sfm Documentation

Release 2.1.0

The George Washington University Libraries

Jan 03, 2019

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.

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

User Guide

Welcome to Social Feed Manager!

Social Feed Manager (SFM) is an open-source tool designed for researchers, archivists, and curious individuals to collect social media data from Twitter, Tumblr, Flickr, or Sina Weibo. See the SFM Overview for a quick look at SFM.

If you want to learn more about what SFM can do, read *What is SFM used for?* This guide is for users who have access to SFM and want to learn how to collect. If you're an administrator setting up SFM for your institution, see admin-documentation.

To get your first collection up and running:

- **Sign up**: On the SFM homepage, click "Sign up." Fill out the form, including a unique email. Once you sign up, you will be automatically logged in.
- **Get credentials**: You'll need to authorize access to the social media platforms using credentials. See *Setting up Credentials*.
- **Create a collection set** and within it a collection, where you'll actually collect data. See *Creating Collections*.
- **Add seeds**: Seeds are the criteria used to collect data. You'll add user accounts or search criteria. See *Adding Seeds*.
- **Set your collections running!**
- **Export your collections** when you want to see and work with your data, or adjust settings. See *Exporting your Data*.

You can always come back to this user guide for help by clicking *Documentation* at the bottom of any SFM page and selecting *User Guide*.

1.1 What is SFM used for?

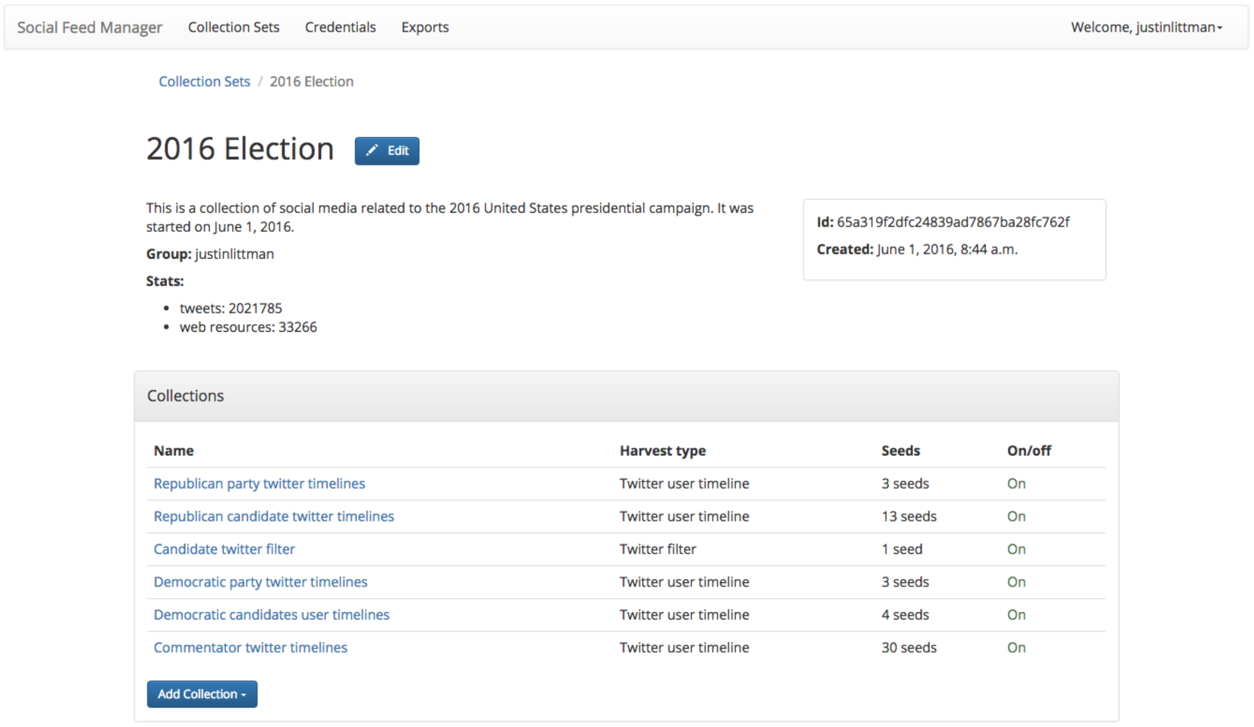
Social Feed Manager (SFM) collects individual posts–tweets, photos, blogs–from social media sites. These posts are collected in their native, raw data format called JSON and can be exported in many formats, including spreadsheets. Users can then use this collected data for research, analysis or archiving.

Some ideas for how to use SFM:

- **Collecting from individual accounts** such as the tweets of every U.S. Senator (*Twitter user timeline*).
- **Gathering Flickr images for analysis** or archiving the photographs from accounts donated to your organization (*Flickr user*).
- **Researching social media use** by retrieving a sample of all tweets (*Twitter sample*), or by filtering by specific search terms (*Twitter filter*).
- **Capturing a major event** by collecting tweets in a specific geographic location or by following specific hashtags.
- **Collecting Tumblr posts** for preserving institutional blogs or the work of online artists. (*Tumblr blog posts*).
- **Archiving posts** from any social media platform for later research.

Note that SFM currently collects social media data from Twitter, Tumblr, Flickr, and Sina Weibo.

Here's a sample of what a collection set looks like:



2016 Election

This is a collection of social media related to the 2016 United States presidential campaign. It was started on June 1, 2016.

Group: justinlittman

Stats:

- tweets: 2021785
- web resources: 33266

Id: 65a319f2dfc24839ad7867ba28fc762f

Created: June 1, 2016, 8:44 a.m.

Collections

Name	Harvest type	Seeds	On/off
Republican party twitter timelines	Twitter user timeline	3 seeds	On
Republican candidate twitter timelines	Twitter user timeline	13 seeds	On
Candidate twitter filter	Twitter filter	1 seed	On
Democratic party twitter timelines	Twitter user timeline	3 seeds	On
Democratic candidates user timelines	Twitter user timeline	4 seeds	On
Commentator twitter timelines	Twitter user timeline	30 seeds	On

1.1.1 Types of Collections

- *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
- *Tumblr blog posts*: Collects blog posts from specific Tumblr blogs

1.1.2 How to use the data

Once you've collected data, there are a few ways to use it:

- You could export it into a CSV or Excel format for a basic analysis (*Exporting your Data*), or load the format into analysis software such as Stata, SPSS, or Gephi.
- You could set up an archive using the JSON files or Excel files.

1.1.3 Privacy and platform policy considerations

Collecting and using data from social media platforms is subject to those platforms' terms (Twitter, Flickr, Sina Weibo, Tumblr), as you agreed to them when you created your social media account. Social Feed Manager respects those platforms' terms as an application (Twitter, Flickr, Sina Weibo, Tumblr).

Social Feed Manager provides data to you for your research and academic use. Social media platforms' terms of service generally do not allow republishing of full datasets, and you should refer to their terms to understand what you may share. Authors typically retain rights and ownership to their content.

