go about meeting our specifications. To begin this process, we researched each aspect of the project: Docker [2], JupyterHub [3], ElasticSearch [4], and Kibana [5].

By looking through the development documents and explanation of the software in terms of installation, use, and examples we were able to start to understand how each of these parts would work together to achieve our goal.

At the time that we had started exploring these technologies we had not gone through the process of evaluating different host servers to implement our software. Because of this, we initially began researching the above technologies by downloading them onto our local machines to understand their implementation and use.

6.2. Host Server

After our team had gathered a general understanding of the technologies to be used in our final software system, we needed to decide on a host for the software. We considered several options such as AWS, other Cloud hosts, and internal Virginia Tech servers.

After speaking with our client, we decided to begin working on our project in the Information Retrieval (IR) server. One of the reasons for choosing this host server was because the text documents produced by the CME team [1] were already being stored there, giving us easy access when it came to ingesting them into ElasticSearch. It was also a good choice of host servers because we could be

given access to it from our client, meaning that we were able to consult him if we had any questions about the server**.**

6.3. Server Maintenance

Once we had decided to proceed with using the IR server to host our project, we needed to make sure the software already installed on the server was updated and would meet the installation requirements of the software that we would be using. The main aspect of server maintenance was updating Python to the latest version (Python3) as the server previously had Python2 installed. In addition, we updated the server to the latest stable version of the CentOS distribution it was running.

6.4. Docker

To set up Docker for later use in containerization of the project, we downloaded Docker following the instructions outlined in the Docker Documentation to install it for CentOS [6], the operating system running on the IR server. Following these instructions made it relatively straightforward to install Docker. We also installed Docker images of the latest versions of ElasticSearch and JupyterHub to facilitate easy implementation of these backend processes.

Once we had it installed, we needed to start working on a Docker Compose file which will act as the launchpad for Docker to containerize and launch our project easily.

The Docker Compose configuration file, a .yml file that stores the configuration settings for the processes it controls, was stored in a public directory. It contains all the settings necessary to launch parallel instances of JupyterHub and ElasticSearch. These processes communicate using the specified ports, and will both be set to store their data in a specified directory should the system be shut down.

Any changes that the user wishes to make to the settings of the backend will most likely take place in this .yml file. Here data storage, network communication, and open ports can be manipulated.

6.5. JupyterHub

Once Docker was installed on the IR server, JupyterHub was able to be installed using the Docker JupyterHub repository [7]. This repository was also used as a tutorial for running JupyterHub from the IR server.

To configure the deployment of Jupyter Notebooks from JupyterHub, the jupyterhub_config.py file was edited using the Configuration Basics documentation for JupyterHub [8].

6.6. ElasticSearch

To be used as the backend for JupyterHub. ElasticSearch was configured as a Docker Compose image. When Docker is run, it runs as a secondary image in combination with the frontend image. However, at this moment the linkage between this image and the frontend is incomplete, and the Python structures in JupyterHub cannot access the ElasticSearch instance. In addition, document ingestion hasn't been completed.

6.7. Jupyter Notebooks

This project was able to provide a text analytics capability to work with the ETDs because the Jupyter Notebook environment allows for researchers to import python packages that provide the functionality for text analytics. While the tool is intended to offer a barebones example of document ingestion and meta-data loading, our goal was not to provide prebuilt analytic code to the researchers, but rather provide a standardized tool for researchers to use as a foundation node for such work. Given the number of potential applications, providing analytical code was not within the scope of our project.

As such, future work by users will almost certainly involve deployment of the analytical tools they desire into their Jupyter Notebooks instance.

7. Testing/ Evaluation/ Assessment

7.1. Deployment of Backend and Frontend

```
[root@ir ETD_Data_Analysis]# docker-compose up -d
Creating network "etd_data_analysis_default" with the default driver
Creating blmetadata-es ... done
Creating frontend      ... done
```

Figure 1: Successful Launch of ElasticSearch and Jupyter Notebook

As we can see from the terminal output (*Figure 1*), the ElasticSearch ('*blmetadata-es*') and Jupyter Notebook ('*frontend*') services were successfully built into containers from the docker-compose.yml file.

7.2. Connecting to the Frontend and Authentication

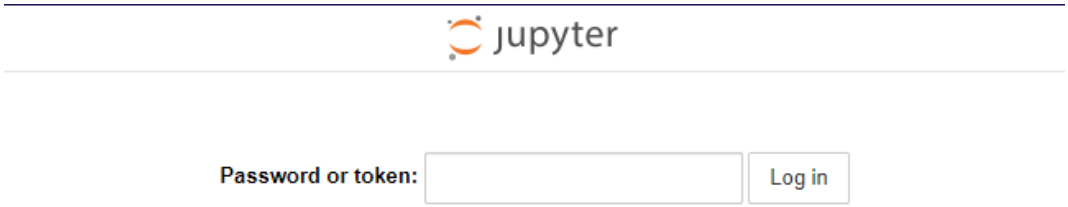

Figure 2: Frontend Authentication Page

The Jupyter Notebook authentication page (*Figure 2*) is reachable from a browser after entering the host's IPv4 address (for this purpose we enter the IR server's IPv4 address) and the port number associated with the frontend (http://128.173.237.40:8888/) into the address bar.

