pyClowder 2

Max Burnette

Rob Kooper

Why pyClowder 2?

- Cleanup of code
- Easier to write extractors
- Easier to use in other applications

Code cleanup

- PEP 8
- No more dict that knows it all
  - Server, key, etc are now variables passed into functions
- Ability to multi-thread extractors
- Ease to add new ways to connect to clowder
  - RabbitMQ, HPC, …
- Use classes to encapsulate
- Module per api section
  - /api/files => files.py
  - /api/datasets => datasets.py

Code example

- pyClowder in **extractors.py**

```
def upload_file_metadata(mdata, parameters):
    """Upload file metadata to Clowder"""
```

- pyClowder2 in **files.py**

```
def upload_metadata(connector, host, key, fileid, metadata):
    """Upload file JSON-LD metadata to Clowder.

    Keyword arguments:
    connector -- connector information, used to get missing parameters and
send status updates
    host -- the clowder host, including http and port, should end with a /
    key -- the secret key to login to clowder
    fileid -- the file that is currently being processed
    metadata -- the metadata to be uploaded
    """
```

Extractors and connectors

- Connectors are classes that connect to Clowder
  - RabbitMQConnector and HPCConnector
  - Listen for messages (listen function)
  - Process status messages (status_update function)
- Extractors are classes that actually extract information
  - Parse command line arguments
  - Start connector(s)
  - Wait for messages (check_message and process_message)

Config and help

- No more config.py!
- Supports environment vars
- `wordcount.py --help`
- Where does it come from?

```
usage: wordcount.py [-h] [--connector [{RabbitMQ,HPC}]] [--logging [LOGGING]]
                    [--num [NUM]]
                    [--pickle [HPC_PICKLEFILE [HPC_PICKLEFILE ...]]]
                    [--register [REGSTRATION_ENDPOINTS]]
                    [--rabbitmqURI [RABBITMQ_URI]]
                    [--rabbitmqExchange [RABBITMQ_EXCHANGE]] [--sslignore]
                    [--version]

WordCount extractor. Counts the number of characters, words and lines in the
text file that was uploaded.

optional arguments:
  -h, --help            show this help message and exit
  --connector [{RabbitMQ,HPC}], -c [{RabbitMQ,HPC}]
                        connector to use (default=RabbitMQ)
  --logging [LOGGING], -l [LOGGING]
                        file or logging coonfiguration (default=None)
  --num [NUM], -n [NUM]
                        number of parallel instances (default=1)
  --pickle [HPC_PICKLEFILE [HPC_PICKLEFILE ...]]
                        pickle file that needs to be processed (only needed
                        for HPC)
  --register [REGSTRATION_ENDPOINTS], -r [REGSTRATION_ENDPOINTS]
                        Clowder registration URL (default=)
  --rabbitmqURI [RABBITMQ_URI]
                        rabbitMQ URI
                        (default=amqp://guest:guest@127.0.0.1/%2f)
  --rabbitmqExchange [RABBITMQ_EXCHANGE]
                        rabbitMQ exchange (default=clowder)
  --sslignore, -s       should SSL certificates be ignores
  --version             show program's version number and exit
```

Logging

- Configure logging
  - from commandline!
  - Pass in file
    - JSON/YAML/python
  - Pass in JSON String

```
{
    "version": 1,
    "formatters": {
        "f": {
            "format": "%(asctime)s %(name)-12s %(levelname)-8s %(message)s"
        }
    },
    "handlers": {
        "h": {
            "class": "logging.StreamHandler",
            "formatter": "f",
            "level": "logging.DEBUG"
        }
    },
    "root": {
        "handlers": [
            "h"
        ],
        "level": "logging.DEBUG",
    }
}
```