Ethical considerations

In addition to respecting the platforms' terms, as a user of Social Feed Manager and data collected within it, it is your responsibility to consider the ethical aspects of collecting and using social media data. Your discipline or professional organization may offer guidance. In addition, take a look at these social media research ethical and privacy guidelines.

1.2 Setting up Credentials

Before you can start collecting, you need **credentials** for the social media platform that you want to use. Credentials are keys used by each platform to control the data they release to you.

You are responsible for creating your own credentials so that you can control your own collection rate and make sure that you are following the policies of each platform.

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

Adding Twitter Credentials
Adding Flickr Credentials
Adding Tumblr Credentials
Adding Weibo Credentials

1.3 Creating Collections

Collections are the basic SFM containers for social media data. Each collection either gathers posts from individual accounts or gathers posts based on search criteria.

Collections are contained in **collection sets**. While collection sets sometimes only include one collection, sets can be used to organize all of the data from a single project or archive–for example, a collection set about a band might include a collection of the Twitter user timelines of each band member, a collection of the band's Flickr, and a Twitter Filter collection of tweets that use the band's hashtag.

Before you begin collecting, you may want to consider these collection development guidelines.

1.3.1 Setting up Collections and Collection Sets

Because collections are housed in collection sets, you must make a collection set first.

Navigate to the Collection Sets page from the top menu, then click the *Add Collection Set* button.

Give the collection set a unique name and description. A collection set is like a folder for all collections in a project.

If you are part of a group project, you can contact your SFM administrator and set up a new group which you can share each collection set with. (This can be changed or added later on).

Once you are in a collection set, click the "Add Collection" dropdown menu and select the collection type you want to add.

Enter a unique collection name and a short description. The description is a great location to describe how you chose what to put in your collection.

Select which credential you want to use. If you need to set up new credentials, see *Setting up Credentials*.

1.3.2 Adding Seeds

Seeds are the criteria used by SFM to collect social media posts. Seeds may be individual social media accounts or search terms used to filter posts.

The basic process for adding seeds is the same for every collection type, except for Twitter Sample and Sina Weibo:

- Turn off the collection.
- Click *Add Seed* for adding one seed or *Add Bulk Seeds* for multiple.
- Enter either the user ids or search criteria and save.
- When you have added all seeds you want, click *Turn on*.

For details on each collection type, see:

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

1.4 Exporting your Data

In order to access the data in a collection, you will need to export it. You are able to download your data in several formats, including Excel (.xlsx) and Comma Separated Values (.csv), which can be loaded into a spreadsheet or data analytic software.

To export:

- At the top of the individual collection, click *Export*.
- Select the file type you want (.csv is recommended; .xlsx types will also be easily accessible).
- Select the export file size you want, based on number of posts per file. You may want to select a number of posts that will work the program that you will be loading the data into, e.g., Excel.
- Select Deduplicate if you only want one instance of every post. This will clean up your data, but will make the export take longer.
- Item start date/end date allow you to limit the export based on the date each post was created. Note that the date you enter will be in the local timezone. The date in posts may be in a different timezone, e.g., UTC. Appropriate adjustments will be made to account for this.

- Harvest start date/end date allow you to limit the export based on the harvest dates.
- When you have the settings you want, click *Export*. You will be redirected to the export screen. When the export is complete, the files, along with a README file describing what was included in the export and the collection, will appear for you to click on and download. You will receive an email when your export completes.
- To help understand each metadata field in the export, see *Data Dictionaries for CSV/Excel Exports*.

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.

2.1 Managing credentials

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

2.1.1 Adding credentials

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

2.1.2 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 `.env`. If additional management is necessary, it can be performed using the Social Accounts section of the Admin interface.

2.2 Platform specifics

Twitter Twitter credentials can be obtained from the Twitter API.

For detailed instructions, see *Adding Twitter Credentials*.

You *must* provide a callback URL which is *http://<SFM hostname>/accounts/twitter/login/callback/*. Note that this should be *http* not *https* even if you are using https.

Also, turn on *Enable Callback Locking* and *Allow this application to be used to Sign in with Twitter*.

It is recommended to change the application permissions to read-only.

Flickr Flickr credentials can be obtained from the Flickr API.

For detailed instructions, see *Adding Flickr Credentials*.

Tumblr Tumblr credentials can be obtained from the Tumblr API.

For detailed instructions, see *Adding Tumblr Credentials*.

2.2.1 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.

2.3 Adding Twitter Credentials

The easiest way to set up Twitter credentials is to connect them to your personal Twitter account (or another Twitter account you control). If you want more fine-tuned control, you can manually set up application-level credentials (see below).

To connect to Twitter credentials, first sign in to Twitter with the account you want to use. Then, on the Credentials page, click *Connect to Twitter*. A window will pop up from Twitter, asking you for authorization. Click authorize, and your credentials will automatically connect.

Once credentials are connected, you can start *Creating Collections*.

Manually adding Twitter Credentials, rather than connecting them automatically using your Twitter account (see above), gives you greater control over your credentials and allows you to use multiple credentials.

To manually add 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.
- **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*

2.4 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.

2.5 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.

2.6 Adding Weibo Credentials

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.

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

3.1 Twitter user timeline

Twitter user timeline collections collect the 3,200 most recent tweets from each of a list of Twitter accounts using Twitter's user_timeline API.

Seeds for Twitter user timelines are individual Twitter accounts.

To identify a user timeline, you can provide a screen name (the string after @, like NASA for @NASA) 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 the first time the harvester runs. The user may change the screen name over time, and the seed will be updated accordingly.

The harvest schedule 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.

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

See *Incremental collecting* to decide whether or not to collect incrementally.

3.2 Twitter search

Twitter searches collect tweets from the last 7-9 days that match search queries, similar to a regular search done on Twitter, using the Twitter Search API. This is **not** a complete search of all tweets; results are limited both by time and arbitrary relevance (determined by Twitter).

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.

See *Incremental collecting* to decide whether or not to collect incrementally.

3.3 Twitter sample

Twitter samples are a random collection of approximately 0.5–1% of 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.

3.4 Twitter filter

Twitter Filter collections harvest 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.

- Note: When entering a comma-separated list of search terms for the track or follow parameters, make sure to use the standard `,` character. When typing in certain languages that use a non-Roman alphabet, a different character is generated for commas. For example, when typing in languages such as Arabic, Farsi, Urdu, etc., typing a comma generates the character. To avoid errors, the Track parameter should use the Roman `,` character; for example: ,

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 website does not provide a way to look up the user ID for a user account. You can use 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).

3.5 Flickr user

Flickr User Timeline collections gather 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).

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.

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.

See *Incremental collecting* to decide whether or not to collect incrementally.

3.6 Tumblr blog posts

Tumblr Blog Post collections harvest posts by specified Tumblr blogs using the Tumblr Posts API.

Seeds are individual blogs for these collections. Blogs can be specified with or without the .tumblr.com extension.

See *Incremental collecting* to decide whether or not to collect incrementally.

3.7 Weibo timeline

Weibo Timeline collections harvest weibos (microblogs) 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.

