Beeg Data - Beeg Meme

Kacper Grzymkowski, Jakub Fołtyn, Mikołaj Malec, Illia Tesliuk

Introduction

This document represents the Beeg Meme project state at the stage of the 2nd milestone delivery in the course of Big Data Analytics. In this document, we will describe all the used data sources in greater detail, present the strategy for data acquisition and transformation, as well as present plans for future data processing and machine learning tasks.

Changes to the 1st document

In this section we will present the changes made to the project with regard to the 1st document.
The changes made:

- Due to the lack of API documentation and severe problems with the Python wrapper library, we have decided to abandon the Ifunny data source, and instead use Tumblr as our new data source.
- We have decided to abandon the KnowYourMeme data source for the time being, as it too was problematic for accessing. It is highly probable that this source is going to be added later on in the project.
- There have been some changes applied to the solution architecture – for instance, we have replaced Google Cloud PubSub with Apache Kafka, as the former did not cooperate with Spark streaming. More details are in the Architecture diagram section.

Data Sources – detailed description

Reddit.com

Data from the Reddit API is acquired through an official API channel, which is supported by the PRAW (The Python Reddit API Wrapper) library. The script which downloads post metadata in JSON format from the API is invoked once every few seconds. The API returns a lot of metadata, a lot of which is technical in nature and matters little for the project. However, all the data returned from the API is passed to further processing in NiFi, in particular to be saved in the master log. A trimmed down (in terms of fields) message will be passed to Apache Kafka for further processing.
Important features of the post (submission, new entry) include:

- Post url (Usually a direct link to an image/video)
- Post title
- Author name
- Post scores (ups/downs/score, however reddit deprecated the downs field a long time ago)
- Number of comments

- Self text (Usually treated as an extension to post title or the body of a “self” post, which do not contain images)
- Awards (paid interactions with the post)
- Date of creation (in seconds since UNIX epoch, UTC timezone)

Due to the relatively low amount of activity in the /r/memes subreddit (one post every few minutes), we will consider expanding the script to use other subreddits, to acquire larger amounts of data.

tumblr.com

Data from the Tumblr API is acquired through an official API channel, which is supported by the official Python API Wrapper pytumblr. The Python script receives information about a predefined number of posts from a particular tag category in a JSON format. Tumblr posts from the #memes category can be represented as images, videos, or texts. Since our models operate only on images, the script filters the received data by the availability of image fields. Each Tumblr image post contains an XML string with URIs of the meme image in different resolutions. The script uses an xml module to parse the string and retrieve a link to the largest resolution image. To compensate for non-image posts filtered out previously, the script sends a new API request with a condition for posts to be older than the previous batch. This way, the predefined number of records is obtained. The maximal number of posts which can be received from a single tag category is limited to 20. To obtain the larger number of posts, the script repeats requests following an above-mentioned approach. Each meme post contains a huge set of meta-data, however, only the following fields can be used in our project:

- blog (author) name
- posted date
- total number of interactions, which is a sum of:
    - number of likes
    - number of replies
    - number of reposts
- description (often empty)
- link to the image

Due to the relatively low amount of activity in the #memes tag category(one post every 10-15 minutes), we will consider expanding the script to use other tags, to acquire larger amounts of data.

Imgur.com

Data is acquired through the use of a dedicated Imgur API Python library – **imgur-python**. This library enables users to both acquire posts from the website, as well as post their own images, comments, and also browse other users’ accounts and comment sections.
Imgur API enables users to get posts from multiple post flows: **top** posts, meaning the posts with highest scores (in the last week, day etc.), **hot** posts, meaning posts that are quickly gaining attention, “trending”, and finally a category called **added by users**, which simply means newest posts.
In general, new posts come in two different variations: a single image, or a gallery of multiple images. The script that we have prepared “unpacks” the gallery, so that each image in the gallery is treated as a single image. What is more, we omit the gif-type images (so animated

images) in our data acquisition. These are the reasons why sometimes when passing the parameter **n** (which corresponds to the number of posts that we want to retrieve), the number of actually retrieved images may vary.
Due to the internal limitations, the API always returns 60 posts. We are acquiring posts from the “added by users” category, as this is the most frequent category, so the category in which we can obtain new images the fastest. API returns information about posts in the form of JSON files. Those files include various information about the post, some of the most important ones are:

- poster name and id
- posted date
- number of comments
- number of ups/downs
- number of views
- post topic (often empty)
- post title
- post description (often empty)
- link to the image (may be gif)
- Image width and height
- author id and name

When investigating, we have noticed that the API returns new posts with a slight delay (meaning that the “newest” posts returned by API have actually been posted around 15 minutes ago). In general, it is hard to measure the number of new posts, as it varies with regard to e.g. the time of the day, but according to our tests, on average around 1-2 new posts are added every minute.

