humiolib

Apr 17, 2023

Contents

CHAPTER 1

Humiolib

The *humiolib* library is a wrapper for Humio's web API, supporting easy interaction with Humio directly from Python. Full documentation for this repository can be found at https://python-humio.readthedocs.io/en/latest/readme.html.

1.1 Vision

The vision for *humiolib* is to create an opinionated wrapper around the Humio web API, supporting log ingestion and log queries. The project does not simply expose web endpoints as Python methods, but attempts to improve upon the usability experience of the API. In addition the project seeks to add non-intrusive quality of life features, so that users can focus on their primary goals during development.

1.2 Governance

This project is maintained by employees at Humio ApS. As a general rule, only employees at Humio can become maintainers and have commit privileges to this repository. Therefore, if you want to contribute to the project, which we very much encourage, you must first fork the repository. Maintainers will have the final say on accepting or rejecting pull requests. As a rule of thumb, pull requests will be accepted if:

- The contribution fits with the project's vision
- All automated tests have passed
- The contribution is of a quality comparable to the rest of the project

The maintainers will attempt to react to issues and pull requests quickly, but their ability to do so can vary. If you haven't heard back from a maintainer within 7 days of creating an issue or making a pull request, please feel free to ping them on the relevant post.

The active maintainers involved with this project include:

- Alexander Brandborg

1.3 Installation

The *humiolib* library has been published on PyPI, so you can use *pip* to install it:

```
pip install humiolib
```

1.4 Usage

The examples below seek to get you going with *humiolib*. For further documentation have a look at the code itself.

1.4.1 HumioClient

The HumioClient class is used for general interaction with Humio. It is mainly used for performing queries, as well as managing different aspects of your Humio instance.

```
from humiolib.HumioClient import HumioClient

# Creating the client
client = HumioClient(
     base_url= "https://cloud.humio.com",
     repository= "sandbox",
     user_token="*****")

# Using a streaming query
webStream = client.streaming_query("Login Attempt Failed", is_live=True)
for event in webStream:
    print(event)

# Using a queryjob
queryjob = client.create_queryjob("Login Attempt Failed", is_live=True)
poll_result = queryjob.poll()
for event in poll_result.events:
    print(event)

# With a static queryjob you can poll it iterativly until it has been exhausted
queryjob = client.create_queryjob("Login Attempt Failed", is_live=False)
for poll_result in queryjob.poll_until_done():
    print(poll_result.metadata)
    for event in poll_result.events:
            print(event)
```

1.4.2 HumioIngestClient

The HumioIngestClient class is used for ingesting data into Humio. While the HumioClient can also be used for ingesting data, this is mainly meant for debugging.

```
from humiolib.HumioClient import HumioIngestClient

# Creating the client
client = HumioIngestClient(
   base_url= "https://cloud.humio.com",
```

(continues on next page)

(continued from previous page)

```
   ingest_token="*****")

# Ingesting Unstructured Data
messages = [
      "192.168.1.21 - user1 [02/Nov/2017:13:48:26 +0000] \"POST /humio/api/v1/ingest/
↪elastic-bulk HTTP/1.1\" 200 0 \"-\" \"useragent\" 0.015 664 0.015",
      "192.168.1..21 - user2 [02/Nov/2017:13:49:09 +0000] \"POST /humio/api/v1/ingest/
↪elastic-bulk HTTP/1.1\" 200 0 \"-\" \"useragent\" 0.013 565 0.013"
   ]

client.ingest_messages(messages)

# Ingesting Structured Data
structured_data = [
      {
         "tags": {"host": "server1" },
         "events": [
               {
                  "timestamp": "2020-03-23T00:00:00+00:00",
                  "attributes": {"key1": "value1", "key2": "value2"}
               }
         ]
      }
   ]

client.ingest_json_data(structured_data)
```

CHAPTER 2

Reference

2.1 HumioClient

class humiolib.HumioClient.**BaseHumioClient**(*base_url*)
Base class for other client types, is not meant to be instantiated

class humiolib.HumioClient.**HumioClient**(*repository*, *user_token*, *base_url='http://localhost:3000'*)
A Humio client that gives full access to the underlying API. While this client can be used for ingesting data, we recommend using the HumioIngestClient made exclusivly for ingestion.

