OCDS Kingfisher Tool

Jan 31, 2019

Contents

OCDS Kingfisher is a tool for downloading, storing and analysing data from publishers of the Open Contracting Data Standard.

It is Python tool that can be used as a library from other Python programmes, or from the command line on your own server.

CHAPTER 1

The Pipeline

The tool runs a pipeline of stages.

1.1 Gather Stage

In this stage, the tool just gathers the URLs that will be downloaded in the next stage.

This stage is designed to be as short as possible. Some HTTP requests may be made at this point, but not many. For this reason it was not designed to be resumable - if it stops half way through for any reason it must be restarted.

Note it is not guaranteed to find all the URL's at this stage - some may be found during the fetch stage.

Some sources are very simple - for instance taiwan and canada_buyandsell simply return a list of hard coded URL's.

Some sources are more complex - for instance uk_contracts_finder will make one request. In the results of that request it will look up what the maximum number of pages is (the API is paginated). With that knowledge, it can generate a full list of URL's to download.

1.2 Fetch Stage

In this stage, the tool downloads all the URL's it can and saves them to disk.

This stage may take a very long time. For that reason, it is designed to be resumable. If it is stopped half way through it can be restarted.

Note as well as downloading the URL, specific sources may do more at this stage. Sometimes, we can download one URL and then extract other URL's we need to download from that data.

These new URL's are then added to the list of current URL's. (One side affect of this is that the count of files downloaded and to download may be inaccurate. For instance, a process may say "downloaded 15 of 20 URL's". But then, when downloading and processing URL number 16, it may find another 10 URL's to download. The process would then say "downloaded 16 of 30 URL's".)

An example of a source that does not do this is uk_contracts_finder - this just downloads the URL and saves it to disk.

An example of a source that does do this is ukraine - sometimes this downloads webpages with lists of data, and from them it extracts the actual URL of the data to download.

1.3 Store Stage

In this stage, the tool simply takes the data from disk and stores it in the database.

This stage may take a long time. For that reason, it is designed to be resumable. If it is stopped half way through it can be restarted.

1.4 Check Stage

In this stage, the tool takes the data in the database and checks it using the same checks as CoVE. The results are stored in the database.

This stage may take a long time. For that reason, it is designed to be resumable. If it is stopped half way through it can be restarted.

1.5 Additional stages - check against a different schema version

You can also select this additional stage manually.

This stage lets you take the data in the database, and then pretend the data is of a different schema version. It is then checked using the same checks as CoVE.

See *Command line tool - check-different-schema-version option* for more details.

This stage may take a long time. For that reason, it is designed to be resumable. If it is stopped half way through it can be restarted.

CHAPTER 2

Sources

The tool comes with sources - definitions of data you can download that already know about all the API work that is needed to get the data.

A full list of sources is:

- afghanistan_records
- afghanistan_releases
- armenia
- australia_nsw
- canada_buyandsell
- canada_montreal
- chile_compra
- colombia
- digiwhist_armenia
- digiwhist_austria
- digiwhist_belgium
- digiwhist_bulgaria
- digiwhist_croatia
- digiwhist_cyprus
- digiwhist_czech_republic
- digiwhist_denmark
- digiwhist_estonia
- digiwhist_finland
- digiwhist_france

- digiwhist_georgia
- digiwhist_germany
- digiwhist_greece
- digiwhist_hungary
- digiwhist_iceland
- digiwhist_ireland
- digiwhist_italy
- digiwhist_latvia
- digiwhist_lithuania
- digiwhist_luxembourg
- digiwhist_malta
- digiwhist_netherlands
- digiwhist_norway
- digiwhist_poland
- digiwhist_portugal
- digiwhist_romania
- digiwhist_serbia
- digiwhist_slovakia
- digiwhist_slovenia
- digiwhist_spain
- digiwhist_sweden
- digiwhist_switzerland
- digiwhist_united_kingdom
- honduras_cost
- indonesia_bandung
- mexico_administracion_publica_federal
- mexico_cdmx
- mexico_grupo_aeroporto
- mexico_inai
- mexico_jalisco
- moldova
- paraguay_dncp
- paraguay_hacienda
- scotland
- taiwan
- uganda

- uk_contracts_finder
- ukraine
- uruguay
- zambia

CHAPTER 3

Parallelism

This explains what you can and can not parallelize in this tool.

On one collection, you should only run one operation at a time. The operations do not currently support parallelism.

For example, running ...

```
python ocdskingfisher-cli run taiwan &
python ocdskingfisher-cli run taiwan &
```

... will not mean the run is twice as fast - it will just break it!

