sfm Documentation

Release 1.6.0

The George Washington University Libraries

Mar 14, 2017

User Documentation

Social Feed Manager is open source software for libraries, archives, cultural heritage institutions and research organizations. It empowers those communities' researchers, faculty, students, and archivists to define and create collections of data from social media platforms. Social Feed Manager will harvest from Twitter, Tumblr, Flickr, and Sina Weibo and is extensible for other platforms. In addition to collecting data from those platforms' APIs, it will collect linked web pages and media.

This site provides documentation for installation and usage of SFM. See the Social Feed Manager project site for full information about the project's objectives, roadmap, and updates.

CHAPTER 1

Quick Start Guide

This quick start guide describes how you can start using Social Feed Manager to select, harvest, explore, export, process and analyze social media data. This covers just the basics of using the software; technical information about installing and administering SFM can be found in the *Admin and Technical Documentation*.

Prerequisites

SFM in operation

This quick start guide assumes SFM is already set up and running. For details about installing and administering SFM, see *Admin and Technical Documentation*.

An SFM account

You can sign up for an account by clicking the *Sign Up* link from within SFM.

If you'd like to set up shared collecting at your institution, you'll need to have your systems administrator set up groups in SFM.

API credentials

You will need API credentials for each of the social media platforms from which you want to collect. This is more than the Twitter/Flickr/Weibo account that you may already have. To get API credentials:

- Request credentials from the social media platform and enter them into Credentials section. The *API Credentials* page provides instructions for each platform.
- For some social media platforms, your administrator may have enabled an option that will allow you to connect your account without leaving SFM. With your permission, SFM will get credentials on your behalf. Click *Credentials* and then *Connect [Twitter, Tumblr, or Weibo] Account*.

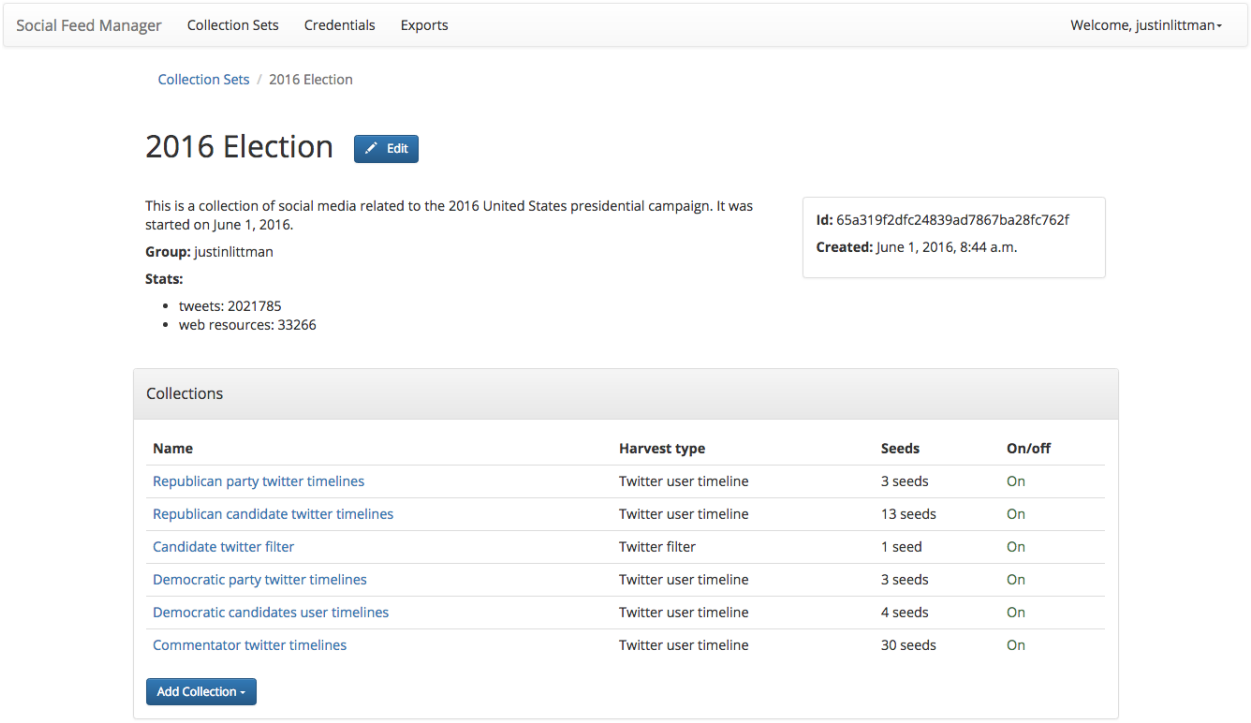
- If you are part of a group, you'll be able to use the credentials already provided by another member of the group.

Setting up collections

Hopefully you've considered what you want to use SFM to collect: which social media accounts, which queries/hashtags/searches/etc., and on which platform(s). You may also have learned a bit about the social media platforms' APIs and best practices for collecting from social media APIs. Now you'd like to set up your collections in SFM.

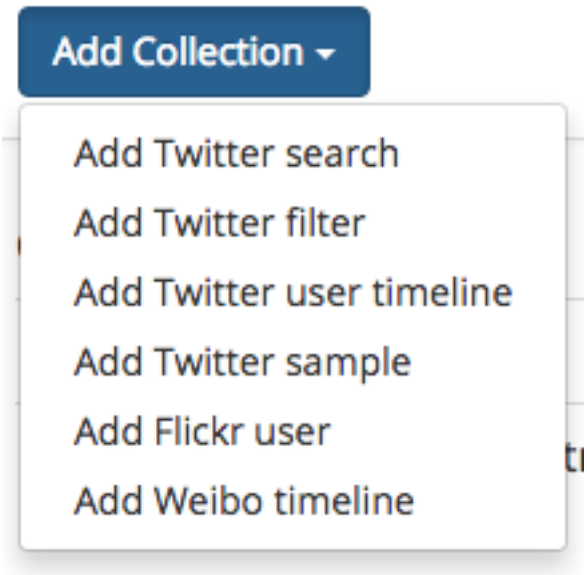
Create a collection set

At the top of the page, go to *Collection Sets* and click the *Add Collection Set* button. A collection set is just a group of collections around a particular topic or theme. For example, you might set up a "2016 U.S. Elections" collection set.



Create a collection

On the collection set detail page, under *Collections* click the *Add Collection* button and select a type.



Collection types differ based on the social media platform and the part of the API from which the social media is to be collected. For more information, see *Collection types*.

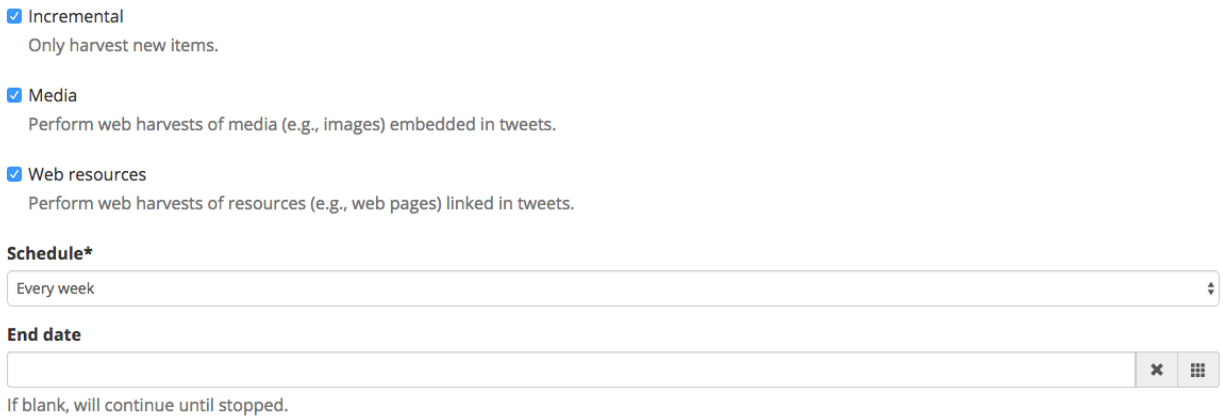
The collection types supported by SFM include:

- *Twitter search*
- *Twitter filter*
- *Twitter user timeline*
- *Twitter sample*
- *Flickr user*
- *Weibo timeline*
- *Tumblr blog posts*

SFM allows you to create multiple collections of each type within a collection set. For example, you might create a "Democratic candidate Twitter user timelines" collection and a "Republican candidate Twitter user timelines" collection. Collections are one way of organizing harvested content.

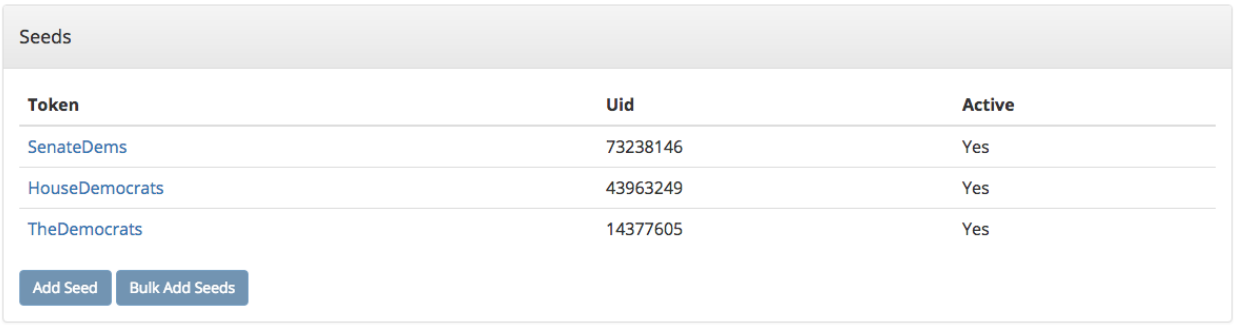
Each collection's harvest type has specific options, which may include:

- Schedule of how often to collect (e.g. daily, monthly). Streaming harvest types such as Twitter filter don't have a schedule – they're either on or off.
- Whether to perform web harvests of images, videos, or web pages embedded or linked from the posts.
- Whether to harvest incrementally. For example, each time a Twitter user timeline harvest runs, it can either collect only new items since the last harvest, or it can try to re-collect each entire timeline.



Add seeds

Some harvest types require seeds, which are the specific targets for collection.



As shown in the chart below, what a seed is and the number of seeds varies by harvest type. Note that some harvest types don't have any seeds.

Harvest type	Seed	How many?
Twitter search	Search query	1 or more
Twitter filter	Track/Follow/Locations	1 or more
Twitter user timeline	Twitter Account Name or ID	1 or more
Twitter sample	None	None
Flickr user	Flickr Account Name or ID	1 or more
Weibo timeline	None	None

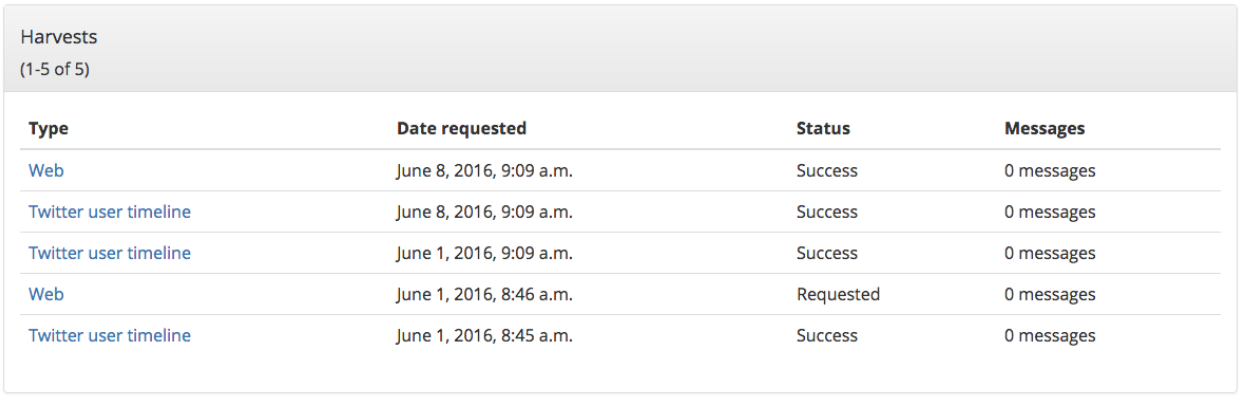
Start harvesting

Each collection's detail page has a *Turn On* button.

Once you turn on the collection, harvesting will proceed in the background according to the collection's schedule. It will stop when it hits the end date or you turn it off.

The collection's detail page will also show a message noting when the next harvest is scheduled for.

As harvesting progresses, SFM will list the results of harvests on the collection's detail page.



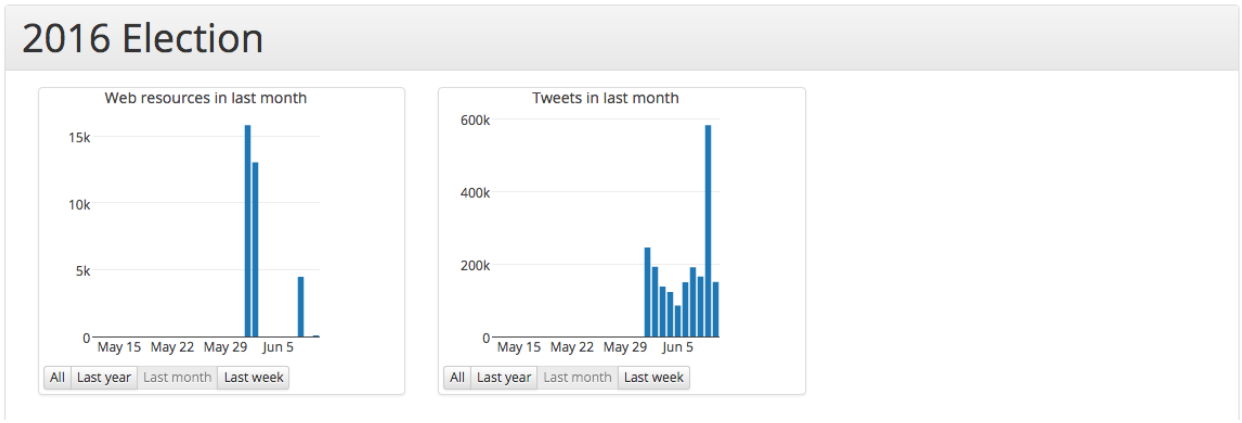
During harvesting

Within SFM, harvesting is performed by (you guessed it) harvesters. Harvesters make calls to the social media platforms' APIs and records the social media data in WARC files. (WARC is a standard file format used for web archiving.)

Depending on the collection options you selected, SFM may also extract URLs from the posts; these URLs link to web resources such as images, web pages, etc. SFM passes the URLs to the web harvester, which will collect these web resources (similar to more traditional web archiving).

To monitor harvesting:

- View details on each harvest in the Harvests section of the collection detail page.
- Check the visualizations of the number of items harvested for each collection on the home page. (Click *Social Feed Manager* in the top left of the page).



If you want to make changes to the collection's options and/or its seeds after harvesting is started, turn off the collection and then click the *Edit* button.

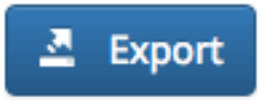
You'll be able to turn it back on and resume collecting afterwards.

Exploring, exporting, processing and analyzing your social media data

SFM provides several mechanisms for exporting collected social media data or feeding the social media data into your own processing pipelines. It also provides some basic tools for exploring and analyzing the collected content within the SFM environment.

Exports

To export collected social media data, click the *Export* button on the collection detail page. Exports are available in a number of formats, including Excel, CSV, and JSON.