3.8 Weibo search

Collects recent weibos that match a search query using the Weibo search_topics API. 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.

3.9 Incremental collecting

The incremental option is the default and will collect tweets or posts that have been published since the last harvest. When the incremental option is not selected, the maximum number of tweets or posts will be harvested each time the harvest runs. If a non-incremental harvest is performed multiple times, there will most likely be duplicates. However, with these duplicates, you may be able to track changes across time in a user's timeline, such as changes in retweet and like counts, deletion of tweets, and follower counts.

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 *Command-line exporting/processing*.

- *Twitter Dictionary*
- *Tumblr Dictionary*
- *Flickr Dictionary*
- *Weibo Dictionary*

4.1 Twitter Dictionary

For more info about source tweet data, see the Twitter API documentation, including Tweet data dictionaries.

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

Field	Description
id	Twitter identifier for the tweet.
tweet_url	URL of the tweet on Twitter's website. If the tweet is a retweet, the URL will be redirected to the o
created_at	Date and time the tweet was created, in Twitter's default format.
parsed_created_at	Date and time the tweet was created, in ISO 8601 format and UTC time zone.
user_screen_name	The unique screen name of the account that authored the tweet, at the time the tweet was posted. So
text	The text of the tweet. Newline characters are replaced with a space.
tweet_type	original, reply, quote, or retweet
coordinates	The geographic coordinates of the tweet. This is only enabled if geotagging is enabled on the accou
hashtags	Hashtags from the tweet text, as a comma-separated list. Hashtags are generally displayed with a #
media	URLs of media objects (photos, videos, GIFs) that are attached to the tweet.

Field	Description
urls	URLs entered by user as part of tweet. Note that URL may be a shortened URL, e.g. from bit.ly.
favorite_count	Number of times this tweet had been favorited/liked by other users at the time the tweet was collect
in_reply_to_screen_name	If tweet is a reply, the screen name of the author of the tweet that is being replied to.
in_reply_to_status_id	If tweet is a reply, the Twitter identifier of the tweet that is being replied to.
in_reply_to_user_id	If tweet is a reply, the Twitter identifier of the author of the tweet that is being replied to.
lang	Language of the tweet text, as determined by Twitter.
place	The user or application-provided geographic location from which a tweet was posted.
possibly_sensitive	Indicates that URL contained in the tweet may reference sensitive content.
retweet_count	Number of times the tweet had been retweeted at the time the tweet was collected.
retweet_or_quote_id	If tweet is a retweet or quote tweet, the Twitter identifier of the source tweet.
retweet_or_quote_screen_name	If tweet is a retweet or quote tweet, the screen name of the author of the source tweet.
retweet_or_quote_user_id	If tweet is a retweet or quote tweet, the Twitter identifier of the author or the source tweet.
source	The application from which the tweet was posted.
user_id	Twitter identifier for the author of the tweet.
user_created_at	Date and time the tweet was created, in Twitter's default format.
user_default_profile_image	URL of the user's profile image.
user_description	The user-provided account description. Newline characters are replaced with a space.
user_favourites_count	Number of tweets that have been favorited/liked by the user.
user_followers_count	Number of followers this account had at the time the tweet was collected.
user_friends_count	Number of users this account was following at the time the tweet was collected.
user_listed_count	Number of public lists that this user is a member of.
user_location	The user's self-described location. Not necessarily an actual place.
user_name	The user's self-provided name.
user_statuses_count	Number of tweets that the user has posted.
user_time_zone	The user-provided time zone. Currently deprecated.
user_urls	URLs entered by user as part of user's description.
user_verified	Indicates that the user's account is verified.

4.2 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

4.3 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/

4.4 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

Command-line 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 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*.

5.1 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.
- csvkit for processing CSVs.
- gron for grepping JSON.

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.

5.2 SFM commandline tools

5.2.1 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.

5.2.2 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] [--harvest-start HARVEST_START]
                     [--harvest-end HARVEST_END] [--warc-start WARC_START]
                     [--warc-end WARC_END] [--api-base-url API_BASE_URL]
                     [--debug [DEBUG]] [--newline]
                     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. Use `--newline` to to separate with a newline instead.)

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`.

5.2.3 Sync scripts

Sync scripts will extract Twitter data from WARC files for a collection and write tweets to to line-oriented JSON files and tweet ids to text files. It is called a "sync script" because it will skip WARCs that have already been processed.

Sync scripts are parallelized, allowing for faster processing.

There are sync scripts for Twitter REST collections (*twitter_rest_sync.sh*) and Twitter stream collections (*twitter_stream_sync.sh*). Usage is *./<script> <collection id> <destination directory> <# of threads>*. For example:

```
cd /opt/processing
mkdir /sfm-processing/test
./twitter_rest_sync.sh e76b140351574015a6aac8999b06dcc7 /sfm-processing/test 2
```

5.2.4 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.

5.3 Recipes

5.3.1 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.

5.3.2 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 --newline 7c37157 > 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>
```

5.3.3 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
```

5.3.4 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

Releasing public datasets

Many social media platforms place limitations on sharing of data collected from their APIs. One common approach for sharing data, in particular for Twitter, is to only share the identifiers of the social media items. Someone can then recreate the dataset be retrieving the items from the API based on the identifiers. For Twitter, the process of extracting tweet ids is often called "dehydrating" and retrieving the full tweet is called "hydrating."

Note that retrieving the entire original dataset may not be possible, as the social media platform may opt to not provide social media items that have been deleted or are no longer public.

This example shows the steps for releasing the Women's March dataset to Dataverse. The Women's March dataset was created by GWU and published on the Harvard Dataverse. These instructions can be adapted for publishing your own collections to the dataset repository of your choice.

Note that the Women's March dataset is a single (SFM) collection. For an example of publishing multiple collections to a single dataset, see the 2016 United States Presidential Election dataset.

6.1 Exporting collection data

1. Access the server where your target collection is located and instantiate a processing container. (See *Command-line exporting/processing*):