Data acquisition strategy

We acquire data from all the sources with the use of Python scripts. We have created three scripts in total, each of them uses their data source’s respective API library:

- **praw** – for accessing data from *reddit* data source
- **imgur-python** – for accessing data from *imgur* data source
- **pytumblr** – for accessing data from *tumblr* data source

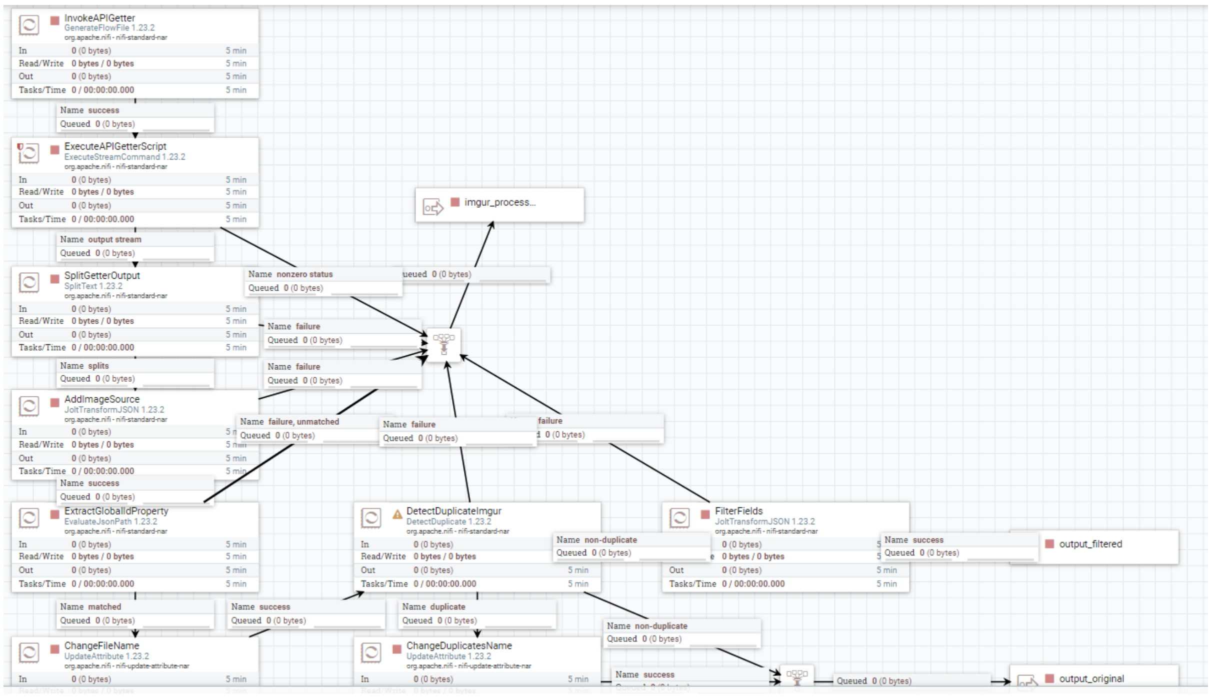
The above libraries have been described in greater detail in their respective data source’s sections.
In addition to source-specific libraries, we are also using some more “common” Python libraries, such as:

- **datetime** – for retrieving dates in UTC ISO format
- **json** – for converting printed data into json objects
- **requests** – for retrieving website data

Each data acquisition script is invoked by the nifi *ExecuteStreamCommand* processor. It is achieved by sending an empty flow file to this processor every 10 seconds. Each time, scripts prints out **n** latest images from its source (where **n** is a parameter). All the images are then further processed using nifi.