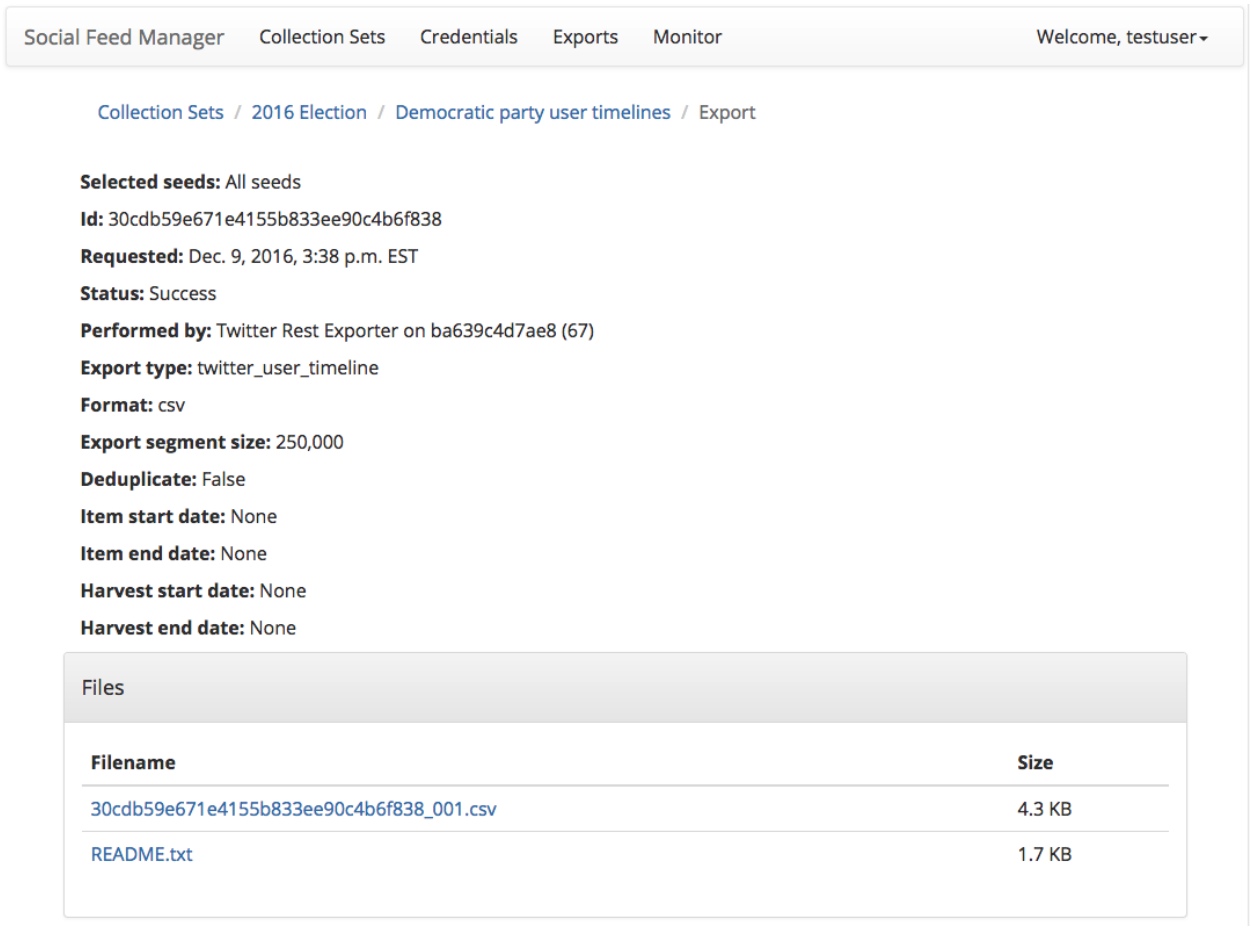


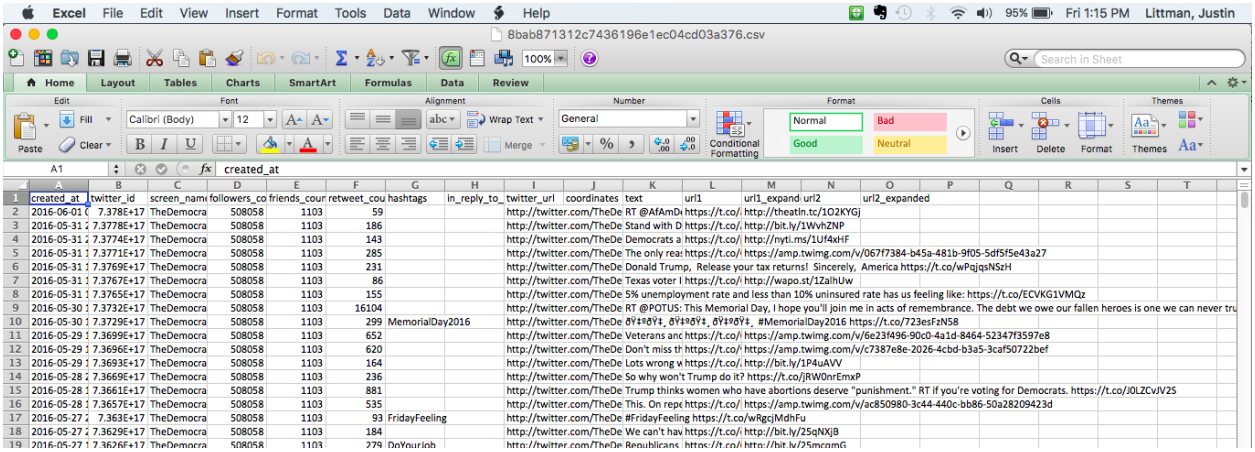
The "Full JSON" format provides the posts (e.g. tweets) in their original form, whereas the other export formats provide a subset of the metadata for each social media item. For example, for a tweet, the CSV export includes the tweet's "coordinates" value but not the "geo" value.

Refer to the *Data Dictionaries for CSV/Excel Exports* for details about each of the columns in the the CSV and Excel exports for Twitter collections.

Dehydration (exporting a list of just the IDs of social media items) is supported for certain data-sharing purposes.

Exports are run in the background, and larger exports may take a significant amount of time. You will receive an email when it is completed or you can monitor the status on the Exports page, where you can vew details about the export. This is also where you will find a link to download the export file once it becomes available. A README file will be created for each export containing documentation on the export and the collection.





Processing

If you've set up a processing container, or if you've installed SFM tools locally, then you have access to the collected social media data from the command line. You can then feed the data into your own processing pipeline and use your own tools.

More on this topic can be found in the *Commandline exporting/processing* section.

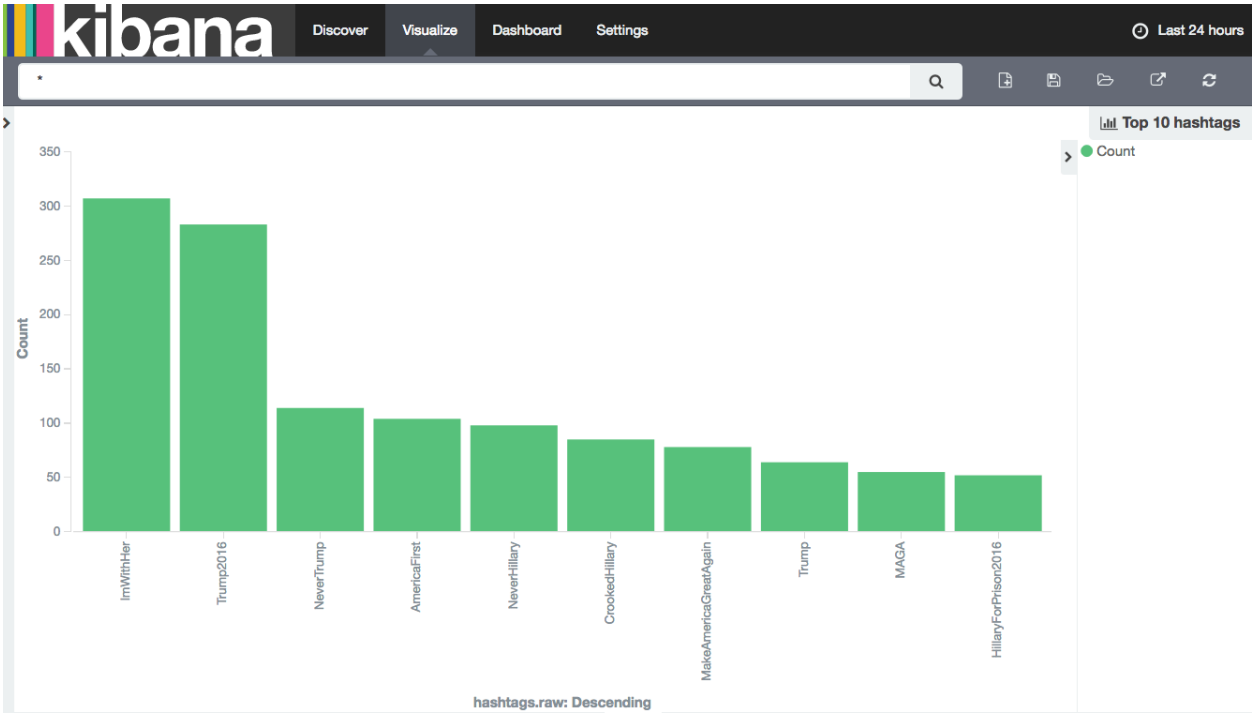
Exploration and analysis

While SFM does not provide a comprehensive toolset for exploring and analyzing the collected social media data, it provides some basic exploration and analysis tools and allows you to export social media data for use with your own tools.

Tools provided by SFM are:

- ELK (Elasticsearch, Logstash, Kibana)

The ELK stack is a general-purpose framework for exploring data. It provides support for loading, querying, analysis, and visualization. SFM provides an instance of ELK that has been customized for exploring social media data, in particular, Twitter and Weibo data.



ELK may be particularly useful for monitoring and adjusting the targets of ongoing social media collections. For example, it can be used to discover additional relevant Twitter hashtags or user accounts to collect, based on what has been collected so far.

ELK requires some additional setup. More on this topic can be found in the *Exploring social media data with ELK* section.

- Processing container

A processing container allows you to have access to the collected social media content from the command line. The processing container has been provisioned with a handful of analysis tools such as Twarc utils.

The following shows piping some tweets into a wordcloud generator from within a processing container:

```
# find_warcs.py 4f4d1 | xargs twitter_rest_warc_iter.py | python /opt/twarc/utils/
↪wordcloud.py
```

More on this topic can be found in the *Commandline exporting/processing* section.

Access and display

SFM does not currently provide a web interface to the collected social media content. However, this should be possible, and we welcome your ideas and contributions.

CHAPTER 2

API Credentials

Accessing the APIs of social media platforms requires credentials for authentication (also knows as API keys). Social Feed Manager supports managing those credentials.

Credentials/authentication allow a user to collect data through a platform's API. For some social media platforms (e.g., Twitter and Tumblr), Limits are placed on methods and rate of collection on a per credential basis.

SFM users are responsible for creating their own new credentials so that they can control their own collection rates and can ensure that they are following each platform's API policies.

Most API credentials have two parts: an application credential and a user credential.(Flickr is the exception – only an application credential is necessary.)

For more information about platform-specific policies, consult the documentation for each social media platform's API.

Managing credentials

SFM supports two approaches to managing credentials: adding credentials and connecting credentials. Both of these options are available from the Credentials page.

Adding credentials

For this approach, a user gets the application and/or user credential from the social media platform and provide them to SFM by completing a form. More information on getting credentials is below.

Connecting credentials

This is the easiest approach for users.

For this approach, SFM is configured with the application credentials for the social media platform by the systems administrator. The user credentials are obtained by the user being redirected to the social media website to give permission to SFM to access her account.

SFM is configured with the application credentials in the `docker-compose.yml`. If additional management is necessary, it can be performed using the Social Accounts section of the Admin interface.

Platform specifics

Twitter Twitter credentials can be obtained from https://apps.twitter.com/.

For detailed instructions, see *Adding Twitter Credentials*.

It is recommended to change the application permissions to read-only. You *must* provide a callback URL, but the URL you provide doesn't matter.

Weibo For instructions on obtaining Weibo credentials, see this guide.

To use the connecting credentials approach for Weibo, the redirect URL must match the application's actual URL and use port 80.

Flickr Flickr credentials can be obtained from https://www.flickr.com/services/api/keys/.

For detailed instructions, see *Adding Flickr Credentials*.

Tumblr Tumblr credentials can be obtained from https://www.tumblr.com/oauth/apps.

For detailed instructions, see *Adding Tumblr Credentials*.

Adding Twitter Credentials

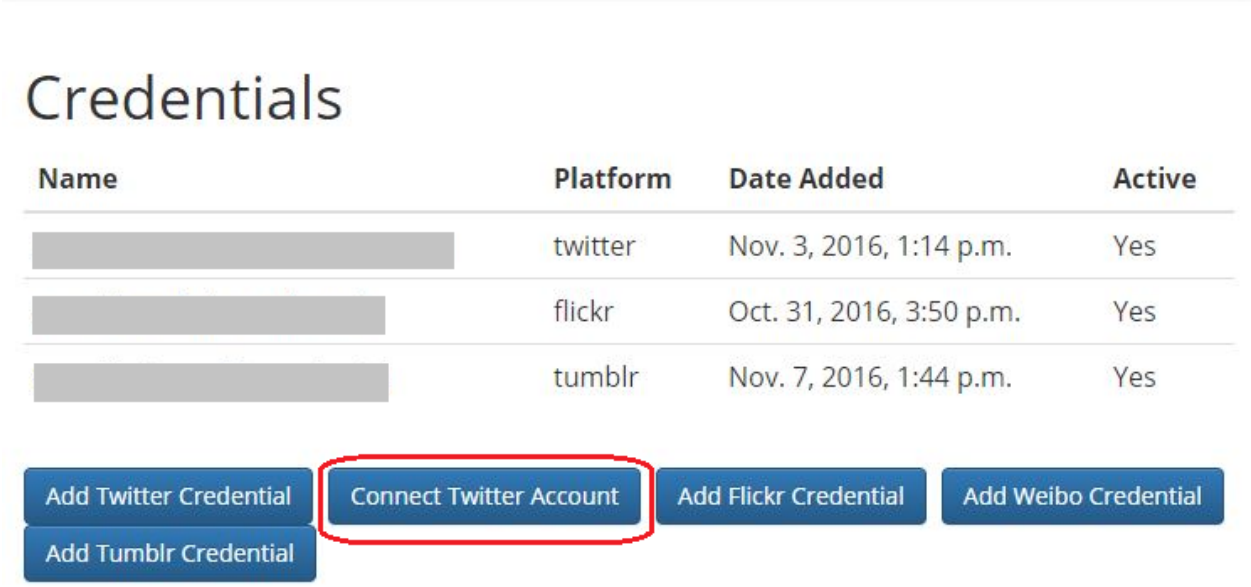
Twitter supports either adding credentials or connecting credentials.

Connecting Twitter credentials

SFM is configured with the application credentials for Twitter. In this approach, the user credentials are obtained by connecting the user to Twitter and the user authorizing SFM to use the user's Twitter credentials.

Note that personal credentials and information are used rather than the application level credentials obtained manually.

- **On the Credentials page of SFM, click** *Connect Twitter Account.*



This will open a Twitter authorization page.

- **Once you are signed in, click** *Authorize.* This will automatically fill out the credential page.

Adding Twitter credentials

In this method, the user manually acquires credentials from Twitter. This should be used particularly if a user wants to manage multiple credentials.

- **Navigate to** https://apps.twitter.com/.
- **Sign in to Twitter and select "Create New App."**
- **Enter a name for the app** like *Social Feed Manager* or the name of a new Collection Set.
- **Enter a description.** You may copy and paste: *This is a social media research and archival tool, which collects data for academic researchers through an accessible user interface.*
- **Enter a Website** such as the SFM url. Any website will work.
- **Enter a Callback URL** such as the same url used for the website field.
- **Review and agree to the Twitter Developer Agreement** and click *Create your Twitter Application.*
- **Recommended:**
  - Click on your new application.
  - Navigate to the *Permissions* tab.
  - Select *Read only* then *Update settings*.
- **Go to the Credentials page of SFM,** and click *Add Twitter Credential*.
- **Fill out all fields:**
  - On the Twitter apps page (https://apps.twitter.com/) click your new application.
  - Navigate to the *Keys and Access Tokens* tab.

  - From the top half of the page, copy and paste into the matching fields in SFM: *Consumer Key* and *Consumer Secret*.
  - From the bottom half of the page, copy and paste into the matching fields in SFM: *Access Token* and *Access Token Secret*.