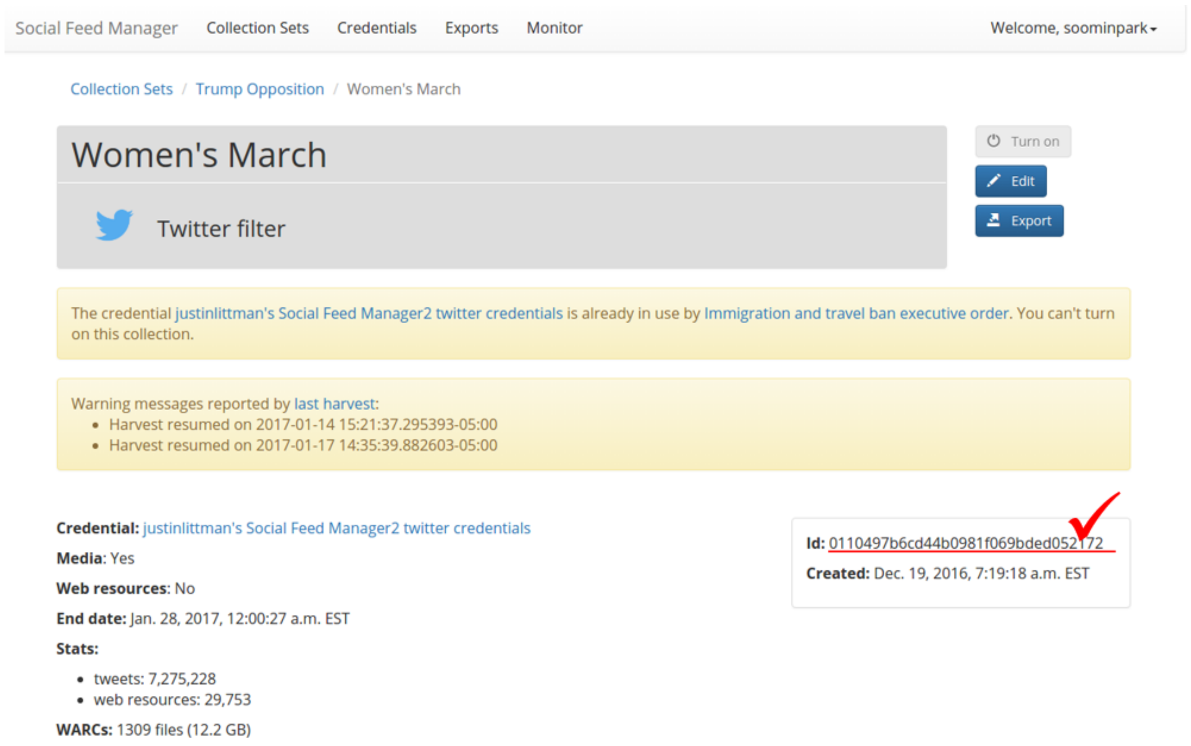
```
ssh sfmserver.org
cd /opt/sfm
docker-compose run --rm processing /bin/bash
```

   Replace `sfmserver.org` with the address of the SFM server that you want export data from.

2. Find a list of WARC files where the data of your target collection are stored, and create a list of WARC files (*source.lst*) and a list of destination text files. (*dest.lst*):

```
find_warcs.py 0110497 | tr ' ' '\n' > source.lst
cat source.lst | xargs basename -a | sed 's/.warc.gz/.txt/' > dest.lst
```

Replace `0110497` with the first few characters of the collection id that you want to export. The collection id is available on the collection detail page in SFM UI. (See the picture below.)



Women's March

Twitter filter

The credential justinlittman's Social Feed Manager2 twitter credentials is already in use by Immigration and travel ban executive order. You can't turn on this collection.

Warning messages reported by last harvest:

- Harvest resumed on 2017-01-14 15:21:37.295393-05:00
- Harvest resumed on 2017-01-17 14:35:39.882603-05:00

Credential: justinlittman's Social Feed Manager2 twitter credentials

Media: Yes

Web resources: No

End date: Jan. 28, 2017, 12:00:27 a.m. EST

Stats:

- tweets: 7,275,228
- web resources: 29,753

WARCs: 1309 files (12.2 GB)

Id: 0110497b6cd44b0981f069bded052172

Created: Dec. 19, 2016, 7:19:18 a.m. EST

3. Write the tweet ids to the destination text files:

```
time parallel -j 3 -a source.lst -a dest.lst --xapply "twitter_stream_warc_iter.
↪py {1} | jq -r '.id_str'  > {2}"
```

This command executes a Twitter Stream WARC iterator to extract the tweets from the WARC files and jq to extract the tweet ids. This shows using *twitter_stream_warc_iter.py* for a Twitter stream collection. For a Twitter REST collection, use *twitter_rest_warc_iter.py*.

Parallel is used to perform this process in parallel (using multiple processors), using WARC files from *source.lst* and text files from *dest.lst*. *-j 3* limits parallel to 3 processors. Make sure to select an appropriate number for your server.

An alternative to steps 1 and 2 is to use a sync script to write tweet id text files and tweet JSON files in one step. (See *Command-line exporting/processing*)

4. Combine multiple files into large files:

The previous command creates a single text file containing tweet ids for each WARC file. To combine the tweets into a single file:

```
cat *.txt > womensmarch.txt
```

- Recommendation: If there are a large number of tweet ids in a file, split into multiple, smaller files. (We limit to 50 million tweet ids per file.)

5. Create a README file that contains information on each collection (management command for sfm ui):

Exit from the processing container, and connect to the UI container and execute the exportreadme management command to create a README file for the dataset:

```
exit
docker-compose exec ui /bin/bash -c "/opt/sfm-ui/sfm/manage.py exportreadme␣
↪0110497 > /sfm-processing/womensmarch-README.txt"
```

6. Copy the files from the server to your local hard drive:

   Exit from the SFM server with `exit` command, move to a location in your local hard drive where you want to store the data, and run the command below:

```
exit
scp -p username@sfmserver.org:/sfm-processing/womensmarch*.txt .
```

   Replace `username` and `sfmserver.org` with your user ID and the address of the SFM server, respectively.

6.2 Publishing collection data on Dataverse

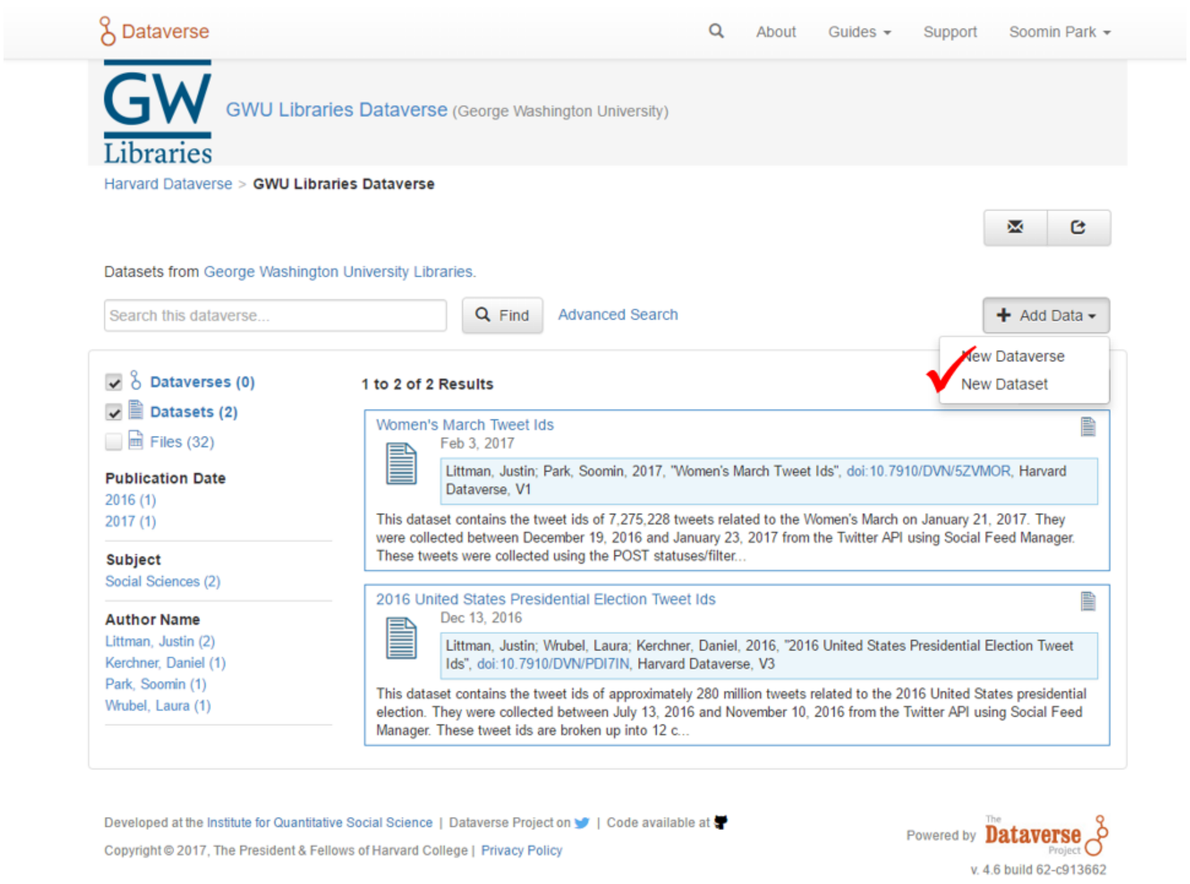
For this example, we will be adding the collection to the GW Libraries Dataverse on the Harvard Dataverse instance.

