CLOWDER WEBINAR OCT. 4TH, 2019

HTTP://CLOWDERFRAMEWORK.ORG/

AGENDA

- Webinars recordings and slides
- Developing simple metadata extractors in Python and R
- ~~Clowder Flutter phone app~~

MONTHLY WEBINAR

- **When**: First Friday of the Month 11am CST
- **Who**: Users and Developers
- **What**: Discuss new and old features
  - 2 presentations by community members
  - Please suggest topics and volunteer to present
- **Where**: https://illinois.zoom.us/j/856788350

WEEKLY DEVELOPER MEETINGS

- **When**: 2nd, 3rd, 4th Friday of the Month 11am CST
- **Who**: Software Developers
- **What**: Technical discussions
- **Where**: https://illinois.zoom.us/j/856788350

SIMPLE EXTRACTORS

ROB KOOPER

OVERVIEW

- **What is an Extractor?**
  - Example extractors
  - Calling extractors
  - Creating an Extractor (the hard way)
- **PyClowder and Extractors**
  - What is PyClowder
  - Creating an Extractor (using PyClowder)
  - Testing an Extractor
- **Simple Extractor**
  - The contract
  - What files are needed?
  - Creating an Extractor (using simple extractor, both python and R)

WHAT IS AN EXTRACTOR?

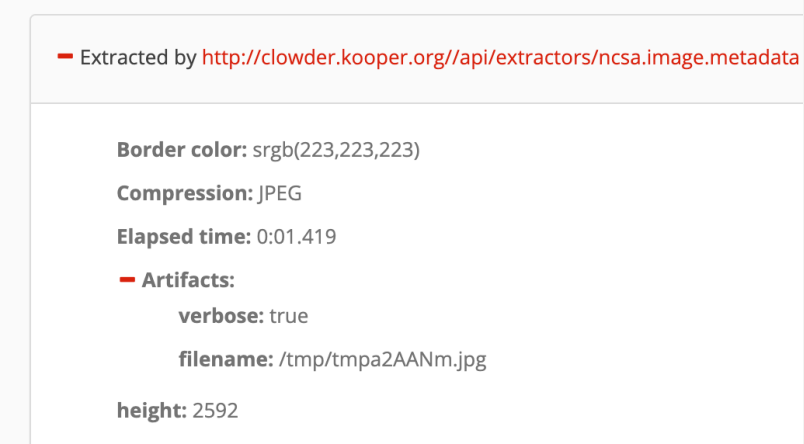
Responsible for extracting information from the data

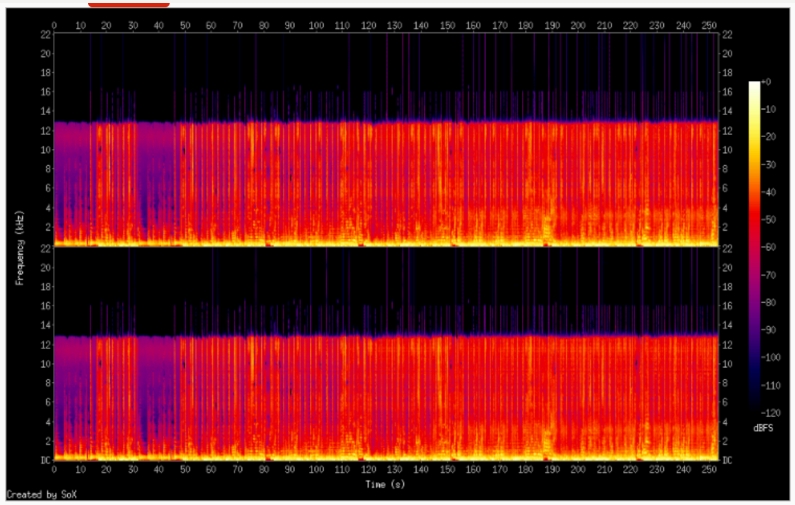
- Can extract information from files/datasets/collections
- Can have many different triggers
  - Upload of files
  - Addition of files to datasets
  - Addition of datasets to collections
  - Manually triggered
- Extractors can have parameters passed in
- Extractors can be grouped (in version 1.8)
- Extractors have a single file to describe them **extractor-info.json**

EXAMPLE EXTRACTORS

Responsible for extracting information from the data

- Information embedded in images
  - EXIF
  - Text in images using OCR
  - Faces in images
  - …
- Can create previews of the data
  - Small preview of images
  - Visual representation of the audio
- Can extract information from group of files
  - Geo spatial bounding box of group of images





CALLING EXTRACTORS

Extractors are triggered by Clowder

- Sends a message to the message bus (RabbitMQ)
  - Contains the data to connect to clowder
  - Contains the data that needs to be operated on

Extractors process their own messages

- Listen on their own queue
  - Each extractor should have their own queue
- Should not remove message from queue until finished
  - If extractors crashes, message is put back on queue

CREATING AN EXTRACTOR

It simple (right)

- Just find a RabbitMQ software
- Connect to the RabbitMQ server
- Listen for messages
- Process each message
- Inform clowder that you started downloading data
- Download data
- Inform clowder that you started processing
- Process data
- When finished upload data back to clowder
- Inform clowder that you finished processing
- Remove message from the queue

PYCLOWDER

Simplifies creating extractors

- Written in python (supports both python 2.7 and python 3)
- Takes care of RabbitMQ connection
- Does most of the grunt work
- Has functions to interact with clowder