On different collections, any operations can be performed at the same time, and the only limit is the capacity of your server.

For example, running ...

```
python ocdskingfisher-cli run taiwan &
python ocdskingfisher-cli run canada_buyandsell &
```

... is fine.

CHAPTER 4

Requirements and Install

4.1 Requirements

Requirements:

- python v3.5 or higher
- Postgresql v10 or higher

4.2 Installation

Set up a venv and install requirements:

```
virtualenv -p python3 .ve
source .ve/bin/activate
pip install -r requirements.txt
pip install -e .
```

4.3 Database

You need to create a UTF8 Postgresql database and create a user with write access.

Once you have created the database, you need to configure the tool to connect to the database.

You can see one way of doing that in the example below, but for other options see *Configuration*.

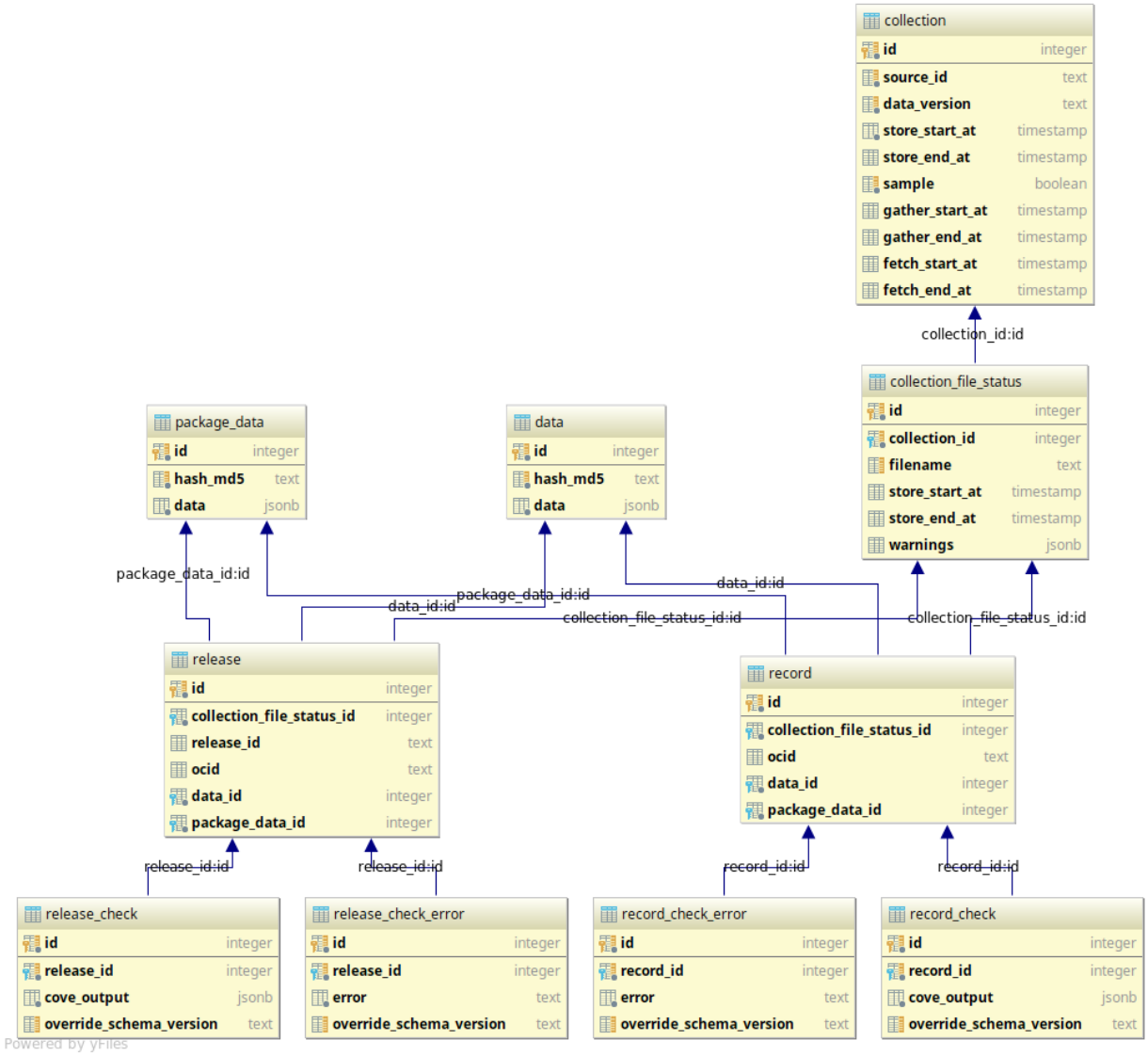
You also have to run a command to create the tables in database.