- **Click** *Save*

Adding Flickr Credentials

- **Navigate to** https://www.flickr.com/services/api/keys/.
- **Sign in to your Yahoo! account.**
- **Click** *Get Another Key*
- **Choose** *Apply for a Non-commercial key,* which is for API users that are not charging a fee.
- **Enter an Application Name** like *Social Feed Manager*
- **Enter Application Description** such as: *This is a social media research and archival tool, which collects data for academic researchers through an accessible user interface.*
- **Check both checkboxes**
- **Click** *Submit*
- **Navigate to the SFM Credentials page** and click *Add Flicker Credential*
- **Enter the Key and Secret** in the correct fields and save.

Adding Tumblr Credentials

- **Navigate to** https://www.tumblr.com/oauth/apps/.
- **Sign in to Tumblr.**
- **Click** *Register Application*
- **Enter an Application Name** like *Social Feed Manager*
- **Enter a website** such as the SFM url
- **Enter Application Description** such as: *This is a social media research and archival tool, which collects data for academic researchers through an accessible user interface.*
- **Enter Administrative contact email.** You should use your own email.
- **Enter default callback url,** the same url used for the website.
- **Click** *Register*
- **Navigate to the SFM Credentials page** and click *Add Tumblr Credential*
- **Enter the OAuth Consumer Key** in the API key field and save.

CHAPTER 3

Collection types

Each collection type connects to one of a social media platform's APIs, or methods for retrieving data. Understanding what each collection type provides is important to ensure you collect what you need and are aware of any limitations. Reading the social media platform's documentation provides further important details.

Collection types

- *Twitter user timeline*: Collect tweets from specific Twitter accounts
- *Twitter search*: Collects tweets by a user-provided search query from recent tweets
- *Twitter sample*: Collects a Twitter provided stream of a subset of all tweets in real time.
- *Twitter filter*: Collects tweets by user-provided criteria from a stream of tweets in real time.
- *Flickr user*: Collects posts and photos from specific Flickr accounts
- *Weibo timeline*: Collects posts from the user and the user's friends
- *Weibo search*: Collects recent weibo posts by a user-provided search query
- *Tumblr blog posts*: Collects blog posts from specific Tumblr blogs
- *Collecting Web resources*: Secondary collections of resources linked to or embedded in social media posts.

Twitter user timeline

Collects tweets and their metadata by particular Twitter user accounts using Twitter's user_timeline API. Twitter provides up to 3,200 of the most recent tweets from each user.

Each Twitter user timeline collection can have multiple seeds, where each seed is a user timeline.

To identify a user timeline, you can provide a screen name (the string after @, like NASA for @NASA, which the user can change) or Twitter user ID (a numeric string which never changes, like 11348282 for @NASA). If you provide one identifier, the other will be looked up and displayed in SFM UI the first time the harvester runs.

The number of user timeline seeds is not limited in collections, but harvests may take longer if the collection exceeds Twitter's rate limits.

SFM will notify you when incorrect or private user timeline seeds are requested; all other valid seeds will be collected.

The incremental option will collect tweets that haven't been harvested before, preventing duplicate tweets. When the incremental option is not selected, the 3,200 most recent tweets will be collected. If a non-incremental harvest is performed multiple times, there will most likely be duplicates. However, you will may be able to track changes across time about a user's timeline, such as retweet and like counts, deletion of tweets, and follower counts.

Scheduling harvests should depend on how prolific the Twitter users are. In general, the more frequent the tweeter, the more frequent you'll want to schedule harvests.

See the *Collecting Web resources* guidance below for deciding whether to collect media or web resources.

Twitter search

Collects tweets from the last 7-9 days that match search queries using the Twitter Search API, similar to a regular search made on Twitter. Based on relevance, this is **not** a complete search of all tweets, limited both by time and arbitrary relevance (determined by Twitter).

Search collections collect tweets from a single search query.

Search queries must follow standard search term formulation; permitted queries are listed in the documentation for the Twitter Search API, or you can construct a query using the Twitter Advanced Search query builder.

Broad Twitter searches may take longer to complete – possibly days – due to Twitter's rate limits and the amount of data available from the Search API. In choosing a schedule, make sure that there is enough time between searches. (If there is not enough time between searches, later harvests will be skipped until earlier harvests complete.) In some cases, you may only want to run the search once and then turn off the collection.

The incremental option will collect tweets that haven't been harvested before, preventing duplicate tweets. When the incremental option is not selected, the search will be performed again, and there will most likely be duplicates.

See the *Collecting Web resources* guidance below for deciding whether to collect media or web resources.

Twitter sample

Collects a random sample of all public tweets using the Twitter sample stream, useful for capturing a sample of what people are talking about on Twitter. The Twitter sample stream returns approximately 0.5-1% of public tweets, which is approximately 3GB a day (compressed).

Unlike other Twitter collections, there are no seeds for a Twitter sample.

When on, the sample returns data every 30 minutes.

Only one sample or *Twitter filter* can be run at a time per credential.

See the *Collecting Web resources* guidance below for deciding whether to collection media or web resources.

Twitter filter

Collects a live selection of public tweets from criteria matching keywords, locations, or users, based on the Twitter filter streaming API. Because tweets are collected live, tweets from the past are not included. (Use a *Twitter search* collection to find tweets from the recent past.)

There are three different filter queries supported by SFM: track, follow, and location.

Track collects tweets based on a keyword search A space between words is treated as 'AND' and a comma is treated as 'OR'. Note that exact phrase matching is not supported. See the track parameter documentation for more information.

Follow collects tweets that are posted by or about a user (not including mentions) from a comma separated list of user IDs (the numeric identifier for a user account). Tweets collected will include those made by the user, retweeting the user, or replying to the user. See the follow parameter documentation for more information.

- Note: The Twitter UI does not provide a way to look up the numeric ID for a user account. You can use the twitter ID converter websites, such as https://tweeterid.com, for this purpose.

Location collects tweets that were geolocated within specific parameters, based on a bounding box made using the southwest and northeast corner coordinates. See the location parameter documentation for more information.

Twitter will return a limited number of tweets, so filters that return many results will not return all available tweets. Therefore, more narrow filters will usually return more complete results.

Only one filter or *Twitter sample* can be run at a time per credential.

SFM captures the filter stream in 30 minute chunks and then momentarily stops. Between rate limiting and these momentary stops, you should never assume that you are getting every tweet.

There is only one seed in a filter collection. Twitter filter collection are either turned on or off (there is no schedule).

See the *Collecting Web resources* guidance below for deciding whether to collection media or web resources.

Flickr user

Collects metadata about public photos by a specific Flickr user, and, optionally, copies of the photos at specified sizes.

Each Flickr user collection can have multiple seeds, where each seed is a Flickr user. To identify a user, you can provide a either a username or an NSID. If you provide one, the other will be looked up and displayed in the SFM UI during the first harvest. The NSID is a unique identifier and does not change; usernames may be changed but are unique.

Usernames can be difficult to find, so to ensure that you have the correct account, use this tool to find the NSID from the account URL (i.e., the URL when viewing the account on the Flickr website).

For each user, the user's information will be collected using Flickr's people.getInfo API and the list of her public photos will be retrieved from people.getPublicPhotos. Information on each photo will be collected with photos.getInfo.

Depending on the image sizes you select, the actual photo files will be collected as well. Be very careful in selecting the original file size, as this may require a significant amount of storage. Also note that some Flickr users may have a large number of public photos, which may require a significant amount of storage. It is advisable to check the Flickr website to determine the number of photos in each Flickr user's public photo stream before harvesting.

If the incremental option is selected, only new photos will be collected.

Weibo timeline

Collects weibos by the user and friends of the user whose credentials are provided using the Weibo friends_timeline API.

Note that because collection is determined by the user whose credentials are provided, there are no seeds for a Weibo timeline collection. To change what is being collected, change the user's friends from the Weibo website or app.

See the *Collecting Web resources* guidance below for deciding whether to collect image or web resources.

Weibo search

Collects recent weibos that match a search query using the **'Weibo search_topics API<http://open.weibo.com/wiki/2/search/topics>'_**. The Weibo API does not return a complete search of all Weibo posts. It only returns the most recent 200 posts matching a single keyword when found between pairs of '#' in Weibo posts (for example: *#keyword#* or *##*)

The incremental option will attempt to only count weibo posts that haven't been harvested before, maintaining a count of non-duplicate weibo posts. Because the Weibo search API does not accept *since_id* or *max_id* parameters, filtering out already-harvested weibos from the search count is accomplished within SFM.

When the incremental option is not selected, the search will be performed again, and there will most likely be duplicates in the count.

See the *Collecting Web resources* guidance below for deciding whether to collect image or web resources.

Tumblr blog posts

Collects blog posts by a specified Tumblr blog using the Tumblr Posts API.

Each Tumblr blog post collection can have multiple seeds, where each seed is a blog. The blog can be specified with or without the .tumblr.com extension.

If the incremental option is selected, only new blog posts will be collected.

See the *Collecting Web resources* guidance below for deciding whether to collect image or web resources.

Collecting Web resources

Most collection types allow you to select an option to collect web resources such as images, web pages, etc. that are included in the social media post. When a social media post includes a URL, SFM will harvest the web page at that URL. It will harvest only that web page, not any pages linked from that page.

Be very deliberate in collecting web resources. Performing a web harvest both takes longer and requires significantly more storage than collecting the original social media post.

CHAPTER 4

Data Dictionaries for CSV/Excel Exports

Social Feed Manager captures a variety of data from each platform. These data dictionaries give explanations for each selected and processed field in exports.

Note that these are subsets of the data that are collected for each post. The full data is available for export by selecting "Full JSON" as the export format or by exporting from the commandline. See *Commandline exporting/processing*.

Twitter Dictionary

For more info about source tweet data, see the Twitter API documentation, including Tweets and Entities.

Documentation about older archived tweets is archived by the Wayback Machine for the Twitter API, Tweets, and Entities.

Field	Description	Example
created_at	Date and time the tweet was created, in ISO 8601 format and UTC time zone.	2016-12-21T19:30:03+00:00
twitter_id	Twitter identifier for the tweet.	114749583439036416
screen_name	The unique screen name of the account that authored the tweet, at the time the tweet was posted. Note that an account's screen name may change over time.	NASA
location	The user's self-described location.	San Francisco, California
followers_count	Number of followers this account had at the time the tweet was harvested.	235
friends_count	Number of users this account was following at the time the tweet was harvested.	114
favorite_count/ like_count	Number of times this tweet had been favorited/liked by other users at the time the tweet was harvested.	12
retweet_count	Number of times this tweet had been retweeted at the time the tweet was harvested.	25
hashtags	Hashtags from the tweet text, as a comma-separated list.	Mars, askNASA
mentions	Other Twitter accounts mentioned in the text of the tweet, separated by comma and space.	NASA_Airborne, NASA_Ice
in_reply_to_screen_name	If the tweet is a reply, the screen name of the original tweet's author.	wiredscience
twitter_url	URL of the tweet. If the tweet is a retweet made using the Twitter retweet feature, the URL will redirect to the original tweet.	http://twitter.com/NASA/status/394883921303056384 retweet redirecting to original tweet: http://twitter.com/NASA/status/394875351894994944
text	The text of the tweet. Newline characters are stripped out.	Observing Hurricane Raymond Lashing Western Mexico: Low pressure System 96E developed quickly over the... http://t.co/YpffdKVrgm
is_retweet	*Yes* if tweet is a retweet of another tweet, according to the tweet's metadata; otherwise *No*.	Yes
is_quote	*Yes* if tweet is a quote of another tweet, according to the tweet's metadata; otherwise *No*.	No
coordinates	The geographic coordinates of the tweet. This is only enabled if geotagging is enabled on the account. The value, if present, is of the form [longitude, latitude]	[-0.22012208, 51.59248806]
url1	First URL in text of tweet, as shortened by Twitter.	http://t.co/WGJ9VmoKME
url1_expanded	Expanded version of *url1*; URL entered by user and displayed in Twitter. Note that the user-entered URL may itself be a shortened URL, e.g. from bit.ly.	http://instagram.com/p/gA_zQ5IaCz/
url2	Second URL in text of tweet, as shortened Twitter.	https://t.co/ZTUQZcikJa
url2_expanded	Expanded version of *url2*; URL entered by user and displayed in Twitter. Note that the user-entered URL may itself be a shortened URL, e.g. from bit.ly.	http://instagram.com/p/gA_zQ5IaCz/
media_url	URL of the media embedded in the tweet. If the media embedded in the tweet is a video, this is the URL of the video's thumbnail image	http://pbs.twimg.com/media/Cyir15CVIAAfAWd.jpg

Tumblr Dictionary

For more info about source tweet data, see the Tumblr API documentation, particularly Posts.

Documentation about older archived posts is archived by the Wayback Machine for the original Tumblr API and the newer Tumblr API.

Field	Description	Example
created_at	Date and time the tweet was created, in ISO 8601 format and UTC time zone.	2016-12-21 19:30:03+00:00
tumblr_id	Tumblr identifier for the blog post	154774150409
blog_name	The short name used to uniquely identify a blog. This is the first part of the blog url, like <nasa.tumblr.com>.	nasa
post_type	The type of post, such as one of the following: text, quote, link, answer, video, audio, photo, or chat.	text
post_slug	Text summary of the post, taken from the final portion of the url.	10-questions-for-our-chief-scientist
post_summary	Text summary of the post, taken from the title of the post.	10 Questions for Our Chief Scientist
post_text	Body of the post text, using html markup.	See https://notepad.pw/w8133kzj
tags	Hashtags from the post as a comma-separated list.	nasa, space, solarsystem, chiefscientist, scientist
tumblr_url	Full url location of the post.	http://nasa.tumblr.com/post/154774150409/10-questions-for-our-chief-scientist
tumblr_short_url	Short url of the post.	https://tmblr.co/Zz_Uqj2G9GXq9

Flickr Dictionary

For more info about source tweet data, see the Flickr API documentation, particularly *People* and *Photos*.

Documentation about older archived posts is archived by the Wayback Machine here.

Field	Description	Example
photo_id	Unique Flickr identifier of the photo.	11211844604
date_posted	Date and time that the post was uploaded to Flickr, in ISO 8601 format and UTC time zone.	2013-12-04 21:39:40+00:00
date_taken	Date and time that media was captured, either extracted from EXIF or from the date posted, in mm/dd/yyyy hh:mm format.	6/7/2014 13:35
license	Licensing allowed for media, given as a numeral according to the following key: • 0 = All Rights Reserved • 1 = Attribution-NonCommercial-Sharealike License • 2 = Attribution-NonCommercial License • 3 = Attribution-NonCommercial NoDerivs License • 4 = Attribution License • 5 = Attribution-ShareAlike License • 6 = Attribution-NoDerivs License • 7 = No known copyright restrictions • 8 = United States Government work • More information at creativecommons.org/licenses	4 *(Attribution license)*
safety_level	Appropriateness of post, given as a numeral according to the following key: • 0 = Safe - Content suitable for everyone • 1 = Moderate - Approximately PG-13 content • 2 = Restricted - Approximately R rated content	0 *(Safe level)*
original_format	File format of uploaded media.	jpg
owner_nsid	Unique Flickr identifier of the owner account.	28399705@N04
owner_username	Unique plaintext username of the owner account.	GW Museum and Textile Museum
title	Title of the post.	Original Museum entrance
description	Short description of the post.	Historic photo courtesy of The Textile Museum Archives.
media	Media type of the post.	photo
photopage	Location url of the post.	https://www.flickr.com/photos/textilemuseum/11211844604/

Weibo Dictionary

For more info about source tweet data, see the Sina Weibo API friends_timeline documentation.

Documentation about older archived tweets is archived by the Wayback Machine here.

Note that for privacy purposes, Weibo dictionary examples are not consistent.

