Writing an Extractor Using Simple Extractor Wrapper

Introduction

Clowder is an open-source research data management system that supports curation of long-tail data and metadata across multiple research domains and diverse data types. It uses a metadata extraction bus to perform data curation. Extractors are software programs that do the extraction of specific metadata from a file or dataset (a group of related files). The Simple Extractor Wrapper is a piece of software being developed to make the process of developing an extractor easier. This document will provide the details of writing an extractor program using the Simple Extractor Wrapper.

Goals of Simple Extractor Wrapper

An extractor can be written in any programming language as long as it can communicate with Clowder using a simple HTTP web service API and RabbitMQ. It can be hard to develop an extractor from the scratch when you also consider the code that is needed for this communication. To reduce this effort and to avoid code duplication, we created libraries written in Python (PyClowder) and Java (JClowder) to make the processing of writing extractors easy in these languages. We chose these languages since they are among the most popular ones and they continue to remain to so. Though this is the case, there is still some overhead in terms of developing an extractor using these libraries. In order to make the process of writing extractors even easier, we created a Simple Extractor Wrapper, that wraps around your existing Python source code and converts your code into an extractor. The main goal of this wrapper is to help create Python extractors with very minimal effort. As the name says, the extractor itself needs to be simple in nature. The extractor will process a file and generate metadata in JSON format and/or create a file preview. Any other Clowder API endpoints are not currently available through the Simple Extractor and the developer would have to fall back to using PyClowder, JClowder or writing the extractor from scratch.

Step-by-Step Instructions

Prerequisites

The step-by-step instructions to create an extractor using the Simple Extractor Wrapper assumes the following:

1. Python is installed on your computer. This document is based on Python 2.7, but it will soon be updated for Python 3 as the former is nearing its end of life.
2. Docker is installed on your computer. You can download and install Docker from https://www.docker.com/products/docker-desktop.
3. You already have a piece of code written in Python that can process a file and generate metadata. Optionally, this can also be used if you have a command-line program written in another programming language, which can be invoked from Python.
4. The extractor that you are trying to create will only generate metadata in JSON format and/or a file preview.
5. Your code has been tested and does what it is supposed to do.

The main function of your Python program needs to accept the string format file path of the input file. It also needs to return a dictionary containing either metadata information ("metadata"), details about file previews ("previews") or both in the following format:

```
{
    "metadata": dict(),
    "previews": list()
}
```

Instructions

Your extractor will contain several files. The ones that will be used by the Simple Extractor Wrapper are listed below. The instructions below will help you to create these files:

- *my_python_program.py* (required): For simplicity, let us call the Python file that contains the main function *my_python_program.py*, the main function *my_main_function*, and your extractor *my_extractor*.
- *extractor_info.json* (required): Contains metadata about the extractor
- *Dockerfile* (required): Contains instructions to create a docker image of your extractor
- *requirements.txt* (optional): Contains names of Python packages that will be installed using the *pip* command.
- *packages.apt* (optional): Contains names of Linux packages that will be installed using the *apt-get* command.

1. Create and save *extractor_info.json* using any text editor in your source directory. This file contains the metadata about the extractor that you are creating. Please fill in the relevant details about the extractor in this file. This document follows the JSON-LD standard. A template *extractor_info.json* has been provided below for reference. As you can see, you can fill in the details like name, version, author, contributors, source code repository, docker image name, the data types on which the extractor will work, external services used, any dependent libraries, BibTex format citation to a list of publications that the extractor is referring to, etc. An example *extractor_info.json* can be found here:

```
{
   "@context": "<context root URL>",
   "name": "<extractor name>",
   "version": "<version number>",
   "description": "<extractor description>",
   "author": "<first name> <last name> <<email address>>",
   "contributors": [
       "<first name> <last name> <<email address>>",
       "<first name> <last name> <<email address>>"
     ],
   "contexts": [
    {
       "<metadata term 1>": "<URL definition of metadata term 1>",
        "<metadata term 2>": "<URL definition of metadata term 2>"
     }
   ],
   "repository": [
      {
    "repType": "git",
         "repUrl": "<source code URL>"
      }
   ],
   "process": {
     "file": [
       "<MIME type/subtype>",
       "<MIME type/subtype>"
     ]
   },
   "external_services": [],
   "dependencies": [],
   "bibtex": []
 }
```