Figure 3: Jupyter Notebook File Directory

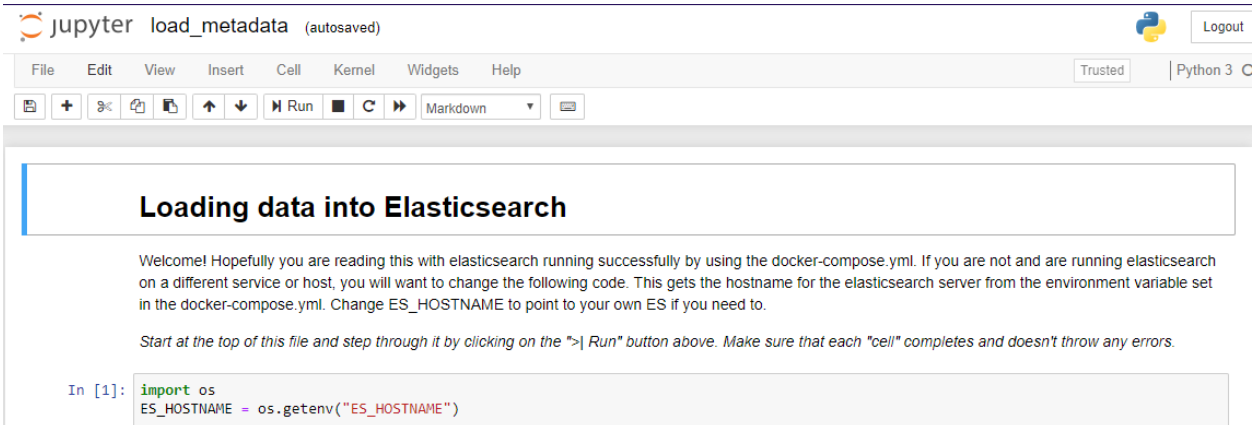

Figure 4: Jupyter Notebook IDE

```
[I 12:49:47.743 NotebookApp] The Jupyter Notebook is running at:
[I 12:49:47.743 NotebookApp] http://ce9f13a493a9:8888/?token=d4942d9f149c1df1c489f7e6d47d2e0b7ee7da5b41f153d3
```

Figure 5: Authentication Token

After retrieving the login token from the host (run '*docker logs frontend'* in the terminal and identify the lines that resemble *Figure 5*) and entering it into the "Password or token:" field in *Figure 2*, we are able to access the Jupyter Notebook file directory page (*Figure 3*). From there, we can click on a Jupyter Notebook file to open it in a new web browser tab (*Figure 4*). Finally, we can establish a connection to our running ElasticSearch instance from the Jupyter Notebook, load in our desired JSON data, create an index and generator in ElasticSearch, use the bulk API ('*elasticsearch.helpers.bulk(client, actions, stats_only=False, *args, **kwargs)*') to index our data into the ElasticSearch database, and query it ('*elasticsearch.search(index='test-index', filter_path=['hits.hits._id', 'hits.hits._type'])*').

8. User's Manual

This section discusses how to use the software system.

8.1. Dependencies

No dependencies are required to use this tool. All the backend interactions will occur in the IR server where all the necessary components reside. All a user needs to access the tool is a web browser.

8.2. Project Setup

To deploy the tool, load into a directory containing the provided Dockerfile, docker-compose.yml file, and load_metadata.ipynb file (*Figure 6*). Ensure that the Dockerfile and docker-compose.yml file are the only ones of their type in the directory.

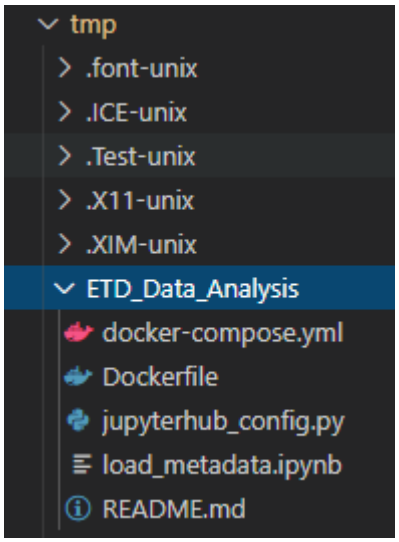

Figure 6: Docker Compose Directory

To start, run docker-compose up (*Figure 1*). This will load two images, one from the ElasticSearch library, and one supported from the British Library Labs library, currently named '*frontend'*. It will also load a JupyterHub instance as a trusted process, and open both the image and the JupyterHub instance to port 8888. The port can easily be changed, and will be outlined in the Developer Manual.

In order to connect to the frontend, enter the hostname of the machine that the Docker services are running on (ir.cs.vt.edu) as well as the port that the frontend is listening on (‘8888’), separated by a colon. In our case, we entered ‘http://ir.cs.vt.edu:8888’ into our web browser. In order to get past the authentication page (*Figure 8*), retrieve a key by running the command ‘*docker logs frontend’* on the host machine’s terminal. This will display the logs of the frontend instance, and near the top, there will be a generated token to access the hub with (*Figure 7*). Enter this token into the “Password or token” field on the authentication page (*Figure 8*). Once a user has access to the hub, they can add a login to avoid having to repeat the task of generating a token.