1. Go to the GW Libraries Dataverse and log in.

   • Note: You should be a Curator for the dataverse to be able to upload data.

2. Open the New Dataset page:

   Click '*Add Data > New Dataset*'.



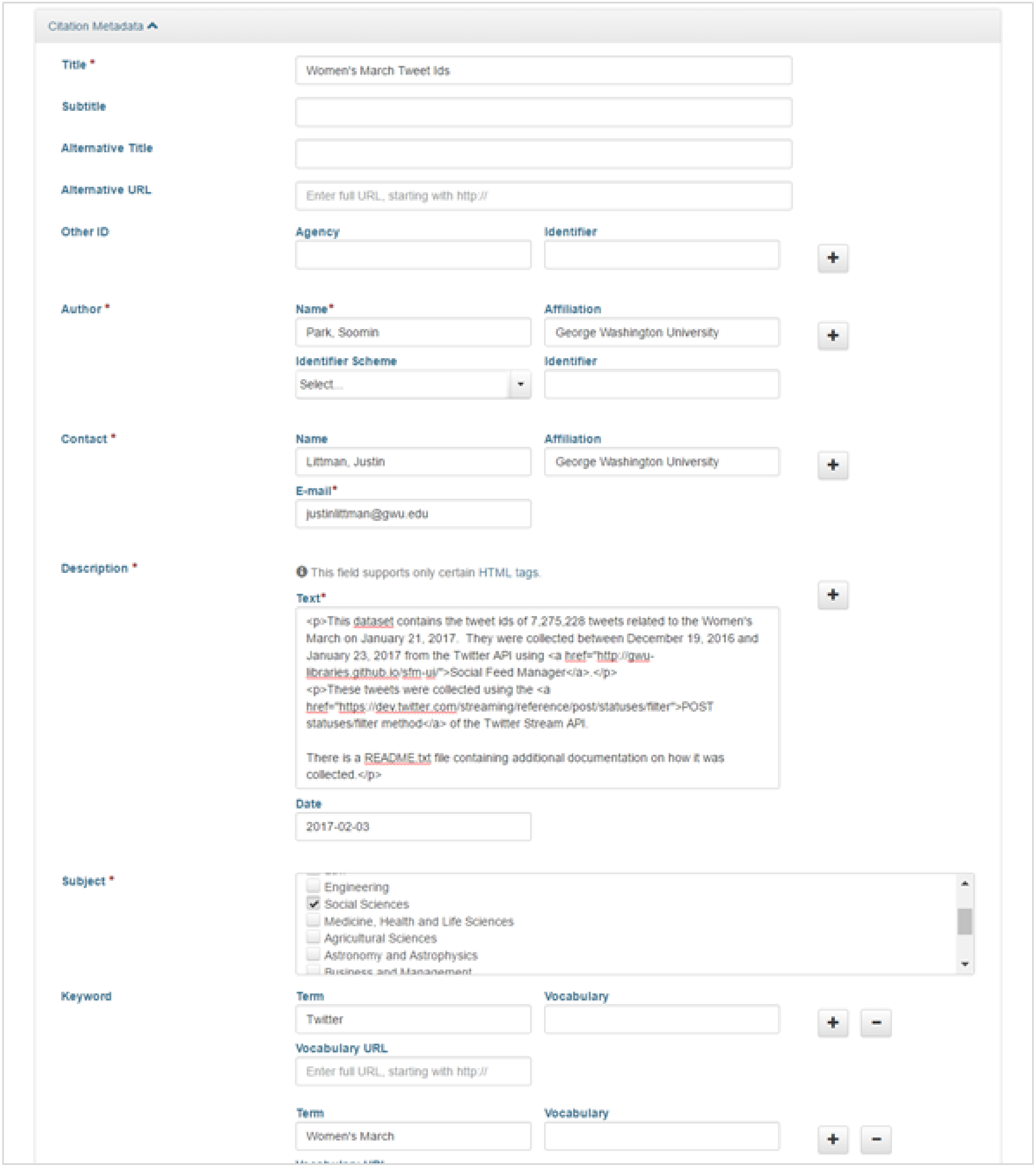
Women's March Tweet Ids
Feb 3, 2017
Littman, Justin; Park, Soomin, 2017, "Women's March Tweet Ids", doi:10.7910/DVN/5ZVMOR, Harvard Dataverse, V1
This dataset contains the tweet ids of 7,275,228 tweets related to the Women's March on January 21, 2017. They were collected between December 19, 2016 and January 23, 2017 from the Twitter API using Social Feed Manager. These tweets were collected using the POST statuses/filter...

2016 United States Presidential Election Tweet Ids
Dec 13, 2016
Littman, Justin; Wrubel, Laura; Kerchner, Daniel, 2016, "2016 United States Presidential Election Tweet Ids", doi:10.7910/DVN/PDI7IN, Harvard Dataverse, V3
This dataset contains the tweet ids of approximately 280 million tweets related to the 2016 United States presidential election. They were collected between July 13, 2016 and November 10, 2016 from the Twitter API using Social Feed Manager. These tweet ids are broken up into 12 c...

Developed at the Institute for Quantitative Social Science | Dataverse Project on | Code available at
Copyright © 2017, The President & Fellows of Harvard College | Privacy Policy
Powered by The Dataverse Project
v. 4.6 build 62-c913662

3. Fill the metadata with proper information (title, author, contact, description, subject, keyword):

   Make sure you input the right number of tweets collected and appropriate dates in the description.



