Extractors

Extractors are services that run silently alongside Clowder. They can be configured to wait for specific file types to be uploaded into Clowder, and automatically execute scripts and processing on those files to extract metadata or generate new files.

Setting up a development environment

In order to develop and test an extractor, it's useful to have a local instance of Clowder running that you can test against. This will allow you to upload target files to trigger your extractor, and verify any outputs are being submitted back into Clowder correctly.

Start a local Clowder instance

The easiest way to get a local Clowder instance up and running is via Docker. We have created a docker image with the full Clowder stack already installed:

- Java (to run the Clowder application itself)
- MongoDB (underlying database storage)
- RabbitMQ (message bus for communication between Clowder and extractors)
- Clowder itself

It's possible to install these elements individually, but unless you want to pursue Clowder development this is unnecessary.

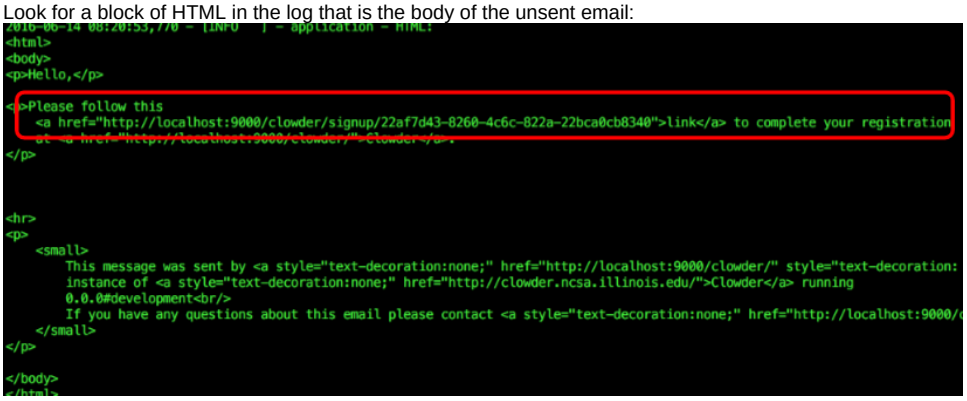
1. Install Docker and download our docker-compose.yml file from GitHub. This file tells Docker how to fetch and start the component containers for Clowder and its dependencies.

2. From a docker-aware terminal, go to the directory where you put the .yml file and run **`docker-compose up`**. This should start Clowder and its components.

3. Now you need to determine which IP address Docker is running on.
   a. Docker's Networking documentation shows how to use **`docker network ls`** and **`ifconfig`** to determine Docker's IP address.
   b. Older installations that use docker-machine may need to use **`docker-machine ip`**.

4. You should be able to now access Clowder at **`<dockerIP>:9000`**.
   You can also access the RabbitMQ management console at **`<dockerIP>:15672`**.

5. Finally, sign up for a local account in Clowder. Because we have not configured an email server, *no confirmation email will be sent* - however we can get the confirmation URL from the Clowder logs:
   a. **`docker ps`** to list running Docker containers
   b. **`docker logs <clowder container name>`** to see the logs for your Clowder container
   c. Look for a block of HTML in the log that is the body of the unsent email:



   d. Copy the link into your browser to activate your account. For example in the screenshot above you would visit: http://localhost:9000/clowder/signup/22af7d43-8260-4c6c-822a-22bca0cb8340

Install pyClowder 2

In addition to Clowder, another package that makes extractor development easier is pyClowder 2. *This package is not required* - extractors can be written in any language that RabbitMQ supports.

pyClowder 2 provides helpful Python wrapper functions for many standard Clowder API calls and communication with RabbitMQ using pika. If you want to write extractors in other languages, you'll need to implement that functionality on your own.

1. Install Git
2. **`git clone https://opensource.ncsa.illinois.edu/bitbucket/scm/cats/pyclowder2.git`**
3. **`python setup.py install`** from inside the new pyclowder2 directory

This will install pyClowder 2 on your system. In Python scripts you will refer to the package as:

- **`import pyclowder`**
- **`from pyclowder.extractors import Extractor`**
- **`import pyclowder.files`**

...etc.

Extractor basics

When certain events occur in Clowder, such as a new file being added to a dataset, messages are generated and sent to RabbitMQ. These messages describe the type of event, the ID of the file/dataset in question, the MIME type of the file, and other information.

Extractors are configured to listen to RabbitMQ for particular types of messages. For example, an extractor can listen for *any* file being added to a dataset, or for specifically *image files* to be added to a dataset. **Clowder event types** below describes some of the available messages. RabbitMQ knows how to route messages coming from Clowder to any extractors listening for messages with that signature, at which point the extractor can examine the message and decide whether to proceed in processing the file/dataset/etc.