add_file_contents(*file_name*, *file_headers*, *changed_rows*, *column_changes=[]*, *offset=0*, *limit=200*)
Add contents to a file

Parameters

- **file_name** (*string*) – Name of file
- **file_headers** (*list*) – Headers of the file
- **changed_rows** (*list*) – Rows within the offset and limit to overwrite existing rows
- **column_changes** (*list, optional*) – Column changes that will be applied to all rows in the file
- **offset** (*int, optional*) – Starting index to replace the old rows with the updated ones.
- **limit** (*int, optional*) – Used to determine when to stop replacing rows, by adding the limit to the offset

Returns Response data to web request as json string

Return type str

create_file(*file_name*)
Create new file.

Parameters **file_name** (*string*) – Name of file

Returns Response data to web request as json string

Return type str

create_queryjob(*query_string*, *start=None*, *end=None*, *is_live=None*, *timezone_offset_minutes=None*, *arguments=None*, *raw_data=None*, ***kwargs*)

Creates a queryjob on Humio, which executes asynchronously of the calling code. The returned QueryJob instance can be used to get the query results at a later time. Queryjobs are good to use for live queries, or static queries that return smaller amounts of data.

Parameters

- **query_string** (*str*) – Humio query
- **start** (*Union[int, str], optional*) – Starting time of query
- **end** (*Union[int, str], optional*) – Ending time of query
- **is_live** (*int, optional*) – Ending time of query
- **is_live** – Timezone offset in minutes
- **argument** (*dict(string->string), optional*) – Arguments specified in query
- **raw_data** (*dict(string->string), optional*) – Additional arguments to add to POST body under other keys

Returns An instance that grants access to the created queryjob and associated results

Return type QueryJob

create_user(*email*, *isRoot=False*)

Create user on Humio instance. Method is idempotent

Parameters

- **email** (*str*) – Email of user to create
- **isRoot** (*bool, optional*) – Indicates whether user should be root

Returns Response to web request as json string

Return type str

delete_file(*file_name*)

Delete an existing file.

Parameters **file_name** (*string*) – Name of file

Returns Response to web request as json string

Return type str

delete_user_by_email(*email*)

Delete user by email.

Parameters **email** (*string*) – Email of user to delete.

Returns Response to web request as json string

Return type str

delete_user_by_id(*user_id*)

Delete user from Humio instance.

Parameters **user_id** (*string*) – Id of user to delete.

Returns Response to web request as json string

Return type str

get_file(*file_name*, *encoding=None*)
Get specific file on repository

Parameters **file_name** (*string*) – Name of file to get.

Returns Response to web request as json string

Return type str

get_file_content(*filename*, *offset=0*, *limit=200*, *filter_string=None*)
Get the contents of a file

Parameters

- **file_name** – Name of file.
- **offset** (*int*) – Starting index to replace the old rows with the updated ones.
- **limit** (*int*) – Used to find when to stop replacing rows, by adding the limit to the offset
- **filter_string** (*string, optional*) – Used to apply a filter string

Returns Response to web request as json string

Return type str

get_status(***kwargs*)
Gets status of Humio instance

Returns Response to web request as json string

Return type str

get_user_by_email(*email*)
Get a user associated with Humio instance by email

Parameters **email** (*str*) – Email of queried user

Returns Response to web request as json string

Return type str

get_users()
Gets users registered to Humio instance

Returns Response to web request as json string

Return type str

ingest_json_data(*json_elements=None*, ***kwargs*)
Ingest structured json data to repository. Structure of ingested data is discussed in: https://docs.humio.com/reference/api/ingest/#structured-data

Parameters

- **messages** (*list(string), optional*) – A list of event strings.
- **parser** (*string, optional*) – Name of parser to use on messages.
- **fields** (*dict(string->string), optional*) – Fields that should be added to events after parsing.
- **tags** (*dict(string->string), optional*) – Tags to associate with the messages.

