Pipeline Tools Documentation

Release 0.30.0

Mint Team

Oct 02, 2018

Contents

Note: This tool is still under active development, so there could be significant changes to its API.

This package provides utilities for retrieving files from the data storage service for the Human Cell Atlas, and for submitting an analysis bundle to the Human Cell Atlas Data Coordination Platform.

It also contains functions for interacting with Google Cloud Storage.

The steps in the submission process are as follows:

- Create analysis.json
- Create submission envelope and upload metadata
- Get URN needed to stage files
- Stage files by using the hca-cli
- Confirm submission

CHAPTER 1

Installing

Install it like this:

```
pip install git+git://github.com/HumanCellAtlas/pipeline-tools.git
```

You can use the cloud storage functions like this:

```
import pipeline_tools
from pipeline_tools import gcs_utils
bucket, object = gcs_utils.parse_bucket_blob_from_gs_link('gs://my-bucket/path/to/
↪object')
```

The rest of the package consists of scripts that are meant to be invoked from the command line as described below.

CHAPTER 2

Usage

2.1 get_analysis_workflow_metadata.py

Utility function fetches required information for creating submission to Ingest service, such as the Cromwell workflow metadata, the UUID of the analysis workflow, and the version of the given pipeline.

Invoke it like this:

```
get-analysis-workflow-metadata \
  --analysis_output_path ${analysis_output_path} \
  --cromwell_url ${cromwell_url} \
  --use_caas ${use_caas} \
  --caas_key_file ${caas_key_file}
```

All arguments are required, except the *–caas_key_file*, which is set to *None* by default.

2.2 create_analysis_metadata.py

Creates both analysis_protocol.json and analysis_process.json files, which are following different versions of the HCA metadata schema. For a full list of the HCA metadata schemas, check here.

Invoke it like this:

```
create-analysis-metadata \
  --analysis_id ${workflow_id} \
  --metadata_json ${metadata_json} \
  --input_bundles ${input_bundle_uuid} \
  --reference_bundle ${reference_bundle} \
  --run_type ${run_type} \
  --method ${method} \
  --schema_url ${schema_url} \
  --analysis_process_schema_version ${analysis_process_schema_version} \
```

(continues on next page)

(continued from previous page)

```
  --analysis_protocol_schema_version ${analysis_protocol_schema_version} \
  --pipeline_version ${pipeline_version} \
  --analysis_file_version ${analysis_file_version} \
  --inputs_file ${write_objects(inputs)} \
  --outputs_file ${write_lines(outputs)} \
  --format_map ${format_map}
```

All arguments are required.

2.3 create_envelope.py

Creates submission envelope and uploads metadata files.

Invoke it like this:

```
create-envelope \
  --submit_url ${submit_url} \
  --analysis_process_path analysis_process.json \
  --analysis_protocol_path analysis_protocol.json \
  --schema_url ${schema_url} \
  --analysis_file_version ${analysis_file_version}
```

All arguments are required.

2.4 get_staging_urn.py

Obtains URN needed for staging files. Queries ingest API until URN is available. The URN (Uniform Resource Name) is a long string that looks like this: hca:sta:aws:staging:{short hash}:{long hash}

It gets decoded by the hca-cli to extract the staging location and credentials needed to stage files.

Invoke it like this:

```
get-staging-urn \
  --envelope_url ${submission_url} \
  --retry_seconds ${retry_seconds} \
  --timeout_seconds ${timeout_seconds} > submission_urn.txt
```

envelope_url is required

2.5 get_files_to_upload.py

Gets a list of files to be uploaded(staged) by the HCA-CLI, writes the list to disk.

Invoke it like this:

```
get-files-to-upload \
    --files ${sep=' ' files} \
    --uploaded_files $uploaded_files
```

Both arguments are required.

2.6 confirm_submission.py

Confirms submission. This causes the ingest service to finalize the submission and create a bundle in the storage service.

Waits until submission status is "Valid", since submission cannot be confirmed until then.