Data transformation and storage

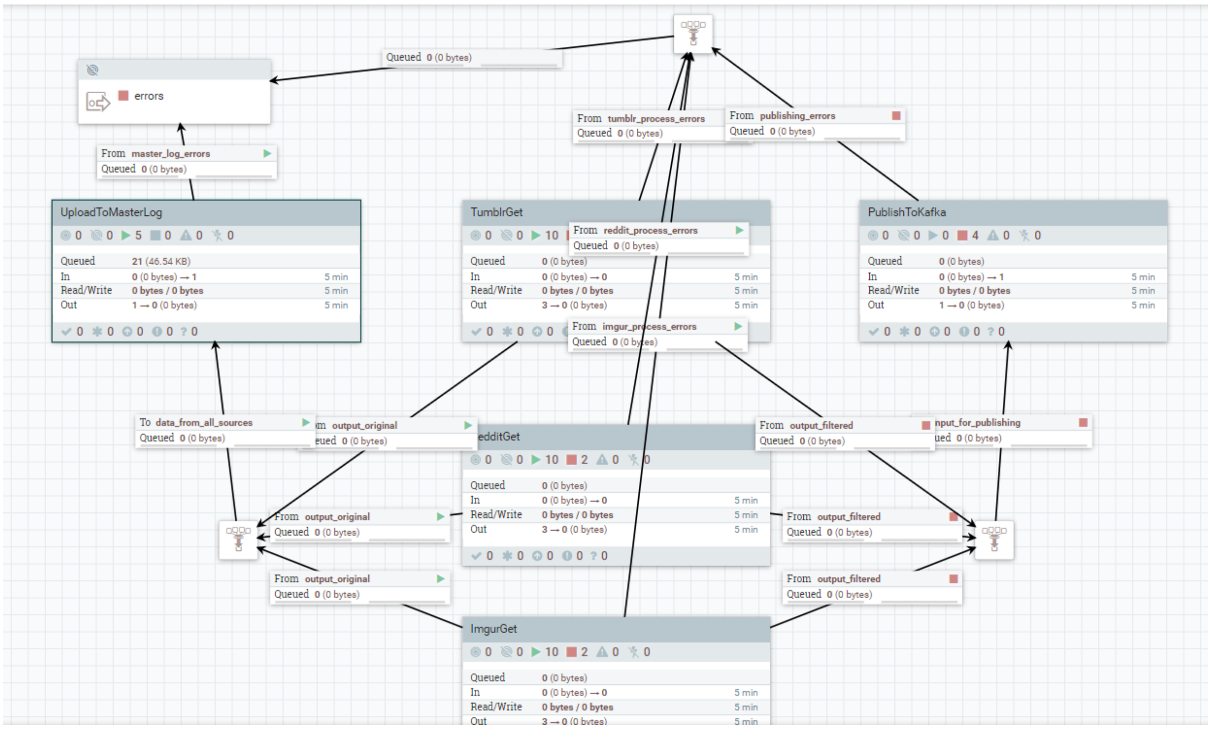
As previously described, data is acquired via Python scripts and transformed using nifi. In general, there are 3 distinct source flows – one for reddit data, one for tumblr data, and one for imgur data. Inside these flows, the operations performed are quite similar.



On the image we can see an example flow for imgur data source. In general, the API invoking script is started by passing an empty flowfile into an ExecuteStreamCommand processor. This processor then produces a single output flowfile with a number of concatenated JSON files. Those files are then split, so that each object is a single flow file, the source of the image is added to the file (in case of of reddit in tumblr, this is also the place in which global_id is added, which is of the form [image_id]-[source]), and then the filename is changed to the global id of the file. The files are then checked for duplication, and non-duplicates are passed to a filtering processor, which chooses only the fields in the JSON file, which are consistent across all data sources. They are:

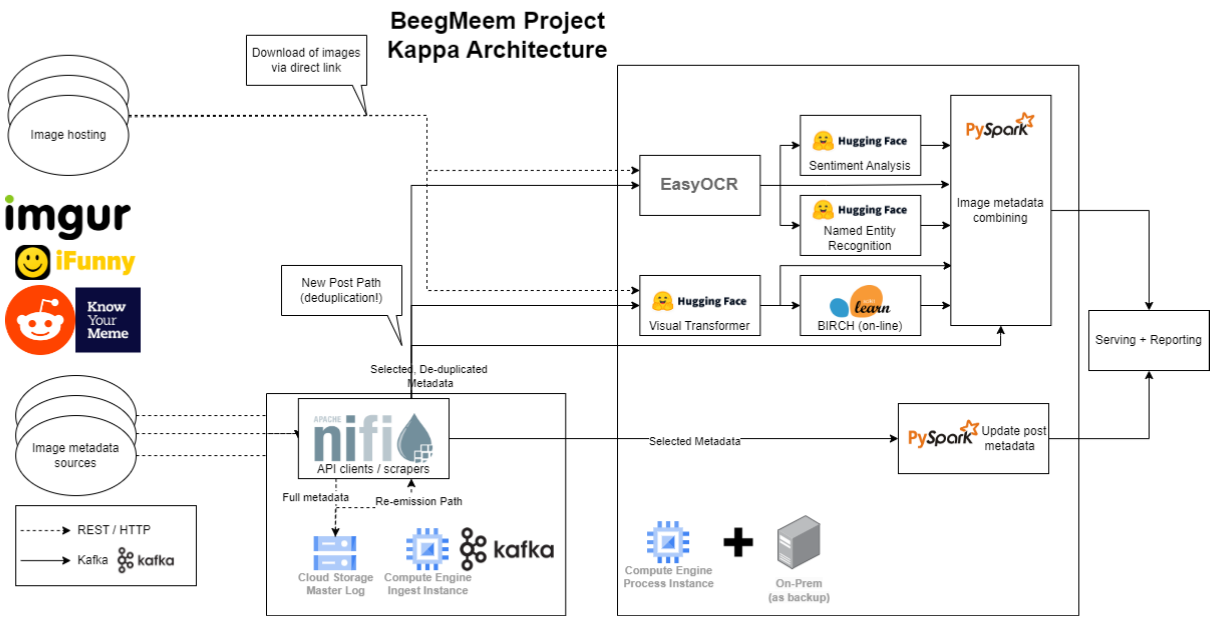
- Author
- Image description
- Date of image creation
- Score (number of upvotes-downvotes)
- Image source
- Image global id