You can see the command in the example below, but for more on that see *Command line tool - upgrade-database option*.

Example of creating an database user, database and setting up the schema:

```
sudo -u postgres createuser ocdskingfisher --pwprompt
sudo -u postgres createdb ocdskingfisher -O ocdskingfisher --encoding UTF8 --template␣
↪template0 --lc-collate en_US.UTF-8 --lc-ctype en_US.UTF-8
export DB_URI='postgres://ocdskingfisher:PASSWORD YOU CHOSE@localhost/ocdskingfisher'
python ocdskingfisher-cli upgrade-database
```

The generated data base will have this tables:



Where:

- **Collection: this table contains the results of a 'run' that pass the storage stage.**
  - source_id: the name of the collection that was run, for example 'canada_buyandsell'
  - data_version: the date and time when the run command was executed
  - store_start_at: the date and time when the store stage started
  - store_end_at: the date and time when the store stage ended

  - sample: a mark that indicates if (false) the collection has all the available data or just a sample of it (true)
  - gather_start_at: the date and time when the gather stage started
  - gather_end_at: the date and time when the gather stage ended
  - fetch_start_at: the date and time when the fetch stage started
  - fetch_end_at: the date and time when the fetch stage ended
- **Collection File Status: this table contains the information about each file downloaded from a collection.**
  - filename: the name of the file that was downloaded
  - store_start_at: the date and time when the store of this file started
  - store_end_at: the date and time when the store of this file ended
  - warnings: a text that indicates any warnings in the process of saving the file, for example encoding issues
- **Package Data: this table contains the meta data information that is included in a release or record package.**
  - hash_md5: a md5 hash to know if the data changes
  - data: the meta data in jsonb format.
- Record: this table contains the ocid and relations with other tables from the downloaded records
- Release: this table contains the ocid, release id and relations with other tables from the downloaded releases
- Data: this table contains the actual data from the releases package and record package. Each row contain a hash and a data column that is a jsonb with a release, that comes from a record: from its releases list or compiledRelease field, or from a release package.
- Record check and Release check: this table contains the result of running the CoVe

validation to a release or record package in the cove_output column.

- Record an Release check error: register any error happened in the check stage

CHAPTER 5

Configuration

5.1 Main database Configuration

Postgresql Database settings can be set using a *~/.config/ocdskingfisher/config.ini* file. A sample one is included in the main directory.

```
[DBHOST]
HOSTNAME = localhost
PORT = 5432
USERNAME = ocdsdata
PASSWORD = FIXME
DBNAME = ocdsdata
```

It will also attempt to load the password from a *~/.pgpass* file, if one is present.

You can also set the *DB_URI* environmental variable to use a custom PostgreSQL server, for example *postgresql://user:password@localhost:5432/dbname*.

The order of precedence is (from least-important to most-important):

- config file
- password from *~/.pgpass*
- environmental variable

5.2 Disk Configuration

This tool will save files to disk as it works.

Where it saves them can be set using a *~/.config/ocdskingfisher/config.ini* file. A sample one is included in the main directory.

```
[DATA]
DIR = /var/ocdskingfisher/data
```

You can also set the *KINGFISHER_DATA_DIR* environmental variable to use a custom directory.

The order of precedence is (from least-important to most-important):

- config file
- environmental variable

5.3 Logging Configuration

This tool will provide additional logging information using the standard Python logging module, with loggers in the "ocdskingfisher" namespace.

When using the command line tool, it can be configured by setting a *~/.config/ocdskingfisher/logging.json* file. A sample one is included in the main directory.

5.4 Backwards compatibility

For backwards compatibility with older versions, *~/.config/ocdsdata/config.ini* and *~/.config/ocdsdata/logging.json* will also work.

CHAPTER 6

Command line tool

You can use the tool with the provided CLI script. There are various sub commands.

You can pass the *verbose* flag to all sub commands, to get more output printed to the terminal.

```
python ocdskingfisher-cli --verbose run ...
```

6.1 Command line tool - run option

The run command actually runs the pipeline.

Run it by passing the name of one or more sources to run:

```
python ocdskingfisher-cli run taiwan
python ocdskingfisher-cli run taiwan canada_buyandsell
```

You can run all sources with the *all* flag.

```
python ocdskingfisher-cli run --all
```

It is not recommended to do this as some of the sources take a very long time to download!

