Parquet Containers

May 11, 2022

Final Report

Andrew Weber
Matthew Coscia

Supervised by:

Dr. Edward A. Fox
Xinyue Wang, Client
Ryan Wood, GTA
Kenneth Neba, GTA
# Contents

**List of Figures** 4  
**List of Tables** 5  

## 1 Abstract 6  

## 2 Introduction 7  

### 2.1 Background Information 7  
#### 2.1.1 Docker Containerization 7  
#### 2.1.2 Podman 8  

### 2.2 Application Information 8  
#### 2.2.1 Apache Spark 8  
#### 2.2.2 Web ARChive File Format 8  
#### 2.2.3 JavaScript Object Notation File Format 9  
#### 2.2.4 Parquet File Format 9  

## 3 Requirements 10  

### 3.1 Project Deliverables 10  
### 3.2 Documentation 11  
### 3.3 OKRs 11  
### 3.4 VTCS Cloud Access 11  
### 3.5 Other Relevant Workflows 11  

## 4 Design 13  

### 4.1 WARC to Parquet 13  
### 4.2 JSON to Parquet 13  
### 4.3 Container Design 14  

## 5 Implementation 15  

### 5.1 WARC to Parquet 15  
### 5.2 JSON to Parquet 15  
#### 5.2.1 Docker Containers 15  

## 6 Testing 17  

### 6.1 Uploading to VT Computer Science Cloud 17  

## 7 User Manual 19  

### 7.1 WARC to Parquet 19  
### 7.2 JSON to Parquet 19  
### 7.3 Accessing the Containers 19  

2
List of Figures

1. Containers vs. Virtual Machines (adapted from [8]) .......................... 7
3. Docker vs. Podman [8] ................................................................ 8
5. Deliverables: Relevant Workflows .................................................. 10
6. The two containers ready to run in the VTCS Cloud ......................... 11
7. Remaining Relevant Workflows ..................................................... 12
8. Original Project Timeline ............................................................ 13
9. WARC to Parquet Dockerfile ....................................................... 16
10. JSON to Parquet Dockerfile .......................................................... 16
11. Kubernetes Deployment .............................................................. 17
12. Containers running in separate pods in the VT Cloud ..................... 18
13. build.sbt ................................................................................. 20
14. Running WARC to Parquet Container ........................................... 22
15. Packages and global variables used in json2parquet.py ................. 22
16. Running JSON to Parquet Container ............................................. 22
17. Agile Sprint Example .................................................................. 25
List of Tables

1 File size comparisons ................................................................. 8
2 File conversion times ................................................................. 9
3 Twitter data additional fields ...................................................... 13
4 Spark submit command .............................................................. 19
5 PySpark command ................................................................. 19
1 Abstract

Archives preserve content and support various types of analysis, study, and use. Archiving the Web and social media (e.g., Twitter) content involves use of representations like Web ARChive (WARC) and JavaScript Object Notation (JSON). Parquet is a newer representation that yields better performance with large collections for many data processing tasks. In support of archive-related research at Virginia Tech in University Libraries and the Digital Library Research Laboratory (DLRL), this project involved the development of software (tested in Linux and Windows, as well as on local and container cluster environments), and its packaging in containers, for converting WARC to Parquet as well as JSON to Parquet. Images of the resulting containers are available to support the research of others.
2 Introduction

Containerization is a rapidly expanding technology. A container is described by the Docker website as, “an abstraction at the app layer that packages code and dependencies together.” [8] It allows users to run applications without the host machine having to individually download and configure the necessary packages and dependencies, which as we experienced can be tedious. A container then shares a small portion of the OS kernel, running as an isolated process in the user space. [8] A container is often compared to a virtual machine, but there are very significant differences. Containers are more lightweight and take up less of the host’s user space. Figure 1 illustrates a few of the differences between virtual machines and containers. In this section we will go over the basics of the project and some of the options we considered for meeting our ultimate goals.