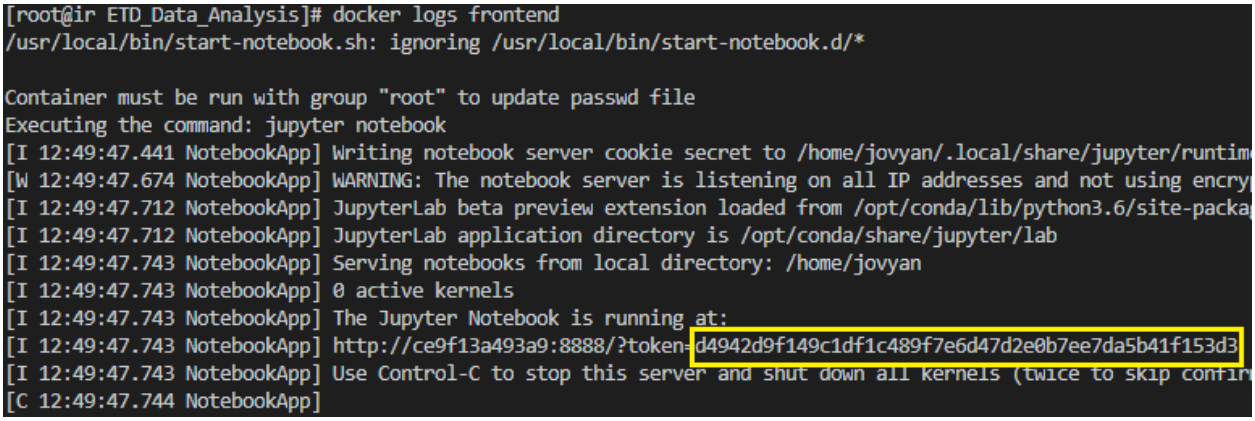

Figure 7: Token from Running ‘docker logs frontend’

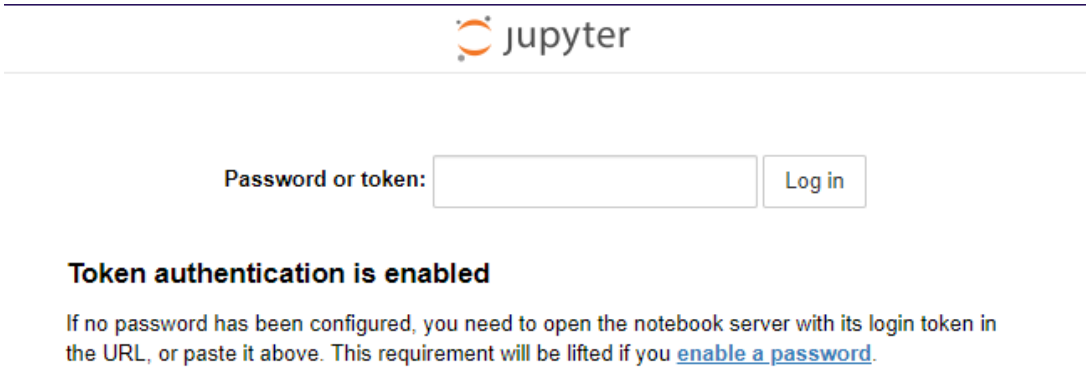

Figure 8: Authentication Page

8.3. Steps to Run and Access Frontend

Here are steps to run and access the frontend, in a more readable format.

1. Login into the IR server
2. Go to the */tmp/ETD_Data_Analysis directory*
3. Run '*sudo docker-compose up -d'*
4. Confirm the software has launched successfully (*Figure 1*)
5. Run '*sudo docker logs frontend'*
6. Copy Token (*Figure 7*)
7. Open a browser and enter 'https://ir.cs.vt.edu:8888' into the address bar
8. Paste token into the "Password or token" field (*Figure 8*) on the authentication page
9. (Recommended) Set up login info

9. Developer's Manual

9.1. File Inventory

'*Dockerfile*'

Launch file for docker-compose. Controls bootup of JupyterHub/port opening

'*docker-compose.yml*'

Configuration file for docker-compose instance. Controls Docker volume to persist work, port opening, Docker images, and several other important functions. Most developer changes can be made here.

'*load_metadata.ipynb*'

Loads basic Python example code. Largely taken from the Humanities Workbench tool. Provides a basic example for loading metadata into ElasticSearch backend for analysis, but currently isn't fully implemented. Breaks on third Python executable statement, since backend ElasticSearch infrastructure is incomplete.

9.2. Dependencies

Download the latest version of Python 3 [9]. Follow the instructions provided to install the software on your machine. Install a currently supported version of Docker, Docker Compose, and images of current versions of ElasticSearch and Jupyterhub.

To ensure you have the correct and up-to-date dependencies, the following commands should be run on a Linux machine.

