Tracking extractor job status

Motivation

We currently track extractor messages by resource - for a dataset or file, the user can view a list of Extractor events that were associated with that file.

The list of events is further categorized by extractor, but this can still be inadequate for tracking multiple jobs running on the same extractor.

The problem is exacerbated exponentially if multiple replicas or threads are running for a particular extractor - this means that 2 jobs can simultaneously be sending updates for the same extractor + file or dataset combination, and makes it impossible to determine which jobs sent which updates.

In addition to resolving the problem stated above, expanding extractor job tracking would also allow extractor developers to better debug and analyze their extractor jobs, or to determine if there are performance enhancements that could be made.

Current Behavior

The API defines the **ExtractorInfo** and **Extraction** models, and offers an **Extractions** API for fetching and updating these events for display. The **MongoDBExtractorService** can be used internally to create and interact with those models within MongoDB.

On Startup

The Clowder API for file uploads includes optional calls to the RabbitMqPlugin. This plugin must be enabled to use extractors.

If the RabbitMQ plugin is enabled, Clowder subscribes to RabbitMQ to receive heartbeat messages from each extractor. Clowder subscribes to these messages to determine if an extractor is still online and functioning. (This is the **ExtractorInfo** model.)

pyClowder, as the name implies, is a framework for writing Python-based extractors for Clowder. It includes, among other utilities, a client for subscribing to the appropriate queues in RabbitMQ. pyClowder knows how to automatically send back heartbeat signals and status updates to Clowder.

When a pyClowder-based extractor starts up, it begins sending heartbeat messages to RabbitMQ indicating the status of the extractor. Clowder subscribes to these messages to determine if an extractor is still online and functioning. (This is the **ExtractorInfo** model.)

At this time, the extractor also creates and/or subscribes to a queue unique to its extractor identifier, to which Clowder will send new upload events to be processed.

On Upload

For each new file upload, Clowder will push an event into the appropriate extractor queue(s) in RabbitMQ based on the types/rules defined in the ExtractorInfo.

If an extractor is idle and sees a new message in the queue it is watching, it will grab the message and start processing.

Whenever a stdout/stderr log event happens or if the Extraction status changes (e.g. SUBMITTED PROCESSED or PROCESSED DONE), pyClowder will send an update back to Clowder on the "reply queue".

When a reply comes back on the "reply queue" containing a file + extractor combo, Clowder creates a new **Extraction** in the database housing this message.

UI

The UIs for viewing a File or Dataset offer an Extractions tab, which simply lists out all **Extraction** objects associated with the file or dataset and groups by extractor type.

Changes Proposed

pyClowder

pyClowder may need minor updates to send more information regarding a specific job. Alongside the user, resource (e.g. dataset or file), and extractor name, pyClowder should also attach a unique identifier (e.g. UUID) to related job messages. That is, when an extractor begins working on a new item from the queue, it should assign a unique identifier to that workload that is unique to each run of the desired extractor.

This way, we can easily create a UI that can filter and sort through these events without making major modifications to the UI and without taking a large performance hit in the frontend.

There is still an open question of who is responsible for creating this unique ID. On the one hand it should be up to Clowder to create and manage these IDs, as it is managing the larger concept of a "Job" which spans multiple status updates. On the other hand, will this account for failover? What if an extractor fails midjob? Is it ok to start this job over with the same ID, or would this require generating an entirely new ID?

API

The API changes proposed should be fairly minimal, as we are simply extending the existing Extraction API to account for the new identifier that will need to be added to pyClowder.

