BibServer Documentation

Release 1.0

Open Knowledge Foundation

August 02, 2016

Contents

BibServer is an open-source RESTful bibliographic data server. BibServer makes it easy to create and manage collections of bibliographic records such as reading lists, publication lists and even complete library catalogs.

Main features:

- Create and manage bibliographic collections simply and easily
- Import (and export) your collection from bibtex, MARC, RIS, BibJSON, RDF or other bibliogrpaphic formats in a matter of seconds
- Browse collection via an elegant faceted interface
- Embed the collection browser in other websites
- Full RESTful API
- Open-source and free to use
- ~~Hosted service available at http://bibsoup.net/~~

CHAPTER 1

Quick Links

- Code: http://github.com/okfn/bibserver
- Mailing list: http://lists.okfn.org/mailman/listinfo/openbiblio-dev
- ~~Live demo: http://dev.bibsoup.net/ (sandbox) or http://bibsoup.net/~~

CHAPTER 2

Installation

2.1 Installation

2.1.1 Simple Setup

1. Install pre-requisites:
    - Python (>= 2.7), pip and virtualenv.
    - git
    - ElasticSearch (> 0.17 series)
2. [optional] Create a virtualenv and enable it:

```
# in bash
virtualenv {myenv}
. {myenv}/bin/activate
```

3. Get the source:

```
# by convention we put it in the virtualenv but you can put anywhere
# mkdir {myenv}/src
# git clone https://github.com/okfn/bibserver {myenv}/src/
git clone https://github.com/okfn/bibserver
```

3. Install the app:

```
# move to your checkout of bibserver
# cd {myenv}/src/bibserver
cd bibserver
# do a development install from current directory
pip install -e .
# alternatively if you do not want a development install
# note there is an error with this at the moment - do dev install
# python setup.py install
```

4. Run the webserver:

```
python bibserver/web.py
```

See doc/deploy.rst or https://bibserver.readthedocs.io/en/latest/deploy.html for more details on a full installation

2.1.2 Install example

Install commands on a clean installation of Ubuntu_11.10:

```
sudo apt-get install python-pip python-dev build-essential
sudo pip install --upgrade pip
sudo pip install --upgrade virtualenv
sudo apt-get install git

wget https://github.com/downloads/elasticsearch/elasticsearch/elasticsearch-0.18.2.tar.gz
tar -xzvf elasticsearch-0.18.2.tar.gz
./elasticsearch-0.18.2/bin/elasticsearch start

virtualenv .
. ./bin/activate

git clone https://github.com/okfn/bibserver
cd bibserver
pip install -e .

python bibserver/web.py
```

You will now find your bibserver running at localhost:5000.

2.2 Deployment

2.2.1 Pre-requisites

This example is for installing bibserver to run bibsoup.net, but applies to other instances - just change relevant parts e.g. domain name and so on.

These instructions work on an ubuntu / debian machine, and explain how to get a stable deployment using:

- git (to get latest copy of code)
- nginx (the web server that proxies to the web app)
- python2.7+, pip, virtualenv (required to run the app)
- gunicorn (runs the web app that receives the proxy from nginx)
- supervisord (keeps everything up and running)

2.2.2 nginx config

Create an nginx site config named e.g. bibsoup.net default location is /etc/nginx/sites-available (for OKF machines should be in ~/etc/nginx/sites-available then symlinked) then symlink from /etc/nginx/sites-enabled

upstream bibsoup_app_server { server 127.0.0.1:5050 fail_timeout=0;

}

server { server_name bibsoup.net;

access_log /var/log/nginx/bibsoup.net.access.log;

server_name_in_redirect off;

client_max_body_size 20M;

```
        location / {
            ## straight-forward proxy proxy_redirect off;
            proxy_connect_timeout 75s; proxy_read_timeout 180s;
            proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://bibsoup_app_server;
        }
    }
```

2.2.3 supervisord config

Create a supervisord config named e.g. bibsoup.net.conf - the default location for this is /etc/supervisor/conf.d (for OKF machines, should be put in ~/etc/supervisor/conf.d then symlinked)