![Figure 1: Containers vs. Virtual Machines (adapted from [8])]({})

2.1 Background Information

2.1.1 Docker Containerization

Docker is a company that manages containers for users everywhere. It does this using a Docker daemon which is a program that runs in the background to manage and create Docker images, and run the containers. [9] A Docker image is an immutable, standalone file with executable code that can create a container on a computing system. It includes system libraries, utilities, and workloads to run in the container. Images can also be layered, allowing components and configurations to be reused or adopted. They are built using a Dockerfile which details the dependencies and configurations that will define the container. [4] Figure 2 is a good high level representation of what happens when you create and run a Docker image as a container.

![Figure 2: Docker Basics [8]]({})
2.1.2 Podman

Podman is a daemon-less alternative to Docker. It gives non-root privileges to containers, which has security advantages. However, Docker now also has a rootless mode. [9] Ultimately, our decision to choose Docker over Podman came down to our own familiarity with the technology as well as the counsel of others experienced in containerization. That being said, there is an interchangeable aspect to the two services considering that the same approach taken to deploy containers locally on our Windows computers equipped with Docker also worked in rlogin which uses Podman to emulate Docker functionality.

![Podman vs Docker](image)

**Figure 3**: Docker vs. Podman [8]

2.2 Application Information

In this section we will overview Apache Spark as well as the inputs and outputs for our containerized applications.

2.2.1 Apache Spark

Apache Spark is a multi-language framework for distributed computing and big data processing. It includes standard SQL support for processing structured data using a dataframe approach and provides an interface for reading and writing several file formats. Additionally, it can be run with Hadoop YARN, Kubernetes, and Docker Swarm. These qualities make it the ideal big data processing tool for Parquet Containers. [15]

<table>
<thead>
<tr>
<th>File</th>
<th>Original Size (Kb)</th>
<th>Parquet File Size (Kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHIVEIT.archive.org-8090.warc.gz</td>
<td>106921</td>
<td>124967</td>
</tr>
<tr>
<td>table_z_1.json</td>
<td>436446</td>
<td>14989</td>
</tr>
<tr>
<td>table_z_10.json</td>
<td>37203</td>
<td>1164</td>
</tr>
<tr>
<td>table_z_11.json</td>
<td>435</td>
<td>31</td>
</tr>
<tr>
<td>table_z_12.json</td>
<td>26642</td>
<td>919</td>
</tr>
<tr>
<td>table_z_13.json</td>
<td>1513</td>
<td>69</td>
</tr>
<tr>
<td>table_z_14.json</td>
<td>181387</td>
<td>8390</td>
</tr>
<tr>
<td>table_z_15.json</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

**Table 1**: File size comparisons

2.2.2 Web ARChive File Format

WARC file format is traditionally used to store “web crawls” as a sequence of content blocks from around the internet. In our case, the data will be sourced from the “vt.edu” namespace. These files are a concatenation of multiple records, each including a header and some content. [18] The support for processing WARC files is made possible by Apache Spark with the help of executable Java ARchive (JAR) files and package imports written for Scala. [16]
<table>
<thead>
<tr>
<th>File</th>
<th>Conversion Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHIVEIT.archive.org-8090.warc.gz</td>
<td>35.3616448</td>
</tr>
<tr>
<td>table_z_1.json</td>
<td>14.614174</td>
</tr>
<tr>
<td>table_z_10.json</td>
<td>1.452997</td>
</tr>
<tr>
<td>table_z_11.json</td>
<td>0.647998</td>
</tr>
<tr>
<td>table_z_12.json</td>
<td>0.995338</td>
</tr>
<tr>
<td>table_z_13.json</td>
<td>0.622997</td>
</tr>
<tr>
<td>tabl_z_14.json</td>
<td>3.243972</td>
</tr>
<tr>
<td>tabl_z_15.json</td>
<td>0.569603</td>
</tr>
</tbody>
</table>

Table 2: File conversion times

2.2.3 JavaScript Object Notation File Format

The JSON files used in this project contain large multi-line objects built using the Twitter API. They fit a unique schema and the support for processing them is made possible with PySpark, the Python API for Apache Spark. [7]

