CrisisNET Architecture

The system consists of three primary components. Sucka, which retrieves data from the Internet or local filesystem, Grimlock, a processing pipeline which transforms/augments documents, and CrisisNET, a REST API.

Sucka

The Sucka component understands how to retrieve data based on Source documents stored in the application database. These documents define where the source's information is located (API endpoints, URIs, etc), authentication credentials, how frequently information should be retrieved from the source, etc. The source document is interpreted by a sucka module (example) that performs the retrieval and transforms the data into the structure expected throughout the system, an Item.

Authors of individual suckas have autonomy to add whatever metadata may be applicable to the ingested data during the transformation step. For example tags/categorization or geospatial information. However, sucka modules should do the bare minimum amount of transformation, using only the third-party data when performing the transformation (so avoid making external requests to geocode an address, for example). Augmentation requiring more complex processing is performed in other parts of the system.

After data has been transformed it is stored in the application database and is therefore immediately available to consumers of the CrisisNET API. Once an Item is stored, a task is published to the system's central job queue denoting that the Item is ready for additional processing.

Sucka is written in Node.js (v0.10.26), and for the time being suckas must also be written as Node.js modules. Retrieval tasks are scheduled in a redis datastore using the kue framework. For example a task can be run every minute, hour, day, etc.

Grimlock

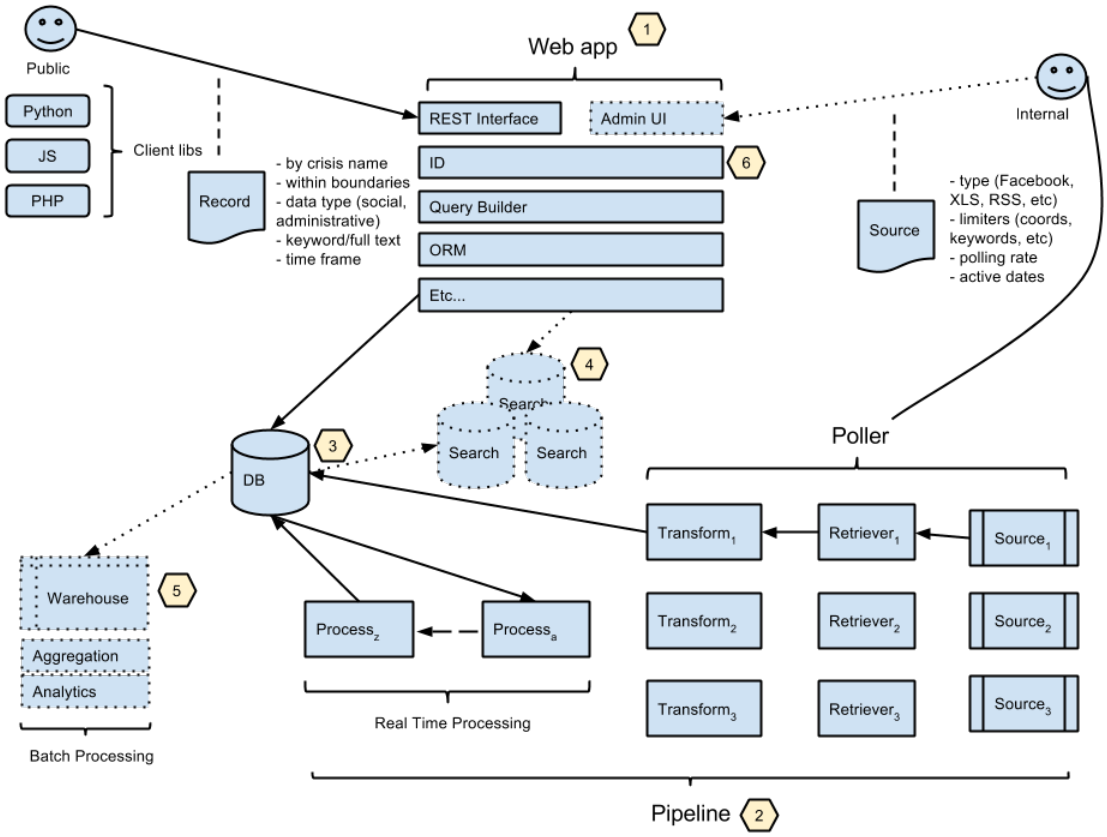
Grimlock is a transformation/processing pipeline. It reads jobs from a redis queue, retrieves Item documents from the application datastore and passes those documents through a series of tasks to add metadata like geospatial information, categorization/classification, etc. Any Python module can be a task, assuming it contains a run method that accepts and returns an Item document. Here's a simple example.

The "pipeline" is essentially a composition of its functions (in reverse order), so if you have tasks A, B and C, the pipeline is C(B(A())) – the last task accepts the return value of the second-to-last task, which accepts the value of the third-to-last task, and so on. It's important to keep this in mind for two reasons: 1) the last task should save the augmented Item back to the datastore, and 2) downstream tasks can build upon the work of tasks executed earlier in the pipeline. So, make tasks as simple and discrete as possible – each task has one responsibility/makes a single transformation.

CrisisNET

The main CrisisNET application handles user registration/authentication, and exposes a REST API for retrieving Item based on query parameters. It is written in Node.js and retrieves Items from MongoDB.

The diagram below offers a high-level overview of the system. Keep in mind it includes a number of planned components that aren't required for basic functionality (and subsequently don't exist yet).


Public
Web app
1
Python
JS
PHP
Client libs
REST Interface
Admin UI
Internal
Record
- by crisis name
- within boundaries
- data type (social, administrative)
- keyword/full text
- time frame
ID
6
Query Builder
ORM
Etc...
Source
- type (Facebook, XLS, RSS, etc)
- limiters (coords, keywords, etc)
- polling rate
- active dates
4
Search
Search
Search
3
DB
Poller
Transform1
Retriever1
Source1
Transform2
Retriever2
Source2
Transform3
Retriever3
Source3
Warehouse
5
Aggregation
Analytics
Batch Processing
Processz
Processa
Real Time Processing
Pipeline
2