Need to extend Extractors class

- Implement process_file function
- Uses extractor_info.json

EXTRACTOR-INFO.JSON

Describes the extractor

- Name of extractor
  - Used as the queue when sending message
  - Should be unique (use company prefix)
- Version number should follow semantic versioning
- Author and Contributes to track who worked on it
- Context
  - List of all metadata that is returned by extractor
  - Might eventually be used to validate metadata
- Process
  - What should trigger the extractor


```
{
  "@context": "http://clowder.ncsa.illinois.edu/contex
  "name": "ncsa.wordcount",
  "version": "1.0",
  "description": "WordCount simple extractor. Counts t
  "author": "Bing Zhang <bing@illinois.edu>",
  "contributors": [],
  "contexts": [
    {
      "lines": "http://clowder.ncsa.illinois.edu/metad
      "words": "http://clowder.ncsa.illinois.edu/metad
      "characters": "http://clowder.ncsa.illinois.edu/
    }
  ],
  "repository": [
    {
      "repType": "git",
      "repUrl": "https://opensource.ncsa.illinois.edu/
    }
  ],
  "process": {
    "file": [
      "text/*",
      "application/json"
    ]
  },
  "external_services": [],
  "dependencies": [],
  "bibtex": [
    "Luigi Marini, Indira Gutierrez-Polo, Rob Kooper,
  ]
}
```

CREATING AN EXTRACTOR (PYCLOWDER)

Becomes simpler

- ~~Just find a RabbitMQ software~~
- ~~Connect to the RabbitMQ server~~
- ~~Listen for messages~~
- ~~Process each message~~
- ~~Inform clowder that you started downloading data~~
- ~~Download data~~
- ~~Inform clowder that you started processing~~
- Process data
- When finished upload data back to clowder
- ~~Inform clowder that you finished processing~~
- ~~Remove message from the queue~~

EXAMPLE EXTRACTOR

Extend Extractor class

Implement process_message

- Get the filename
- Process message
- Create metadata result to upload
- Upload result to clowder

Make sure to start the extractor

```python
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)
    result = result.decode('utf-8')

    # 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


if __name__ == "__main__":
    print(wordcount("/etc/passwd"))
```

TESTING EXTRACTOR (PYCLOWDER)

Start clowder

- Use docker to start the full clowder stack
- Wait for mongo, RabbitMQ and clowder to be up and running

Start your extractor

- Wait for it to connect to clowder

Upload your test file to clowder

- See message come into extractor
- See extractor run (hopefully)
- Check result in clowder

SIMPLE EXTRACTOR

What can we do to make this even easier

- Remove last bit of boiler plate code
  - Create the extractor class
  - Create the metadata message
  - Upload the metadata to clowder

Simplify testing of extractors

- Remove requirement of clowder/pyclowder
- Call function with a file, return result

SIMPLE EXTRACTOR CONTRACT

Should implement a function that takes a file as input argument

Simple Extractors will return a well formed map of results

- List of metadata to return
- List of previews images to return

Metadata should map to context in extractor_info.json

```
{
  "metadata": {
    "lines": 11,
    "words": 20,
    "characters": 500
  },
  "previews": [
    "file1.jpg",
    "file2.jpg"
  ]
}
```

```
"contexts": [
  {
    "lines": "http://clowder.ncsa
    "words": "http://clowder.ncsa
    "characters": "http://clowde
  }
],
```

SIMPLE EXTRACTOR

Provide file with actual code

- No dependencies on clowder

Create extractor_info.json

- Same as before

Create/Copy Dockerfile

```
FROM clowder/extractors-simple-extractor-python3:onbuild

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

SIMPLE EXTRACTOR - PYTHON

Create function to do the processing

- Takes file as input
- Returns dict with results

Can have main for testing

```python
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)
    result = result.decode('utf-8')

    # 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

if __name__ == "__main__":
    extractor = WordCount()
    extractor.start()
```

SIMPLE EXTRACTOR - PYTHON

Can have packages.apt

- List of Debian packages to install
- Will be installed first by docker build

Can have requirements.txt

- List your python dependencies
- Will be installed by docker build

Automatic copy of python files and extractor_info.json

Can extend Dockerfile but will be run **AFTER** steps above

SIMPLE EXTRACTOR - R

Same as python extractor but uses R

- Uses python code to call R as subprocess
- Uses helper code in R to call your function and serialize list to JSON
- Python code will deserialize JSON and send results back to clowder

```
FROM clowder/extractors-simple-r-extractor-python3:onbuild

ENV R_SCRIPT="wordcount.R" \
    R_FUNCTION="process_file"
```

```r
process_file <- function(filename) {
  lines <- 0
  words <- 0
  characters <- 0

  con <- file(filename, "r")
  while (TRUE) {
    line = readLines(con, n = 1)
    if ( length(line) == 0 ) {
      break
    }
    lines <- lines + 1
    line <- strsplit(line, '\\s+')[[1]]
    if (length(line) != 0) {
      words <- words + length(line)
      characters <- characters + sum(sapply(line, nchar))
    }
  }
  close(con)

  list(
    metadata=list(
      lines=lines,
      words=words,
      characters=characters
    )
  )
}
```

SIMPLE EXTRACTOR - R

Can have packages.apt

- List of Debian packages to install
- Will be installed first by docker build

Can have docker.R

- Script to install any R packages
- Will be installed by docker build

Automatic copy of all files and extractor_info.json

Can extend Dockerfile but will be run **AFTER** steps above

HTTP://CLOWDERFRAMEWORK.ORG/

HTTPS://GITHUB.COM/CLOWDER-FRAMEWORK/PYCLOWDER