Invoke it like this:

```
confirm-submission \
  --envelope_url ${submission_url} \
  --retry_seconds ${retry_seconds} \
  --timeout_seconds ${timeout_seconds}
```

envelope_url is required

CHAPTER 3

Testing

3.1 Running unit tests

To run unit tests, first create a virtual environment with the requirements:

```
virtualenv test-env
source test-env/bin/activate
pip install -r requirements.txt -r test-requirements.txt
```

Then, run unit tests from the root of the pipeline-tools repo like this:

```
bash test.sh
```

To run schema integration tests, do:

```
export TEST_SUITE="latest_schema"
bash test.sh
```

CHAPTER 4

API documentation

pipeline_tools.dcp_utils.**get_auth_token**(*http_requests*, *url='https://danielvaughan.eu.auth0.com/oauth/token'*, *client_id='Zdsog4nDAnhQ99yiKwMQWAPc2qUDlR99'*, *client_secret='t-OAE-GQk_nZZtWn-QQezJxDsLXmU7VSzlAh9cKW5vb87i90qlXGTvVNAjfT9weF'*, *audience='http://localhost:8080'*, *grant_type='client_credentials'*)

Request and get the access token for a trusted client from Auth0.

Note: We have hard-coded some test credentials here temporarily, which do not give any special permissions in the ingest service.

Parameters

- **http_requests** (HttpRequests) – the HttpRequests object to use
- **url** (*str*) – the url to the Auth0 domain oauth endpoint.
- **client_id** (*str*) – the value of the Client ID field of the Non Interactive Client of Auth0.
- **client_secret** (*str*) – the value of the Client Secret field of the Non Interactive Client of Auth0.
- **audience** (*str*) – the value of the Identifier field of the Auth0 Management API.
- **grant_type** (*str*) – type of OAuth 2.0 flow you want to run. e.g. client_credentials

Returns

A dict containing the JWT (JSON Web Token) and its expiry (24h by default), the scopes granted, and the token type.

Return type auth_token (dict)

Raises requests.HTTPError – for 4xx errors or 5xx errors beyond timeout

pipeline_tools.dcp_utils.**get_file_by_uuid**(*file_id*, *dss_url*, *http_requests*)

Retrieve a JSON file from the Human Cell Atlas data storage service by its id. Retry with exponentially increasing wait times between requests if there are any failures.

Parameters

- **file_id** (*str*) – the id of the file to retrieve.
- **dss_url** (*str*) – the url for the HCA data storage service, e.g. "https://dss.staging.data.humancellatlas.org/v1".
- **http_requests** (HttpRequests) – the HttpRequests object to use

Returns dict representing the contents of the JSON file

Return type dict

Raises requests.HTTPError – for 4xx errors or 5xx errors beyond timeout

pipeline_tools.dcp_utils.**get_manifest**(*bundle_uuid*, *bundle_version*, *dss_url*, *http_requests*)

Retrieve manifest JSON file for a given bundle uuid and version.

Retry with exponentially increasing wait times between requests if there are any failures.

TODO: Reduce the number of lines of code by switching to use DSS Python API client.

Instead of talking to the DSS API directly, using the DSS Python API can avoid a lot of potential issues, especially those related to the Checkout Service. A simple example of using the DSS Python client and the metadata-api to get the manifest would be:

‘‘‘python from humancellatlas.data.metadata.helpers.dss import download_bundle_metadata, dss_client

client = dss_client() version, manifest, metadata_files = download_bundle_metadata(client, ‘gcp’, bundle_uuid, directurls=True) ‘‘‘

Parameters

- **bundle_uuid** (*str*) – the uuid of the bundle
- **bundle_version** (*str*) – the bundle version, e.g. "2017-10-23T17:50:26.894Z"
- **dss_url** (*str*) – The url for the Human Cell Atlas data storage service,
- **"https** (*e.g.*) – //dss.staging.data.humancellatlas.org/v1"
- **http_requests** (HttpRequests) – the HttpRequests object to use

Returns A dict representing the full bundle manifest, with *directurls* for each file.

Return type dict

Raises requests.HTTPError – for 4xx errors or 5xx errors beyond timeout

pipeline_tools.dcp_utils.**make_auth_header**(*auth_token*)

Make the authorization headers to communicate with endpoints which implement Auth0 authentication API.

Parameters **auth_token** (*dict*) – a dict obtained from the Auth0 domain oauth endpoint, containing the signed JWT (JSON Web Token), its expiry, the scopes granted, and the token type.

Returns

A dict representing the headers with necessary token information to talk to Auth0 authentication required endpoints.