2.2.4 Parquet File Format

The Apache Parquet website describes Parquet format as, “an open source, column-oriented data file format designed for efficient data storage and retrieval.” [3] Core characteristics include the column-based format and advanced nested data structures. The approach leverages the fact that data in a single column will be of the same datatype, thus data from the same column is organized together on a unique page in virtual memory. This optimizes the time needed to query a particular column from a data table because the solution will be contained in a known memory location. This contrasts with row-based files like CSV which would need to load the entire table from memory to satisfy the same query. Furthermore, Parquet supports efficient compression and encoding schemes. Compression of a single column is straightforward given that the data types will be quite similar. Not only does Parquet format optimize query time, it has been shown to reduce storage requirements on large datasets. [6]
3 Requirements

3.1 Project Deliverables

There are two deliverables for this project:

- A containerized application that converts WARC files to Parquet files
- A containerized application that converts JSON files to Parquet files

We were given sample scripts with details regarding additional columns the client wanted included in the Parquet files for each individual application. We then translated the scripts into programs that could be executed from the command line rather than in a Spark shell. Additional requirements such as documentation, objectives and key results (OKRs), and Virginia Tech Computer Science (VTCS) cloud access are described in the following sections. To make certain we achieved these goals, we designed project workflows as found in Figure 5.

![Figure 5: Deliverables: Relevant Workflows]
3.2 Documentation

Ample documentation is one of the key requirements for this project. We want future developers to be easily able to pick up where we left off and have success executing our applications. To this end our code has to be thoroughly documented and we have included informative manuals for users and developers. Along the way, we kept a Discord server and Google Doc to stay up to date with the latest updates and share material. Notes were added after each meeting with the client to ensure we stayed on track with meeting objectives.

3.3 OKRs

In order to stay on track for this semester long project, we made use of a website called Ally.io that provided a framework for us to create Objectives and Key Results (OKRs). This is a common practice in many technology firms today. It allows group members to clearly lay out the overall goals of a project and divide them into smaller key results that must be accomplished to complete the bigger picture of the project. If used correctly, this technique is known to increase productivity. It was also a requirement for the project. [2]

3.4 VTCS Cloud Access

Another major requirement of the capstone project was that we upload our containers to the Virginia Tech Computer Science Cloud. This is so it is of use for clients interested in working with files in the DLRL. To do this we had to push both of our Docker images to a image registry in container.cs.vt.edu. From there we could pull the images to a namespace in cloud.cs.vt.edu to deploy our two containers in their own pods. Figure 6 shows the two pods running in our cloud.cs.vt.edu namespace. We will go more into depth about this in a later section.

![Figure 6: The two containers ready to run in the VTCS Cloud](image)

3.5 Other Relevant Workflows

Figure 7 shows the workflows of the other project goals that were achieved.
Figure 7: Remaining Relevant Workflows
4 Design

Figure 8 outlines our projected timeline in terms of OKRs distributed throughout the semester. The rest of this section will focus on the design decisions pertaining to each step.

![Figure 8: Original Project Timeline](image)

4.1 WARC to Parquet

We received the project with a functional script for converting WARC to Parquet using Apache Spark written in Scala. The goal was to create a compiled version of code to be run outside of the Spark Shell. The core challenge in doing so was to infer critical dependencies and replicate the environment inside the shell. This proved to be a sizable challenge, especially given our lack of familiarity with Apache Spark and Scala. In addition to performing the conversion, the program had to add an additional column to the data with a fixed timestamp as defined by the client. The fix timestamp method extracts the timestamp field from the content header and represents it in a more human readable format. It fell on us to write code to meet this objective in the general case. Lastly, a directory based approach was taken to convert files for an entire directory in a single execution of the program rather than running the code for one file at a time.

4.2 JSON to Parquet

Although it shares the same goal of converting archive files into Parquet format, there are fundamental differences when compared to the WARC to Parquet program. First, JSON to Parquet is written in Python, leveraging PySpark. This allows it to achieve the same basic goal with fewer dependencies. Just like in the other program, the output required additional columns as described by the client. In this case, the goal was to flatten the hashtags in a new field, cast the unique tweet identifier to long type, and include a UNIX timestamp. The same directory based approach as described in Section 4.2 was taken in this program as well.