1. '*sudo yum -y update*'
2. '*sudo yum install -y docker*'
3. '*sudo service docker start*'
4. '*sudo groupadd docker*'
5. '*sudo usermod -a -G docker* your_username'

    You must log out and log back in order to be added to the Docker group. The purpose of this is so that you do not have to have root user permissions to execute Docker commands.

If you are still unable to run Docker commands (i.e. '*docker info*'), log in as a root user or run the following command: '*sudo su -*' to avoid having to specify '*sudo*' for every Docker command.

6. '*sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose*'
7. '*sudo chmod +x /usr/local/bin/docker-compose*'
8. '*docker pull britishlibrarylabs/hwb_elasticsearch*'
9. '*docker pull docker.elastic.co/elasticsearch/elasticsearch:6.1.4*'

9.3. Project Setup

While in the IR server, change directory to the *tmp* folder and create a new folder to place all the docker-compose components into. Place the '*Dockerfile*', '*docker-compose.yml*', and '*load_metadata.ipynb*' files from the *ETD_Data_Analysis* directory into the new directory and launch Docker with '*docker-compose up*' when starting up Docker for the first time. This will create new containers for the images specified in the '*docker-compose.yml*' file. When complete, run the command '*docker-compose stop*' to stop the services and have the Jupyter Notebook work be saved in the specified volumes in the '*docker-compose.yml*' file. To restart services, run '*docker-compose start*'.

In order to adjust open ports, you can specify individual ports or ranges of ports in both the '*docker-compose.yml*' file and the '*Dockerfile*'. When making these changes ensure that your changes match between the files, as differences will prevent the Docker images from listening on the proper ports. Volumes can be adjusted from the '*docker-compose.yml*' file in the same way.

9.4. Project Methodology

In this section we'll outline the type of user we anticipate using this system, and outline their goals, and the methodology behind their anticipated workflow.

9.4.1. User Types/Goals

We realistically only have one type of user. The product we're developing is relatively single purpose, compared to many of our peers' projects, so the user will be a researcher who is using this product to analyze a large body of documents, whether they are Electronic Theses and Dissertations (ETDs) or some other type of lengthy text, using a variety of research tools. Since the researchers will be using this for a variety of purposes, each may have their own body of documents they're researching, and will therefore be performing their own data ingestion and system maintenance, obviating the purpose of a separate system administrator/maintainer role user persona.

As a result of this single purpose design and single user persona, our prototypical user will have a wide range of goals. The goals of every user of our system can be broken down into one of four different types.

1. To have access to the ETDs they want to analyze

2. To manipulate the ETD data (Ingest more data, remove data from analysis set, modify ETDs and attached metadata in the data set)

3. To analyze the ETD documents (Deep learning using data set, visualizing with analysis tools, writing their own analysis scripts)

4. System maintenance and administration (Keeping system on line on their local machine/institutional server, moving complete product to new systems)

9.4.2. User Goal Breakdowns

Goal 1: Have access to the ETDs they want to analyze

1. This can be described by the task of accessing the system front end, and accessing the backend database. This breaks down further into two sub-tasks:
    a. Logging into frontend
    b. Accessing backend ETDs from frontend

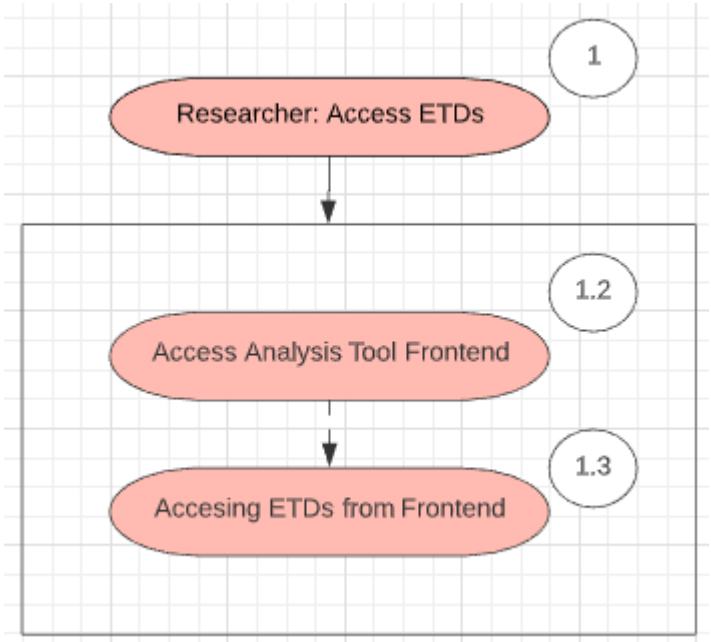

Figure 9: Graph for Goal 1 Task Breakdown

Goal 2: To manipulate the ETD data (Ingest more data, remove data from analysis set, modify ETDs and attached metadata in the data set)
This goal can be broken down into three tasks, each with their own set of subtasks.