2. Download the Docker Compose file from
    a. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/raw/docker-compose.yml
    b. You can also use curl command to download it from a terminal:

    ```
    curl https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/raw/docker-compose.yml --output docker-compose.yml
    ```

3. Start up the Clowder services stack (Clowder, RabbitMQ, MongoDB, and ElasticSearch) by running the following command from the directory containing the downloaded *docker-compose.yml* file. This may take a few minutes when running for the first time:

    ```
    docker-compose -p clowder up
    ```

4. Create and save a Dockerfile in your existing source code directory. This can be done using any text editor in your computer. The content of the Dockerfile needs to be the following, where you should replace *my_python_program.py* and *my_main_function* with their actual names:

    ```
    FROM clowder/extractors-simple-extractor:onbuild
    ENV EXTRACTION_FUNC="my_main_function"
    ENV EXTRACTION_MODULE="my_python_program"
    ```

5. If there are any Python or Linux packages that are required by your code, please add them to two files named *requirements.txt* and *packages.apt* i n the source code directory. Each package entry should be added to a separate line in these files.

6. Now, create the Docker image for your extractor using the command below. Please note that there is a dot (.) at the end of the command. You will need to open a terminal client and change to your Dockerfile directory using the *cd* command before running the command below (this will also install the Python packages from *requirements.txt* and Linux *apt-get* packages from *packages.apt*):

    ```
    docker build -t my_extractor .
    ```

    In the terminal, you should be able to see the logs of the services that are part of the Clowder stack.

7. From another terminal window, you can now run your extractor using the following command:

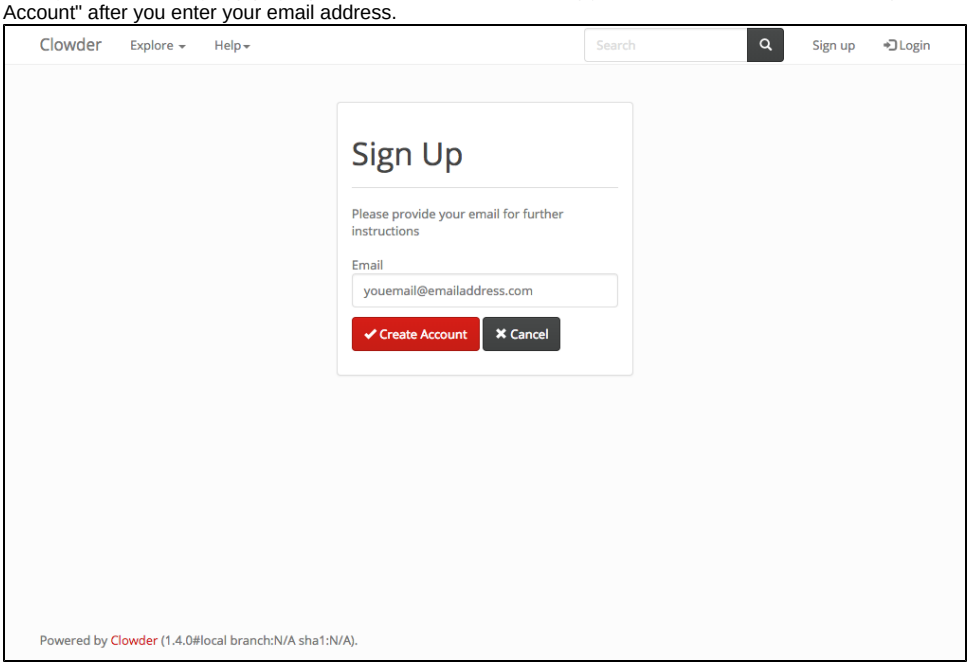
    ```
    docker run -t -i --rm --network clowder_clowder my_extractor
    ```

    You should be able to see the logs related to the starting extractor in this terminal window.