Clowder event types

Extractors use the RabbitMQ message bus to communicate with Clowder instances. Queues are created for each extractor, and the queue bindings filter the types of Clowder event messages the extractor is notified about. The following non-exhaustive list of events exist in Clowder (messages begin with an asterisk because the exchange name is not required to be 'clowder'):

message type	trigger event	message payload	examples
***.file.#**	when any file is uploaded	• added file ID • added filename • destination dataset ID, if applicable	clowder.file.image.png clowder.file.text.csv clowder.file.application.json
***.file.image.#** ***.file.text.#** **...**	when any file of the given MIME type is uploaded (this is just a more specific matching)	• added file ID • added filename • destination dataset ID, if applicable	*see above*
***.dataset.file.added**	when a file is added to a dataset	• added file ID • dataset ID • full list of files in dataset	clowder.dataset.file.added
***.dataset.file.removed**	when a file is removed from a dataset	• removed file ID • dataset ID • full list of files in dataset	clowder.dataset.file.removed
***.metadata.added**	when metadata is added to a file or dataset	• file or dataset ID • the metadata that was added	clowder.metadata.added
***.metadata.removed**	when metadata is removed from a file or dataset	• file or dataset ID	clowder.metadata.removed

Typical extractor structure

In a pyClowder 2 context, extractor scripts will have 3 parts:

- **`main()`** will set up the connection with RabbitMQ and begin listening for messages. This typically will not change across extractors.
- **`check_message(parameters)`** receives the message from RabbitMQ and includes information about the message in the parameters argument. Extractors can count the number of files, look for particular file extensions, check metadata and so on.
  - Return 'true' or 'download' to download the file(s) process with process_message (below)
  - Return 'bypass' to process with process_message, but skip downloading the file automatically
  - Return 'false' or 'skip' to decline processing the file.
- **`process_message(parameters)`** receives the message and, if specified in check_message(), the file(s) themselves. Here the actual extractor code is called on the files. Outputs can also be uploaded back to Clowder as files and/or metadata, for example.

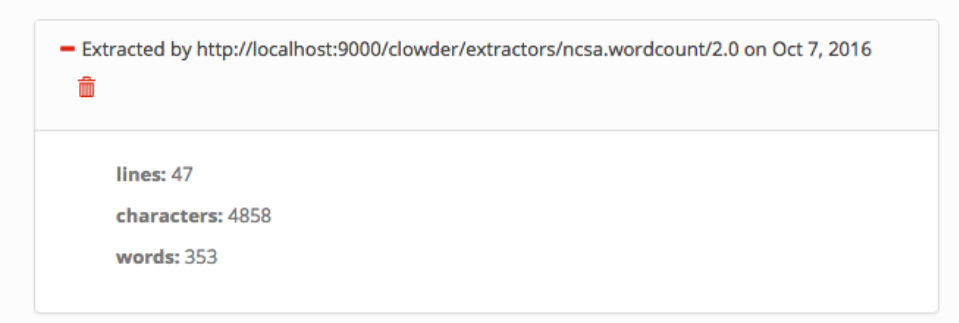
In addition to the extractor script itself:

- **`extractor_info.json`** contains some metadata about the extractor for registration and documentation.
- Many extractors will also include a Dockerfile for creating docker images of the extractor.

Running a sample extractor

Now that we have our necessary dependencies, we can try running a simple extractor to make sure we've installed things correctly. The **wordcount** extractor is included with pyClowder 2 and will add metadata to text files when they are uploaded to Clowder.

1. Go to **`/pyclowder2/sample-extractors/wordcount/`**
2. Run the extractor
   a. **`python wordcount.py`** is basic example
   b. If you're running Docker, you'll need to specify the correct RabbitMQ URL because Docker is not localhost:
   `python wordcount.py --rabbitmqURI amqp://guest:guest@<dockerIP>/%2f`
   c. You can use **`python wordcount.py -h`** to get other commandline options.
3. When the extractor reports "`Starting to listen for messages`" you are ready.
4. Upload a .txt file into Clowder
   a. Create > Datasets
   b. Enter a name for the dataset and click Create
   c. Select Files > Upload
5. Verify the extractor triggers and metadata is added to the file, e.g.:



You'll be able to see some activity in the console where you launched the extractor if done correctly.

Writing an extractor

Once you can run the sample extractor, you are ready to develop your own extractor. Much of this section will be specific to Python extractors using pyClowder 2, but the concepts apply to all extractors.

Extractor vs. Command-line - Calling Your Scripts

Often you will have a script that already performs the desired operations, perhaps by providing a directory of input and output files on the command line. The goal will be to call the correct parts of your existing script from within the **`process_message()`** function in your extractor, and to push the outputs from those methods back into Clowder.

Things to keep in mind:

- *Extractors do not necessarily operate on a single directory of files*. Extractors can be run on any machine, anywhere in the world, as long as they can communicate with RabbitMQ. To support this, pyClowder knows how to download all necessary data into a temporary directory. But this means the files may be in different locations when the extractor is called - your extractor will simply receive a list of paths to where your files are located on disk, even in a /tmp location. **If your script expects the files to be in one directory and you don't want to generalize it, you will need to check for that condition first and move the temporary files around if not.**
- *You are responsible for handling your script outputs.* If your extractor generates new files, you must upload them into Clowder before finishing. If your extractor generates metadata, you must attach them to the file or dataset of interest. pyClowder 2 does not handle outputs automatically, although we provide methods to make uploading easy.

common requirements

```
sudo -s
export RABBITMQ_URL="amqp://guest:guest@localhost:5672/%2F"
export EXTRACTORS_HOME="/home/clowder"

apt-get -y install git python-pip
pip install pika requests

cd ${EXTRACTORS_HOME}
git clone https://opensource.ncsa.illinois.edu/stash/scm/cats/pyclowder.git
chown -R clowder.users pyclowder
```

start extractors

```
cd /etc/init
for x in clowder-*.conf; do
  start `basename $x .conf`
done
```

Converting from pyClowder to pyClowder2

Given an extractor that is written to use pyClowder 1, the process of migrating to pyClowder 2 is fairly straightforward.