Duplicated images have their filename modified (a timestamp is added), and are passed with the non-duplicated ones in their original form to be posted to master log. Master log is a cloud storage bucket. The files are merged (a 100 files at once, it is possible that later on in the project we will increase this number to a 1000) into a TAR file and then uploaded into the bucket.
On the other hand, the deduplicated and filtered files are sent to be published by a KafkaPublisher processor. Currently, we have created a temporary Kafka topic, as no process subscribes to it yet.



The above image shows the general nifi data flow.

Architecture diagram


BeegMeem Project
Kappa Architecture
Download of images via direct link
Image hosting
imgur
iFunny
Know Your Meme
New Post Path (deduplication!)
Selected, De-duplicated Metadata
Image metadata sources
API clients / scrapers
Full metadata
Re-emission Path
Cloud Storage Master Log
Compute Engine Ingest Instance
kafka
REST / HTTP
Kafka
EasyOCR
Hugging Face Sentiment Analysis
Hugging Face Named Entity Recognition
Hugging Face Visual Transformer
BIRCH (on-line)
PySpark Image metadata combining
Serving + Reporting
Selected Metadata
PySpark Update post metadata
Compute Engine Process Instance
On-Prem (as backup)

Planned ML tasks

We plan to use various models for data mining of the meme images. Firstly, we plan to perform **Optical Character Recognition** using the EasyOCR framework. OCR is the process that recognizes text inside an image and converts it into a machine-readable text

format. Most memes contain short text therefore with the help of EasyOCR module we would be able to extract this data from the images for further processing, namely Sentiment Analysis and Named Entity Recognition.
Sentiment Analysis is a natural language processing technique that identifies the polarity of a given text and labels it into positive, negative and neutral. We are planning to use a pretrained sentiment-analysis pipeline from HuggingFace platform. The results of this operation will help us to understand the sentiment of the meme's image text and can be used in further data analysis, e.g. for finding an average sentiment for a particular person or event over a specific period of time.
Usually, sports events like football matches generate a lot of memes over a short period of time, so the gathered and preprocessed data can be used for determining the predominant sentiment for a particular player. Information about this person or an event can be retrieved with the help of **Named Entity Recognition** - component of natural language processing that identifies predefined categories of objects in a body of text. We are planning solve this task with the help of a pretrained model from the HuggingFace platform such as fine-tuned bert-base-ner BERT model that achieves state-of-the-art performance for NER task.
The results of NER, sentiment analysis and optical character recognition tasks will be saved in Spark alongside the meme image and its metadata.
Finally, we are going to perform a **clustering** task on the input meme images. This task consists of first, passing a batch of memes through the HuggingFace-based pretrained Visual Transformer (ViT) and applying the BIRCH clustering algorithm from the scikit-learn library. BIRCH is an online clustering method, which makes it a natural choice for the tasks with streaming data. Clustering results will be saved together with the meme image and the outputs of above-mentioned NLP tasks.

Stream/batch processing

Stream processing will include the above ML tasks, as well as combining the resulting streams. Additionally, windowing and aggregations will be performed for the serving layer, where necessary.

We will not have specialized batch processing in our system, we will use the Kappa architecture to treat batch data as streams. This will be done via NiFi that will replay a part of the master log to the rest of the system.

Serving layer plan

Storage for processes data:
Processed data, including OCR results, sentiment scores, NER outputs and clustering results will be stored in JSON format within the MongoDB environment. It will anable fast write to the database and fast analytics from the database.

Data will be partitioned by day.

Visualization tools:

We will use PowerBI Desktop rapport and connect to database via ODBC data source.

Work division

Jakub Fołtyn:

- Nifi data acquisition, transformation
- Master log set-up
- Imgur data acquisition
- Nifi testing

Kacper Grzymkowski:

- Infrastructure management
- Kafka set-up
- Reddit data acquisition

Illia Tesliuk:

- Tumblr data acquisition
- Pubsub/Pubsub lite/Kafka feasibility assessment
- Description of ML tasks

Mikołaj Malec

- KnowYourMeme
- Serving layer
- PowerBI rapport