Citation Metadata

Title *	Women's March Tweet Ids	
Subtitle		
Alternative Title		
Alternative URL	Enter full URL, starting with http://	
Other ID	Agency:	Identifier:
Author *	Name*: Park, Soomin	Affiliation: George Washington University
	Identifier Scheme: Select...	Identifier:
Contact *	Name: Littman, Justin	Affiliation: George Washington University
	E-mail*: justinlittman@gwu.edu	

Description *

This field supports only certain HTML tags.

Text*

```
<p>This dataset contains the tweet ids of 7,275,228 tweets related to the Women's
March on January 21, 2017.  They were collected between December 19, 2016 and
January 23, 2017 from the Twitter API using <a href="http://gwu-
libraries.github.io/sfm-ui/">Social Feed Manager</a>.</p>
<p>These tweets were collected using the <a
href="https://dev.twitter.com/streaming/reference/post/statuses/filter">POST
statuses/filter method</a> of the Twitter Stream API.

There is a README.txt file containing additional documentation on how it was
collected.</p>
```

Date: 2017-02-03

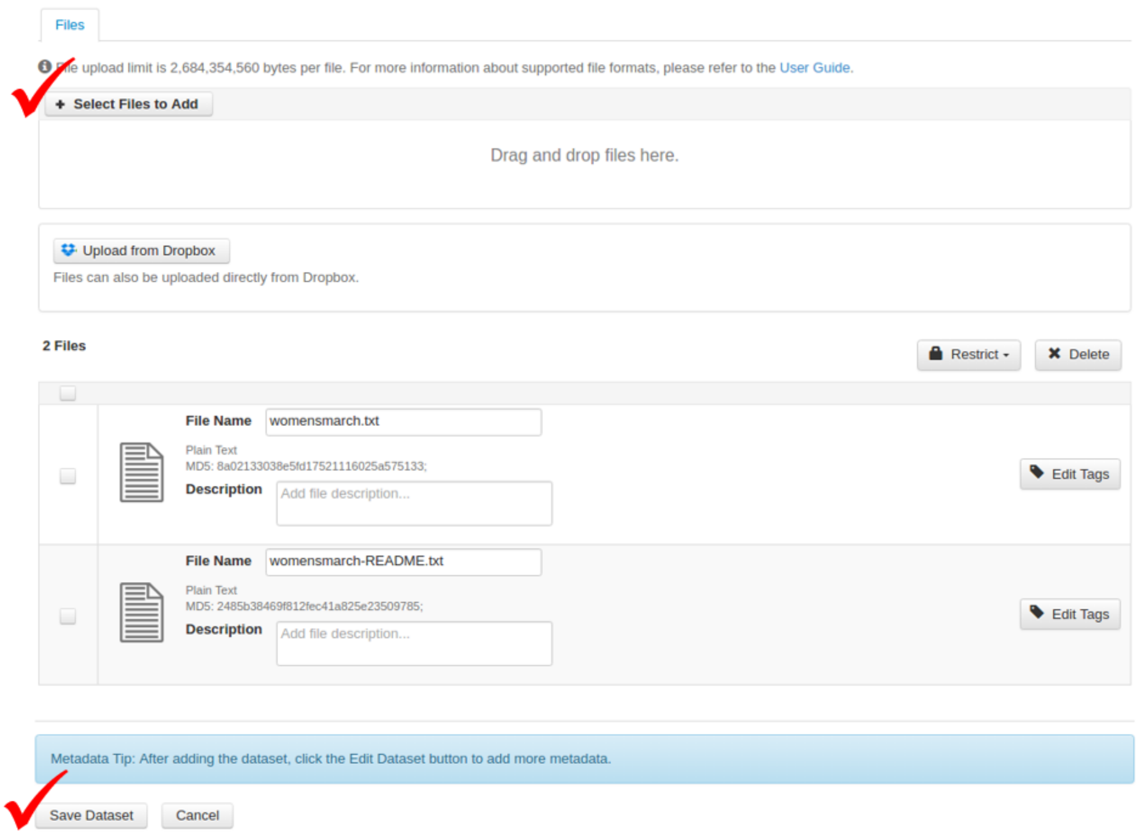
Subject *

- [ ] Engineering
- [x] Social Sciences
- [ ] Medicine, Health and Life Sciences
- [ ] Agricultural Sciences
- [ ] Astronomy and Astrophysics
- [ ] Business and Management

Keyword	Term: Twitter	Vocabulary:
	Vocabulary URL: Enter full URL, starting with http://	
	Term: Women's March	Vocabulary:

4. Upload the files (both data and README files) and save the dataset:

   - Note: The dataset will be saved as a draft.



File upload limit is 2,684,354,560 bytes per file. For more information about supported file formats, please refer to the User Guide.

Drag and drop files here.

Files can also be uploaded directly from Dropbox.

2 Files

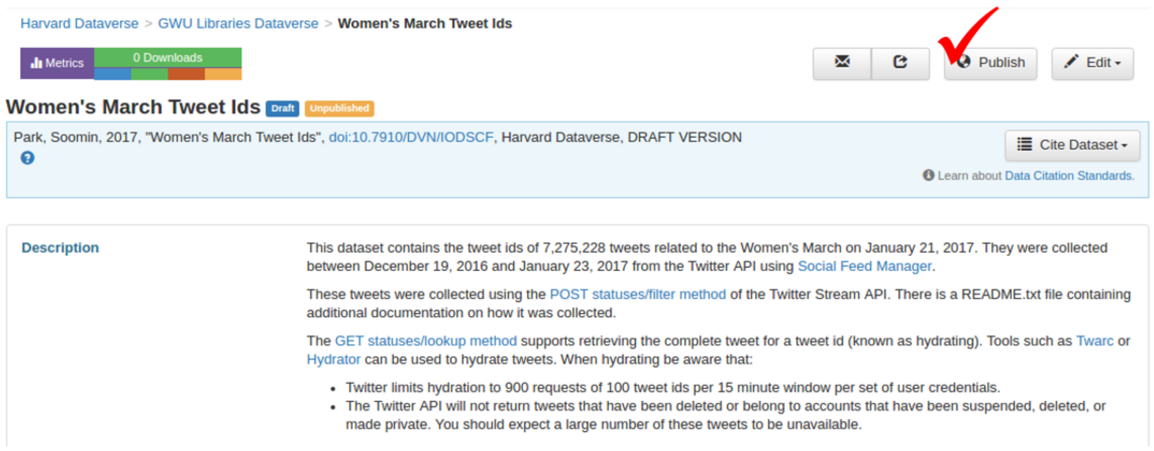
File Name: womensmarch.txt
Plain Text
MD5: 8a02133038e5fd17521116025a575133;
Description: Add file description...

File Name: womensmarch-README.txt
Plain Text
MD5: 2485b38469f812fec41a825e23509785;
Description: Add file description...

Metadata Tip: After adding the dataset, click the Edit Dataset button to add more metadata.

5. Publish the dataset:

   Go to the page of the draft that was just saved, and click '*Publish*' button.


