Harvester

The Harvester Service provides a service-oriented framework to support the processing and routing of content and metadata from a source Data Provider to a target application. It potentially makes management of distributed harvests simpler by providing a single harvest application which can service many clients wishing to perform harvesting without the overhead of writing their own embedded harvester.

The Harvester is an extensible Python module that enables harvesting capabilities within the RDA Registry.

The plugin-architecture enables the development of further modules that support additional harvest methods and metadata schemas and/or profiles. Further modules can be ‘plugged-in’ to this architecture, enabling the creation of RDA Registry records from any web resource.

The following harvest methods are currently supported:

- GET (aka http get) - allows the harvest of individual files from any web resource, in any format (e.g. json or xml)
    - The GET harvester can be given any url with complete parameter list
- CKAN - (json metatdata over HTTP)
    - the CKAN harvester attempts to get a list of Identifiers and retrieve the json data for each record.
      it converts the entire set as one XML document (json serialised as XML)
- CKANQUERY - (json metatdata over HTTP)
    - The CKAN QUERY sends a query string to the CKAN server and retrieves all content using the start and rows params Converts the json response to serialised XML
- OAI-PMH - xml
    - retrieves all records in the metadataFormat requested by the datasource owner usingthe ListRecords endpoint
- CSW (Catalogue Services for the Web) - xml
    - retrieves datasets using the CSW protocol (using the outputSchema, in batches of 100)
- PURE (a simple dataset harvester using the PURE API)
    - requesting pages of 100 datasets until completed.
- JSONLD (a sitemap crawler and jsonld content extractor)
    - the sitemap crawler requires a sitemap file, it could be text or xml (either <sitemapindex> or <urlset>)
    - using asynchronous request (max 5)
    - attempts to extract json-ld from all pages
    - combine the result into batches of 400
- ARCQUERY (json metatdata over HTTP)
    - retrieves all records from the ARC data portal to construct a list of grant Ids
    - queries the portal again by each specific Id to obtain rich json formatted metadata
    - combines the results into batches of 400
- OPEN DATA API (json metatdata over HTTP)
    - The OPEN DATA API Harvester retrieves JSON from any service that implements the US Government Project Open Data API (for dataSets)
    - Combines the results into batches of 400
- MAGDAQUERY (json metatdata over HTTP)
    - The MAGDA QUERY Harvester retrieves JSON from any service that implements MAGDA SOLR API (for dataSets), by limit of 400
    - Combines the results into batches of 400

Whilst the Harvester can retrieve metadata in any format, it must be transformed into RIF-CS XML to be compatible with the RDA Registry ingest process; where a transform is required, an XSL transformation can be incorporated within the ‘plug-in’ module.

When json data is provided the harvester attempts to serialize the json objects as xml documents, so they can be crasswalked to rifcs using XSLT.

The Harvester can perform simultaneous harvests; the maximum number of concurrent harvests can be set within its configuration.

Interaction with the Registry

The registry can requests a harvest by interacting with a shared database table with the Harvester. This table is `dbs_registry.harvests`. This table is checked every 30 seconds by the harvesters for any new jobs to do. Once a harvester found a new harvests job to do, the harvester will spawn a new thread to do that job. After the job is done, it will poll the Metadata Registry via the `response_url` URL (for eg http://registry-url/registry/import/put). The registry at this point will pick up the files harvested by the harvester and start the importing process. This overall process can be monitored by viewing the Data Source Dashboard in the Metadata Registry.

The status of the Harvester can also be viewed in the Registry Maintenance Dashboard, usually located at http://registry-url/registry/maintenance.

Installation & Basic Usage

The ARDC Harvester requires Python 3.4 to be installed on the server. The instructions on how to install Python 3.4 on a CentOS machine are available for viewing.