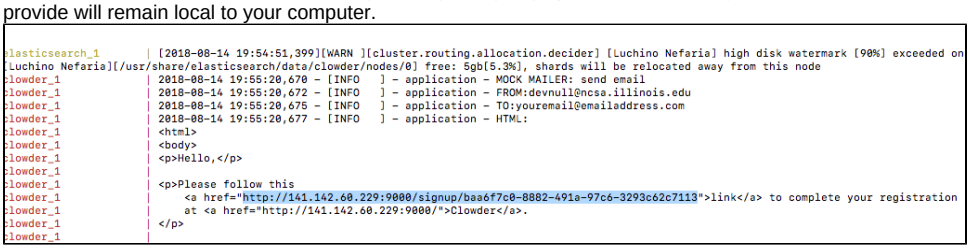
8. You can always test your python code before wrapping it as an extractor. To test your built extractor, you will need to sign up and create an account in your local Clowder instance. Please follow the steps below:

a. Open your web browser and go to *http://<ip_address>:9000/signup*, where *<ip_address>* needs to be replaced by your computer's IP address. You can run *ifconfig* (Mac/Linux) or *ipconfig* (Windows) command from a terminal window to find your computer's IP address.

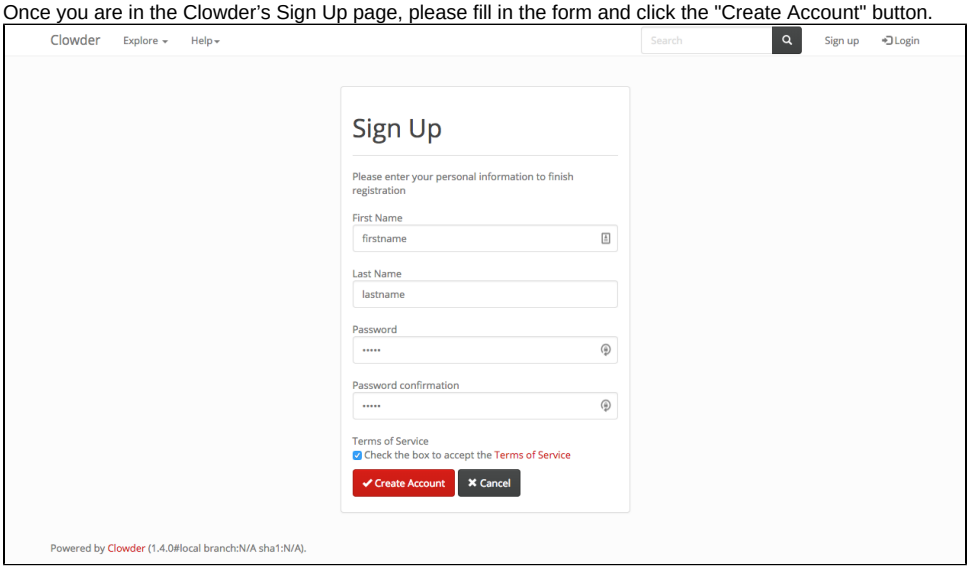
b. Once you are in the sign up page, please create an account using your email address as shown in figure above. Click on the "Create Account" after you enter your email address.



c. Now, you need go back to the terminal where you launched Clowder services stack and check the log (for concise presentation, the below screenshot has been partially cropped on left side). You will see a URL (highlighted) of the Sign Up form. Please go to that URL from your web browser. Please note that you are just signing up for an account in your local Clowder instance and the information you provide will remain local to your computer.



d. Once you are in the Clowder's Sign Up page, please fill in the form and click the "Create Account" button.



e. Now you can login to your local Clowder instance your email address and password that you set up during the signup process. After you login, you can create a dataset and upload a file for testing. After the extractor processes your file, you will be able to see the generated metadata in the Clowder file page. You will also see some relevant messages in the terminal window where the extractor is running.