Harvard Dataverse > GWU Libraries Dataverse > Women's March Tweet Ids
Metrics
0 Downloads
Publish
Edit
Draft
Unpublished
Cite Dataset

Women's March Tweet Ids

Park, Soomin, 2017, "Women's March Tweet Ids", doi:10.7910/DVN/IODSCF, Harvard Dataverse, DRAFT VERSION

Learn about Data Citation Standards.

Description

This dataset contains the tweet ids of 7,275,228 tweets related to the Women's March on January 21, 2017. They were collected between December 19, 2016 and January 23, 2017 from the Twitter API using Social Feed Manager.

These tweets were collected using the POST statuses/filter method of the Twitter Stream API. There is a README.txt file containing additional documentation on how it was collected.

The GET statuses/lookup method supports retrieving the complete tweet for a tweet id (known as hydrating). Tools such as Twarc or Hydrator can be used to hydrate tweets. When hydrating be aware that:

- Twitter limits hydration to 900 requests of 100 tweet ids per 15 minute window per set of user credentials.
- The Twitter API will not return tweets that have been deleted or belong to accounts that have been suspended, deleted, or made private. You should expect a large number of these tweets to be unavailable.

6.3 Adding link to Dataverse dataset

Once you have published your collection data on Dataverse, you can add to it from SFM. This will allow other SFM users to find the public version of your collection.

1. Go to the collection page for your collection in SFM and click Edit.

2. Add the Dataverse link to in the “Public Link” field and click Save.

CHAPTER 7

Citing SFM and datasets

7.1 Citing SFM

The recommended citation for Social Feed Manager (i.e., the software) is:

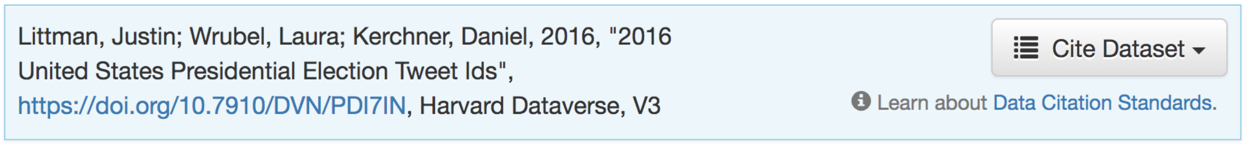
```
George Washington University Libraries. (2016). Social Feed Manager. Zenodo. https://
↪doi.org/10.5281/zenodo.597278
```

For more guidance on citing SFM, see SFM in Zenodo.

7.2 Citing a public dataset

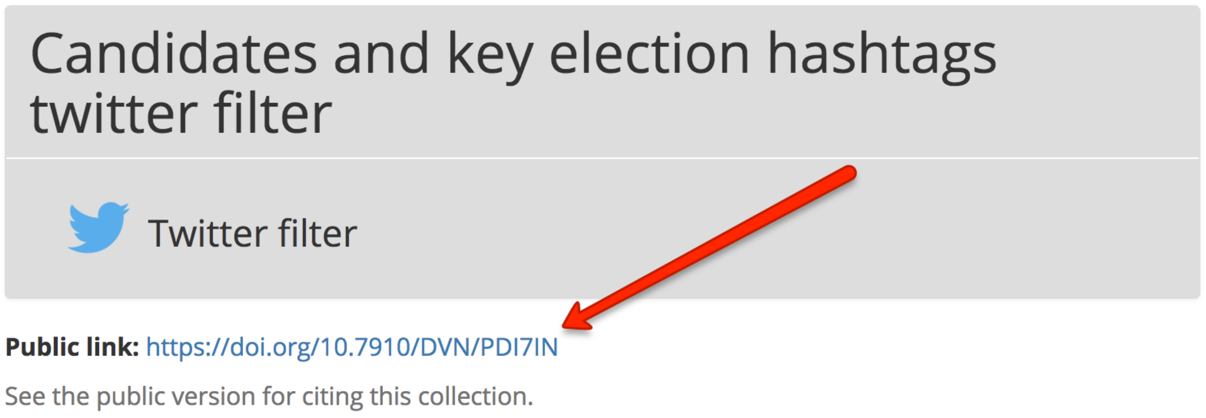
Some SFM collections have been released as public datasets, usually by depositing them in a data repository. (See *Releasing public datasets*).

Usually the public version will provide guidance on citing. For example, the 2016 United States Presidential Election collection is deposited in Harvard's Dataverse, which offers the following assistance on citing:



Littman, Justin; Wrubel, Laura; Kerchner, Daniel, 2016, "2016 United States Presidential Election Tweet Ids", https://doi.org/10.7910/DVN/PDI7IN, Harvard Dataverse, V3

Within SFM, a link may be provided to the public version of a dataset.


Candidates and key election hashtags twitter filter

Twitter filter

Public link: https://doi.org/10.7910/DVN/PDI7IN

See the public version for citing this collection.

7.3 Citing your own dataset

To make your dataset citable and reusable by others, you are encouraged to release it as public dataset. (See *Releasing public datasets*). You are also encouraged to cite SFM within your dataset release and your publication.

CHAPTER 8

Installation and configuration

8.1 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.

8.2 Local installation

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

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

```
git clone https://github.com/gwu-libraries/sfm-docker.git
cd sfm-docker
# Replace 2.1.0 with the correct version.
git checkout 2.1.0
cp example.prod.docker-compose.yml docker-compose.yml
cp example.env .env
```

or just download `example.prod.docker-compose.yml` and `example.env` (replacing 2.1.0 with the correct version):

```
curl -L https://raw.githubusercontent.com/gwu-libraries/sfm-docker/2.1.0/example.prod.
↪docker-compose.yml > docker-compose.yml
curl -L https://raw.githubusercontent.com/gwu-libraries/sfm-docker/2.1.0/example.env >
↪ .env
```

2. Update configuration in `.env` as described in *Configuration*.
3. Download containers and start SFM:

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

It may take several minutes for the images to be downloaded and the containers to start. These images are large (roughly 12GB) so make sure you have enough disk space and a high-speed connection is recommended.

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

   ```
   docker-compose scale twitterrestharvester=2 twitterpriorityrestharvester=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*.
- For help with other Docker commands (e.g., to stop SFM) see *Helpful commands*.

8.3 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*. Also, in the curl statements change *master* to the correct version, e.g., *2.1.0*:

```
#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://raw.githubusercontent.com/gwu-libraries/sfm-docker/2.1.0/example.
↪prod.docker-compose.yml > docker-compose.yml
 - curl -L https://raw.githubusercontent.com/gwu-libraries/sfm-docker/2.1.0/example.
↪env > .env
```

(continues on next page)

(continued from previous page)

```
# 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 >> .env
 - echo PROCESSING_VOLUME=/sfm-processing:/sfm-processing >> .env
# 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
 - docker-compose up -d
 - docker-compose scale twitterrestharvester=2 twitterpriorityrestharvester=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.

8.4 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`, and `POSTGRES_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`.

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.

