Centralized Reporting & Monitoring

Overview

There are several parallel efforts to capture information about Clowder metrics:

- User activity (backend)
    - resources created or uploaded
    - bytes added or removed
    - extractors triggered & runtime
- User activity (frontend)
    - page views
    - ...
- System health
    - Response time of ping Clowder website.
    - Response time of download Clowder website and bytes of homepage.
    - Uptime of Clowder website
    - ...

The goal is to minimize number of moving parts to capture and store this data. Below is summary of our discussion from 12/7.

Recon / Planning Phase

RabbitMQ Queue & Flask API

Use queue to store data points. Not an extractor queue, but a special new system queue.

- Clowder can write messages directly to RabbitMQ
- Lightweight Flask API we run in a python container that also connects to RabbitMQ
    - Other code can post datapoints to this API, that get forwarded to RabbitMQ

Flask API design notes - ideally these endpoints also match the calls on the new backend SinkService:

- Seems lightweight enough for possibly a single generic endpoint to enqueue an item
    - Can expand to an endpoint-per-event-type as API evolves or as needs grow
    - Use Swagger from the start, as best practice... should make it easier to alter/scale/sync API server/clients when necessary
- Requires authentication via API key or some other mechanism to prevent spam or potentially malicious fake messages
    - Cannot fetch / auth using Clowder (since this is for handling the case when Clowder is down)

Internal Clowder events service

For the user activity (Max's reporting part and Mike's Clickstream stuff basically) we can call an internal RabbitMQ service for the events that we want to capture, to generate datapoints.

Current (frontend) tracking:

- Allows for configuration of Amplitude API key
- If configured, tracking snippet added to every view (via `index.html`)
- If configured, events tracked in the Javascript via `amplitude.logEvent()`
- Events tracked:
    - Resource views (files, datasets, collections)
    - Files / datasets submitted to extractor
    - File uploads

Proposed changes:

- Allow for configuration of Amplitude API key (no change)
- If configured, tracking snippet added to every view (no change)

- For the tracked events above, call the new backend SinkService, which will check for configured integrations with Amplitude/Google Analytics/etc and delegate appropriately:
    - This will be a new piece of code that will submit to our special RabbitMQ queue/exchange
    - If Amplitude is configured, also send to Amplitude via the REST API
    - If GA is configured, also send to Google Analytics (NOTE: may be difficult or impossible - see https://stackoverflow.com/questions/15530487/restful-api-and-google-analytics)
- Bonus points: add a backend action that automatically tracks API calls and sends to the SinkService

Clowder health monitor(s)

Bing's external monitor can't call Clowder, because it has to operate even when Clowder is down. Instead the monitors in different regions can collect and post their datapoints to the Flask API, which can go around Clowder into RabbitMQ directly.

We run a service as docker container periodically fetch the statistics data of Clowder service, e.g., the uptime, response time and a number of active connections to Clowder, etc. And those data will be stored in the backend services e.g., influxdata (this will need the extra endpoints of service), and grafana will retrieve those data and render them on the grafana website for the visualization.

The uptime of Clowder website can ensure we understand the liveness of Clowder service and this metric will be collected by sending ping to the target Clowder website with a certain timeout.

Response time: meanwhile, we collect the statistics of the response time of the ping command. And the elapsed time of downloading Clowder homepage.

The number of connections: It would be good to see how many connections to Clowder website. we can measure the number of connections within a period of time. We would analyze the NGINX log to get those information.

Database monitor(s)

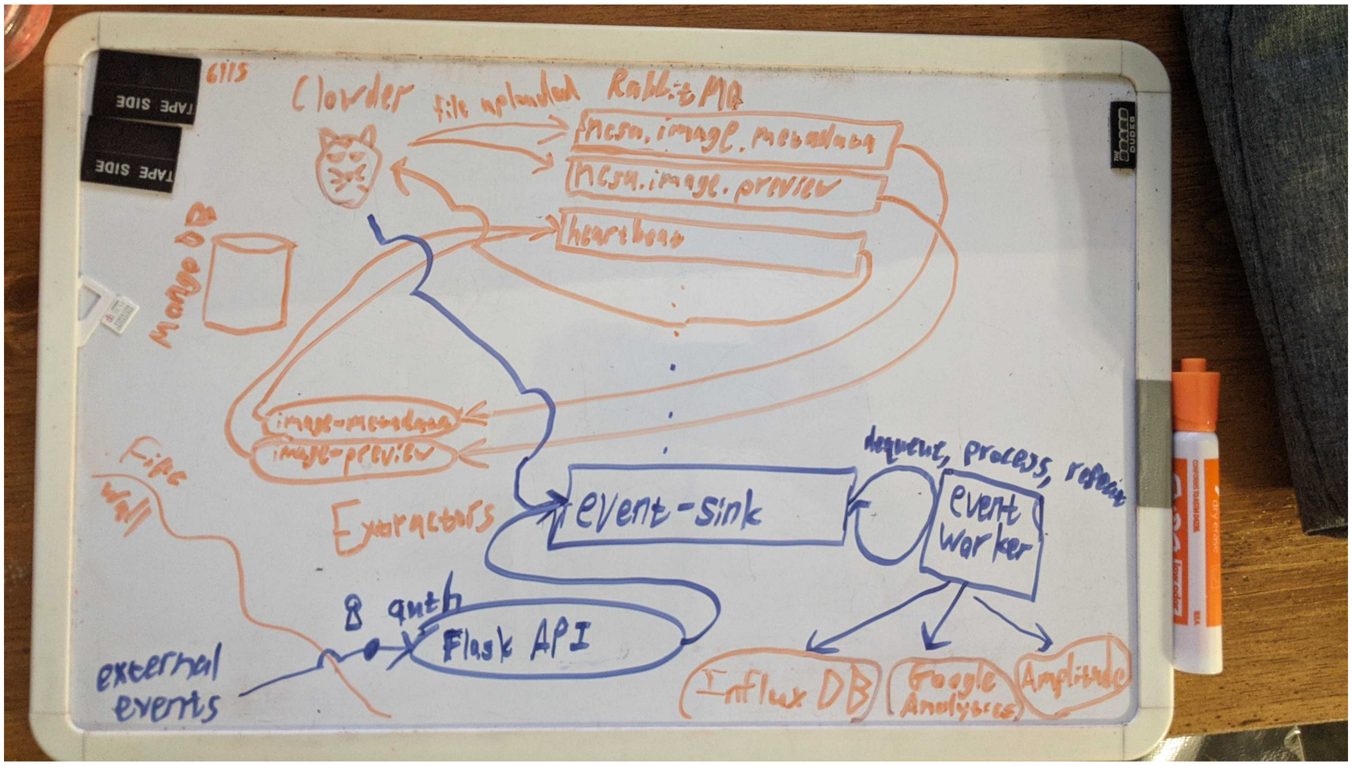
Finally, we need a service to actually pull the messages from RabbitMQ and write them into a database, whether that is MongoDB or InfluxDB or whatever. Maybe these could register with Clowder like extractors even, so that they each get a separate queue and multiples can log to different destinations at once.