9. To stop the Clowder services stack, you will need to open a terminal client and change to your Clowder *docker-compose.yml* directory using the *cd* command before running the command below:

```
docker-compose -p clowder down
```

10. To stop the extractor, you will need to go to the terminal where the extractor is running and press **Control + C** from keyboard.

Creating a Word Count Extractor Using Simple Extractor Wrapper

Now, following the instructions in the previous section we can create a word count extractor using the Simple Extractor Wrapper. There are three files in this extractor, namely, *Dockerfile*, *extractor_info.json*, and *wordcount.py*. The word count extractor utilizes the Ubuntu built in command *wc* to find the number of lines, words, and characters in a text or JSON format file. No additional packages are needed by this extractor and hence there are no package installation files (*requirements.txt*, *packages.apt*).

1. As the first step, let us create an *extractor_info.json* file that contains the details about the word count extractor as shown in Figure 1. Add this to the source code directory containing the word count Python main program.

```
{
  "@context": "http://clowder.ncsa.illinois.edu/contexts/extractors.jsonld",
  "name": "ncsa.wordcount",
  "version": "1.0",
  "description": "WordCount simple extractor. Counts the number of characters, words and lines in the text file that was uploaded.",
  "author": "Bing Zhang <bing@illinois.edu>",
  "contributors": [],
  "contexts": [
    {
      "lines": "http://clowder.ncsa.illinois.edu/metadata/ncsa.wordcount#lines",
      "words": "http://clowder.ncsa.illinois.edu/metadata/ncsa.wordcount#words",
      "characters": "http://clowder.ncsa.illinois.edu/metadata/ncsa.wordcount#characters"
    }
  ],
  "repository": {
    "repType": "git",
    "repUrl": "https://opensource.ncsa.illinois.edu/stash/scm/cats/pyclowder.git"
  },
  "process": {
    "file": [
      "text/*",
      "application/json"
    ]
  },
  "external_services": [],
  "dependencies": [],
  "bibtex": []
}
```

Figure 1. Word count extractor_info.json. In extractor_info.json file, users need to give unique name(e.g., ncsa.wordcount) to the "name" field. In "contexts" field, you can give the semantic definition URL of metadata (e.g., the definition of character in wordcount extractor means the alphabetic letter instead of a person in a movie). In process filed, you can specify the mime type files which is applicable to word count extractor(e.g., wordcount can process any type of text files and JSON files).

2. Please follow the steps 2 and 3 in the previous section to setup and start the Clowder stack using Docker compose.

3. Now, create and save a Dockerfile for the wordcount extractor. Two environment files need to be set in the Dockerfile. EXTRACTION_FUNC is used to store the name of the main function in the Python program and EXTRACTION_MODULE is used to store the name of the Python main program file. In this example we set EXTRACTION_FUNC and EXTRACTION_MODULE to "wordcount". You will later see that the Python file and main function names are the same in this example:

```
FROM clowder/extractors-simple-extractor:onbuild

ENV EXTRACTION_FUNC="wordcount"
ENV EXTRACTION_MODULE="wordcount"
```

Figure 2. Word count extractor Dockerfile.

```python
import subprocess


def wordcount(input_file_path):
    """
    This function calculates the number of lines, words, and characters in a text format file.

    :param input_file_path: Full path to the input file
    :return: Result dictionary containing metadata about lines, words, and characters in the input file
    """

    # Execute word count command on the input file and obtain the output
    result = subprocess.check_output(['wc', input_file_path], stderr=subprocess.STDOUT)

    # Split the output string into lines, words, and characters
    (lines, words, characters, _) = result.split()

    # Create metadata dictionary
    metadata = {
        'lines': lines,
        'words': words,
        'characters': characters
    }

    # Store metadata in result dictionary
    result = {
        'metadata': metadata
    }

    # Return the result dictionary
    return result
```

Figure 3. Word count program file and main function. The word count program calculates the count of lines, words, and characters and stores it into a "metadata" dictionary object (lines 19 ~ 22), which is then stored in a "result" dictionary object (lines 26 ~ 28) and returned.