1. Ingest documents into database
    a. Access backend
    b. Select documents to ingest
    c. Execute Optical Character Recognition process to extract text and meta-data from document
    d. Add formatted text file, metadata, and original file to backend database

2. Remove data from database
    a. Access backend
    b. Select documents to remove from backend database
    c. Remove documents while leaving file system/other documents intact
3. Modify data
    a. Access backend
    b. Select documents to modify
    c. Modify documents while leaving file system/other documents intact

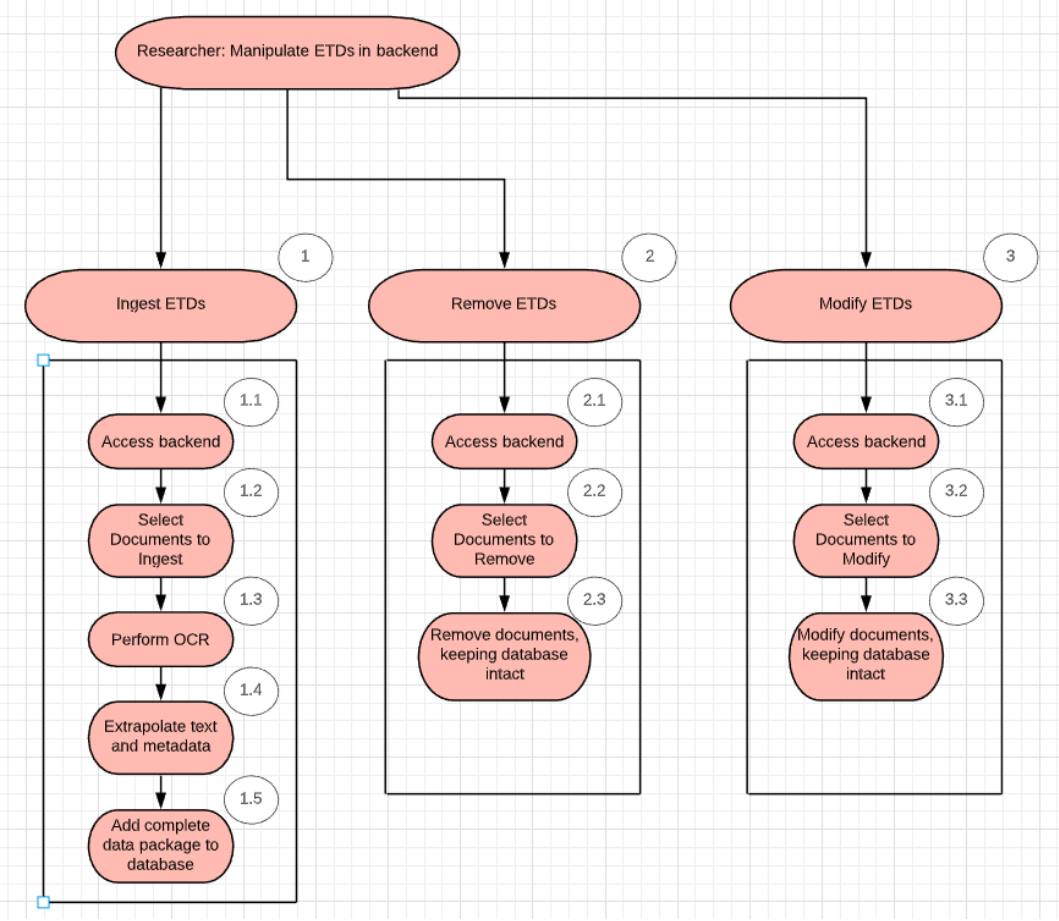

Figure 10: Graph for Goal 2 Task Breakdown

Goal 3: To analyze the ETD documents (Deep learning using data set, visualizing with analysis tools, writing their own analysis scripts)

This goal can be broken down into a single task. While there are doubtless dozens of potential ways this task can be granularized, it is practically not possible for us to analyze the myriad number of tools that a researcher may wish to bring to bear on the problem of ETD or document. As such, the task outlined below covers a generalized task of reaching a point where analysis is possible, and then exiting cleanly, with analysis intact, leaving the actual form of analysis to the researcher.

1. Analyze ETD documents
    a. Access frontend
    b. Bring up analysis tools
    c. (Potential step) Learn how to apply analysis tools
    d. Display results of analysis
    e. Exit frontend while retaining analysis data

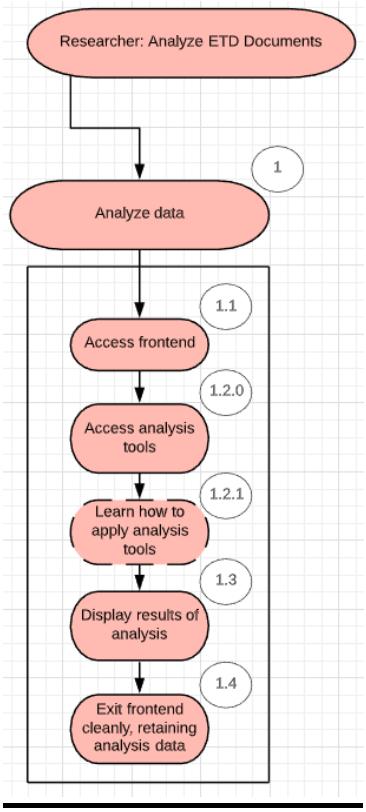

Figure 11: Graph for Goal 3 Task Breakdown

Goal 4: System maintenance and administration

This goal can be broken down into two general tasks. First, any researcher using this system will obviously need to be able to bring it online, and shut it down cleanly, as well as keep the system online and operational when in use. This constitutes the first task, system maintenance.