Database design

Let's consider some different types of events. Assume **user** and **timestamp** for all data captured too.

component	event type	data captured	notes
storage	file uploaded file deleted	fileid, datasetid, spaceid, bytes	
extractions	extraction event	message, type (queued or working)	do we care about data traffic downloaded to the extractor containers?
traffic	page views resource downloads	url, resourceid bytes	do we care about every page view? this is currently tracking which resources are being viewed but without the full url
health	ping update	response time, queue length, other?	

The Full Picture



Legend

- Orange: existing technology
- Blue: likely a new piece that needs to be written

Refactoring RabbitMQ

Clowder currently has `RabbitMqPlugin.scala` which contains logic for submitting files and datasets to the appropriate queues in RabbitMQ for scheduling extractions. A client (e.g. pyClowder) can then subscribe to the queue(s) to systematically work through related extraction jobs in a scalable fashion.

We want to preserve the above relationship, while making the underlying code slightly more generalized such that we can use it send an arbitrary message to an arbitrary queue in the configured RabbitMQ instance.

This new generalized service (name TBD, but for now I will call this `RabbitMqService.scala`) can be used by the existing `RabbitMqPlugin.scala` (or optionally a simplified rewrite) to submit jobs to extractors in the same way. The new `RabbitMqService.scala` can also be utilized by new code to send events to the event sink system.

The Event Sink System

As described above, the "event sink" is simply a special exchange in RabbitMQ that is configured to fanout to multiple queues. This way we can siphon different types of events into specific queues based on the intended target.

The plan is to create a set of very thin AMQP event worker clients, which will start up and subscribe to the queue(s) to systematically work through delivering the metrics/event payloads to interested parties.

Emergency Flask API

In order to track events surrounding Clowder downtime, we should offer an alternative/emergency API (via Flask, or similar). This way if Clowder is down or has an outage, we can still continue to collect metrics data by using the exposed Flask API endpoint(s) to submit to the event sink exchange in RabbitMQ, as described above.

The Flask API only needs to be instrumented to collect/submit events/metrics related to Clowder uptime/downtime, maintenance, and outages.

The Flask API only needs to submit directly to RabbitMQ, and does not need to integrate directly with Clowder itself.

Event Workers

MongoDB Worker

A debug worker that will simply echo the given message into a new collection in MongoDB.

This can be used for debugging purposes and to provide a simple pattern/template for creating future event workers.

InfluxDB Worker

Our centralized receiver that will receive all events of all types and parse them into InfluxDB.

The ultimate goal is to use Grafana/InfluxDB as a centralized place from which to view/store all metrics/event data. Grafana then allows us to build graphs, dashboards, and alerts based off of these metrics

Experimental Workers

The tricky part here is that typically these types of analytics/clickstream services tend to integrate directly with the user's browser via JavaScript, and **not** via a backend service. We have no guarantee that these proposed workers will function as described, but we are hopeful that the resources offered by the associated SDKs will allow for them to fit into our solution.

Amplitude Worker

Amplitude supposedly offers a REST API for submitting events directly from the server-side:

- https://help.amplitude.com/hc/en-us/articles/115000959052-For-Developers-Getting-Started#h_c5787e3e-5f2a-450e-b2b1-d51a4859e903

If possible, this will send UI-driven or event-related metrics to Amplitude to track user analytics:

- Demographic (e.g. Age/Location/Language)
- Device Info (e.g. Hardware/OS/Browser)
- User Interactions (e.g. Page Views/Events)

Google Analytics Worker

Google Analytics supposedly once offered an API for submitting events directly from the server-side:

- https://cloud.google.com/appengine/docs/standard/php7/integrating-with-analytics#server-side_analytics_collection

Unfortunately, briefly attempting to follow the links above yielded some 404 errors, so some exploratory work will be necessary here.

If possible, this will send UI-driven or event-related metrics to Google Analytics to track user analytics:

- Demographic (e.g. Age/Location/Language)
- Device Info (e.g. Hardware/OS/Browser)
- User Interactions (e.g. Page Views/Events)