Key differences

- config.py is no longer used or needed.
  - Several of the common entries in config.py are accessible to all extractors via the basic Extractor class: https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/extractors.py#66 (here you can also see defaults)
  - You can implement your own command line arguments to include any special parameters in config.py. Another option is to read them from environment variables.
  - the 'messageType' parameter (telling what types of messages to listen for) will go into extractor_info.json and uses a a more MIME-like definition format.
- your extractor will now be an extension of pyClowder2's Extractor class, which contains many useful methods.
  - *init* is where you can define custom command line arguments beyond the standard ones.
  - check_message and process_message now get explicit parameters such as clowder host and secret key, rather than embedding them in a 'params' object. information about the entity in clowder that triggered the extraction (file, dataset, etc.) is in the 'resource' parameter. The old 'parameters' is kept for back compatibility, but is deprecated.
- As a result of config.py going away, you should provide parameters at runtime
  - python my_extractor.py --rabbitmqExchange="terra" --rabbitmqURI="rabbitmw.ncsa.illinois.edu/clowder-dev"
- new cleaner functions in pyClowder 2 for interacting with clowder, including packages for files, datasets, etc.
  - OLD - extractors.upload_file_to_dataset(outfile, parameters)
  - NEW - pyclowder.files.upload_to_dataset(connector, host, secret_key, resource['id'], outfile)

Migration steps

1. If there are parameters in config.py that don't use the default values in the link under Key differences, they should be listed as command line parameters in your new extractor class __init__ or simply coded into the script. It's possible to make the parameters read from environment variables as well.
   a. https://github.com/terraref/extractors-stereo-rgb/pull/3/files - in this example,
      i. https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-6be4f9dea03b90eac1407a1012cdf34eL42 is moved to
      ii. https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-f53b0090553dbecd9e15f5eb59549c00R32
   b. ...and below the self.parser.add_argument the input values can be adjusted before assinging to self.args (e.g. cast a string to an int):
      i. https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-f53b0090553dbecd9e15f5eb59549c00R48
   c. Add the messageType from config.py into extractor_info.json
      i. Before: https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-38d737ae3b969ee995bd1b34ebe93be4L25
      ii. After: https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-40099abc8fb726838bb4c7a44b8b5958R10
2. Move your extractor python functions into a new Extractor subclass
   a. https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-924a575b0595fcd52d5531433471b109R23 Here a new extractor class called StereoBin2JpgTiff is created.
   b. main() -> __init__(self) (but only for handling inputs)
   c. check_message() and process_message() must be named as such now, and receive explicit inputs:
      i. self, connector, host, secret_key, resource, parameters
      ii. typically, old references to parameters['xyz'] can be replaced either with resource['xyz'] or with secret_key, host, etc.

            iii. if you aren't sure when writing, you can use print(resource) in your extractor testing to see what fields are included.
3. Modify old extractor.method() to use the new pyclowder.files.method() or pyclowder.datasets.method()
    a. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/files.py
    b. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/datasets.py
    c. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/collections.py
    d. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/sections.py
    e. https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/pyclowder2/browse/pyclowder/utils.py
    f. more to come
4. finally, the call to main() is replaced with a simple instantiation of your extractor class.
    a. https://github.com/terraref/extractors-stereo-rgb/pull/3/files#diff-924a575b0595fcd52d5531433471b109R174
    b. extractor = StereoBin2JpgTiff(); extractor.start()

Registering an Extractor

If you create an extractor and want to register it for widespread use Clowder has a catalog of extractors. To let the community know about your extractor you may submit it to the catalog (registration required) for publication alongside the extractors created by the Clowder developers, and other Clowder community developers.

1. If you haven't registered you may do so at the registration page (to be linked soon). Registration will need to be approved, so this will not be an instantaneous process. Once approved you will be able to sign in, and submit extractors.
2. Sign in to the catalog at the sign in page.
3. Click the Contribute link at the top, or visit the contribution page.
4. Paste your extractor info into the text box on the contribution page. The extractor info has all the information that we use in the catalog, so having a fully detailed extractor info is important for the catalog.
5. Select Extractor from the radio buttons under the text box (Converter is for converting one file type to another, and there may be more options added for other tools at a later date).
6. Submit the extractor.
7. On the home page you may not see your extractor immediately. Once submitted it must be approved by one of the Clowder development team to show up on the home page. As long as your links to where to get the extractor check out you will likely be approved, and if there are correctable issues we may contact you.