- Python 3.4 to be installed. (guide)
- A well configured and installed RDA Registry Software. (guide)

The instruction will install the harvester in `/usr/local/harvester` with the harvested contents being in `/var/harvested_contents`

Checking out the repository

```
cd /usr/local/
sudo git clone https://github.com/au-research/ANDS-Harvester.git harvester
```

Configure the harvester

```
cd harvester
sudo cp myconfig.sample myconfig.py
sudo vi myconfig.py
```

Sample configurations

```
ssl_context = ssl.create_default_context()
# how many records the harvester will gather before stopping if it's mode is set to "TEST"
test_limit = 50
# how often (seconds interval) the harvester will check for a new harvest
polling_frequency = 3
# how long (in seconds) can a harvest run before deemed to be pulled
max_up_seconds_per_harvest = 7200
# the path to itself TODO: refer to abs_path instead
run_dir = '/opt/ands/harvester/'
# abs_path is a new way to do it, but we left the old way in (too busy to retest)
abs_path = os.path.abspath(".")
# not used but was planning to send an alert in case a harvest fails
admin_email_addr = "u4187959@uni.edu.au"
# the url the registry is accepting import commands (once the harvest is completed)
response_url='http://localhost/api/import'
# not used but was introduced in case temporary files need to be stored
data_dir= '/tmp/data/harvester/'
# the main folder where all harvested content is stored in a ds_id/batch_id hierarchy
data_store_path= '/tmp/harvested_contents/'
# the directory where the harvester is writing its logs
log_dir= "/tmp/log/harvester/"
# the log "INFO, ERROR, DEBUG" level (DEBUG) is very verbose so try not to use it unless developing
log_level = "ERROR"
# the redis poster the harvester can message subscribers (eg the registry)
#redis_poster_host = 'test.ands.org.au'
redis_poster_host = ''
# java home is needed to run saxon and XSLT 2.0 transforms
java_home='path_to_java'
# the XSLT processor
saxon_jar='/path_to/saxon8.jar'
# database of the registry
db_host='db_host.org'
# the user name that the harvester connects
db_user="the db user's name"
db_passwd="the db user's password"
# the database name
db='dbs_registry'
# the database port
db_port=3306
# the harvets table
harvest_table ='harvests'
# max number of  asynchronous connection that the harvester will hit any given client
# increase it to too much and they might just blacklist us
tcp_connection_limit=5
# datasource attributes that are needed and not in the harvests table :-( TODO should bring them all into one
harvester_specific_datasource_attributes = "'xsl_file','title','harvest_method','uri','provider_type'," \
                                           "'advanced_harvest_mode','oai_set', 'last_harvest_run_date', " \
                                           "'harvest_params','user_defined_params', 'harvest_frequency'"
```

Download Saxon 8

For the harvester to initiate an XSLT transform after a successful harvest, `saxon8.jar` is required. Use a search engine to find saxon8.jar and install it as `/usr/share/java/saxon8.jar`

Make sure the log directory is created and writable

```
cd /usr/local/harvester
sudo mkdir log
sudo chmod 777 -R log
```

Make sure the harvested contents directory is created and writable

```
cd /var/
sudo mkdir harvested_contents
sudo chmod 777 -R harvested_contents
```

Install ARDC Harvester as a Linux Service

This will enable automatic restarts of the Harvester upon server restarts and enable a faster and easy way to start and stop the service

```
cd /usr/local/harvester
sudo cp ands-harvester /etc/init.d
sudo chmod 755 /etc/init.d/ands-harvester
sudo chkconfig --add ands-harvester
sudo chkconfig ands-harvester on
```

Start the harvester with

```
sudo service ands-harvester start
```

The harvester should start writing to its log at `/usr/local/harvester/log`, monitor this log for any activity or harvests errors.

Advanced Harvest Methods

Record providers will have the ability to select from three Advanced Harvest Modes to change the behaviour of the ARDC Harvester. These harvest modes include:

- **Standard** - will harvest all records from the data source feed (this will become the default mode)
- **Incremental** - will use OAI-PMH to support harvesting on a "from and until" basis. This means that the Harvester will only add new records that have been created or modified "from" the last harvest date, "until" the date of the new harvest.
- **Refresh** - will consider the latest Harvest to be the "complete set of records" and remove all previously harvested records that are no longer in the feed. Records created manually using the Add Registry Object screens will not be removed.

The "Incremental" harvest mode will only function correctly with OAI-PMH providers that correctly support the "from and until" parameters as per the OAI-PMH Specification.

For existing providers, the mode will be set to "Incremental" (which has identical behaviour to the current harvester).

Any new providers will have the mode default to "Standard". ARDC strongly recommends that providers who support OAI-PMH Incremental Harvesting should update the Advanced Harvest Mode to "Incremental" to ensure that their harvests perform optimally.

Harvester will ignore multiple copies of the same record (i.e. multiple records with identical keys) when they are received in the same harvest. The first record will be harvested and any duplicates found in that same harvest will be ignored.

« Access Management System Solr »