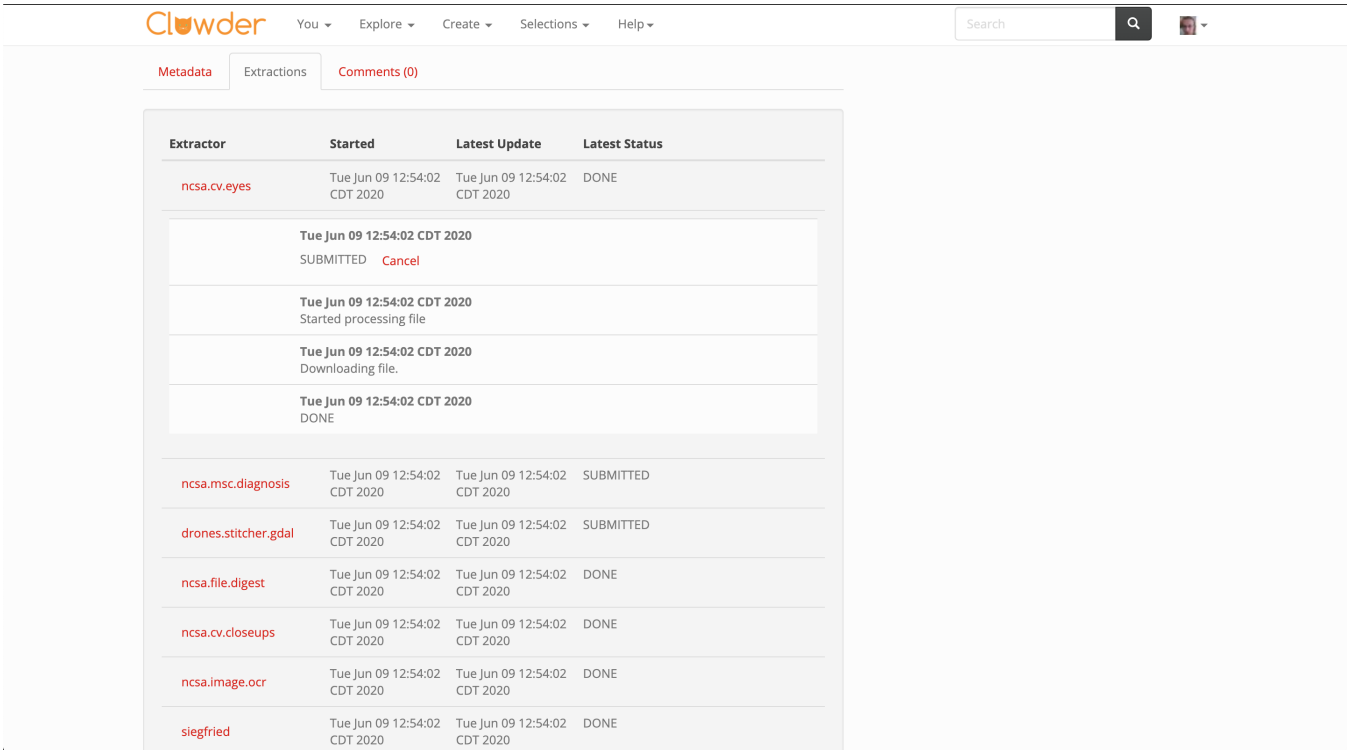
NOTE: **Extraction** already has a field named **id**, which is unique per-status update. We may want to create a new field (named **job_id**?) instead of changing existing behavior and maybe causing compatibility issues.

Further changes would likely break the existing implementation of Extractors/Extractions, and should be considered thoroughly before execution.

UI

The UI takes on a few changes here, adding further grouping/categorization based on the new job ID.

Displaying extractor events currently looks as follows:



We offer a tab that lists all extractors that have run on the file or dataset.

The user can click an extractor to expand a subsection containing individual events from that extractor.