Return type headers (dict)

class pipeline_tools.http_requests.**HttpRequests**(*write_dummy_files=True*)

Wraps requests library and adds ability to record requests and responses.

When in record mode, will record requests and responses by writing them to files.

Requests made through this class will raise an error if response status code indicates an error.

should_record
bool – whether to record requests and responses.

record_dir
str – the directory where requests and responses should be written.

retry_timeout
int – time in seconds after which we will stop retrying

retry_max_tries
int – maximum number of retries before giving up

retry_multiplier
float – multiplier A for exponential retries, e.g. interval will be: A x 2^i

retry_max_interval
float – ceiling for retry interval

individual_request_timeout
int – time in seconds after which each request will timeout

static check_status(*response*)

Check that the response status code is in range 200-299, or 409. Raises a ValueError and prints response_text if status is not in the expected range. Otherwise, just returns silently. :param response.status: The actual HTTP status code. :type response.status: int :param response.text: Text to print along with status code when mismatch occurs :type response.text: str

Raises

- `requests.HTTPError` – for 5xx errors and prints response_text if status is not in the expected range.
- `Otherwise,`
- just returns silently.

4.0.0.0.1 Examples

check_status(200, 'foo') passes check_status(404, 'foo') raises error check_status(301, 'bar') raises error

get(**args*, ***kwargs*)

Calls requests.get function.

In addition to calling requests.get, this function will record the request and response if the HttpRequests object's should_record attribute is True.

Parameters **arguments that requests.get accepts are accepted here.** (*All*) –

Returns An object representing the response to the request.

Return type requests.Response

Raises

- `requests.HTTPError` – if 5xx error occurs

- `tenacity.RetryError` – if retry limit is reached and condition specified by
- retry kwarg is still not met

post(**args*, ***kwargs*)
Calls requests.post function.

In addition to calling requests.post, this function will record the request and response if the HttpRequests object's should_record attribute is True.

Parameters

- **arguments that requests.post accepts are accepted here except** (*All*) –
- **'body' -- use 'json' instead.** (*for*) –

Returns An object representing the response to the request.

Return type requests.Response

Raises

- `requests.HTTPError` – if 5xx error occurs
- `tenacity.RetryError` – if retry limit is reached and condition specified by
- retry kwarg is still not met

put(**args*, ***kwargs*)
Calls requests.put function.

In addition to calling requests.put, this function will record the request and response if the HttpRequests object's should_record attribute is True.

Parameters

- **arguments that requests.put accepts are accepted here except** (*All*) –
- **'body' -- use 'json' instead.** (*for*) –

Returns An object representing the response to the request.

Return type requests.Response

Raises

- `requests.HTTPError` – if 5xx error occurs
- `tenacity.RetryError` – if retry limit is reached and condition specified by
- retry kwarg is still not met

class pipeline_tools.http_requests.**RetryPolicy**(*retry_after_status_codes={301}*, ***kwargs*)
Wrapper around the urllib3.retry module that overrides which status codes should follow the retry_after header.

retry_after_status_codes
frozenset – Which status codes follow the retry_after header

classmethod from_int(*retries*, *redirect=True*, *default=None*)
Backwards-compatibility for the old retries format.

get_backoff_time()
Formula for computing the current backoff

Return type float

get_retry_after(*response*)
Get the value of Retry-After in seconds.

increment(*method=None*, *url=None*, *response=None*, *error=None*, *_pool=None*, *_stacktrace=None*)
Return a new Retry object with incremented retry counters.

Parameters

- **response** (HTTPResponse) – A response object, or None, if the server did not return a response.
- **error** (*Exception*) – An error encountered during the request, or None if the response was received successfully.

Returns A new Retry object.

is_exhausted()
Are we out of retries?

is_retry(*method*, *status_code*, *has_retry_after=False*)
Is this method/status code retryable? (Based on whitelists and control variables such as the number of total retries to allow, whether to respect the Retry-After header, whether this header is present, and whether the returned status code is on the list of status codes to be retried upon on the presence of the aforementioned header)

sleep(*response=None*)
Sleep between retry attempts.

This method will respect a server's Retry-After response header and sleep the duration of the time requested. If that is not present, it will use an exponential backoff. By default, the backoff factor is 0 and this method will return immediately.

CHAPTER 5

Table of Contents

- genindex
- modindex
- search

Python Module Index

p

Index