<table>
<thead>
<tr>
<th>New Column</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>entities_hashtags</td>
<td>Extract hashtags into a single string</td>
</tr>
<tr>
<td>id_long</td>
<td>Cast numeric identifier to long type</td>
</tr>
<tr>
<td>timestampUTC_int64</td>
<td>Include int64 unix timestamp</td>
</tr>
</tbody>
</table>

*Table 3: Twitter data additional fields*
4.3 Container Design

The design for the container was relatively straightforward. What we needed to do was to find a base image that supports Apache Spark, and then include all of the necessary dependencies to run the program. The WARC conversion container is built from a base image of Apache Spark packaged by Bitnami that allowed us to run Spark version 2.4.3. [11] The JSON conversion container uses a different base image because it is designed for PySpark. In this case, our choice was a base image from project 'datamechanics.' [13] The remainder of the Dockerfiles are purposely simplistic, only including files and installations of critical importance.
5 Implementation

5.1 WARC to Parquet

This program uses support from Java, Scala, and Apache Spark packages. Additionally, two outside libraries are included in the project for reading WARC files and correctly mapping the content according to a schema. They are the web archive utility library, ia-web-commons [14], and the Archives Unleashed Toolkit. [1] Regarding the code itself, the majority of it pertains to setting up a Spark context and configuring IO. Reading into a dataframe, adding the aforementioned fixed timestamp column, and writing to an output file are all accomplished in one line each. Furthermore, the conversion itself was delegated to a helper method that is accessed via the main method. The purpose of the main method is to locate targets for the conversion from within a user supplied directory and call the conversion method accordingly.

5.2 JSON to Parquet

This program is relatively simplistic compared to the WARC program. It requires less setup and involves fewer dependencies. All of the needed support comes from PySpark; the default writing configurations proved to be sufficient. The logic follows a similar flow as the WARC program and the only unique challenge was determining the schema from sample JSON files. This was accomplished by reading the data into a dataframe and printing the schema. We could then use this schema to ensure files meet the expected structural outline. Once that challenge was met, the code to perform the conversion and data fixing was succinct. In both programs, the first and most important step was setting up a Spark context.

5.2.1 Docker Containers

For each containerized application, a custom Dockerfile was written. Both layered the specific application on top of an image capable of running Apache Spark by populating the file directory with the critical files. Test files were also included in each container. The challenges we faced in this part of the project came after successfully building the images. For instance, command line interfaces did not work the same way in the container as they did locally. An early iteration of the programs used true command line interfaces that requested user inputs throughout execution to find the next target for conversion. We found that even when run interactively with a bash shell, the containers could not facilitate a line of communication with a user. The solution was to design programs that take one argument and execute all the way to completion. Furthermore, in the case of the WARC program written in Scala, heap space was an issue in the container. The Docker run command had to include a Java option to expand the heap in order for it to work as expected. Consequently, this caused the container to abort when run on rlogin through a Linux machine because we do not possess the permissions to modify system variables.
Figure 9: WARC to Parquet Dockerfile

```
FROM gcr.io/datamechanics/spark:platform-3.1-latest

LABEL maintainers="matthewcoscia@vt.edu & atweber@vt.edu"

ENV PYSPARK_MAJOR_PYTHON_VERSION=3

RUN pip install pyspark

USER root

RUN adduser user

USER user

COPY --chown=user:user json_examples /home/user/json_examples
COPY --chown=user:user json2parquet.py /home/user

WORKDIR /home/user
```

Figure 10: JSON to Parquet Dockerfile
6 Testing

Our primary mode of confirming expected results was to view the generated Parquet files in a Spark Shell. We frequently used this approach to check columns, particularly the ones that we added to the original data. This approach also allowed us to confirm schema. If the hard coded schema that our read methods expected were not met, the outputted Parquet file would be empty and each cell would contain null values. Output from mid-execution was often sufficient in telling us that an error took place during a conversion. In these cases, the Parquet files would be empty and the programs would have crashed.