Migration Steps

1. **Move old extractor functions into new Extractor subclass**

2. Replace parameters[] references with explicit parameters or resource[] properties

3. Update pyClowder imports & calls to new modular structure

4. Define any necessary command line args your script need & update extractor_info

```
class WordCount(Extractor):
    """Count the number of characters, words and lines in a text file."""
    def __init__(self):
        Extractor.__init__(self)

        # add any additional arguments to parser
        # self.parser.add_argument('--max', '-m', type=int, nargs='?', default=-1,
        #                          help='maximum number (default=-1)')

        # parse command line and load default logging configuration
        self.setup()

        # setup logging for the exctractor
        logging.getLogger('pyclowder').setLevel(logging.DEBUG)
        logging.getLogger('__main__').setLevel(logging.DEBUG)
```

```
def check_message(self, connector, host, secret_key, resource, parameters)
```

```
def process_message(self, connector, host, secret_key, resource, parameters)
```

Migration Steps

1. Move old extractor functions into new Extractor subclass
2. **Replace parameters[] references with explicit references or resource[] properties**
3. Update pyClowder imports & calls to new modular structure
4. Define any necessary command line args your script need & update extractor_info

```python
def process_message(self, connector, host, secret_key, resource, parameters):
    # Process the file and upload the results

    logger = logging.getLogger(__name__)
    inputfile = resource["local_paths"][0]
    file_id = resource['id']

    # call actual program
    result = subprocess.check_output(['wc', inputfile], stderr=subprocess.STDOUT)
    (lines, words, characters, _) = result.split()

    # store results as metadata
    result = {
        'lines': lines,
        'words': words,
        'characters': characters
    }
    metadata = self.get_metadata(result, 'file', file_id, host)
    logger.debug(metadata)

    # upload metadata
    pyclowder.files.upload_metadata(connector, host, secret_key, file_id, metadata)
```

Migration Steps

1. Move old extractor functions into new Extractor subclass

2. Replace parameters[] references with explicit parameters or resource[] properties

3. **Update pyClowder imports & calls to new modular structure**

4. Define any necessary command line args your script need & update extractor_info

WordCount Extractor

```
#!/usr/bin/env python

"""Example extractor based on the clowder code."""

import logging
import subprocess

from pyclowder.extractors import Extractor
import pyclowder.files

class WordCount(Extractor):
    """Count the number of characters, words and lines in a text file."""
    def __init__(self):
        Extractor.__init__(self)

        # add any additional arguments to parser

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

pyclowder.files

```
def download(connector, host, k
def download_info(connector, ho
def download_metadata(connector
def upload_metadata(connector,
# pylint: disable=too-many-argu
def upload_preview(connector, h
def upload_tags(connector, host
def upload_thumbnail(connector,
def upload_to_dataset(connector
```

pyclowder.datasets

```
def create_empty(connector,
def download(connector, host
def download_metadata(connec
def get_info(connector, host
def get_file_list(connector,
def remove_metadata(connecto
def upload_metadata(connecto
```

pyclowder.geostreams

```
def create_sensor(connector, hos
def create_stream(connector, hos
def create_datapoint(connector,
def get_sensor_by_name(connector
def get_sensors_by_circle(connec
def get_sensors_by_polygon(conne
def get_stream_by_name(connector
def get_streams_by_circle(connec
def get_streams_by_polygon(conne
```

Migration Steps

1. Move old extractor functions into new Extractor subclass

2. Replace parameters[] references with explicit parameters or resource[] properties

3. Update pyClowder imports & calls to new modular structure

4. **Define any necessary command line args your script need & update extractor_info**

pyClowder 2 includes default args