[program:bibsoup.net] command=/home/okfn/var/srvc/%(program_name)s/bin/gunicorn -w 4 -b 127.0.0.1:5050 bibserver.web:app user=www-data directory=/home/okfn/var/srvc/%(program_name)s/src/bibserver stdout_logfile=/var/log/supervisor/%(program_name)s-access.log stderr_logfile=/var/log/supervisor/%(program_name)s-error.log autostart=true

2.2.4 Install bibserver

Create a virtualenv and get the latest bibserver code installed. Bibserver requires python2.7+ so make sure that is available on your system, then start a virtualenv to run it in

virtualenv -p python2.7 bibsoup.net –no-site-packages cd bibsoup.net mkdir src cd bin source activate cd ../src git clone https://github.com/okfn/bibserver cd bibserver python setup.py install

Currently, setup.py install does not result in running system because config.json cannot be found. So, do dev install. This will be fixed asap

pip install -e .

Then install gunicorn into the virtualenv

pip install gunicorn

Now create a local_config.json with details as necessary for example check the ES index you with to use for this instance (default is bibserver)

{ “debug”: false, “port”: 5050, “ELASTIC_SEARCH_DB” : “bibserver_something”, “ELASTIC_SEARCH_HOST” : “localhost:9200”

}

Now run bibserver directly to check it is working - this requires elasticsearch to be up and running, as it attempts to create indices.

If it works, you should see confirmation of creation of the index and the mappings; if all good, kill it and move on. If not, debug the issues.

python bibserver/web.py

If the above step failed to push the mappings, you can do so manually. A command such as the following, augmented for your ES index URL and your index name, should do the job for you (default mappings are in config.json) (remember to do record and collection)

```
curl -X PUT localhost:9200/bibserver/record/_mapping -d '{
    "record" [{] "date_detection" : false, "dynamic_templates" : [
        {
          "default" [{] "match" : "*", "match_mapping_type": "string", "mapping" : {
              "type" : "multi_field", "fields" : {
                "{name}" : {"type" : "{dynamic_type}", "index" : "analyzed",
                "store" : "no"}, "exact" : {"type" : "{dynamic_type}", "index" :
                "not_analyzed", "store" : "yes"}
              }
            }
          }
        }
    ]
  }
}'
```

2.2.5 Enable everything

In the case of OKF service deployment, make symbolic links from the supervisor and nginx files which should be in the ~/etc folder into the /etc/nginx/sites-available and /etc/supervisor/conf.d folders, then make symbolic link from /etc/nginx/sites-available into /etc/nginx/sites-enabled - if you do not use this pattern, just put the config directly in /etc/nginx/sites-available and symlink from there into sites-enabled

cd /etc/nginx/sites-available ln -s ~/etc/nginx/sites-available/bibsoup.net . cd /etc/supervisor/conf.d ln -s ~/etc/supervisor/conf.d/bibsoup.net.conf .

Then enable the new nginx and supervisor settings

cd /etc/nginx/sites-enabled ln -s ../sites-available/bibsoup.net . /etc/init.d/nginx reload supervisorctl reread supervisorctl update

Configure your domain name to point at your server, and it should work.

2.3 Configuration

Configuration is managed via some key files. To make changes, just use local_config.json.

2.3.1 config.json

The main default config - does not need to be altered. Instead, it is updated by local_config.json

2.3.2 local_config.json

This is where configurations should be set for you particular instances. These are the various values you can set:

2.3.3 config.py

This is the config class - it loads config in from config.json, then overrides with updates from local_config.json. The config object is then made available and can be imported elsewhere in the app. Changes to config need only happen in local_config.json.

2.3.4 core.py

This is where configure_app and create_app are definedm which prepare the flask settings for the app to run. This is imported by web.py when the app is created. Settings are read from config.

2.3.5 default_settings.py

Contains a "secure-key" setting for flask config

2.4 Uploading collections

When a bibserver instance is configured to allow uploads, it is possible to upload from a source URL or file from PC into the instance via the /upload page.

A typical bibserver will support upload from the parsers it has available to it - read more about the parsers and running them independently, or writing new ones, on the parsers documentation page - *Parsing sources*

2.4.1 The upload page

To upload, just go to the upload page. Provide a URL or file, a collection name and description, confirm the license and the format type.