Returns Response to web request as json string

Return type str

ingest_messages(*messages=None*, *parser=None*, *fields=None*, *tags=None*, ***kwargs*)
Ingest unstructred messages to repository. Structure of ingested data is discussed in: https://docs.humio.com/reference/api/ingest/#parser

Parameters

- **messages** (*list(string), optional*) – A list of event strings.
- **parser** (*string, optional*) – Name of parser to use on messages.
- **fields** (*dict(string->string), optional*) – Fields that should be added to events after parsing.
- **tags** (*dict(string->string), optional*) – Tags to associate with the messages.

Returns Response to web request as json string

Return type str

list_files()
List uploaded files on repository

Returns Response to web request as json string

Return type str

remove_file_contents(*file_name*, *offset=0*, *limit=200*)
Remove contents of a file

Parameters

- **file_name** (*string*) – Name of file
- **offset** (*int, optional*) – Starting index to replace the old rows with the updated ones.
- **limit** (*int, optional*) – Used to find when to stop replacing rows, by adding the limit to the offset

Returns Response data to web request as json string

Return type str

streaming_query(*query_string*, *start=None*, *end=None*, *is_live=None*, *timezone_offset_minutes=None*, *arguments=None*, *raw_data=None*, ***kwargs*)
Humio Query type that opens up a streaming socket connection to Humio. This is the preferred way to do static queries with large result sizes. It can be used for live queries, but not that if data is not passed back from Humio for a while, the connection will be lost, resulting in an error.

Parameters

- **query_string** (*str*) – Humio query
- **start** (*Union[int, str], optional*) – Starting time of query
- **end** (*Union[int, str], optional*) – Ending time of query
- **is_live** (*bool, optional*) – Ending time of query
- **timezone_offset_minutes** (*int, optional*) – Timezone offset in minutes
- **argument** (*dict(string->string), optional*) – Arguments specified in query

- **raw_data** (*dict(string->string), optional*) – Additional arguments to add to POST body under other keys

Returns A generator that returns query results as python objects

Return type Generator

class humiolib.HumioClient.**HumioIngestClient**(*ingest_token*, *base_url='http://localhost:3000'*)

A Humio client that is used exclusivly for ingesting data

ingest_json_data(*json_elements=None*, ***kwargs*)

Ingest structured json data to repository. Structure of ingested data is discussed in: https://docs.humio.com/reference/api/ingest/#structured-data

Parameters **json_elements** (*str*) – Structured data that can be parsed to a json string.

Returns Response to web request as json string

Return type str

ingest_messages(*messages=None*, *parser=None*, *fields=None*, *tags=None*, ***kwargs*)

Ingest unstructred messages to repository. Structure of ingested data is discussed in: https://docs.humio.com/reference/api/ingest/#parser

Parameters

- **messages** (*list(string), optional*) – A list of event strings.
- **parser** (*string, optional*) – Name of parser to use on messages.
- **fields** (*dict(string->string), optional*) – Fields that should be added to events after parsing.
- **tags** (*dict(string->string), optional*) – Tags to associate with the messages.

Returns Response to web request as json string

Return type str

2.2 QueryJob

class humiolib.QueryJob.**BaseQueryJob**(*query_id*, *base_url*, *repository*, *user_token*)

Base QueryJob class, not meant to be instantiated. This class and its children manage access to queryjobs created on a Humio instance, they are mainly used for extracting results from queryjobs.

poll(***kwargs*)

Polls the queryjob for the next segment of data, and handles edge cases for data polled

Returns A data object that contains events of the polled segment and metadata about the poll

Return type *PollResult*

class humiolib.QueryJob.**LiveQueryJob**(*query_id*, *base_url*, *repository*, *user_token*)

Manages a live queryjob

class humiolib.QueryJob.**PollResult**(*events*, *metadata*)

Result of polling segments of queryjob results. We choose to return these clusters of data, rather than just a list of events, as the metadata returned changes between polls.

class humiolib.QueryJob.**StaticQueryJob**(*query_id*, *base_url*, *repository*, *user_token*)

Manages a static queryjob

poll(***kwargs*)
Polls next segment of result

Returns A data object that contains events of the polled segment and metadata about the poll

Return type *PollResult*

poll_until_done(***kwargs*)
Create generator for yielding poll results

Returns A generator for query results

Return type Generator

2.3 WebCaller

class humiolib.WebCaller.**WebCaller**(*base_url*)
Object used for abstracting calls to the Humio API

call_graphql(*headers=None*, *data=None*, ***kwargs*)
Call Humio's GraphQL endpoint