Field	Description	Example
created_at	Date and time the tweet was created, in ISO 8601 format and UTC time zone.	2016-12-21T19:30:03+00:00
weibo_id	Sina Weibo identifier for the tweet.	4060309792585658
screen_name	The unique screen name of the account that authored the weibo, at the time the weibo was posted.	
followers_count	Number of followers this account had at the time the weibo was harvested.	3655329
friends_count	Number of users this account was following at the time the weibo was harvested.	2691
reposts_count	Number of times this weibo had been reposted at the time the weibo was harvested.	68
topics	Topics (similar to hashtags) from the weibo text as a comma-separated list.	
in_reply_to_screen_name	If the weibo is a reply, the screen name of the original weibo's author. (This is not yet supported by Sina Weibo.)	
weibo_url	URL of the weibo. If the tweet is a retweet made	http://m.weibo.cn/1618051664/4060300716095462
text	The text of the weibo.	
url1	First URL in text of weibo, as shortened by Sina Weibo.	http://t.cn/RM2xyx6
url2	Second URL in text of weibo, as shortened by Sina Weibo.	http://t.cn/Rc52gDY
retweeted_text	Text of original weibo when the collected weibo is a repost.	
retweeted_url1	First URL in text of original weibo, as shortened by Sina Weibo.	http://t.cn/RVR4cAQ
retweeted_url2	Second URL in text of original weibo, as shortened by Sina Weibo.	http://t.cn/RMAJISP

CHAPTER 5

Commandline exporting/processing

While social media data can be exported from the SFM UI, in some cases you may want to export from the commandline. These cases include:

- Exporting very large datasets. (Export via the UI is performed serially; export via the commandline can be performed in parallel, which may be much faster.)
- Performing more advanced filtering or transformation that is not supported by the UI export.
- Integrating with a processing/analysis pipeline.

To support export and processing from the commandline, SFM provides a processing container. A processing container is a Linux shell environment with access to the SFM's data and preloaded with a set of useful tools.

Using a processing container requires familiarity with the Linux shell and shell access to the SFM server. If you are interested in using a processing container, please contact your SFM administrator for help.

When exporting/processing data, remember that harvested social media content and and web resources are stored in `/sfm-data`. `/sfm-processing` is provided to store your exports, processed data, or scripts. Depending on how it is configured, you may have access to `/sfm-processing` from your local filesystem. See *Storage*.

Processing container

To bootstrap export/processing, a processing image is provided. A container instantiated from this image is Ubuntu 14.04 and pre-installed with the warc iterator tools, `find_warcs.py`, and some other useful tools. (Warc iterators and `find_warcs.py` are described below.) It will also have read-only access to the data from `/sfm-data` and read/write access to `/sfm-processing`.

The other tools available in a processing container are:

- jq for JSON processing.
- twarc for access to the Twarc utils.
- JWAT Tools for processing WARCs.
- warctools for processing WARCs.

- parallel for parallelizing processing.

To instantiate a processing container, from the directory that contains your `docker-compose.yml` file:

```
docker-compose run --rm processing /bin/bash
```

You will then be provided with a bash shell inside the container from which you can execute commands:

```
root@0ac9caaf7e72:/sfm-processing# find_warcs.py 4f4d1 | xargs twitter_rest_warc_iter.
↪py | python /opt/twarc/utils/wordcloud.py
```

Note that once you exit the processing container, the container will be automatically removed. However, if you have saved all of your scripts and output files to `/sfm-processing`, they will be available when you create a new processing container.

SFM commandline tools

Warc iterators

SFM stores harvested social media data in WARC files. A warc iterator tool provides an iterator to the social media data contained in WARC files. When used from the commandline, it writes out the social media items one at a time to standard out. (Think of this as `cat`-ing a line-oriented JSON file. It is also equivalent to the output of Twarc.)

Each social media type has a separate warc iterator tool. For example, `twitter_rest_warc_iter.py` extracts tweets recorded from the Twitter REST API. For example:

```
root@0ac9caaf7e72:/sfm-data# twitter_rest_warc_iter.py
usage: twitter_rest_warc_iter.py [-h] [--pretty] [--dedupe]
                                 [--print-item-type]
                                 filepaths [filepaths ...]
```

Here is a list of the warc iterators:

- `twitter_rest_warc_iter.py`: Tweets recorded from Twitter REST API.
- `twitter_stream_warc_iter.py`: Tweets recorded from Twitter Streaming API.
- `flickr_photo_warc_iter.py`: Flickr photos
- `weibo_warc_iter.py`: Weibos
- `tumblr_warc_iter.py`: Tumblr posts

Warc iterator tools can also be used as a library.

Find Warcs

`find_warcs.py` helps put together a list of WARC files to be processed by other tools, e.g., warc iterator tools. (It gets the list of WARC files by querying the SFM API.)

Here is arguments it accepts:

```
root@0ac9caaf7e72:/sfm-data# find_warcs.py
usage: find_warcs.py [-h] [--include-web] [--harvest-start HARVEST_START]
                     [--harvest-end HARVEST_END] [--api-base-url API_BASE_URL]
                     [--debug [DEBUG]]
                     collection [collection ...]
```

For example, to get a list of the WARC files in a particular collection, provide some part of the collection id:

```
root@0ac9caaf7e72:/sfm-data# find_warcs.py 4f4d1
/sfm-data/collection_set/b06d164c632d405294d3c17584f03278/
↪4f4d1a6677f34d539bbd8486e22de33b/2016/05/04/14/515dab00c05740f487e095773cce8ab1-
↪20160504143638715-00000-47-88e5bc8a36a5-8000.warc.gz
```

(In this case there is only one WARC file. If there was more than one, it would be space separated.)

The collection id can be found from the SFM UI.

Note that if you are running `find_warcs.py` from outside a Docker environment, you will need to supply `--api-base-url`.

READMEs

The *exportreadme* management command will output a README file that can be used as part of the documentation for a dataset. The README contains information on the collection, including the complete change log. Here is an example of creating a README:

```
docker-compose exec ui /bin/bash -c "/opt/sfm-ui/sfm/manage.py exportreadme 4f4d1 > /
↪sfm-processing/README.txt"
```

For examples, see the README files in this open dataset.

Note that this is a management command; thus, it is executed differently than the commandline tools described above.

Recipes

Extracting URLs

The "Extracting URLs from #PulseNightclub for seeding web archiving" blog post provides some useful guidance on extracting URLs from tweets, including unshortening and sorting/counting.

Exporting to line-oriented JSON files

This recipe is for exporting social media data from WARC files to line-oriented JSON files. There will be one JSON file for each WARC. This may be useful for some processing or for loading into some analytic tools.

This recipe uses parallel for parallelizing the export.

Create a list of WARC files:

```
find_warcs.py 7c37157 | tr ' ' '\n' > source.lst
```

Replace *7c37157* with the first few characters of the collection id that you want to export. The collection id is available on the colllection detail page in SFM UI.

Create a list of JSON destination files:

```
cat source.lst | xargs basename -a | sed 's/.warc.gz/.json/' > dest.lst
```

This command puts all of the JSON files in the same directory, using the filename of the WARC file with a .json file extension.

If you want to maintain the directory structure, but use a different root directory:

```
cat source.lst | sed 's/sfm-data\/collection_set/sfm-processing\/export/' | sed 's/.
↪warc.gz/.json/'
```

Replace *sfm-processing/export* with the root directory that you want to use.

Perform the export:

```
parallel -a source.lst -a dest.lst --xapply "twitter_stream_warc_iter.py {1} > {2}"
```

Replace *twitter_stream_warc_iter.py* with the name of the warc iterator for the type of social media data that you are exporting.

You can also perform a filter on export using jq. For example, this only exports tweets in Spanish:

```
parallel -a source.lst -a dest.lst --xapply "twitter_stream_warc_iter.py {1} | jq -c
↪'select(.lang == \"es\")' > {2}"
```

And to save space, the JSON files can be gzip compressed:

```
parallel -a source.lst -a dest.lst --xapply "twitter_stream_warc_iter.py {1} | gzip >
↪{2}"
```

You might also want to change the file extension of the destination file to ".json.gz" by adjusting the commmand use to create the list of JSON destination files. To access the tweets in a gzipped JSON file, use:

```
gzip -c <filepath>
```

Counting posts

wc -l can be used to count posts. To count the number of tweets in a collection:

```
find_warcs.py 7c37157 | xargs twitter_stream_warc_iter.py | wc -l
```

To count the posts from line-oriented JSON files created as described above:

```
cat dest.lst | xargs wc -l
```

wc -l gotcha: When doing a lot of counting, *wc -l* will output a partial total and then reset the count. The partial totals must be added together to get the grand total. For example:

```
[Some lines skipped ...]
    1490 ./964be41e1714492bbe8ec5793e05ec86-20160725070757217-00000-7932-62ebe35d576c-
↪8002.json
    4514 ./5f78a79c6382476889d1ed4734d6105a-20160722202703869-00000-5110-62ebe35d576c-
↪8002.json
   52043 ./417cf950a00d44408458c93f08f0690e-20160910032351524-00000-1775-c4aea5d70c14-
↪8000.json
54392684 total
[Some lines skipped ...]
   34778 ./30bc1c34880d404aa3254f82dd387514-20160806132811173-00000-21585-
↪62ebe35d576c-8000.json
   30588 ./964be41e1714492bbe8ec5793e05ec86-20160727030754726-00000-10044-
↪62ebe35d576c-8002.json
21573971 total
```

Using jq to process JSON

For tips on using jq with JSON from Twitter and other sources, see:

- Getting Started Working with Twitter Data Using jq
- Recipes for processing Twitter data with jq
- Reshaping JSON with jq

CHAPTER 6

Exploring social media data with ELK

The ELK (Elasticsearch, Logstash, Kibana) stack is a general-purpose framework for exploring data. It provides support for loading, querying, analysis, and visualization.

SFM provides an instance of ELK that has been customized for exploring social media data. It currently supports data from Twitter and Weibo.

One possible use for ELK is to monitor data that is being harvested to discover new seeds to select. For example, it may reveal new hashtags or users that are relevant to a collection.

Though you can use Logstash and Elasticsearch directly, in most cases you will interact exclusively with Kibana, which is the exploration interface.

Enabling ELK

ELK is not available by default; it must be enabled as described here.

You can enable one or more ELK Docker containers. Each container can be configured to be loaded with all social media data or the social media data for a single collection set.

To enable an ELK Docker container it must be added to your `docker-compose.yml` and then started by:

```
docker-compose up -d
```

An example container is provided in `example.docker-compose.yml` and `example.prod.docker-compose.yml`. These examples also show how to limit to a single collection set by providing the collection set id.

By default, Kibana is available at http://<your hostname>:5601/app/kibana. (Also, by default Elasticsearch is available on port 9200 and Logstash is available on port 5000.)

If enabling multiple ELK containers, add multiple containers to your `docker-compose.yml`. Make sure to give each container a unique name and a unique `hostname:` value, and make sure that each container maps to different ports.

Loading data

ELK will automatically be loaded as new social media data is harvested. (Note, however, that there will be some latency between the harvest and the data being available in Kibana.)

Since only new social media data is added, it is recommended that you enable the ELK Docker container before beginning harvesting.

If you would like to load social media data that was harvested before the ELK Docker container was enabled, use the `resendwarccreatedmsgs` management command:

```
usage: manage.py resendwarccreatedmsgs [-h] [--version] [-v {0,1,2,3}]
                                       [--settings SETTINGS]
                                       [--pythonpath PYTHONPATH] [--traceback]
                                       [--no-color]
                                       [--collection-set COLLECTION_SET]
                                       [--harvest-type HARVEST_TYPE] [--test]
                                       routing_key
```

The `resendwarccreatedmsgs` command resends warc_created messages which will trigger the loading of data by ELK.

To use this command, you will need to know the routing key. The routing key is `elk_loader_<hostname>.warc_created`. The hostname is available as part of the definition of the ELK container in the `docker-compose.yml` file.

The loading can be limited by collection set (`--collection-set`) and/or (`--harvest-type`). You can get collection set ids from the collection set detail page. The available harvest types are twitter_search, twitter_filter, twitter_user_timeline, twitter_sample, and weibo_timeline.

This shows loading the data limited to a collection set:

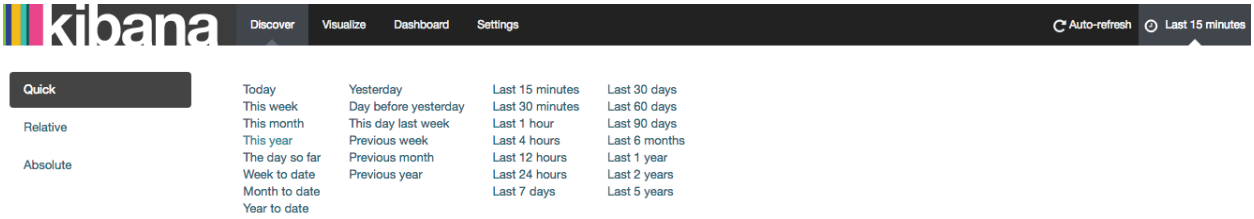
```
docker exec docker_sfmuiapp_1 python sfm/manage.py resendwarccreatedmsgs --collection-
↪set b438a62cbcf74ad0adc09be3b07f039e elk_loader_myproject_elk.warc_created
```

Overview of Kibana

The Kibana interface is extremely powerful. However, with that power comes complexity. The following provides an overview of some basic functions in Kibana. For some advanced usage, see the Kibana Reference or the Kibana 101: Getting Started with Visualizations video.

When you start Kibana, you probably won't see any results.

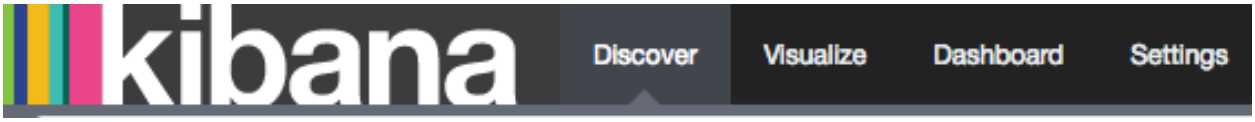
This is because Kibana defaults to only showing data from the last 15 minutes. Use the date picker in the upper right corner to select a more appropriate time range.



Tip: At any time, you can change the date range for your query, visualization, or dashboard using the date picker.

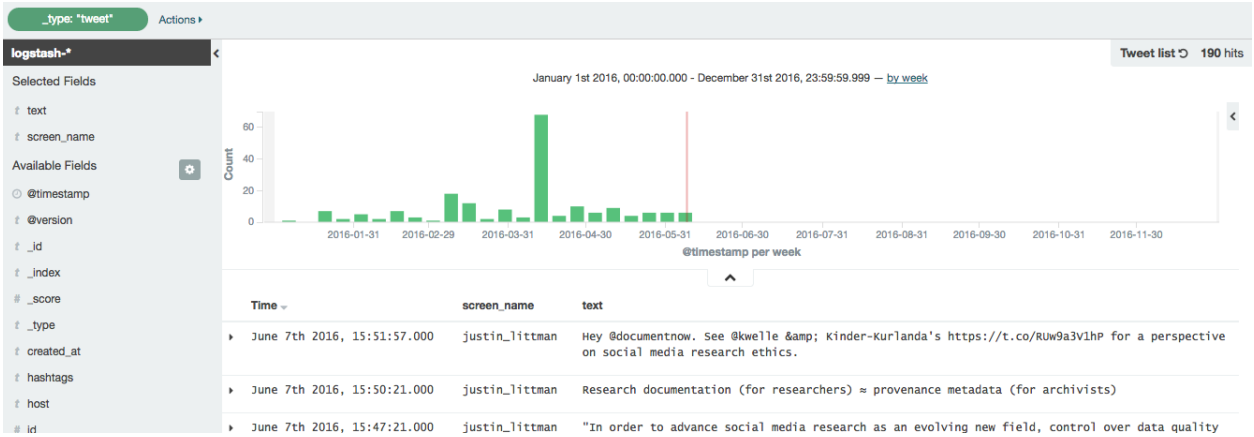
Discover

The Discover tab allows you to query the social media data.



By default, all social media types are queried. By limit to a single type (e.g., tweets), click the folder icon and select the appropriate filter.

You will now only see results for that social media type.



Notice that each social media item has a number of fields.

Time	screen_name	text
June 7th 2016, 15:51:57.000	justin_littman	Hey @documentnow. See @kwelle & Kinder-Kurlanda's https://t.co/RUw9a3V1hP for a perspective on social media research ethics.