```
# create the actual extractor
self.parser = argparse.ArgumentParser(description=self.extractor_info['description'])
self.parser.add_argument('--connector', '-c', type=str, nargs='?', default="RabbitMQ",
                         choices=["RabbitMQ", "HPC"],
                         help='connector to use (default=RabbitMQ)')
self.parser.add_argument('--logging', '-l', nargs='?', default=None,
                         help='file or logging coonfiguration (default=None)')
self.parser.add_argument('--num', '-n', type=int, nargs='?', default=1,
                         help='number of parallel instances (default=1)')
self.parser.add_argument('--pickle', type=file, nargs='*', dest="hpc_picklefile",
                         default=None, action='append',
                         help='pickle file that needs to be processed (only needed for HPC)')
self.parser.add_argument('--register', '-r', nargs='?', dest="regstration_endpoints",
                         default=registration_endpoints,
                         help='Clowder registration URL (default=%s)' % registration_endpoints)
self.parser.add_argument('--rabbitmqURI', nargs='?', dest='rabbitmq_uri', default=rabbitmq_uri,
                         help='rabbitMQ URI (default=%s)' % rabbitmq_uri.replace("%", "%%"))
self.parser.add_argument('--rabbitmqExchange', nargs='?', dest="rabbitmq_exchange", default=rabbitmq_exchange,
                         help='rabbitMQ exchange (default=%s)' % rabbitmq_exchange)
self.parser.add_argument('--mounts', '-m', dest="mounted_paths", default=mounted_paths,
                         help="dictionary of {'remote path':'local path'} mount mappings")
self.parser.add_argument('--sslignore', '-s', dest="sslverify", action='store_false',
                         help='should SSL certificates be ignores')
self.parser.add_argument('--version', action='version', version='%(prog)s 1.0')
```

```
# add any additional arguments to parser
# self.parser.add_argument('--max', '-m', type=int, nargs='?', default=-1,
#                          help='maximum number (default=-1)')
self.parser.add_argument('--output', '-o', dest="output_dir", type=str, nargs='?',
                         default="/home/extractor/sites/ua-mac/Level_1/demosaic",
                         help="root directory where timestamp & output directories will be created")
self.parser.add_argument('--mongo', dest="mongo_storage", type=str, nargs='?',
                         default=False,
                         help="if true, ignore file path logic because paths are mongo upload paths")
self.parser.add_argument('--overwrite', dest="force_overwrite", type=bool, nargs='?', default=False,
                         help="whether to overwrite output file if it already exists in output directory")
self.parser.add_argument('--betyURL', dest="bety_url", type=str, nargs='?',
                         default="https://terraref.ncsa.illinois.edu/bety/api/beta/traits.csv",
                         help="traits API endpoint of BETY instance that outputs should be posted to")
self.parser.add_argument('--betyKey', dest="bety_key", type=str, nargs='?', default=False,
                         help="API key for BETY instance specified by betyURL")
self.parser.add_argument('--plots', dest="plots_shp", type=str, nargs='?',
                         default="/home/extractor/extractors-metadata/sensorposition/shp/sorghumexpfall2016v5/sorghumexpfall2016v5_l
                         help=".shp file containing plots")

# parse command line and load default logging configuration
self.setup()

# setup logging for the exctractor
logging.getLogger('pyclowder').setLevel(logging.DEBUG)
logging.getLogger('__main__').setLevel(logging.DEBUG)

# assign other arguments
self.output_dir = self.args.output_dir
self.mongo_storage = self.args.mongo_storage
self.force_overwrite = self.args.force_overwrite
self.bety_url = self.args.bety_url
self.bety_key = self.args.bety_key
self.plots_shp = self.args.plots_shp
```

extractor_info

```
{
  "@context": "http://clowder.ncsa.illinois.edu/contexts/extractors.jsonld",
  "name": "ncsa.wordcount",
  "version": "2.0",
  "description": "WordCount extractor. Counts the number of characters, words and lines in the text file that was uploaded
  "author": "Rob Kooper <kooper@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": []
}
```

RabbitMQ queue

Help Information

JSON-LD Context

RabbitMQ registration type
*.file.text.#
*.file.application.json.#
extractors.ncsa.wordcount

pyClowder 2 going forward

- Support more Clowder API calls
  - Not just for extractors, but for any application that interacts with Clowder
  - Geostreams in-progress
- Support wider extractor varieties
  - trigger on a dataset and process a group of files
  - trigger on a collection and process a group of datasets
  - Scheduled extraction events (e.g. execute this large extraction at 1am daily)

- What else do people need?