There is a sample mode. This only fetches a small amount of data for each source. (If you use the *all* flag we strongly recommend the *sample* flag too!)

```
python ocdskingfisher-cli run --sample ...
```

It will look for existing collections with the same source and sample flag as you specify, and by default resume the latest one.

To make sure you start a new collection, pass the *newversion* flag.

```
python ocdskingfisher-cli run --newversion  ...
```

To select an existing collection, pass the *dataversion* flag.

```
python ocdskingfisher-cli run --dataversion 2018-07-31-16-03-50 ...
```

By default, it will run all stages of the pipeline.

You can specify that only one stage should be run with the following flags:

- onlygather
- onlyfetch
- onlystore
- onlycheck

You can specify that stages should be skipped with the following flags:

- ignoregather
- ignorefetch
- ignorestore
- ignorecheck

For example:

```
python ocdskingfisher-cli run --skipstore --skipcheck ...
```

6.2 Command line tool - status option

During or after a run you can use the status command to check on the progress.

Run *ocdskingfisher-cli status* with the source as the publisher you want to see. Pass the sample flag too, if it's a sample run.

```
python ocdskingfisher-cli status taiwan
python ocdskingfisher-cli status --sample taiwan
```

By default it will show the progress for the latest run, but you can pass the dataversion flag to see a specify version.

```
python ocdskingfisher-cli status --dataversion  2018-07-31-16-03-50 ...
```

6.3 Command line tool - upgrade-database option

This tool will setup from scratch or update to the latest versions the tables and structure in the Postgresql database.

The connection settings should be configured before running it.

```
python ocdskingfisher-cli upgrade-database
```

If you want to delete all the existing tables before setting up empty tables, pass the *deletefirst* flag.

```
python ocdskingfisher-cli upgrade-database --deletefirst
```

6.4 Command line tool - check-different-schema-version option

In the check stage of the pipeline, the data will be checked against the most appropriate version of the standard.

In most cases, this will just be the version reported in the data. (One source gets data with a badly reported source, and in that case the version is corrected to the intended one.)

You may want to check the data against a different schema version - for example you may have some *1.0* data and want to check it against the *1.1* schema version.

The check-different-schema-version command does this.

Note it does not upgrade the data properly; the only change it makes is to change the version field in the data to the version you specify.

You must specify the source you want as the first argument. You can also include the *sample* flag.

```
python ocdskingfisher-cli check-different-schema-version taiwan
python ocdskingfisher-cli check-different-schema-version taiwan --sample
```

It will look for existing collections with the same source and sample flag as you specify, and by default check the latest one.

To select a specific existing collection, pass the *dataversion* flag.

```
python ocdskingfisher-cli check-different-schema-version --dataversion 2018-07-31-16-
↪03-50 ...
```

By default, it will convert to the latest version of the schema, but you can change that with the *schemaversion* option.

```
python ocdskingfisher-cli check-different-schema-version --schemaversion 1.0 ...
```

6.5 Command line tool - list-collections option

This command lists all the collections this install of the app knows about, both on disk and in the database.

A database connection is needed to run this command.

```
python ocdskingfisher-cli list-collections
DB-ID DISK SOURCE-ID                              DATA-VERSION          SAMPLE
    1 -    taiwan                                 2018-09-11-10-03-53   Full
-     Disk canada_buyandsell                      2018-09-11-10-07-58   Full
```

6.6 Command line tool - force-fetch-to-store option

This command takes a collection that is somehow stuck in the fetch stage and forces it to instantly move on to the store stage.

The next time you *run* this collection, it should start at the store stage with no problems.

Note this forcing process is irreversible - once done, it can not be undone.

```
python ocdskingfisher-cli --verbose run australia_nsw
..... get bored, stop process .....
python ocdskingfisher-cli force-fetch-to-store australia_nsw
python ocdskingfisher-cli --verbose run australia_nsw
```

This might be needed:

- because there were errors in the fetch stage.
- because the fetch stage is just taking far to long, and we want to store the data we already have for analysis.

CHAPTER 7

Developing this tool

7.1 Run tests

Run *py.test* from root directory.

7.2 Main Database - Postgresql

Create DB Migrations with Alembic - http://alembic.zzzcomputing.com/en/latest/

```
alembic --config=mainalembic.ini revision -m "message"
```

Add changes to new migration, and make sure you update database.py table structures and delete_tables to.

7.3 Meta Database - SQLite

During Gather and Fetch stages, a local SQLite DB is used to track progress.

Create DB Migrations with Alembic - http://alembic.zzzcomputing.com/en/latest/

```
alembic --config=metaalembic.ini revision -m "message"
```

Add changes to new migration, and make sure you update metadata_db.py table structures to.