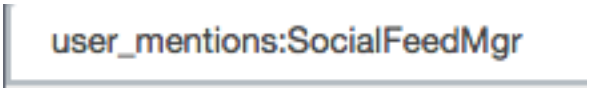
Table JSON Link to /logstash-2016.06.07/tweet/740270089644056600

@timestamp	June 7th 2016, 15:51:57.000
@version	1
_id	740270089644056600
_index	logstash-2016.06.07
_score	
_type	tweet
created_at	Tue Jun 07 19:51:57 +0000 2016
hashtags	
host	26ce21fa2e43
id	740,270,089,644,056,448
screen_name	justin_littman
sm_type	tweet
text	Hey @documentnow. See @kwelle & Kinder-Kurlanda's https://t.co/RUw9a3V1hP for a perspective on social media research ethics.
urls	http://dl.acm.org/citation.cfm?doid=2908131.2908172
user_id	481186914
user_mentions	documentnow, kwelle

You can search against a field. For example, to find all tweets containing the term "archiving":

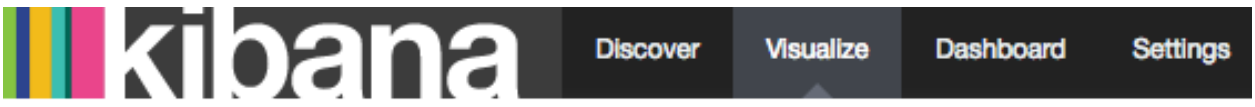
or having the hashtag #SaveTheWeb:

or mentioning @SocialFeedMgr:



Visualize

The Visualize tab allows you to create visualizations of the social media data.



The types of visualizations that are supported include:

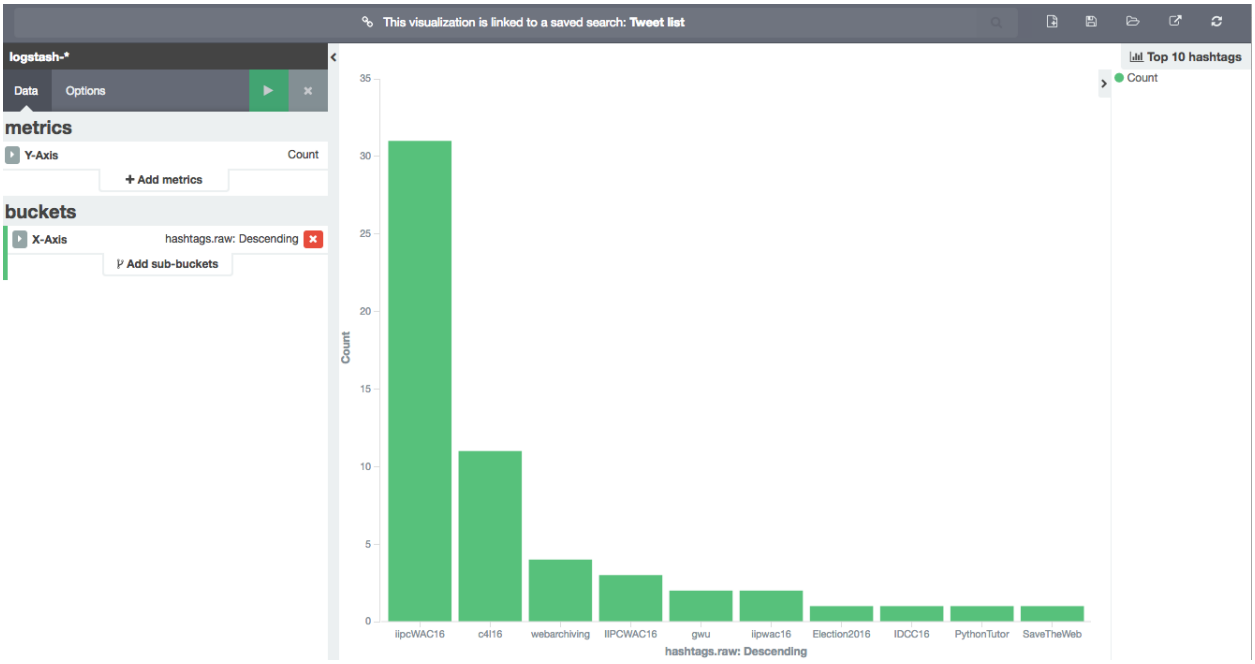
- Area chart
- Data table
- Line chart
- Pie chart

- Map
- Vertical bar chart

Describing how to create visualizations is beyond the scope of this overview.

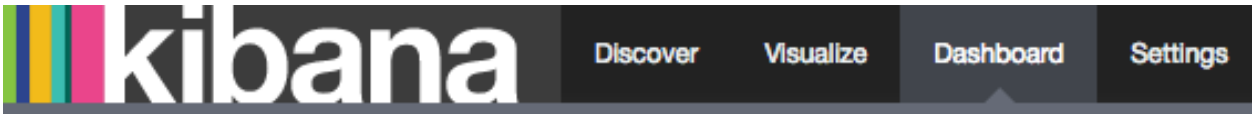
A number of visualizations have already been created for social media data. (The available visualizations are listed on the bottom of the page.)

For example, here is the Top 10 hashtags visualization:

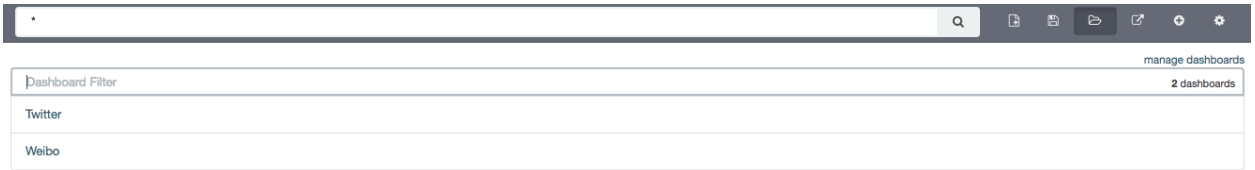


Dashboard

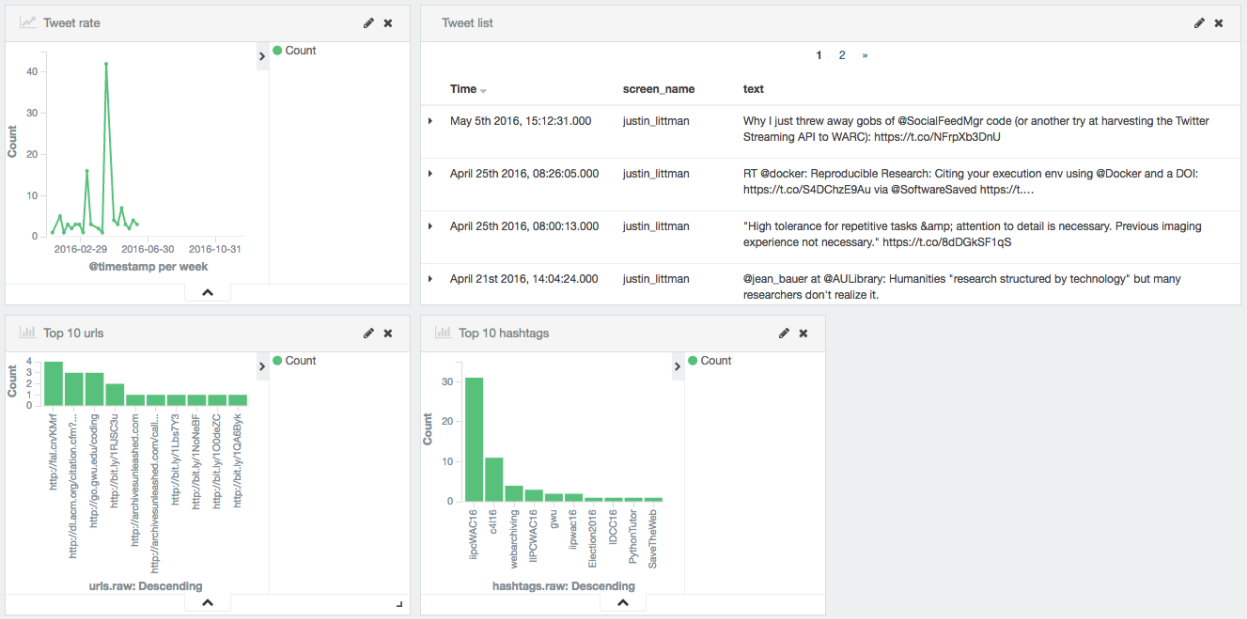
The Dashboard tab provides a summary view of data, bringing together multiple visualizations and searches on a single page.



A number of dashboards have already been created for social media data. To select a dashboard, click the folder icon and select the appropriate dashboard.



For example, here is the top of the Twitter dashboard:



Caveats

- This is experimental. We have not yet determined the level of development that will be performed in the future.
- Approaches for administering and scaling ELK have not been considered.
- No security or access restrictions have been put in place around ELK.

CHAPTER 7

Installation and configuration

Overview

The supported approach for deploying SFM is Docker containers. For more information on Docker, see *Docker*.

Each SFM service will provide images for the containers needed to run the service (in the form of `Dockerfile` s). These images will be published to Docker Hub. GWU created images will be part of the GWUL organization and be prefixed with *sfm-*.

sfm-docker provides the necessary `docker-compose.yml` files to compose the services into a complete instance of SFM.

The following will describe how to setup an instance of SFM that uses the latest release (and is suitable for a production deployment.) See the development documentation for other SFM configurations.

SFM *can* be deployed without Docker. The various `Dockerfile` s should provide reasonable guidance on how to accomplish this.

Local installation

Installing locally requires Docker and Docker-Compose. See *Installing Docker*.

1. Either clone the sfm-docker repository and copy the example configuration files:

```
git clone https://github.com/gwu-libraries/sfm-docker.git
cd sfm-docker
cp example.prod.docker-compose.yml docker-compose.yml
cp example.env .env
```

or just download `example.prod.docker-compose.yml` and `example.env`:

```
curl -L https://github.com/gwu-libraries/sfm-docker/raw/master/example.prod.docker-
↪compose.yml > docker-compose.yml
curl -L https://github.com/gwu-libraries/sfm-docker/raw/master/example.env > .env
```

2. Update configuration in `.env` as described in *Configuration*.
3. Bring up the containers:

```
docker-compose up -d
```

It may take several minutes for the images to be downloaded and the containers to start.

4. It is also recommended that you scale up the Twitter REST Harvester container:

```
docker-compose scale twitterrestharvester=2
```

Notes:

- The first time you bring up the containers, their images will be pulled from Docker Hub. This will take several minutes.
- For instructions on how to make configuration changes *after* the containers have been brought up, see *Configuration*.
- To learn more about scaling , see *Scaling up with Docker*.
- For suggestions on sizing your SFM server, see *Server sizing*.

Amazon EC2 installation

To launch an Amazon EC2 instance running SFM, follow the normal procedure for launching an instance. In *Step 3: Configure Instance Details*, under *Advanced Details* paste the following in user details and modify as appropriate as described in *Configuration*:

```
#cloud-config
repo_update: true
repo_upgrade: all

packages:
 - python-pip

runcmd:
 - curl -sSL https://get.docker.com/ | sh
 - usermod -aG docker ubuntu
 - pip install -U docker-compose
 - mkdir /sfm-data
 - mkdir /sfm-processing
 - cd /home/ubuntu
# This brings up the latest production release. To bring up master, remove prod.
 - curl -L https://github.com/gwu-libraries/sfm-docker/raw/master/example.prod.docker-
↪compose.yml > docker-compose.yml
 - curl -L https://github.com/gwu-libraries/sfm-docker/raw/master/example.env > .env
# Set config below by uncommenting.
# Don't forget to escape $ as \$.
# COMMON CONFIGURATION
# - echo TZ=America/New_York >> .env
# VOLUME CONFIGURATION
# Don't change these.
 - echo DATA_VOLUME=/sfm-data:/sfm-data
 - echo PROCESSING_VOLUME=/sfm-processing:/sfm-processing
# SFM UI CONFIGURATION
# Don't change this.
```

```
 - echo SFM_HOSTNAME=`curl http://169.254.169.254/latest/meta-data/public-hostname` >>
↪ .env
 - echo SFM_PORT=80 >> .env
# Provide your institution name display on sfm-ui footer
# - echo SFM_INSTITUTION_NAME=yourinstitution >> .env
# Provide your institution link
# - echo SFM_INSTITUTION_LINK=http://library.yourinstitution.edu >> .env
# To send email, set these correctly.
# - echo SFM_SMTP_HOST=smtp.gmail.com >> .env
# - echo SFM_EMAIL_USER=someone@gmail.com >> .env
# - echo SFM_EMAIL_PASSWORD=password >> .env
# An optional contact email at your institution that is provided to users.
# - echo SFM_CONTACT_EMAIL=sfm@yourinstitution.edu >> .env
# To enable connecting to social media accounts, provide the following.
# - echo TWITTER_CONSUMER_KEY=mBbq9ruffgEcfsktgQztTHUir8Kn0 >> .env
# - echo TWITTER_CONSUMER_SECRET=Pf28yReB9Xgz0fpLVO4b46r5idZnKCKQ6xlOomBAjD5npFEQ6Rm >
↪> .env
# - echo WEIBO_API_KEY=13132044538 >> .env
# - echo WEIBO_API_SECRET=68aea49fg26ea5072ggec14f7c0e05a52 >> .env
# - echo TUMBLR_CONSUMER_KEY=Fki09cW957y56h6fhRtCnig14QhpM0pjuHbDWMrZ9aPXcsthVQq >> .
↪env
# - echo TUMBLR_CONSUMER_SECRET=aPTpFRE2O7sVl46xB3difn8kBYb7EpnWfUBWxuHcB4gfvP >> .env
# For automatically created admin account
# - echo SFM_SITE_ADMIN_NAME=sfmadmin >> .env
# - echo SFM_SITE_ADMIN_EMAIL=nowhere@example.com >> .env
# - echo SFM_SITE_ADMIN_PASSWORD=password >> .env
# RABBIT MQ CONFIGURATION
# - echo RABBITMQ_USER=sfm_user >> .env
# - echo RABBITMQ_PASSWORD=password >> .env
# - echo RABBITMQ_MANAGEMENT_PORT=15672 >> .env
# DB CONFIGURATION
# - echo POSTGRES_PASSWORD=password >> .env
# WEB HARVESTER CONFIGURATION
# - echo HERITRIX_USER=sfm_user >> .env
# - echo HERITRIX_PASSWORD=password >> .env
# - echo HERITRIX_ADMIN_PORT=8443 >> .env
# - echo HERITRIX_CONTACT_URL=http://library.myschool.edu >> .env
 - docker-compose up -d
 - docker-compose scale twitterrestharvester=2
```

When the instance is launched, SFM will be installed and started.

Note the following:

- Starting up the EC2 instance will take several minutes.
- This has been tested with *Ubuntu Server 14.04 LTS*, but may work with other AMI types.
- For suggestions on sizing your SFM server, see *Server sizing*.
- If you need to make additional changes to your `docker-compose.yml`, you can ssh into the EC2 instance and make changes. `docker-compose.yml` and `.env` will be in the default user's home directory.
- Make sure to configure a security group that exposes the proper ports. To see which ports are used by which services, see example.prod.docker-compose.yml.
- To learn more about configuring EC2 instances with user data, see the AWS user guide.

Configuration

Configuration is documented in `example.env`. For a production deployment, pay particular attention to the following:

- Set new passwords for `SFM_SITE_ADMIN_PASSWORD`, `RABBIT_MQ_PASSWORD`, `POSTGRES_PASSWORD`, and `HERITRIX_PASSWORD`.
- The data volume strategy is used to manage the volumes that store SFM's data. By default, normal Docker volumes are used. To use a host volume instead, change the `DATA_VOLUME` and `PROCESSING_VOLUME` settings. Host volumes are recommended for production because they allow access to the data from outside of Docker.
- Set the `SFM_HOSTNAME` and `SFM_PORT` appropriately. These are the public hostname (e.g., sfm.gwu.edu) and port (e.g., 80) for SFM.
- Email is configured by providing `SFM_SMTP_HOST`, `SFM_EMAIL_USER`, and `SFM_EMAIL_PASSWORD`. (If the configured email account is hosted by Google, you will need to configure the account to "Allow less secure apps." Currently this setting is accessed, while logged in to the google account, via https://myaccount.google.com/security#connectedapps).
- Application credentials for social media APIs are configured in by providing the `TWITTER_CONSUMER_KEY`, `TWITTER_CONSUMER_SECRET`, `WEIBO_API_KEY`, `WEIBO_API_SECRET`, and/or `TUMBLR_CONSUMER_KEY`, `TUMBLR_CONSUMER_SECRET`. These are optional, but will make acquiring credentials easier for users. For more information and alternative approaches see *API Credentials*.
- Set an admin email address with `SFM_SITE_ADMIN_EMAIL`. Problems with SFM are sent to this address.
- Set an SFM contact email address with `SFM_CONTACT_EMAIL`. Users are provided with this address.
- For branding in the SFM UI, provide `SFM_INSTITUTION_NAME` and `SFM_INSTITUTION_LINK`.
- Provide a contact URL (e.g., http://library.gwu.edu) to be used when web harvesting with `HERITRIX_CONTACT_URL`.