The Upload form can either be given a URL from which the Bibserver will retrieve the data to import, or a user can upload a file from her local machine to be imported. Bibserver tries to guess the format of the supplied URL by looking at the filename extension of the supplied URL. (this is unreliable and might be removed in future). If the fileformat can not be guessed, a list of supported fileformats for that install of Bibserver is shown.

If a converter of the right format is not available, your source file can be converted elsewhere into JSON following the bibJSON standard, then the JSON file can be imported directly.

2.4.2 Upload from other online services

Examples of providing URLs for uploading directly from other online sources such as bibsonomy.

2.4.3 Monitoring tickets

Explain the tickets page, and the info that can be got there.

2.4.4 Viewing an uploaded collection

On upload, a tidy version of collection name is made for URL.

Once a collection has uploaded, it can be found at /username/collection.

2.4.5 Multiple files to same collection

Confirm what happens when uploading mutliple files of different content to the same collection name

2.4.6 Overwriting records and internal IDs

Mention the method by which internal IDs are allocated, and how records can be overwritten if an upload is performed of records that are somewhat identical to current records. Point out what this means for local edits.

2.5 The Bibserver frontend

2.5.1 The standard pages

Explain the various standard pages, how they are built from the templates, how they are presented via flask to the end user, what they typically have on them

2.5.2 Facetview

Explain in more detail the facetview javascript / jquery app, and how it embeds onto various pages and its functionality

2.5.3 Embedding remotely

Explain how facetview can be embedded in a remote web page and still call back to a bibserver instance, enabling display of search results in any web page.

2.6 The web API

2.6.1 web.py and search.py

Detail the workings of the web.py file.

Explain the search.py file too, which is used by web.py extensively.

2.6.2 URL routes

/

methods GET

returns HTML

The front page of the web service.

/faq

methods GET

returns HTML

The frequently asked questions page of the web service.

/account

- /account/register
- /account/login
- /account/logout

returns HTML

Esed for account creation via a web browser, although it would be possible to register a new account by POSTing to the registration endpoint.

/query

- /query/record
- /query/collection

methods GET or POST

returns JSON

Exposes the query endpoints of the elasticsearch backend indices, so queries can be issued against it directly.

For exmaple /query/record/_search?q=* will return all records in the standard ten results at a time.

/users

requires authorisation

methods GET

returns HTML or JSON

Provides a list of users

/collections

- /collections/<username>
- /collections/<username>/<collection>

methods GET

returns HTML or JSON

Provides a list of all collections, a list of collections for a user, or a particular collection for a user.

/upload

methods GET (to return the browser form) or POST

requires authorisation

LIST THE PARAMS

For uploading from source files into collections.

/create

not implemented

For creating new records.

/<username>

- /<username>/collections
- /<username>/<collection>
- /<username>/<collection>/<record>

methods GET or POST or DELETE

returns HTML or JSON

requires authorisation for retrieval of user data via GET, and for POSTs

List as JSON all collections for a given user (same as /collections/<username>).

Access information about a user, a collection, or a record. Update the records by POSTing updated versions. Delete the users, collections, records by DELETE.

/<implicitfacet>/<implicitvalue>

methods GET

returns HTML or JSON

Anything that is not matched to a known endpoint but that can be matched to a key in the record index will cause an attempt to interpret it as an implicit value. For example, /year/2012 will attempt to return all records in the index (therefore across all collections) that have a a key called "year" where the value equals "2012".

/search

methods GET

returns HTML or JSON

The search endpoint allows for search across the full record index of the instance.

/<anything>

Any route that cannot be matched to any previous endpoint, including an implicit facet, performs the same as the /search endpoint.

2.6.3 Programmatic access

The API endpoints can be queried either via a web browser - which defaults to providing HTML output, of course - or programmatically by requests e.g. via cURL.

Requests for access to data operate as usual - but requests to insert data require authentication; this is achieved via API keys. Every user account has an API key assigned to it, which can be retrieved from the /username page; it can then be provided as a parameter to any request that attempts to submit data into the system - e.g. a request to the /upload endpoint.