A more conclusive way to perform tests would be to read both the input and output files to a unique data frame and compare results. With the exception of the newly added columns, the data frames should match. Tests could also be performed on the new columns because they are derived from data in the input files. With structured tests and CI/CD principles[10], we could improve confidence in our applications going forward.

6.1 Uploading to VT Computer Science Cloud

Once we finished creating the containers and successfully ran them locally, we needed to move on to deployment. It did not make any sense to just have the containers available to us, we wanted many people to be able to make use of our work. This was one of our original goals, and we felt that if we were able to help researchers have easier and more efficient access to this large data, that we would be directly impacting the advancement of technology within the Virginia Tech Computer Science department. To achieve this, we had to work very closely with the VTCS Tech staff to get our work deployed to cloud.cs.vt.edu. Figure 11 shows a high level overview of what it looks like for a container to be deployed. The VTCS cloud belongs in the middle of the diagram with docker and kubernetes. The cloud is a container cluster management software that allows containerized applications to be accessed by everyone across the vt.edu domain. All VT Computer Science students have access to the discovery cluster. With the help of the VTCS tech staff, we were able to use the discovery cluster to hold our containers. All we had to do was create a namespace to hold our separate pods. Figure 12 shows both of our containers running in separate pods in the VT Cloud.

As you can see, we are working in the school’s discovery cluster, and we created our own namespace named
atweber-parquetcontainers2. This is following naming guidelines in manuals to upload containers to the VT Cloud. A namespace is just a tool used to isolate resources within a cluster for a small number of pods. Below that you can see that our two containers are up and running.

**Figure 12:** Containers running in separate pods in the VT Cloud
7 User Manual

7.1 WARC to Parquet

The compiled program is run using a `spark-submit` command that includes two supporting libraries and a single directory name parameter. The conversion code will only execute if the parameter maps to a valid directory in the user space that contains at least one compressed WARC file. The program does not notify the user if a valid input was provided. Rather, Spark will report none of the extensive tell-tale standard output corresponding to the task. It will simply exit the program. This process is identical through either the command line or an interactive container running with a bash shell.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spark-submit --class &quot;Warc2Parquet&quot; --master local[*] --jars lib/aut-0.17.1-DLRL.jar,lib/bin/ia-web-commons-1.1.0-jar-with-dependencies.jar target/scala-2.11/warc-2-parquet_2.11-1.0.jar warc_examples</code></td>
</tr>
</tbody>
</table>

Table 4: Spark submit command

7.2 JSON to Parquet

Written in Python, the program leverages PySpark to perform data processing. It accepts the name of a directory as the argument and performs file format conversions according to a schema on any JSON files. It is up to the user to provide a directory containing properly formatted JSON files to ensure meaningful results. The results from the program will go into a new directory. They can be verified by viewing the generated files in a data frame post execution. It is important to note that files that do not fit the schema will not trigger an error in the program and will result in an empty Parquet file. The command to run the program is like any standard Python program. See Table 5.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>python json2parquet.py json_examples</code></td>
</tr>
</tbody>
</table>

Table 5: PySpark command

7.3 Accessing the Containers

To access the containers, log in to cloud.cs.vt.edu and search for the namespace atweber-parquetcontainers2 in the discovery cluster. From there you will see the containers running in their respective pods and you can follow the steps listed above to convert your files to parquet format.
8 Developer Manual

8.1 WARC to Parquet

Spark supports execution of pre-compiled Scala files through the `spark-submit` command. Special considerations must be made in organizing the directory structure that will eventually contain the compiled JAR file. There are several steps to build and run the program with Spark.