Note that if you make a change to configuration *after* SFM is brought up, you will need to restart containers. If the change only applies to a single container, then you can stop the container with `docker kill <container name>`. If the change applies to multiple containers (or you're not sure), you can stop all containers with `docker-compose stop`. Containers can then be brought back up with `docker-compose up -d` and the configuration change will take effect.

Stopping

To stop the containers gracefully:

```
docker-compose stop -t 180
```

SFM can then be restarted with `docker-compose up -d`.

Server restarts

If Docker is configured to automatically start when the server starts, then SFM will start. (This is enabled by default when Docker is installed.)

SFM will even be started if it was stopped prior to the server reboot. If you do not want SFM to start, then configure Docker to not automatically start.

To configure whether Docker is automatically starts, see *Stopping Docker from automatically starting*.

Upgrading

Following are general instructions for upgrading SFM versions. Always consult the release notes of the new version to see if any additional steps are required.

1. Stop the containers gracefully:

```
docker-compose stop -t 180
```

This may take several minutes.

2. Make a copy of your existing `docker-compose.yml` and `.env` files:

```
cp docker-compose.yml old.docker-compose.yml
cp .env old.env
```

3. Get the latest `example.prod.docker-compose.yml`. If you previously cloned the sfm-docker repository then:

```
git pull
cp example.prod.docker-compose.yml docker-compose.yml
```

otherwise:

```
curl -L https://github.com/gwu-libraries/sfm-docker/raw/master/example.prod.docker-
↪compose.yml > docker-compose.yml
```

4. If you customized your previous `docker-compose.yml` file (e.g., for SFM ELK containers), make the same changes in your new `docker-compose.yml`.

5. Make any changes in your `.env` file prescribed by the release notes. This should include changing `SFM_VERSION`.

6. Bring up the containers:

```
docker-compose up -d
```

It may take several minutes for the images to be downloaded and the containers to start.

Server sizing

While we have not performed any system engineering analysis of optimal server sizing for SFM, the following are different configurations that we use:

Use	Server type	Processors	RAM (gb)
Production		6	16
Sandbox	m4.large (AWS)	2	8
Use in a class	m4.xlarge (AWS)	4	16
Continuous integration	t2.medium (AWS)	2	4
Heavy dataset processing	m4.4xlarge (AWS)	16	64
Development	Docker for Mac	2	3

CHAPTER 8

Monitoring

There are several mechanisms for monitoring (and troubleshooting) SFM.

For more information on troubleshooting, see *Troubleshooting*.

Monitor page

To reach the monitoring page, click "Monitor" on the header of any page in SFM UI.

The monitor page provides status and queue lengths for SFM components, including harvesters and exporters.

The status is based on the most recent status reported back by each harvester or exporter (within the last 3 days). A harvester or exporter reports its status when it begins a harvest or export. It also reports its status when it completes the harvest or exporter. Harvesters will also provide status updates periodically during a harvest.

Note that if there are multiple instances of a harvester or exporter (created with *docker-compose scale*), each instance will be listed.

The queue length lists the number of harvest or export requests that are waiting. A long queue length can indicate that additional harvesters or exporters are needed to handle the load (see *Scaling up with Docker*) or that there is a problem with the harvester or exporter.

The queue length for SFM UI is also listed. This is a queue of status update messages from harvesters or exporters. SFM UI uses these messages to update the records for harvests and exports. Any sort of a queue here indicates a problem.

Logs

It can be helpful to peek at the logs to get more detail on the work being performed by a harvester or exporter.

Docker logs

The logs for harvesters and exporters can be accessed using Docker's *log* commands.

First, determine the name of the harvester or exporter using `docker ps`. In general, the name will be something like *sfm_twitterrestharvester_1*.

Second, get the log with `docker logs <name>`.

Add *-f* to follow the log. For example, `docker logs -f sfm_twitterrestharvester_1`.

Add *–tail=<number of lines* to get the tail of the log. For example, `docker logs --tail=100 sfm_twitterrestharvester_1`.

Side note: To follow the logs of all services, use `docker-compose logs -f`.

Twitter Stream Harvester logs

Since the Twitter Stream Harvester runs multiple harvests on the same host, accessing its logs are a bit different.

First, determine the name of the Twitter Stream Harvester and the container id using `docker ps`. The name will probably be *sfm_twitterstreamharvester_1* and the container id will be something like *bffcae5d0603*.

Second, determine the harvest id. This is available from the harvest's detail page.

Third, get the stdout log with `docker exec -t <name> cat /sfm-data/containers/<container id>/log/<harvest id>.out.log`. To get the stderr log, substitute *.err* for *.out*.

To follow the log, use *tail -f* instead of *cat*. For example, `docker exec -t sfm_twitterstreamharvester_1 tail -f /sfm-data/containers/bffcae5d0603/log/d4493eed5f4f49c6a1981c89cb5d525f.err.log`.

Management consoles

Several of the services used by SFM offer management consoles that can be useful for monitoring.

For each of these, the username, password, and port are available from your *.env* file.

RabbitMQ

The RabbitMQ Admin is usually available on port 15672. For example, http://localhost:15672/.

Heritrix

The Heritrix management console is usually available on port 8443. For example, https://localhost:8443/.

Note that you must used HTTPS to reach the management console. You may be warned by your browser about the certificate; it is safe to proceed.

CHAPTER 9

Administration

Admin Interace

Designated users have access to SFM UI's Django Admin interface by selecting Welcome > Admin on the top right of the screen. This interface will allow adding, deleting, or changing database records for SFM UI. Some of the most salient uses for this capability are given below.

Managing groups

To allow for multiple users to control a collection set: 1. Create a new group. 2. Add users to the group. (This is done from the user's admin page, not the group's admin page.) 3. Assign the collection set to the group. This is done from the collection set detail page or from the collection set admin page.

Deleting items

Records can be deleted using the Admin Interface. It is recommended to minimize deletion; in particular, collections should be turned off and seeds made inactive.

Note the following when deleting: * Cascades delete, i.e., when a record is deleted any other records that depend on it will also be deleted. Before the deletion is performed, you will be informed what dependent records will be deleted. * When deleting collection sets, collections, harvests, WARCs, and exports *the corresponding files will be deleted*. Thus, if you delete a collection set *all* data and metadata will be deleted. **Be careful.**

Allowing access to Admin Interface

To allow a user to have access to the Admin Interface, give the user Staff status or Superuser status. This is done from the user's admin page.

CHAPTER 10

Authentication

Social Feed Manager allows users to self-sign up for accounts. Those accounts are stored and managed by SFM. Future versions of SFM will support authentication against external systems, e.g., Shibboleth.

By default, a group is created for each user and the user is placed in group. To create additional groups and modify group membership use the Admin interface.

In general, users and groups can be administered from the Admin interface.

The current version of SFM is not very secure. Future versions of SFM will more tightly restrict what actions users can perform and what they can view. In the meantime, it is encouraged to take other measures to secure SFM such as restricting access to the IP range of your institution.

CHAPTER 11

Docker

This page contains information about Docker that is useful for installation, administration, and development.

Installing Docker

Docker Engine and Docker Compose

On OS X:

- Install Docker for Mac.
- If you are using Docker Toolbox, switch to Docker for Mac.

On Ubuntu:

- If you have difficulties with the `apt` install, try the `pip` install.
- The docker group is automatically created. Adding your user to the docker group avoids having to use sudo to run docker commands. Note that depending on how users/groups are set up, you may need to manually need to add your user to the group in `/etc/group`.

While Docker is available on other platforms (e.g., Windows, Red Hat Enterprise Linux), the SFM team does not have any experience running SFM on those platforms.

Helpful commands

`docker-compose up -d` Bring up all of the containers specified in the docker-compose.yml file. If a container has not yet been pulled, it will be pulled. If a container has not yet been built it will be built. If a container has been stopped ("killed") it will be re-started. Otherwise, a new container will be created and started ("run").

`docker-compose pull` Pull the latest images for all of the containers specified in the docker-compose.yml file with the *image* field.

docker-compose build Build images for all of the containers specified in the docker-compose.yml file with the *build* field. Add `--no-cache` to re-build the entire image (which you might want to do if the image isn't building as expected).

docker ps List running containers. Add `-a` to also list stopped containers.

docker-compose kill Stop all containers.

docker kill <container name> Stop a single container.

docker-compose rm -v --force Delete the containers and volumes.

docker rm -v <container name> Delete a single container and volume.

docker rm $(docker ps -a -q) -v Delete all containers.

docker-compose logs List the logs from all containers. Add `-f` to follow the logs.

docker logs <container name> List the log from a single container. Add `-f` to follow the logs.

docker-compose -f <docker-compose.yml filename> <command> Use a different docker-compose.yml file instead of the default.

docker exec -it <container name> /bin/bash Shell into a container.

docker rmi <image name> Delete an image.

docker rmi $(docker images -q) Delete all images

docker-compose scale <service name>=<number of instances> Create multiple instances of a service.

Scaling up with Docker

Most harvesters and exporters handle one request at a time; requests for exports and harvests queue up waiting to be handled. If requests are taking too long to be processed you can scale up (i.e., create additional instances of) the appropriate harvester or exporter.

To create multiple instances of a service, use docker-compose scale.

The harvester most likely to need scaling is the Twitter REST harvester since some harvests (e.g., broad Twitter searches) may take a long time. To scale up the Twitter REST harvester to 3 instances use:

```
docker-compose scale twitterrestharvester=3
```

To spread containers across multiple containers, use Docker Swarm.

Using compose in production provides some additional guidance.

Stopping Docker from automatically starting

Docker automatically starts when the server starts. To control this:

Ubuntu 14 (Upstart)

Stop Docker from automatically starting:

```
echo manual | sudo tee /etc/init/docker.override
```

Allow Docker to automatically start:

```
sudo rm /etc/init/docker.override
```

Manually start Docker:

```
sudo service docker start
```

Ubuntu 16 (Systemd)

Stop Docker from automatically starting:

```
sudo systemctl disable docker
```

Allow Docker to automatically start:

```
sudo systemctl enable docker
```

Manually start Docker:

```
sudo systemctl start docker
```

CHAPTER 12

Collection set / Collection portability

Overview

Collections and collection sets are portable. That means they can be moved to another SFM instance or to another environment, such as a repository. This can also be used to backup an SFM instance.

A collection includes all of the social media items and web resources (stored in WARCs) and the database records for the collection sets, collections, users, groups, credentials, seeds, harvests, and WARCs, as well as the history of collection sets, collections, credentials, and seeds. The database records are stored in JSON format in the `records` subdirectory of the collection. Each collection has a complete set of JSON database records to support loading it into a different SFM instance.

Here are the JSON database records for an example collection:

```
[root@1da93afd43b5:/sfm-data/collection_set/4c59ebf2dcdc4a0e9660e32d004fa846/
↪072ff07ea9954b39a1883e979de92d22/records# ls
collection.json      groups.json     historical_collection.json      historical_seeds.
↪json  users.json
collection_set.json  harvest_stats.json  historical_collection_set.json  info.json  ␣
↪      warcs.json
credentials.json     harvests.json   historical_credentials.json     seeds.json
```

Thus, moving a collection set only requires moving/copying the collection set's directory; moving a collection only requires moving/copying a collection's directory. Collection sets are in `/sfm-data/collection_set` and are named by their collection set ids. Collections are subdirectories of their collection set and are named by their collection ids.

A `README.txt` is automatically created for each collection and collection set. Here a `README.txt` for an example collection set:

```
This is a collection set created with Social Feed Manager.

Collection set name: test collection set
Collection set id: 4c59ebf2dcdc4a0e9660e32d004fa846
```

```
This collection set contains the following collections:
* test twitter sample (collection id 59f9ff647ffd4fa28fd7e5bc4d161743)
* test twitter user timeline (collection id 072ff07ea9954b39a1883e979de92d22)


Each of these collections contains a README.txt.

Updated on Oct. 18, 2016, 3:09 p.m.
```

Preparing to move a collection set / collection

Nothing needs to be done to prepare a collection set or collection for moving. The collection set and collection directories contain all of the files required to load it into a different SFM instance.

The JSON database records are refreshed from the database on a nightly basis. Alternatively, they can be refreshed used the `serializecollectionset` and `serializecollection` management commands:

```
root@1da93afd43b5:/opt/sfm-ui/sfm# ./manage.py serializecollectionset 4c59ebf2d
```

Loading a collection set / collection

1. Move/copy the collection set/collection to `/sfm-data/collection_set`. Collection sets should be placed in this directory. Collections should be placed into a collection set directory.
2. Execute the `deserializecollectionset` or `deserializecollection` management command:

```
root@1da93afd43b5:/opt/sfm-ui/sfm# ./manage.py deserializecollectionset /sfm-data/
↪collection_set/4c59ebf2dcdc4a0e9660e32d004fa846
```

Note:

- If loading a collection set, all of the collection set's collections will also be loaded.
- When loading, all related items are also loaded. For example, when a collection is loaded, all of the seeds, harvests, credentials, and their histories are also loaded.
- If a database record already exists for a collection set, loading will not continue for the collection set or any of its collections or related records (e.g., groups).
- If a database record already exists for a collection, loading will not continue for the collection or any of the related records (e.g., users, harvests, WARCs).
- If a database record already exists for a user or group, it will not be loaded.
- Collections that are loaded are turned off.
- Users that are loaded are set to inactive.
- A history note is added to collection sets and collections to document the load.

Moving an entire SFM instance

1. Stop the source instance: `docker-compose stop`.

2. Copy the `/sfm-data` directory from the source server to the destination server.
3. If preserving processing data, also copy the `/sfm-processing` directory from the source server to the destination server.
4. Copy the `docker-compose.yml` and `.env` files from the source server to the destination server.
5. Make any changes necessary in the `.env` file, e.g., `SFM_HOSTNAME`.
6. Start the destination instance: `docker-compose up -d`.

If moving between AWS EC2 instances and `/sfm-data` is on a separate EBS volume, the volume can be detached from the source EC2 instances and attached to the destination EC2 instance.

CHAPTER 13

Storage

Storage volumes

SFM stores data on 2 volumes:

- sfm-data: The data volume is where SFM stores the harvested social media content and web resources, the db files, and exports. This is described in more detail below. It is available within containers as /sfm-data.
- sfm-processing: The processing volume is where processed data is stored when using a processing container. (See *Commandline exporting/processing*.) It is available within containers as /sfm-processing.

Volume types

There are 2 types of volumes:

- Internal to Docker. The files on the volume will only be available from within Docker containers.
- Linked to a host location. The files on the volumes will be available from within Docker containers and from the host operating system.

The type of volume is specified in the *.env* file. When selecting a link to a host location, the path on the host environment must be specified:

```
# Docker internal volume
DATA_VOLUME=/sfm-data
# Linked to host location
#DATA_VOLUME=/src/sfm-data:/sfm-data
# Docker internal volume
PROCESSING_VOLUME=/sfm-processing
# Linked to host location
#PROCESSING_VOLUME=/src/sfm-processing:/sfm-processing
```

We recommend that you use an internal volume only for development; for other uses linking to a host location is recommended. This make it easier to place the data on specific storage devices (e.g., NFS or EBS) and to backup the data.

File ownership

SFM files are owned by the sfm user (default uid 990) in the sfm group (default gid 990). If you use a link to a host location and list the files, the uid and gid may be listed instead of the user and group names.

If you shell into a Docker container, you will be the root user. Make sure that any operations you perform will not leave behind files that do not have appropriate permissions for the sfm user.

Note then when using Docker for Mac and linking to a host location, the file ownership may not appear as expected.