Running wordcount.py as regular python code

On the terminal, you can run the wordcount.py as a regular python code, e.g., you can use the below command to run the wordcount.py to extract the counts of lines, words and characters from the input text format file. For example, let us create a text file called *poem.txt* containing the first stanza of the poem "The Road Not Taken" by Robert Frost:

```
"Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;"
```

Now we can run the following command to test the word count code:

```
python -c "import wordcount; print wordcount.wordcount(\"poem.txt\")"
```

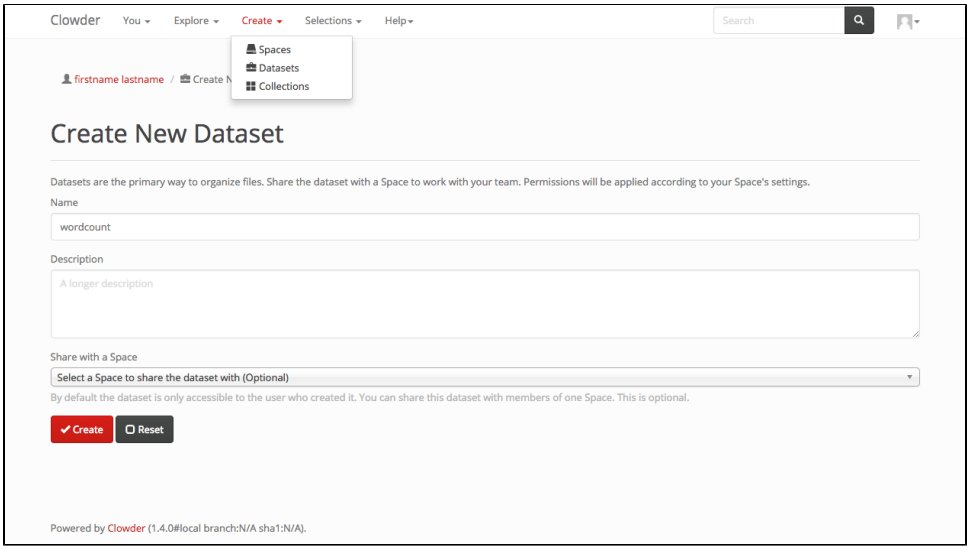
It will show the output as:

```
{'metadata': {'lines': 5', 'characters': 182', 'words': '37'}}
```