1. System maintenance:
    a. Bring system online
    b. Link backend and frontend
    c. Load pre-existing and selected data sets, for both ETD and analysis data
    d. Shut down system cleanly, retaining previously stated data sets for next bootup.

The second goal can be described best as system deployment. The tool we're creating is designed as a containerized tool that can be deployed and used by a variety of researchers across a number of institutions, who may be using any of a number of platform options, ranging from Windows and Linux based servers to personal or local machines. This necessitates a task of deploying the packaged tool onto a new platform.

1. System deployment:
    a. Deploying container architecture
    b. Including backend images
    c. Including configuration files
    d. Configuring backend database data sets
    e. Bringing system online
    f. Testing functionality

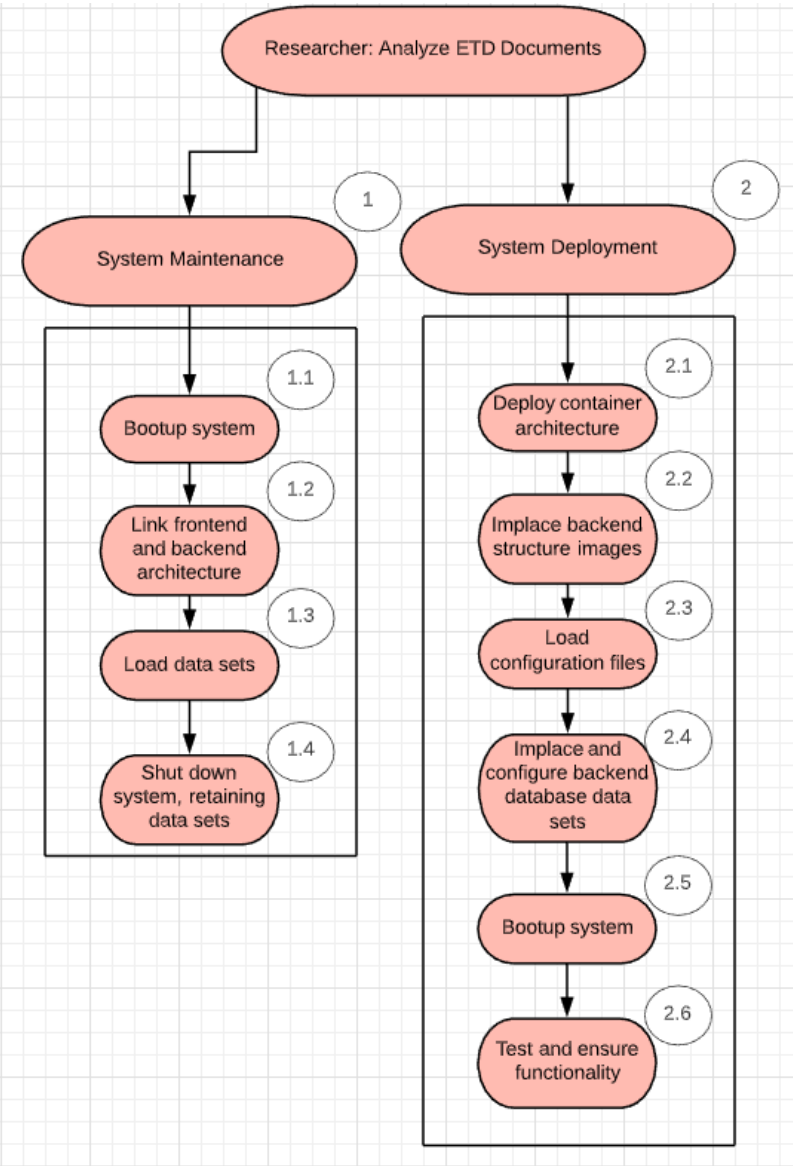

Figure 12: Graph for Goal 4 Task Breakdown

9.4.3. Goal Implementation

G1T1: Accessing ETD documents will be a relatively simple process. While the system is online, it will maintain a hub, connected by the Docker Compose architecture to a network with an ElasticSearch database. Users will use Jupyter Notebook to access this node, and will be provided with a starting base that includes the access to the database and library of Python functionality. In order to access the documents, all the user has to do is connect to the hub, and they'll be granted access to whatever database is attached to the instance of the hub.

G2T1: Ingesting ETDs into the database will utilize a script and OCR system developed by our predecessor team, and outlined in the final report. Users will need access to the system that the database is stored on, and access to the original ETD documents, and will then run that script, directing output to the file system that the ElasticSearch backend uses.