Directory structure of sfm-data

The following is a outline of the structure of sfm-data:

```
/sfm-data/
    collection_set/
        <collection set id>
            README.txt (README for collection set)
            <collection id>/
                README.txt (README for collection)
                state.json (Harvest state record)
                heritrix_job/
                    Web harvest state records
                records/
                    JSON records for the collection metadata
                <year>/<month>/<day>/<hour>/
                    WARC files
    containers/
        <container id>/
            Working files for individual containers
    elk/
        <container id>/
            ELK files
    export/
        <export id>/
            Export files
    postgresql/
        Postgres db files
```

Space warnings

SFM will monitor free space on sfm-data and sfm-processing. Administrators will be notified when the amount of free space crosses a configurable threshold. The threshold is set in the *.env* file:

```
# sfm-data free space threshold to send notification emails,only ends with MB,GB,TB.␣
↪eg. 500MB,10GB,1TB
DATA_VOLUME_THRESHOLD=10GB
```

```
# sfm-processing free space threshold to send notification emails,only ends with MB,
↪GB,TB. eg. 500MB,10GB,1TB
PROCESSING_VOLUME_THRESHOLD=10GB
```

Moving from a Docker internal volume to a linked volume

These instructions are for Ubuntu. They may need to be adjusted for other operating systems.

1. Stop docker containers:

```
docker-compose stop
```

2. Copy sfm-data contents from inside the container to a linked volume:

```
sudo docker cp sfm_data_1:/sfm-data /
```

3. Set ownership:

```
sudo chown -R 990:990 /sfm-data/*
```

4. Change .env:

```
#DATA_VOLUME=/sfm-data
DATA_VOLUME=/sfm-data:/sfm-data
```

5. Restart containers:

```
docker-compose up -d
```

CHAPTER 14

Limitations and Known Issues

To make sure you have the best possible experience with SFM, you should be aware of the limitations and known issues:

- SFM is does not currently run with HTTPS (Ticket #361).
- Because of the need to link a Heritrix container and a web harvester container, the web harvester cannot be scaled with `docker-compose scale command` (Ticket 408)
- Changes to the hostname of server (e.g., from the reboot of an AWS EC2 instance) are not handled (Ticket 435)

We are planning to address these in future releases. In the meantime, there are work-arounds for many of these issues. For a complete list of tickets, see https://github.com/gwu-libraries/sfm-ui/issues

In addition, you should be aware of the following:

- The current implementation of web harvesting is not optimal and requires significant additional work or reconsideration. In particular: (1) It does not scale: under normal collecting scenarios, web harvesting can lag far behind social media collecting. (2) It is not reliable: Heritrix requires more fiddling and testing.
- Access to the Weibo API is limited, so make sure you understand what can be collected.
- SFM does not currently provide a web interface for "replaying" the collected social media or web content.
- ELK is only experimental. Scaling and administration of ELK have not been considered.

CHAPTER 15

Troubleshooting

General tips

- Upgrade to the latest version of Docker and Docker-Compose.
- Make sure expected containers are running with `docker ps`.
- Check the logs with `docker-compose logs` and `docker logs <container name>`.
- Additional information is available via the admin interface that is not available from the UI. To access the admin interface, log in as an account that has superuser status and under "Welcome, <your name>," click Admin. By default, a superuser account called *sfmadmin* is created. The password can be found in `.env`.

Specific problems

Skipped harvests

A new harvest will not be requested if the previous harvest has not completed. Instead, a harvest record will be created with the status of skipped. Some of the reasons that this might happen include:

- Harvests are scheduled too closely together, such that the previous harvest cannot complete before the new harvest is requested.
- There are not enough running harvesters, such that harvest requests have to wait too long before being processed.
- There is a problem with harvesters, such that they are not processing harvest requests.
- Something else has gone wrong, and a harvest request was not completed.

After correcting the problem to resume harvesting for a collection, void the last (non-skipped) harvest. To void a harvest, go to that harvest's detail page and click the void button.

Connection errors when harvesting

If harvests from a container fail with something like:

```
HTTPSConnectionPool(host='api.flickr.com', port=443): Max retries exceeded with url: /
↪services/rest/?user_id=148553609%40N08&nojsoncallback=1&method=flickr.people.
↪getInfo&format=json (Caused by ProxyError('Cannot connect to proxy.', error('Tunnel␣
↪connection failed: 500 [Errno -3] Temporary failure in name resolution',)))
```

then stop and restart the container. For example:

```
docker-compose stop flickrharvester
docker-compose up -d
```

Bind error

If when bringing up the containers you receive something like:

```
ERROR: driver failed programming external connectivity on endpoint docker_sfmuiapp_1␣
↪(98caab29b4ba3c2b08f70fdebad847980d80a29a2c871164257e454bc582a060): Bind for 0.0.0.
↪0:8080 failed: port is already allocated
```

it means another application is already using a port configured for SFM. Either shut down the other application or choose a different port for SFM. (Chances are the other application is Apache.)

Bad Request (400)

If you receive a Bad Request (400) when trying to access SFM, your `SFM_HOST` environment variable is not configured correctly. For more information, see ALLOWED_HOSTS.

Social Network Login Failure for Twitter

If you receive a Social Network Login Failure when trying to connect a Twitter account, make sure that the Twitter app from which you got the Twitter credentials is configured with a callback URL. The URL you provide doesn't matter.

Docker problems

If you are having problems bringing up the Docker containers (e.g., `driver failed programming external connectivity on endpoint`), restart the Docker service. On Ubuntu, this can be done with:

```
# service docker stop
docker stop/waiting
# service docker status
docker stop/waiting
# service docker start
docker start/running, process 15039
```

Web harvesting / Heritrix problems

If you are encountering problems with web harvesting, check the logs of the web harvester container (`docker-compose logs webharvester`) and the heritrix container (`docker-compose logs heritrix`).

If you see a line like `heritrix:8443 not available after wait.` in the web harvester logs and various Java exceptions in the heritrix container logs then kill, remove, and restart the containers:

```
docker-compose kill webharvester heritrix
docker-compose rm -vf webharvester heritrix
docker-compose up -d
```

CSV export problems

Excel for Mac has problems with unicode characters in CSV files. As a work-around, export to Excel (XLSX) format.

Still stuck?

Contact the SFM team. We're happy to help.

CHAPTER 16

Development

Setting up a development environment

SFM is composed of a number of components. Development can be performed on each of the components separately.

For SFM development, it is recommended to run components within a Docker environment (instead of directly in your OS, without Docker).

Step 1: Install Docker and Docker Compose

See *Installing Docker*.

Step 2: Clone sfm-docker and create copies of docker-compose files

For example:

```
git clone https://github.com/gwu-libraries/sfm-docker.git
cd sfm-docker
cp example.docker-compose.yml docker-compose.yml
cp example.env .env
```

For the purposes of development, you can make changes to `docker-compose.yml` and `.env`. This will be described more below.

Step 3: Clone the component repos

For example:

```
git clone https://github.com/gwu-libraries/sfm-ui.git
```

Repeat for each of the components that you will be working on. Each of these should be in a sibling directory of sfm-docker.

Running SFM for development

To bring up an instance of SFM for development, change to the sfm-docker directory and execute:

```
docker-compose up -d
```

You may not want to run all of the containers. To omit a container, simply comment it out in `docker-compose.yml`.

By default, the code that has been committed to master for each of the containers will be executed. To execute your local code (i.e., the code you are editing), you will want to link in your local code. To link in the local code for a container, uncomment the volume definition that points to your local code. For example:

```
volumes:
    - "../sfm-twitter-harvester:/opt/sfm-twitter-harvester"
```

sfm-utils and warcprox are dependencies of many components. By default, the code that has been committed to master for sfm-utils or warcprox will be used for a component. To use your local code as a dependency, you will want to link in your local code. Assuming that you have cloned sfm-utils and warcprox, to link in the local code as a dependency for a container, change `SFM_REQS` in `.env` to "dev" and comment the volume definition that points to your local code. For example:

```
volumes:
    - "../sfm-twitter-harvester:/opt/sfm-twitter-harvester"
    - "../sfm-utils:/opt/sfm-utils"
    - "../warcprox:/opt/warcprox"
```

Note: * As a Django application, SFM UI will automically detect code changes and reload. Other components must be killed and brought back up to reflect code changes.

Running tests

Unit tests

Some components require a `test_config.py` file that contains credentials. For example, sfm-twitter-harvester requires a `test_config.py` containing:

```
TWITTER_CONSUMER_KEY = "EHdoTksBfgGflP5nUalEfhaeo"
TWITTER_CONSUMER_SECRET = "ZtUpemtBkf2cEmaqiy52Dd343ihFu9PAiLebuMOmqN0QtXeAlen"
TWITTER_ACCESS_TOKEN = "411876914-c2yZjbk1np0Z5MWEFYYQKSQNFFGBXd8T4k90YkJ1"
TWITTER_ACCESS_TOKEN_SECRET = "jK9QOmn5VRF5mfgAN6KgfmCKRqThXVQ1G6qQg8BCejvp"
```

Note that if this file is not present, unit tests that require it will be skipped. Each component's README will describe the `test_config.py` requirements.

Unit tests for most components can be run with:

```
python -m unittest discover
```

The notable exception is SFM UI, which can be run with:

```
cd sfm
./manage.py test --settings=sfm.settings.test_settings
```

Integration tests

Many components have integration tests, which are run inside docker containers. These components have a `ci.docker-compose.yml` file which can be used to bring up a minimal environment for running the tests.

As described above, some components require a `test_config.py` file.

To run integration tests, bring up SFM:

```
docker-compose -f docker/dev.docker-compose.yml up -d
```

Run the tests:

```
docker exec docker_sfmtwitterstreamharvester_1 python -m unittest discover
```

You will need to substitute the correct name of the container. (`docker ps` will list the containers.)

And then clean up:

```
docker-compose -f docker/dev.docker-compose.yml kill
docker-compose -f docker/dev.docker-compose.yml rm -v --force
```

For reference, see each component's `.travis.yml` file which shows the steps of running the integration tests.

Smoke tests

sfm-docker contains some smoke tests which will verify that SFM is running correctly.

To run the smoke tests, first bring up SFM:

```
docker-compose up -d
```

and then run the tests:

```
docker-compose -f docker-compose.yml -f smoketests.docker-compose.yml run --rm␣
↪smoketests python -m unittest discover
```

Note that the smoke tests are not yet complete.

For reference, the continuous integration deploy instructions shows the steps of running the smoke tests.

Requirements files

This will vary a depending on whether a project has warcprox and sfm-utils as a dependency, but in general:

- `requirements/common.txt` contains dependencies, except warcprox and sfm-utils.
- `requirements/release.txt` references the last released version of warcprox and sfm-utils.
- `requirements/master.txt` references the master version of warcprox and sfm-utils.
- `requirements/dev.txt` references local versions of warcprox and sfm-utils in development mode.

To get a complete set of dependencies, you will need `common.txt` and either `release.txt`, `master.txt` or `dev.txt`. For example:

```
virtualenv ENV
source ENV/bin/activate
pip install -r requirements/common.txt -r requirements/dev.txt
```

Development tips

Admin user accounts

Each component should automatically create any necessary admin accounts (e.g., a django admin for SFM UI). Check `.env` for the username/passwords for those accounts.

RabbitMQ management console

The RabbitMQ management console can be used to monitor the exchange of messages. In particular, use it to monitor the messages that a component sends, create a new queue, bind that queue to *sfm_exchange* using an appropriate routing key, and then retrieve messages from the queue.

The RabbitMQ management console can also be used to send messages to the exchange so that they can be consumed by a component. (The exchange used by SFM is named *sfm_exchange*.)

For more information on the RabbitMQ management console, see *RabbitMQ*.

Blocked ports

When running on a remote VM, some ports (e.g., 15672 used by the RabbitMQ management console) may be blocked. SSH port forwarding can help make those ports available.

Django logs

Django logs for SFM UI are written to the Apache logs. In the docker environment, the level of various loggers can be set from environment variables. For example, setting *SFM_APSCHEDULER_LOG* to *DEBUG* in the *docker-compose.yml* will turn on debug logging for the apscheduler logger. The logger for the SFM UI application is called ui and is controlled by the *SFM_UI_LOG* environment variable.

Apache logs

In the SFM UI container, Apache logs are sent to stdout/stderr which means they can be viewed with *docker-compose logs* or *docker logs <container name or id>*.

Initial data

The development and master docker images for SFM UI contain some initial data. This includes a user ("testuser", with password "password"). For the latest initial data, see *fixtures.json*. For more information on fixtures, see the Django docs.

Runserver

There are two flavors of the the development docker image for SFM UI. *gwul/sfm-ui:master* runs SFM UI with Apache, just as it will in production. *gwul/sfm-ui:master-runserver* runs SFM UI with runserver, which dynamically reloads changed Python code. To switch between them, change *UI_TAG* in *.env*.

Note that as an byproduct of how runserver dynamically reloads Python code, there are actually 2 instances of the application running. This may produce some odd results, like 2 schedulers running. This will not occur with Apache.

Job schedule intervals

To assist with testing and development, a 5 minute interval can be added by setting *SFM_FIVE_MINUTE_SCHEDULE* to *True* in the *docker-compose.yml*.

Connecting to the database

To connect to postgres using psql:

```
docker exec -it sfm_db_1 psql -h db -U postgres -d sfmdatabase
```

You will be prompted for the password, which you can find in *.env*.

Docker tips

Building vs. pulling

Containers are created from images. Images are either built locally or pre-built and pulled from Docker Hub. In both cases, images are created based on the docker build (i.e., the Dockerfile and other files in the same directory as the Dockerfile).

In a docker-compose.yml, pulled images will be identified by the *image* field, e.g., *image: gwul/sfm-ui:master*. Built images will be identified by the *build* field, e.g., *build: app-dev*.

In general, you will want to use pulled images. These are automatically built when changes are made to the Github repos. You should periodically execute *docker-compose pull* to make sure you have the latest images.

You may want to build your own image if your development requires a change to the docker build (e.g., you modify fixtures.json).

Killing, removing, and building in development

Killing a container will cause the process in the container to be stopped. Running the container again will cause process to be re-started. Generally, you will kill and run a development container to get the process to be run with changes you've made to the code.

Removing a container will delete all of the container's data. During development, you will remove a container to make sure you are working with a clean container.

Building a container creates a new image based on the Dockerfile. For a development image, you only need to build when making changes to the docker build.

CHAPTER 17

Writing a harvester

Requirements

- Implement the *Messaging Specification* for harvesting social media content. This describes the messages that must be consumed and produced by a harvester.
- Write harvested social media to a WARC, following all relevant guidelines and best practices. The message for announcing the creation of a WARC is described in the Messaging Specification. The WARC file must be written to *<base path>/<harvest year>/<harvest month>/<harvest day>/<harvest hour>/*, e.g., */data/test_collection_set/2015/09/12/19/*. (Base path is provided in the harvest start message.) Any filename may be used but it must end in *.warc* or *.warc.gz*. It is recommended that the filename include the harvest id (with file system unfriendly characters removed) and a timestamp of the harvest.
- Extract urls for related content from the harvested social media content, e.g., a photo included in a tweet. The message for publishing the list of urls is described in the Messaging Specification.
- Document the harvest types supported by the harvester. This should include the identifier of the type, the API methods called, the required parameters, the optional parameters, what is included in the summary, and what urls are extracted. See the Flickr Harvester as an example.
- The smoke tests must be able to prove that a harvester is up and running. At the very least, the smoke tests should check that the queues required by a harvester have been created. (See test_queues().)
- Be responsible for its own state, e.g., keeping track of the last tweet harvested from a user timeline. See sfmutils.state_store for re-usable approaches to storing state.
- Create all necessary exchanges, queues, and bindings for producing and consuming messages as described in *Messaging*.
- Provide master and production Docker images for the harvester on Docker Hub. The master image should have the *master* tag and contain the latest code from the master branch. (Setup an automated build to simplify updating the master image.) There must be a version specific production images, e.g., *1.3.0* for each release. For example, see the Flickr Harvester's dockerfiles and Docker Hub repo.

Suggestions

- See sfm-utils for re-usable harvester code. In particular, consider subclassing BaseHarvester.
- Create a development Docker image. The development Docker images links in the code outside of the container so that a developer can make changes to the running code. For example, see the Flickr harvester development image.
- Create a development *docker-compose.yml*. This should include the development Docker image and only the additional images that the harvester depends on, e.g., a Rabbit container. For example, see the Flickr harvester development docker-compose.yml.
- When possible, use existing API libraries.
- Consider write integration tests that test the harvester in an integration test environment. (That is, an environment that includes the other services that the harvester depends on.) For example, see the Flickr Harvester's integration tests.
- See the Twitter harvester unit tests for a pattern on configuring API keys in unit and integration tests.