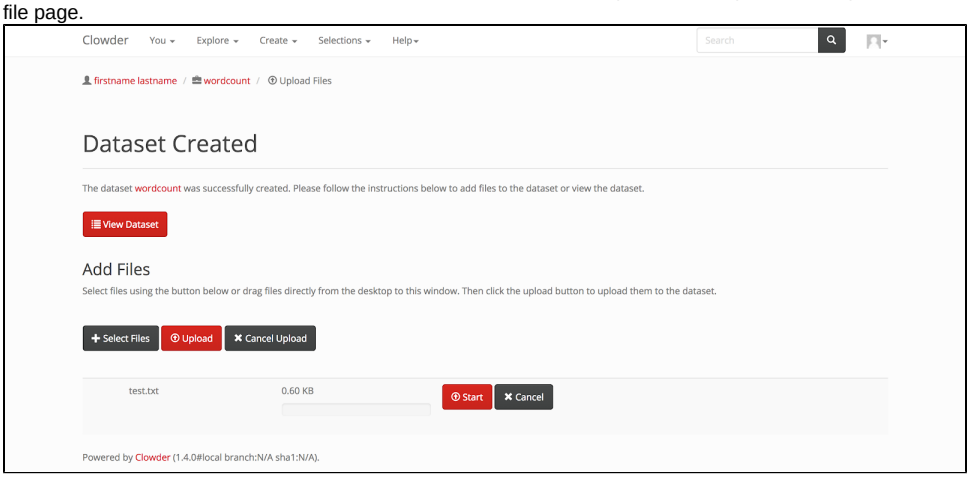
Running wordcount.py as extractor

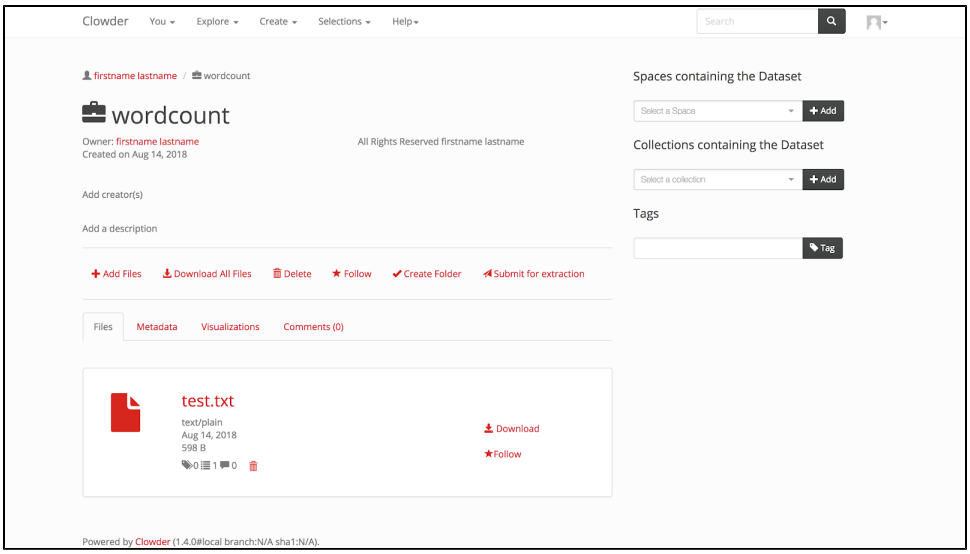
Word count extractor runs as docker container. Thus, if you want to use word count extractor, please refer to instruction steps to start Clowder services stack and then launch the word count extractor. Then you can submit files from Clowder Web GUI and please follow the steps listed below:

a. After logging into Clowder, you need to create a dataset (in this example, we create a dataset and name it wordcount) and upload files into that dataset (you can select multiple files and click "Upload" button to submit to Clowder in a batch).