G2T2: Removing ETDs will be a straightforward process of removing reports from the sorted file system that the ElasticSearch backend connects to. This will, however, require access to the platform the system is hosted on. As a secondary option, researchers can remove ETDs from their analysis with the included tools in the library that will come prepackaged with their Jupyter Notebook access. This second method, however, will not remove the actual files, merely neglect to include them in analysis.

G2T3: Modifying ETDs will be a challenge, and not something that most users are anticipated to do. We expect most of the analysis to happen on the frontend, and most of the modification to come in the form of changing queries, or modifying metadata upon ingestion. If, however, the user needs to change a document or piece of data, they will need access to the system hosting platform, and will access the file system, directly modifying the files contained therein using a system-specific tool.

G3T1: Analysis of ETDs will use the following implementation. The system is designed for long term deployment on a system that stays up for extended periods of time, rather than being booted up every time a user needs to access analysis. Therefore the implementation of ETD analysis will start with the user accessing the frontend through a personal Jupyter Notebook instance, much as in G1T1, where they will be presented with a Jupyter page already configured with access to the database and functionality. They will then perform analysis as they see fit using this library, and will be able to display their results either in text form, or visually, using the Kibana visualization suite our implementation will include with the ElasticSearch library. Exiting Jupyter Notebooks will save their analysis, and if the system is shut down correctly, JupyterHub will save their data in a file location specified in the accompanying .YAML configuration file.

G4T1: System maintenance will require access to the hosting platform. Startup will be accomplished by running a .YAML file configured Docker Compose image, which will spool up images of both JupyterHub and ElasticSearch, and their respective data pools, and link them together. Shutdown will also be implemented through Docker Compose, to ensure a clean and orderly disengagement of the processes, and storage of associated files.

G4T2: System deployment will take place almost entirely within the Docker Compose architecture. Users will configure a Docker Compose process on their system, using a platform-specific variant as necessary, and then retrieve images of both ElasticSearch and JupyterHub. An attached .YAML file will provide all the necessary configuration, and they can execute a Docker command to boot up the processes in sync. Users can also modify the data pool in the configuration file as they wish, and will need to set up the ElasticSearch files. Once this is done, startup and shutdown of the backend can be accomplished as in G4T1.

9.4.4. Goal Workflow Description

Goal 1: Workflow 1 = JupyterNotebook + ElasticSearch Python Library
Goal 2: Workflow 1 = Platform File System + OCR/Metadata Extraction Python Script + ElasticSearch
Goal 2: Workflow 2 = Platform File System + ElasticSearch
Goal 2: Workflow 3 = Platform File System + Platform Specific Editing Tool
Goal 3: Workflow 1 = JupyterNotebook + Backend + ElasticSearch Python Library + Kibana Visualization Toolset
Goal 4: Workflow 1 = Hosting Platform Access + Docker Compose + .YAML Config File
Goal 4: Workflow 2 = Hosting Platform Access + Docker Compose + .YAML Config File + ElasticSearch DC Image + JupyterHub DC Image + ETD File Access + JupyterNotebook Instance

10. Document Ingestion

Before we begin the process of ingesting documents into ElasticSearch, we need to ensure they are in the correct format by preprocessing these documents. ElasticSearch ingests data only if it is in a particular format. Fortunately, the CME team from last semester had converted the ETD metadata and text pairs into single JSON files. The next step is to load the JSON files stored in the IR server (‘*/mnt/ceph/cme/30Ketds/updated_dateformat_metadata.json*’). ElasticSearch expects records to have a few fields like "*_type*" and "*_index*" to collate the data we upload. We create an index in ElasticSearch and provide a generator that takes the ETD data and generates suitable records for it to index. Then we make use of the ElasticSearch bulk API to begin indexing the data into the ElasticSearch database. Then, we are ready to perform queries on our data.

11. Lessons Learned

This section discusses the timeline of our project, problems we faced while working on our project, solutions we found to overcome these problems, and future work that could be done to improve our final product.

11.1. Timeline

At the start of our work on this project we had discussed and were working to meet the goals set in the following proposed timeline:

- February
    - Communicate with AWS Team
    - Wednesday, 02/12: Establish ElasticSearch database on local machine
        - Verify database functionality
        - Test data analysis and visualization tools
        - Test multiple user access
    - Wednesday, 02/19: Establish JupyterHub server on a local machine
        - Setup JupyterHub host
        - Establish and test user access
- March
    - Wednesday, 03/04: Move ElasticSearch database functionality to a persistent server environment
        - Select server environment that will best suit client needs
        - Establish server functionality
        - Test previously required functionality
    - Wednesday, 03/25: Move JupyterHub host to persistent server environment
        - Test JupyterHub functionality with remote login of users to server
- April
    - Wednesday, 04/08: Connect JupyterHub host to ElasticSearch database
        - Test analysis tools
        - Test multiple user connections

- Wednesday, 04/22: Encapsulation of project in Docker
  - Verify functionality through Docker
  - Test functionality on multiple server solutions to provide flexibility for client usage cases

However, as we continued to work on the project, the timeline of events we had originally intended to follow was vastly altered by the knowledge we gained as well as external factors we did not have control over.