Parameters

- **headers** (*dict, optional*) – Http headers
- **data** (*dict, optional*) – Post request body for GraphQL

Returns Response to web request

Return type Response Object

call_rest(*verb*, *endpoint*, *headers=None*, *data=None*, *files=None*, *stream=False*, ***kwargs*)
Call one of Humio's REST endpoints

Parameters

- **verb** (*str*) – Http verb
- **endpoint** (*str*) – Called Humio endpoint
- **headers** (*dict, optional*) – Http headers
- **data** (*dict, optional*) – Post request body
- **files** (*dict, optional*) – Files to be posted
- **stream** (*bool, optional*) – Indicates whether a stream request should be made

Returns Response to web request

Return type Response Object

static response_as_json(*func*)
Wrapper to take the raw requests responses and turn them into json

Parameters **func** (*Function*) – Function to be wrapped.

Returns Result of function, parsed into python objects from json

Return type dict

class humiolib.WebCaller.**WebStreamer**(*connection*)
Wrapper for a web request stream. Its main purpose is to catch errors during stream and raise them again as custom Humio exceptions.

2.4 HumioExceptions

exception humiolib.HumioExceptions.**HumioConnectionDroppedException**

exception humiolib.HumioExceptions.**HumioConnectionException**

exception humiolib.HumioExceptions.**HumioException**

exception humiolib.HumioExceptions.**HumioHTTPException**(*message*, *status_code=None*)

exception humiolib.HumioExceptions.**HumioQueryJobExhaustedException**

exception humiolib.HumioExceptions.**HumioQueryJobExpiredException**

exception humiolib.HumioExceptions.**HumioTimeoutException**

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1 Ways To Contribute

There are many different ways, in which you may contribute to this project, including:

- Opening issues by using the issue tracker, using the correct issue template for your submission.
- Commenting and expanding on open issues.
- Propose fixes to open issues via a pull request.

We suggest that you create an issue on GitHub before starting to work on a pull request, as this gives us a better overview, and allows us to start a conversation about the issue. We also encourage you to separate unrelated contributions into different pull requests. This makes it easier for us to understand your individual contributions and faster at reviewing them.

3.2 Setting Up *humiolib* For Local Development

1. Fork python-humio (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com/humio/python-humio.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

4. Install *humiolib* from your local repository:

```
pip install -e .
```

Now you can import *humiolib* into your Python code, and you can make changes to the project locally.

5. As your work progresses, regularly commit to and push your branch to your own fork on GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

3.3 Running Tests locally

Testing is accomplished using the pytest library. This should automatically be installed on your machine, when you install the *humiolib* package. To run tests simply execute the following command in the *tests* folder:

```
pytest
```

Humio API calls made during tests have been recorded using vcr.py and can be found in the *tests/cassettes* folder. These will be *played back* when tests are run, so you do not need to set up a Humio instance to perform the tests. Please do not re-record cassettes unless you're really familiar with vcr.py.

3.4 Building Documentation From Source

If you're contributing to the documentation, you need to build the docs locally to inspect your changes.

To do this, first make sure you have the documentation dependencies installed:

```
pip install -r docs/requirements.txt
```

Once dependencies have been installed build the HTML pages using sphinx:

```
sphinx-build -b html docs build/docs
```

You should now find the generated HTML in `build/docs`.

3.5 Making A Pull Request

When you have made your changes locally, or you want feedback on a work in progress, you're almost ready to make a pull request.

If you have changed part of the codebase in your pull request, please go through this checklist:

1. Write new test cases if the old ones do not cover your new code.
2. Update documentation if necessary.
3. Add yourself to `AUTHORS.rst`.

If you have only changed the documentation you only need to add yourself to `AUTHORS.rst`.

When you've been through the applicable checklist, push your final changes to your development branch on GitHub. Afterwards, use the GitHub interface to create a pull request to the official repository.

3.6 Publishing the Library to PyPI

This section describes the manual process of publishing this library to PyPI. This is a task only done by maintainers of the repository, and it is always done from the `master` branch.