Each endpoint can return HTML or JSON; JSON can be requested either by appending .json to the URL portion, or adding format=json to the URL parameters, or by setting the "accept" headers on your request to "application/json".

Here is an example of retrieving some records from a collection via cURL:

ADD EXAMPLE

Here is an example of submitting a new collection via cURL:

ADD EXAMPLE

2.7 Parsing sources

The most common way of importing bibliographic records into Bibserver is using the Upload form in the web interface. See *Uploading collections*. When you run the Bibserver application from the command line, an ingest system is started in a separate process, which handles the processing of uploads asynchronously. This is done to keep the web interface responsive as some data uploads may involved downloading and processing very large data files.

To run the ingest process manually separate from the Bibserver application, start it up with a -d flag. For example:

```
python bibserver/ingest.py -d
```

Note that it is normally not required to run the ingest manually, the startup of ingest should be done by the main bibserver command line web script. See: *Deployment*

2.7.1 The parsers

For each kind of file that can be imported into a Bibserver, a 'parser' exists. A parser is an executable file that accepts a file format on standard input and always outputs Bibjson. The parsers are stored in a directory called 'parserscrapers_plugins' by default.

When the importing subsystem of Bibserver (named 'ingest') is initialised, all the executable files found in the parserscrapers_plugins directory are executed with a -bibserver command line parameter. A parser **must** recognise this parameter and output a JSON config section in response, indicating if this is a valid Bibserver parser and the format that is supported. All the parsers found are stored in a json data snippet named 'plugins.json' which can be used to determine what the current list of supported data formats for a given instance are. (this is used for example in the Upload forms)

2.7.2 The download cache

When a data import is submitted to Bibserver, a 'ticket' is made which tracks the progress of the upload. The ticket lists the format of the imported data, who requested it, the time it was made and the progress of the import. When an import is completed, it is also possible to see the raw data, plus the resulting Bibjson data.

The ingest tickets, downloaded data plus resulting Bibjson are all stored in a directory named 'download_cache' by default. (this location can be changed in the config.json file) The list of tickets in a system can be viewed on the /ticket/

URL. Each ticket has an ID, and one could then view either the source data /ticket/<ticket id>/data or the resulting Bibjson /ticket/<ticket id>/bibjson

All the data in a Bibserver instance can be listed by looking at a URL: /data.txt. This produces a text file with the URL for each Bibjson data file per line. This can be used for automated buld data downloads of Bibserver data.

2.7.3 Making a new parser

Even though Bibserver is written in Python, it is not necessary to write a parser in Python - it can be written in any programming language. At the time of writing there is one example parser written in Perl to support the MARC format, which is commonly found in library automation systems.

To make a new parser:

- you should be able to make standalone executable in the parserscrapers_plugins that can be called from a shell
- the parser must support the -bibserver command line paramater which gives details about the data format supported
- read data from standard input, parse and convert the data to Bibjson, and print the resulting Bibjson to standard output.

TODO: how to submit a pull request or email to include it in the repo.

2.8 Authorisation and superusers

2.8.1 Users

Explain how users create accounts and login, and how they get and can use their API key - this is just an overview as the details should be in the API pages and so on.

Also point out that users can add other users to their collection as admins - giving them the same controls as the owner has.

2.8.2 Superusers

Refer to config settings for how superusers can be listed, and how a superuser can go to any collection or user page and edit them.

2.9 About bibJSON

To Do:

- Brief detail of bibjson and link to http://okfnlabs.org/bibjson/ for more info.
- Also link to open biblio principles.
- Also mention the licenses documentation page.

2.10 Licensing issues

2.10.1 Software

Bibserver is open source MIT. It relies on other open source packages.

2.10.2 Metadata

Bibserver is a tool that enables easy sharing of bibliographic metadata. It uses the bibJSON format, and converts from other typical formats into bibJSON.

Mention open biblio principles, and refer to bibjson docs page. Point out that content uploaded should be freely available, but licenses can be attached to collections if desired.

Typically, public domain / open access should be licensed as such.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search