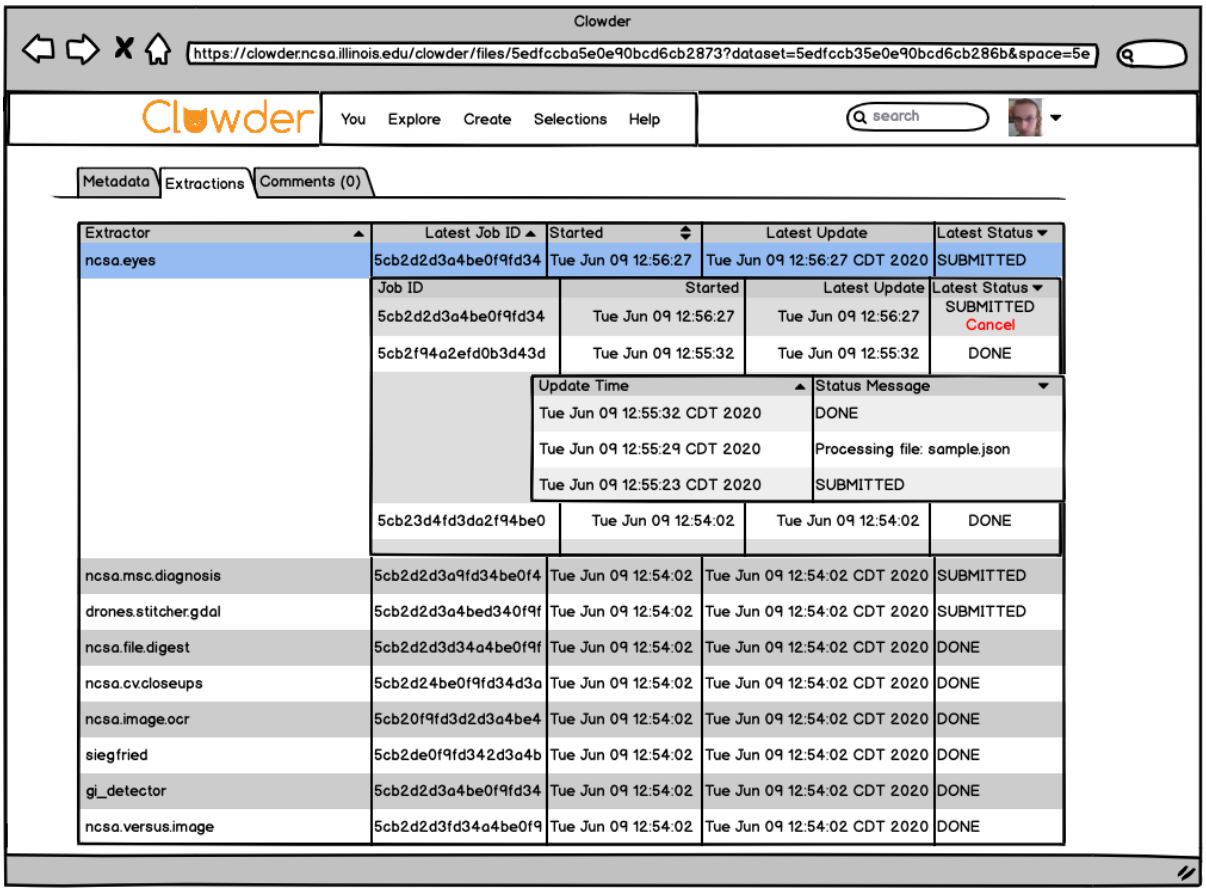
We propose altering the UI with the following changes:

1. Categorize further based on the unique identifier for each job - this allows user
2. For finished extractor jobs (?), show the elapsed time it took to complete the job

Nice to have:

1. Display the user that initiated the job - for new uploads, should we show the user who uploaded the file?

The final product might look something like this:



Open Questions

- What do we do about past extractions/jobs that are missing this identifier? Is it worth preserving the existing display in the UI for this purpose?
- Extraction already has an **id** field that is unique per-status update - would adding a **job_id** field work in a way that keeps this data/view backward-compatible as possible?
- Should Clowder or pyClowder create this unique identifier? What about cases where an extractor dies mid-job? Can/should another extractor pick this up with the same ID, or should this generate a new ID instead?