8.5 HTTPS

To run SFM with HTTPS:

1. Create or acquire a valid certificate and private key.
2. In `docker-compose.yml` uncomment the nginx-proxy container and set the paths under `volumes` to point to your certificate and key.
3. In `.env` change `USE_HTTPS` to True and `SFM_PORT` to 8080. Make sure that `SFM_HOSTNAME` matches your certificate.
4. Start up SFM.

Note:

- HTTPS will run on 443. Port 80 will redirect to 443.

- For more information on nginx-proxy, including advanced configuration see https://github.com/jwilder/nginx-proxy.
- If you receive a 502 (bad gateway), wait until SFM UI has completely started. If the 502 continues, troubleshoot SFM UI.

8.6 Stopping

To stop the containers gracefully:

```
docker-compose stop -t 180 twitterstreamharvester
docker-compose stop -t 45
```

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

8.7 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*.

8.8 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 twitterstreamharvester
   docker-compose stop -t 45
   ```

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
   # Replace 2.1.0 with the correct version.
   git checkout 2.1.0
   cp example.prod.docker-compose.yml docker-compose.yml
   ```

otherwise, replacing 2.1.0 with the correct version:

```
curl -L https://raw.githubusercontent.com/gwu-libraries/sfm-docker/2.1.0/example.prod.
↪docker-compose.yml > docker-compose.yml
```

4. If you customized your previous `docker-compose.yml` file, make the same changes in your new `docker-compose.yml`.

5. Make any changes in your `.env` file prescribed by the release notes.
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.

7. Deleting images from the previous version is recommended to prevent Docker from filling up too much space. Replacing 1.5.0 with the correct previous version:

```
docker rmi $(docker images | grep "1.5.0" | awk '{print $3}')
```

You may also want to periodically clean up Docker (>= 1.13) with `docker system prune`.

8.9 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 9

Monitoring

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

For more information on troubleshooting, see *Troubleshooting*.

9.1 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.

9.2 Logs

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

9.2.1 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`.

9.2.2 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`.

9.3 RabbitMQ management console

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

CHAPTER 10

Administration

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.

10.1 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.

10.2 Deactivating collections

Deactivating a collection indicates that you have completed collecting data for that collection. Deactivated collections will be removed from some of the lists in SFM UI and will not appear in the harvest status emails.

Collections can be deactivated using the "Deactivate" button on the collection detail page.

Note:

- A deactivated collection can be re-activated from the collection detail page. A deactivated collection must be re-activated before it can be edited or turned on.
- A collection set is considered deactivated when it has no active collections. It will also be removed from some of the lists in SFM UI and not appear in harvest status emails.

10.3 Sharing collections

Changing the visibility of a collection to "Other users" will allow the collection to be viewed by all SFM users.

The visibility of a collection can be changed by editing the collection.

Note: * A collection set is shared when it has a shared collection. * Shared collection sets will be listed on a separate tab of the collection set list page.

10.4 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.**

10.5 Moving collections

Collections can be moved from one collection set to another. This is done by changing the collection set for the collection in the Admin Interface.

Note the following when moving collections:

- The collections files are moved as well, as the directory structure includes the collection set's identifier.
- The path for WARC files in WARC records are updated.
- Make sure harvesting is turned off and all harvests and exports are completed before moving.
- Previous exports will become unavailable after the move.

10.6 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 11

Docker

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

11.1 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.

11.2 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. A**
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.

11.3 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.

11.4 Stopping Docker from automatically starting

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

11.4.1 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
```

11.4.2 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

12.1 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 (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
```

(continues on next page)

(continued from previous page)

```
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.
```

12.2 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
```

12.3 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.

12.4 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

13.1 Storage volumes

SFM stores data on 2 volumes:

- sfm-data: The data volume is where SFM stores the harvested social media content, 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 *Command-line exporting/processing*.) It is available within containers as /sfm-processing.

13.2 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.

13.3 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.

13.4 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)
                records/
                    JSON records for the collection metadata
                <year>/<month>/<day>/<hour>/
                    WARC files
    containers/
        <container id>/
            Working files for individual containers
    export/
        <export id>/
            Export files
    postgresql/
        Postgres db files
```

13.5 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
```

13.6 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:

- Changes to the hostname of server (e.g., from the reboot of an AWS EC2 instance) are not handled (Ticket 435)

For a complete list of tickets, see https://github.com/gwu-libraries/sfm-ui/issues

In addition, you should be aware of the following:

- 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.

CHAPTER 15

Troubleshooting

15.1 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`.

15.2 Specific problems

15.2.1 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.

15.2.2 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
```

15.2.3 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.)

15.2.4 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.

15.2.5 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 should be *http://<SFM hostname>/accounts/twitter/login/callback/*.

If you have made a change to the credentials configured in `.env`, try deleting twitter from Social Applications in the admin interface and restarting SFM UI (`docker-compose stop ui` then `docker-compose up -d`).

15.2.6 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
```

15.2.7 CSV export problems

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

15.3 Still stuck?

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

CHAPTER 16

Development

16.1 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).

16.1.1 Step 1: Install Docker and Docker Compose

See *Installing Docker*.

16.1.2 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.

16.1.3 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.

16.2 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.

16.3 Running tests

16.3.1 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-c2yZjbk1np0Z5MWEFYYQKSQNFFGBXd8T4k90YkJl"
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
```

16.3.2 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.

16.3.3 Smoke tests

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

To run the smoke tests, first bring up SFM:

```
docker-compose -f example.docker-compose.yml -f smoketests.docker-compose.yml up -d
```

wait, and then run the tests:

```
docker-compose -f example.docker-compose.yml -f smoketests.docker-compose.yml run --
↪rm smoketests /bin/bash -c "appdeps.py --port-wait mq:5672 --port-wait ui:8080 &&␣
↪python -m unittest discover"
```

Note that the smoke tests are not yet complete and require test fixtures that are only available in a development deploy.

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

16.4 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
```

16.5 Development tips

16.5.1 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.

16.5.2 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*.

16.5.3 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.

16.5.4 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.

16.5.5 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>*.

16.5.6 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.

16.5.7 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.

16.5.8 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*.

16.5.9 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*.

16.6 Docker tips

16.6.1 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).

16.6.2 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

17.1 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.

17.2 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.

17.3 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

18.1 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`.

18.2 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.

18.3 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()
```

18.4 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

19.1 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.

19.2 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.

19.3 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.

19.3.1 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 sample 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.

19.3.2 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) Writes the api calls to a warc.
3. Upon completing the api harvest, the harvester:
   (a) Publishes a warc created message.
   (b) 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) 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 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 warc created message.
   (c) 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.

19.3.3 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"
    },
```

(continues on next page)

(continued from previous page)

```
    "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.

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": {
```

(continues on next page)

(continued from previous page)

```
        "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": {
```

(continues on next page)

(continued from previous page)

```
        "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.

19.4 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.

19.4.1 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.

19.4.2 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",
```

(continues on next page)

(continued from previous page)

```
    "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

20.1 Funding history

- Development of this project was 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 was 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.