A major change made to the original timeline above was the decision about the host server for our project being made well before we expected to transfer our work to the server. Early in the project we discussed what host server option would be the best suited for our project and were given access to the Informational Retrieval (IR) server soon after that meeting. Because we already had this access to the host server, we decided to mostly bypass installing and configuring various software on our local machines and to start installing this software on the IR server. For the most part, we worked on our project through the IR server with the exception of the initial research into the software we would be using at the very beginning of the semester

Another change that was made to the original timeline was that we saw the benefits of using the Docker software to install JupyterHub and then ElasticSearch instead of installing and configuring Docker at the end of the project just to use for containerization. The use of Docker as a base for installing JupyterHub was able to provide a foundation for configuring these different types of software to work together.

11.2. Problems

Overall, the process we initially outlined to meet the requirements specified by our client worked rather well. Other than having to move around our timeline a bit the actual process of working on the project did not pose many significant problems. One technical problem we did run into was that the server we were implementing our project on was being used for research simultaneously by a Virginia Tech graduate student. As they were working on their research some of the data we were using in our system got moved around, which interfered with the software that we had configured already.

A major obstacle our team faced was the decision made by Virginia Tech to move our classes online due to the health concerns that occurred during the course of the project. This was a problem initially because the transition to virtual classes was difficult in terms of getting everything to a point where our team felt like we could properly communicate with each other.

One major issue is that due to effects beyond our control, we've been unable to effectively complete much of the ElasticSearch functionality. This means that the frontend is incapable of correctly reaching the ElasticSearch instance, and we've been unable to complete the document ingestion into the ElasticSearch backend.

11.3. Solutions

In order to overcome the problem our team encountered with the graduate student we first went through the data on the server and were able to transfer it back to where it was before, so that our software could work correctly as it had before. Then, we were put in contact with the graduate student through our client to ensure we could contact them if we had other similar issues later in the semester.

To overcome the obstacle of all team communication and communication with our professor being shifted online we came to heavily rely on apps allowing us to video conference with each other and talk to each other in real time as we couldn't physically meet in person. Another large part of this obstacle was solved quickly after it arose as a problem by our client. This obstacle was that of our bi-weekly client meetings having to be over video conferences. Our client quickly provided a link to the meeting in order for our schedule to be maintained despite the changes to our lifestyle that we were experiencing.

11.4. Project Extensibility

Due to unforeseen circumstances our group faced this semester while working on our project, we were unable to optimize the efficiency of the final product and truly make it the best it could be. In the future, those who choose to add to our project could explore ways to streamline the components of our final product.

Ideally, during the time we were working on this project we would have been able to more thoroughly represent the capabilities of our software system through demonstrations of using the Jupyter Notebook environment to perform text analytics. We had originally aimed to be able to use the ETDs on the IR server that were in both PDF and JSON formats to perform analytics as an example of our project. However, due to limitations in our final product as well as unforeseen circumstances we were unable to accomplish this.

In addition to improving efficiency of the project, an interactive interface that allows researchers to launch Jupyter Notebooks from a website hosted by those in charge of the JupyterHub aspect of the software could be useful. In addition, linking this with Kibana would work further towards a product that is even easier and more convenient to use by researchers, allowing use of visualization tools to aid researchers in analysis of documents.

As outlined in the problems section, the project as it stands doesn't meet several of the base requirements, specifically for ElasticSearch access and document ingestion. In order to solve this, the project needs proper linking between the ElasticSearch backend and the JupyterHub frontend.

12. Acknowledgements

Client: William Ingram, Asst. Dean and IT Director, University Libraries
waingram@vt.edu

Advisor: Dr. Edward Fox, Professor, Virginia Tech
fox@vt.edu

Funding: IMLS LG-37-19-0078-19

13. References

[1] Collection Management of Electronic Theses and Dissertations (CME) CS 5604 Fall 2019, Date Accessed: May 5, 2020
http://hdl.handle.net/10919/96484

[2] Docker, Date Accessed: May 5, 2020
https://www.docker.com

[3] JupyterHub, Date Accessed: May 5, 2020
https://jupyterhub.readthedocs.io/en/stable/

[4] Elasticsearch, Date Accessed: May 5, 2020
https://www.elastic.co/elasticsearch/

[5] Kibana, Date Accessed: May 5, 2020
https://www.elastic.co/kibana

[6] Get Docker Engine - Community for CentOS, Date Accessed: May 5, 2020
https://docs.docker.com/install/linux/docker-ce/centos/

[7] JupyterHub Docker Repository, Date Accessed: May 5, 2020
https://hub.docker.com/r/jupyterhub/jupyterhub/

[8] JupyterHub Configuration Documentation, Date Accessed: May 5, 2020

https://jupyterhub.readthedocs.io/en/stable/getting-started/config-basics.html#configure-for-various-deployment-environments

[9] Download Python | Python.org, Date Accessed: May 5, 2020
https://www.python.org/downloads/

[10] British Library Labs, Date Accessed: May 5, 2020
https://www.bl.uk/projects/british-library-labs

[11] British Library Labs - Humanities Workbench, Date Accessed: May 5, 2020
https://github.com/BL-Labs/humanitiesworkbench