File Structure

- `./lib/aut-0.17.1-DLRL.jar` (Archives Unleashed Toolkit with support for reading WARC files)
- `./lib/bin/ia-web-commons-1.1.0-jar-with-dependencies.jar` (Web archiving utility library with support for AUT)
- `./src/main/scala/Warc2Parquet.scala` (Source code)
- `./build.sbt` (Build file)
- `./example-input.warc.gz` (Compressed WARC file)

Build with SBT

The program is compiled into a JAR by running the command, `sbt package`. The command looks for source files in the `src` directory and links additional libraries by searching the `lib` directory.

```
name := "WARC 2 Parquet"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.3.4"
scalaOptions += "-deprecation"
```

Figure 13: build.sbt

Run with spark-submit

This step allows us to define the entry point for the application, define the number of logical cores we want to allocate to execution, and link additional libraries. Each specification is included in the run command. The files will be input during execution. Table 5 on page 19 contains the command to execute the code.

8.2 WARC to Parquet Containerized

The container is designed to be completely self contained. The user can run the container and observe the outputs in the local file system. Be sure to run the commands shown in table 14 in a directory with the proper Dockerfile. Examine the contents of the Dockerfile to ensure correct directory structure otherwise.
8.3 JSON to Parquet

Python applications leveraging Apache Spark are made possible by installing the package PySpark.

The application can be run on the command line using a standard python command; python json2parquet.py [directory name]. It is important to note that this implementation achieves results on a per-directory basis. Correct usage would be to supply a valid directory containing one or more JSON files following the expected schema as a program argument. Table 4 on page 19 contains the command to execute the code.

8.4 JSON to Parquet Containerized

The container image is built with all of the necessary dependencies. Run the container and observe the output in your local filesystem. Just like with WARC program, be sure to run the commands shown in table 16 in a directory with the proper Dockerfile. Examine the contents of the Dockerfile to ensure correct directory structure otherwise.

Future improvements to this program include error checking for schema and allowing the user to specify an output folder of choice.

8.5 Adding Columns to a Dataframe

This functionality is supported by Apache Spark through the “withColumn” method. The method allows developers to create a new column based on the data in another column according to function. [17]
Container Commands:
docker build -t warc2parquet .
docker run --mount type=bind,source="%cd%",target=/home/user warc2parquet

Figure 14: Running WARC to Parquet Container

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
import os
import sys
from os import path
import ntpath