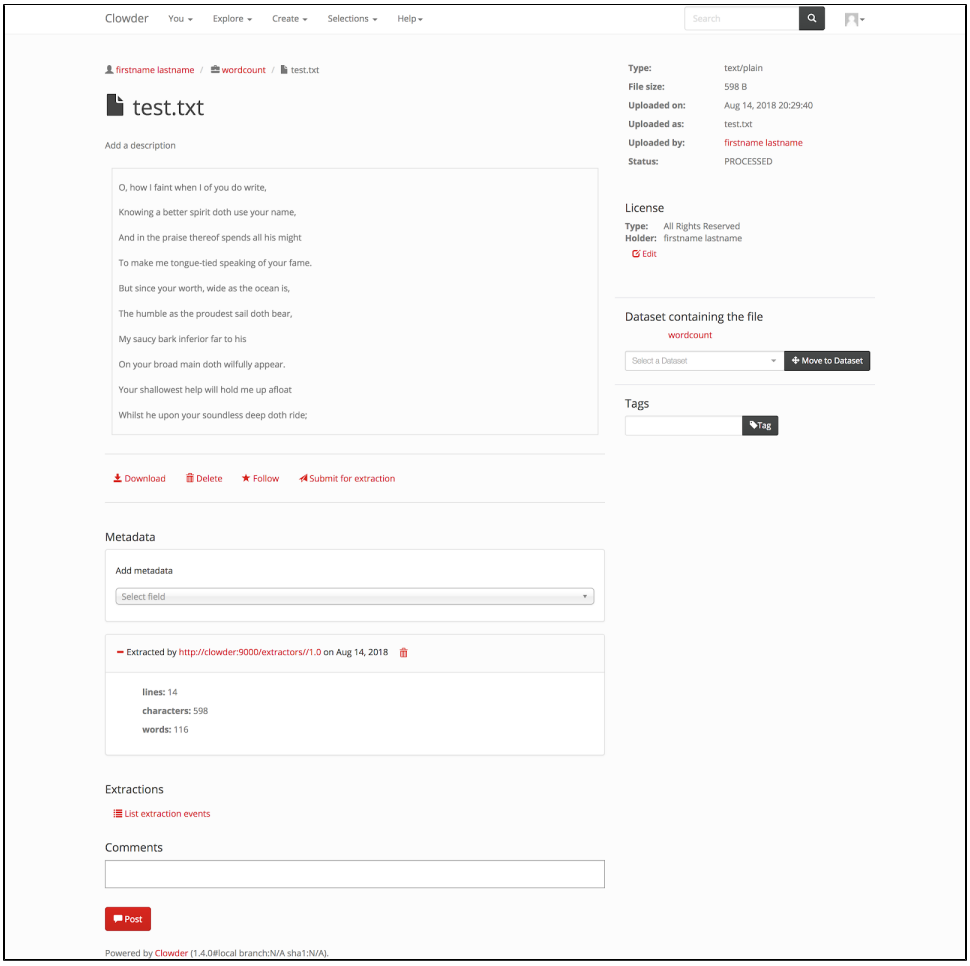


b. After successfully uploading files to Clowder, you can browse the uploaded files by clicking the "View Dataset" button. There is one file named "test.txt" in the "wordcount" dataset. To view the metadata outputs on this uploaded file, please click the "test.txt" link to go to the file page.





c. On the file page, you can see the metadata output of word count extractor. Sometimes, the extractor will take some time to upload extracted metadata to Clowder, so you may need to refresh the browser page to view the output metadata.



d. Now, you need to go back to the terminal where you launched the word count extractor. On the terminal, you can see the log of the extractor processing the uploading files (for concise presentation, the below screenshot has been partially cropped on left side).

```
2018-08-14 20:29:28,668 [MainThread     ] INFO    : pika.adapters.base_connection - Connecting to 172.23.0.5:5672
2018-08-14 20:29:28,680 [MainThread     ] INFO    : pika.adapters.blocking_connection - Created channel=1
2018-08-14 20:29:28,761 [MainThread     ] INFO    : pyclowder.extractors - Waiting for messages. To exit press CTRL+C
2018-08-14 20:29:28,762 [Connector-0    ] INFO    : pyclowder.connectors - Starting to listen for messages.
2018-08-14 20:29:40,380 [Thread-1       ] INFO    : pyclowder.connectors - [5b733bb4e4b0fa9aad7215c3] : StatusMessage.start: Started processing
2018-08-14 20:29:40,397 [Thread-1       ] INFO    : pyclowder.connectors - [5b733bb4e4b0fa9aad7215c3] : StatusMessage.processing: Downloading file.
2018-08-14 20:29:40,449 [Thread-1       ] INFO    : __main__ - upload metadata
2018-08-14 20:29:40,449 [Thread-1       ] INFO    : pyclowder.connectors - [5b733bb4e4b0fa9aad7215c3] : StatusMessage.processing: Uploading file metadata.
2018-08-14 20:29:41,944 [Thread-1       ] INFO    : pyclowder.connectors - [5b733bb4e4b0fa9aad7215c3] : StatusMessage.done: Done processing
```

The source code of the word count extractor created using the simple extractor wrapper can be found here: https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/sample-extractors/wordcount-simple-extractor