Before the package can be published, you need to bump the semantic version of the library. This is done using the program `bump2version`, which can be installed as such:

```
pip3 install bump2version
```

You can now bump the library to either a new patch, minor or major version, using the following command:

```
bumpversion (patch | minor | major)
```

This will bump the version across library as specified in `.bumpversion.cfg`.

Once the version has been bumped, add a descriptive entry to `CHANGELOG.rst` about what has changed in the new version of the library.

You will not need to change any more tracked files during the publishing process, so create a new commit to encompass the changes made by your version bump now.

To build the library into a package run:

```
python3 setup.py bdist_wheel sdist
```

This will create a build and source distribution of the library within the `/dist` folder.

To upload these files to PyPI you need to install `twine`, which can be done using the following command:

```
pip3 install twine
```

Now upload the contents of `/dist` to PyPI by entering the following command and following the prompt on the screen:

```
twine upload dist/*
```

Congratulations! The new version of the package should now be live on PyPI for all to enjoy.

3.7 Terms of Service For Contributors

For all contributions to this repository (software, bug fixes, configuration changes, documentation, or any other materials), we emphasize that this happens under GitHubs general Terms of Service and the license of this repository.

3.7.1 Contributing as an individual

If you are contributing as an individual you must make sure to adhere to:

The GitHub Terms of Service **Section D. User-Generated Content,** Subsection: 6. Contributions Under Repository License :

Whenever you make a contribution to a repository containing notice of a license, you license your contribution under the same terms, and you agree that you have the right to license your contribution under those terms. If you have a separate agreement to license your contributions under different terms, such as a contributor license agreement, that agreement will supersede. Isn't this just how it works already? Yep. This is widely accepted as the norm in

the open-source community; it's commonly referred to by the shorthand "inbound=outbound". We're just making it explicit."

3.7.2 Contributing on behalf of a Corporation

If you are contributing on behalf of a Corporation you must make sure to adhere to:

The GitHub Corporate Terms of Service **Section D. Content Responsibility; Ownership; License Rights,** subsection 5. Contributions Under Repository License:

Whenever Customer makes a contribution to a repository containing notice of a license, it licenses such contributions under the same terms and agrees that it has the right to license such contributions under those terms. If Customer has a separate agreement to license its contributions under different terms, such as a contributor license agreement, that agreement will supersede

CHAPTER 4

Authors

4.1 Current Maintainer(s)

- Alexander Brandborg, @alexanderbrandborg

4.2 Original Author and First Commit

- Sergey Grigorev, @xorsnn

4.3 Contributors (alpha by username)

- Anders Fogh Eriksen @Fogh
- Hanne Moa @hmpf
- Kristian Gausel @KGausel
- Peter Mechlenborg @pmech
- Sam @samgdf
- Chris Fraser @swefraser
- Vishal Kuo @vishalkuo

CHAPTER 5

Changelog

5.1 0.2.0 (2020-03-30)

Initial real release to PyPI

Added:

- Tests, mocking out API calls with vcr.py
- Custom error handling to completly wrap url library used
- QueryJob class

Changed:

- Whole API interface has been updated
- Updated Sphinx documentation

Removed:

- A few configuration files left over from earlier versions

5.2 0.2.2 (2020-05-19)

Bugfixing to ensure that static queryjobs can be polled for all their results

Added:

- Static queryjobs can now be queried for more than one segment

Changed:

- Upon polling from a QueryJob it will now stall until it can poll data from Humio, ensuring that an empty result is not returned prematurely.

Removed:

- The poll_until_done method has been removed from live query jobs, as this does not make conceptual sense to do, in the same manner as a static query job.

5.3 0.2.3 (2021-08-13)

Smaller bugfixes Changed:

- Fix urls in docstrings in HumioClient.py
- Propagate kwargs to poll functions in QueryJob.py

5.4 0.2.4 (2022-08-15)

Smaller file related bugfixes Changed:

- upload_file function no longer attempts a cast to json
- list_files function now works on newer versions of humio

5.5 0.2.5 (2023-04-17)

Expand file functionality Changed:

- Added additional endpoints for manipulating files via GraphQL

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

P

R

S

W