# Global Variables
spark = SparkSession.builder.master("local[*]").appName("Read Big JSON").getOrCreate()
sc = spark.sparkContext
```

Figure 15: Packages and global variables used in json2parquet.py

Container Commands:
docker build -t json2parquet .
docker run --mount type=bind,source="%cd%",target=/home/user json2parquet

Figure 16: Running JSON to Parquet Container
9 Future Work

Now that the containers are functional and available in Virginia Tech's cloud space, future work should focus on deployment and persistence.

9.1 Kubernetes

Rather than separate deployment, the applications can be deployed in a single pod managed by Kubernetes. This would better reflect the connection between the applications and make them easier to use alongside one another. If the containers are running in the same pod, that means they can communicate with each other. This would be helpful to pass information back and forth, and the user would not need to submit twitter collection and WARC files separately anymore. A future developer could make this addition.

9.2 Ceph

Ceph is an open source, distributed storage system which would provide increased reliability and persistence to the applications. Considering that containers can be spun up and down frequently and sometimes unexpectedly, persistent storage would provide a link between different sessions. [5] If you look back at 12 you can see that the JSON to Parquet container already had 18 pod restarts even though it was only created 18 hours ago. This shows that containers are constantly being taken down and re-deployed, as resources for them are not always available. Ceph would provide a great opportunity to keep that Parquet data somewhere safe and reliable.
10 Lessons Learned

Parquet Containers exposed us to several new technologies and a modern data challenge. Along the way, we gained experience meeting both client-defined and self-defined goals. The process demanded that we gain new skills and a working understanding of the concepts in multiple contexts. For example, we learned how different designing software to be run on Windows and on Linux can be. Additionally, we were able to observe different ways to give commands to a container and used that knowledge to settle on the interactive approach. The remainder of this section goes into specific lessons we learned about productivity.

10.1 Group Work Sessions

After the first couple of weeks of excitement about our project, we sensed a slight lull in our productivity as a team. It was about the time that Heidi Gilbert came and spoke to us about how to operate in a team. We thought we were working well as a team, but Heidi presented us with several concrete ideas on how to improve our productivity that we hadn't yet thought of. A problem we found was that the project deadline was still so far away that we had trouble allocating enough time in our week to actually work on it, since there was always another homework that was due before it. After Heidi's presentation, Matthew had an idea that we should have bi-weekly group sessions on Wednesday and Friday afternoons. That meant 2-3 hours where we could meet as a team, talk over what we were working on, and just have that time allocated for only working on the capstone. This helped us focus and increase our productivity immensely. It is a major factor as to why we are on track to finish the project early.

10.2 Agile Sprints/Scrum

After spring break we heard from Christian Johnson about Agile Sprints and Scrum and we decided it would be a great option for us moving forward. We would allocate a little bit of our first meeting of the week (whether it be in class or in our bi-weekly meeting) to have a scrum and discuss what our short term goals were for the week. It was a great option and the team benefited greatly from it. Figure 17 is an illustration regarding one of our sprints.
## AGILE PROJECT PLAN TEMPLATE

<table>
<thead>
<tr>
<th>AT #</th>
<th>CASE NAME</th>
<th>FEATURE TYPE</th>
<th>RESPONSIBLE</th>
<th>STORY POINTS</th>
<th>START</th>
<th>FINISH</th>
<th>DURATION (DAYS)</th>
<th>STATUS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analyze JDOH schema and files</td>
<td>Matt</td>
<td>3/21/22</td>
<td>3/24/22</td>
<td>1</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Get started with jpspark</td>
<td>Matt</td>
<td>3/23/22</td>
<td>3/25/22</td>
<td>2</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Write basic JDOH to Parquet input</td>
<td>Matt</td>
<td>3/24/22</td>
<td>3/28/22</td>
<td>4</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>JDOH -&gt; Pa, Cu</td>
<td>Matt</td>
<td>3/28/22</td>
<td>3/30/22</td>
<td>2</td>
<td>In Progress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Script 2</td>
<td>Matt</td>
<td>3/22/22</td>
<td>3/29/22</td>
<td>7</td>
<td>In Progress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Run and interact with container</td>
<td>Andrew</td>
<td>3/23/22</td>
<td>3/24/22</td>
<td>1</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Use dockerfile to import files</td>
<td>Andrew</td>
<td>3/23/22</td>
<td>3/25/22</td>
<td>2</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Use dockerfile to download stages</td>
<td>Andrew</td>
<td>3/24/22</td>
<td>3/28/22</td>
<td>2</td>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Run JDOH -&gt; Pa, Cu in container</td>
<td>Andrew</td>
<td>3/28/22</td>
<td>3/29/22</td>
<td>3</td>
<td>In Progress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Use WARC -&gt; Pa, Cu in container</td>
<td>Andrew</td>
<td>3/27/22</td>
<td>3/30/22</td>
<td>3</td>
<td>In Progress</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Container 1 Documentation</td>
<td>Andrew</td>
<td>3/23/22</td>
<td>3/30/22</td>
<td>7</td>
<td>Not Started</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Update WARC -&gt; Pa, Cu</td>
<td>Matt</td>
<td>3/27/22</td>
<td>3/30/22</td>
<td>3</td>
<td>Not Started</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 17:** Agile Sprint Example
11 Acknowledgements

We would first like to thank Dr. Edward A. Fox for being nothing but helpful thus far throughout our semester-long project. Despite having 20 other projects to look after, we feel he always knows what’s going on with our group and stays up to date with our progress. We would also like to thank our client Xinyue Wang for being a terrific guide to us throughout the semester. He is always quick to respond to emails and willing to meet with us no matter how late notice. We surely would not be as successful if it wasn’t for those two people. Additionally we would like to acknowledge the TA assigned to our project, Ryan Wood. He has always been available whenever we run into a small technical issue. Finally, we would like to extend our appreciation to the VT CS Tech staff, specifically Chris Arnold, for helping us troubleshoot issues with rlogin and the VT CS cloud. All of the aforementioned people have played a huge role in us being able to complete our project.
References