Notes

- Harvesters can be written in any programming language.
- Changes to gwu-libraries/* repos require pull requests. Pull requests are welcome from non-GWU developers.

CHAPTER 18

Messaging

RabbitMQ

RabbitMQ is used as a message broker.

The RabbitMQ managagement console is exposed at `http://<your docker host>:15672/`. The username is `sfm_user`. The password is the value of `RABBITMQ_DEFAULT_PASS` in `secrets.env`.

Publishers/consumers

- The hostname for RabbitMQ is `mq` and the port is 5672.
- It cannot be guaranteed that the RabbitMQ docker container will be up and ready when any other container is started. Before starting, wait for a connection to be available on port 5672 on `rabbit`. See appdeps.py for docker application dependency support.
- Publishers/consumers may not assume that the requisite exchanges/queues/bindings have previously been created. They must declare them as specified below.

Exchange

`sfm_exchange` is a durable topic exchange to be used for all messages. All publishers/consumers must declare it.:

```
#Declare sfm_exchange
from kombu import Connection

exchange = Exchange(name="sfm_exchange,
                    type="topic", durable=True)
exchange(channel).declare()
```

Queues

All queues must be declared durable.:

```
#Declare harvester queue
from kombu import Queue
queue = Queue(name="harvester",
              exchange=exchange,
              channel=channel,
              durable=True)
queue.declare()
queue.bind_to(exchange=exchange,
              routing_key="harvest.status.*.*")
```

CHAPTER 19

Messaging Specification

Introduction

SFM is architected as a number of components that exchange messages via a messaging queue. To implement functionality, these components send and receive messages and perform certain actions. The purpose of this document is to describe this interaction between the components (called a "flow") and to specify the messages that they will exchange.

Note that as additional functionality is added to SFM, additional flows and messages will be added to this document.

General

- Messages may include extra information beyond what is specified below. Message consumers should ignore any extra information.
- RabbitMQ will be used for the messaging queue. See the Messaging docs for additional information. It is assumed in the flows below that components receive messages by connecting to appropriately defined queues and publish messages by submitting them to the appropriate exchange.

Harvesting social media content

Harvesting is the process of retrieving social media content from the APIs of social media services and writing to WARC files. It also includes extracting urls for other web resources from the social media so that they can be harvested by a web harvester. (For example, the link for an image may be extracted from a tweet.)

Background information

- A requester is an application that requests that a harvest be performed. A requester may also want to monitor the status of a harvest. In the current architecture, the SFM UI serves the role of requester.

- A stream harvest is a harvest that is intended to continue indefinitely until terminated. A harvest of a Twitter public stream is an example of a stream harvest. A stream harvest is different from a non-stream harvest in that a requester must both start and optionally stop a stream harvest. Following the naming conventions from Twitter, a harvest of a REST, non-streaming API will be referred to as a REST harvest.
- Depending on the implementation, a harvester may produce a single warc or multiple warcs. It is likely that in general stream harvests will result in multiple warcs, but REST harvest will result in a single warc.

Flow

The following is the flow for a harvester performing a REST harvest and creating a single warc:

1. Requester publishes a harvest start message.
2. Upon receiving the harvest message, a harvester:
   (a) Makes the appropriate api calls.
   (b) Extracts urls for web resources from the results.
   (c) Writes the api calls to a warc.
3. Upon completing the api harvest, the harvester:
   (a) Publishes a web harvest message containing the extracted urls.
   (b) Publishes a warc created message.
   (c) Publishes a harvest status message with the status of *completed success* or *completed failure*.

The following is the message flow for a harvester performing a stream harvest and creating multiple warcs:

1. Requester publishes a harvest start message.
2. Upon receiving the harvest message, a harvester:
   (a) Opens the api stream.
   (b) Extracts urls for web resources from the results.
   (c) Writes the stream results to a warc.
3. When rotating to a new warc, the harvester publishes a warc created message.
4. At intervals during the harvest, the harvester:
   (a) Publishes a web harvest message containing extracted urls.
   (b) Publishes a harvest status message with the status of *running*.
5. When ready to stop, the requester publishes a harvest stop message.
6. Upon receiving the harvest stop message, the harvester:
   (a) Closes the api stream.
   (b) Publishes a final web harvest message containing extracted urls.
   (c) Publishes a final warc created message.
   (d) Publishes a final harvest status message with the status of *completed success* or *completed failure*.

- Any harvester may send harvest status messages with the status of *running* before the final harvest status message. A harvester performing a stream harvest must send harvest status messages at regular intervals.
- A requester should not send harvest stop messages for a REST harvest. A harvester performing a REST harvest may ignore harvest stop messages.

Messages

Harvest start message

Harvest start messages specify for a harvester the details of a harvest. Example:

```
{
    "id": "sfmui:45",
    "type": "flickr_user",
    "path": "/sfm-data/collections/3989a5f99e41487aaef698680537c3f5/
↪6980fac666c54322a2ebdbcb2a9510f5",
    "seeds": [
        {
            "id": "a36fe186fbfa47a89dbb0551e1f0f181",
            "token": "justin.littman",
            "uid": "131866249@N02"
        },
        {
            "id": "ab0a4d9369324901a890ec85f00194ac",
            "token": "library_of_congress"
        }
    ],
    "options": {
        "sizes": ["Thumbnail", "Large", "Original"]
    },
    "credentials": {
        "key": "abddfe6fb8bba36e8ef0278ec65dbbc8",
        "secret": "1642649c54cc3ebe"
    },
    "collection_set": {
        "id": "3989a5f99e41487aaef698680537c3f5"
    },
    "collection": {
        "id": "6980fac666c54322a2ebdbcb2a9510f5"
    }
}
```

Another example:

```
{
    "id": "test:1",
    "type": "twitter_search",
    "path": "/sfm-data/collections/3989a5f99e41487aaef698680537c3f5/
↪6980fac666c54322a2ebdbcb2a9510f5",
    "seeds": [
        {
            "id": "32786222ef374eb38f1c5d56321c99e8",
            "token": "gwu"
        },
        {
            "id": "0e789cddd0fb41b5950f569676702182",
            "token": "gelman"
        }
    ],
    "credentials": {
        "consumer_key": "EHde7ksBGgflbP5nUalEfhaeo",
        "consumer_secret": "ZtUpemtBkf2maqFiy52D5dihFPAiLebuMOmqN0jeQtXeAlen",
        "access_token": "481186914-c2yZjgbk13np0Z5MWEFQKSQNFBXd8T9r4k90YkJl",
```

```
        "access_token_secret": "jK9QOmn5Vbbmfg2ANT6KgfmKRqV8ThXVQ1G6qQg8BCejvp"
    },
    "collection_set": {
        "id": "3989a5f99e41487aaef698680537c3f5"
    },
    "collection": {
        "id": "6980fac666c54322a2ebdbcb2a9510f5"
    }
}
```

- The routing key will be *harvest.start.<social media platform>.<type>*. For example, *harvest.start.flickr.flickr_photo*.
- *id*: A globally unique identifier for the harvest, assigned by the requester.
- *type*: Identifies the type of harvest, including the social media platform. The harvester can use this to map to the appropriate api calls.
- *seeds*: A list of seeds to harvest. Each seed is represented by a map containing *id*, *token* and (optionally) *uid*. Note that some harvest types may not have seeds.
- *options*: A name/value map containing additional options for the harvest. The contents of the map are specific to the type of harvest. (That is, the seeds for a flickr photo are going to be different than the seeds for a twitter user timeline.)
- *credentials*: All credentials that are necessary to access the social media platform. Credentials is a name/value map; the contents are specific to a social media platform.
- *path*: The base path for the collection.

Web resource harvest start message

Harvesters will extract urls from the harvested social media content and publish a web resource harvest start message. This message is similar to other harvest start messages, with the differences noted below. Example:

```
{
    "id": "flickr:45",
    "parent_id": "sfmui:45",
    "type": "web",
    "path": "/sfm-data/collections/3989a5f99e41487aaef698680537c3f5/
↪6980fac666c54322a2ebdbcb2a9510f5",
    "seeds": [
        {
            "id": "3724fd97e85345ee84f5175eee09748d",
            "token": "http://www.gwu.edu/"
        },
        {
            "id": "aba6033aafce4fbabd846026ca47f13e",
            "token": "http://library.gwu.edu/"
        }
    ],
    "collection_set": {
        "id": "3989a5f99e41487aaef698680537c3f5"
    },
    "collection": {
        "id": "6980fac666c54322a2ebdbcb2a9510f5"
    }
}
```

- The routing key will be *harvest.start.web*.
- *parent_id*: The id of the harvest from which the urls were extracted.

Harvest stop message

Harvest stop messages tell a harvester perform a stream harvest to stop. Example:

```
{
    "id": "sfmui:45"
}
```

- The routing key will be *harvest.stop.<social media platform>.<type>*. For example, *harvest.stop.twitter.filter*.

Harvest status message

Harvest status messages allow a harvester to provide information on the harvests it performs. Example:

```
{
    "id": "sfmui:45"
    "status": "completed success",
    "date_started": "2015-07-28T11:17:36.640044",
    "date_ended": "2015-07-28T11:17:42.539470",
    "infos": []
    "warnings": [],
    "errors": [],
    "stats": {
        "2016-05-20": {
            "photos": 12,
        },
        "2016-05-21": {
            "photos": 19,
        },
    },
    "token_updates": {
        "a36fe186fbfa47a89dbb0551e1f0f181": "j.littman"
    },
    "uids": {
        "ab0a4d9369324901a890ec85f00194ac": "671366249@N03"
    },
    "warcs": {
        "count": 3
        "bytes": 345234242
    },
    "service": "Twitter Harvester",
    "host": "f0c3c5ef7031",
    "instance": "39",
}
```

- The routing key will be *harvest.status.<social media platform>.<type>*. For example, *harvest.status.flickr.flickr_photo*.
- *status*: Valid values are *completed success*, *completed failure*, or *running*.
- *infos*, *warnings*, and *errors*: Lists of messages. A message should be an object (i.e., dictionary) containing a *code* and *message* entry. It may optionally contain a *seed_id* entry giving the seed id to which the messages applies. Codes should be consistent to allow message consumers to identify types of messages.

- *stats*: A count of items that are harvested by date. Items should be a human-understandable labels (plural and lower-cased). Stats is optional for in progress statuses, but required for final statuses.
- *token_updates*: A map of uids to tokens for which a token change was detected while harvesting. For example, for Twitter a token update would be provided whenever a user's screen name changes.
- *uids*: A map of tokens to uids for which a uid was identified while harvesting at not provided in the harvest start message. For example, for Flickr a uid would be provided containing the NSID for a username.
- *warcs*.'count': The total number of WARCs created during this harvest.
- *warcs*.'bytes': The total number of bytes of the WARCs created during this harvest.
- *service*, *host*, and *instance* identify what performed the harvest. *service* is the name of the harvester. *host* is the Docker container id. *instance* is the harvest process identifier (PID) within the container. This is useful in cases where there are multiple instances of a service on a host.

Warc created message

Warc created message allow a harvester to provide information on the warcs that are created during a harvest. Example:

```
{
    "warc": {
        "path": "/sfm-data/collections/3989a5f99e41487aaef698680537c3f5/
↪6980fac666c54322a2ebdbcb2a9510f5/2015/07/28/11/harvest_id-2015-07-28T11:17:36Z.warc.
↪gz",,
        "sha1": "7512e1c227c29332172118f0b79b2ca75cbe8979",
        "bytes": 26146,
        "id": "aba6033aafce4fbabd846026ca47f13e",
        "date_created": "2015-07-28T11:17:36.640178"
    },
    "collection_set": {
        "id": "3989a5f99e41487aaef698680537c3f5"
    },
    "collection": {
        "id": "6980fac666c54322a2ebdbcb2a9510f5"
    },
    "harvest": {
        "id": "98ddaa6e8c1f4b44aaca95bc46d3d6ac",
        "type": "flickr_user"
    }
}
```

- The routing key will be *warc_created*.
- Each warc created message will be for a single warc.

Exporting social media content

Exporting is the process of extracting social media content from WARCs and writing to export files. The exported content may be a subset or derivate of the original content. A number of different export formats will be supported.

Background information

- A requester is an application that requests that an export be performed. A requester may also want to monitor the status of an export. In the current architecture, the SFM UI serves the role of requester.

- Depending on the nature of the export, a single or multiple files may be produced.

Flow

The following is the flow for an export:

1. Requester publishes an export start message.
2. Upon receiving the export start message, an exporter:
   (a) Makes calls to the SFM REST API to determine the WARC files from which to export.
   (b) Limits the content is specified by the export start message.
   (c) Writes to export files.
3. Upon completing the export, the exporter publishes an export status message with the status of *completed success* or *completed failure*.

Export start message

Export start messages specify the requests for an export. Example:

```
{
    "id": "f3ddcbfc5d6b43139d04d680d278852e",
    "type": "flickr_user",
    "collection": {
        "id": "005b131f5f854402afa2b08a4b7ba960"
    },
    "path": "/sfm-data/exports/45",
    "format": "csv",
    "dedupe": true,
    "segment_size": 100000,
    "item_date_start": "2015-07-28T11:17:36.640178",
    "item_date_end": "2016-07-28T11:17:36.640178",
    "harvest_date_start": "2015-07-28T11:17:36.640178",
    "harvest_date_end": "2016-07-28T11:17:36.640178"
}
```

Another example:

```
{
    "id": "f3ddcbfc5d6b43139d04d680d278852e",
    "type": "flickr_user",
    "seeds": [
        {
            "id": "48722ac6154241f592fd74da775b7ab7",
            "uid": "23972344@N05"
        },
        {
            "id": "3ce76759a3ee40b894562a35359dfa54",
            "uid": "85779209@N08"
        }
    ],
    "path": "/sfm-data/exports/45",
    "format": "json",
    "segment_size": null
}
```

- The routing key will be *export.start.<social media platform>.<type>*. For example, *export.start.flickr.flickr_user*.
- *id*: A globally unique identifier for the harvest, assigned by the requester.
- *type*: Identifies the type of export, including the social media platform. The export can use this to map to the appropriate export procedure.
- *seeds*: A list of seeds to export. Each seed is represented by a map containing *id* and *uid*.
- *collection*: A map containing the *id* of the collection to export.
- Each export start message must have a *seeds* or *collection* but not both.
- *path*: A directory into which the export files should be placed. The directory may not exist.
- *format*: A code for the format of the export. (Available formats may change.)
- *dedupe*: If true, duplicate social media content should be removed.
- *item_date_start* and *item_date_end*: The date of social media content should be within this range.
- *harvest_date_start* and *harvest_date_end*: The harvest date of social media content should be within this range.
- *segment_size*: Maximum number of items to include in a single file. *null* means that all items should be placed in a single file.

Export status message

Export status messages allow an exporter to provide information on the exports it performs. Example:

```
{
    "id": "f3ddcbfc5d6b43139d04d680d278852e"
    "status": "completed success",
    "date_started": "2015-07-28T11:17:36.640044",
    "date_ended": "2015-07-28T11:17:42.539470",
    "infos": []
    "warnings": [],
    "errors": [],
    "service": "Twitter Harvester",
    "host": "f0c3c5ef7031",
    "instance": "39",
}
```

- The routing key will be *export.status.<social media platform>.<type>*. For example, *export.status.flickr.flickr_user*.
- *status*: Valid values are *running*, *completed success* or *completed failure*.
- *infos*, *warnings*, and *errors*: Lists of messages. A message should be an object (i.e., dictionary) containing a *code* and *message* entry. Codes should be consistent to allow message consumers to identify types of messages.
- *service*, *host*, and *instance* identify what performed the harvest. *service* is the name of the harvester. *host* is an identifier for the location of the harvest, e.g., the Docker container id. *instance* is an identifier for the process of the service on the host, e.g., the PID. The is helps in cases there may be multiple instances of a service on a host.

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Funding history

- Development of this project has been supported by a grant (#NARDI-14-50017-14) from the National Historical Publications & Records Commission to George Washington University Libraries from 2014-2017.
- Development of the Sina Weibo harvester is supported by a grant from the Council on East Asian Libraries.
- **Prior development of SFM under the previous repository** was supported by a grant (#LG-46-13-0257-13) from the Institute of Museum and Library Services to George Washington University Libraries from 2013-2014.