Python Atlas client Documentation

Release 1.0.0

Atlas client

Aug 19, 2019

Contents

Contents:

CHAPTER 1

Installation

Install the package with pip:

```
$ pip install atlasclient
```

CHAPTER 2

Apache Atlas Client in Python

Apache Atlas client in Python. Only compatible with Apache Atlas REST API **v2**.

- Free software: Apache Software License 2.0
- Documentation: https://atlasclient.readthedocs.io.

2.1 Get started

```
>>> from atlasclient.client import Atlas
>>> client = Atlas('<atlas.host>', port=21000, username='admin', password='admin')
>>> client.entity_guid(<guid>).status
>>> params = {'typeName': 'DataSet', 'attrName': 'name', 'attrValue': 'data', 'offset
↪': '1', 'limit':'10'}
>>> search_results = client.search_attribute(**params)
>>> for s in search_results:
...     for e in s.entities:
...         print(e.name)
...         print(e.guid)
```

2.2 Features

- Lazy loading: requests are only performed when data are required and not yet available

- Resource object relationships: REST API from sub-resources are done transparently for the user, for instance the user does not have to know that it needs to trigger a different REST request for getting the classifications of a specific entity.

2.3 TODO features

- allow multiprocessing

2.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 3

Usage

To use atlasclient:

```
import atlasclient
```

This Python client is based on the Apache Atlas REST API v2.

The following groups of resources can be accessed:

- DiscoveryREST
- EntityREST
- LineageREST
- RelationshipREST
- TypesREST
- AdminREST

Below a few examples to access some of the resources.

Make sure atlasclient is properly installed (see here).

First you need to create a connection object:

```
from atlasclient.client import Atlas
client = Atlas(your_atlas_host, port=21000, username='admin', password='admin')
```

Replace *your_atlas_host* by the actual host name of the Atlas server. Note that port 21000 might also be different in your case. Port 21000 is default port when using HTTP with Atlas, and 21443 for HTTPS.

To access the list of entry points:

```
from atlasclient.client import ENTRY_POINTS
ENTRY_POINTS
```

You'll get a dictionary with ('key': 'value') corresponding to ('client method': 'model class'): *{'entity_guid': <class 'atlasclient.models.EntityGuid'>, ...}*. For example, we can use:

```
client.entity_guid(GUID)
```

'entity_guid' is used as a method of the 'client' object.

3.1 DiscoveryREST

This section explains how you can search for entities per attribute name, or search using a SQL-like query, and more ;).

3.1.1 Search by attribute

To search for entities with a special attribute name:

```
params = {'typeName': 'DataSet', 'attrName': 'name', 'attrValue': 'data', 'offset': '1
↪', 'limit': '10'}
search_results = client.search_attribute(**params)
#  Info about all entities in one dict
for s in search_results:
    print(s._data)
#  Getting name and guid of each entity
for s in search_results:
    for e in s.entities:
        print(e.name)
        print(e.guid)
```

3.1.2 Search with basic terms

To retrieve data for the specified full text query:

```
params = {'attrName': 'name', 'attrValue': 'data', 'offset': '1', 'limit': '10'}
search_results = client.search_basic(**params)
for s in search_results:
    for e in s.entities
        print(e.guid)
```

Attribute based search *(POST /v2/search/basic)* for entities satisfying the search parameters:

```
data = {'attrName': 'name', 'attrValue': 'data', 'offset': '1', 'limit': '10'}
search_results = client.search_basic.create(data=data)
for e in search_results.entities:
    print(e.guid)
```

3.1.3 Search by DSL

To retrieve data for the specified DSL:

```
params = {'typeName': 'hdfs_path', 'classification': 'Confidential'}
search_results = client.search_dsl(**params)
for s in search_results:
    for e in s.entities:
```

(continues on next page)

(continued from previous page)

```
        print(e.classificationNames)
        print(e.attributes)
```

DSL Search has a helper function available when you specify a SELECT clause or attribute in your search query.

_search_collection = client.search_dsl(**dsl_param) for collection in _search_collection:

attributes = collection.flatten_attrs()

3.2 SavedSearchREST

This section explains how to get, create saved search, update or delete them.

3.2.1 Get all saved search for user

To retrieve saved search for the Atlas user:

```
search_saved = client.search_saved()
for s in search_saved:
    print(s._data)
    print(s.name)
```

3.2.2 Get saved search by name (for user)

To retrieve saved search for the Atlas user by name:

```
search_saved = client.search_saved(NAME)
print(s.name)
print(s.ownerName)
```

3.2.3 Create saved search by name (for user)

To create saved search for the Atlas user by name:

```
payload = """{
"name": "trying",
"ownerName": "svc_data_catalog_api",
"searchType": "BASIC",
"searchParameters": {
    "typeName": "rdbms_db",
    "excludeDeletedEntities": true,
    "includeClassificationAttributes": false,
    "includeSubTypes": true,
    "includeSubClassifications": true,
    "limit": 0,
    "offset": 0
},
    "uiParameters": "Select::0,Name::1,Owner::2,Description::3,Type::4,
↪Classifications::5,Term::6,Db::7"
}"""
```

(continues on next page)

(continued from previous page)

```
response = client.search_saved.create(data=json.loads(payload))
```

3.2.4 Update saved search by guid (for user)

To create saved search for the Atlas user by name:

```
payload = """{"guid": "fa1f15f0-09fc-403d-8ad7-3bcac379c3f9", "name": "trying2"}"""
response = client.search_saved.update(data=json.loads(payload))
```

3.2.5 To delete saved search by guid (for user)

To delete saved search for the Atlas user by guid:

```
client.search_saved.delete(guid=GUID)
```

3.3 EntityREST

This section explains how to create entities, update or delete them.

3.3.1 Create Entity

To create an entity, one needs to create a Python dictionary which will define the entity. This can be done from a json file:

```
import json
with open('my_entity_file.json') as json_file:
    entity_dict = json.load(json_file)
```

One can also just define the dictionary in Python. Note that if the user wants to pass a 'null' value, he should assign a value None in Python dictionary. It will be automatically convert to 'null' when requesting.

Once the entity dictionary is created, the entity can actually be created on Atlas with:

```
client.entity_post.create(data=entity_dict)
```

3.3.2 Get entity by GUID

If you know the GUID of the entity you want to fetch, you can follow these steps to get all info about this entity:

```
entity = client.entity_guid(GUID)
entity._data
```

To access some specific attribute of that entity, say the description:

```
entity.entity['attributes']['description']
```

It shows up as a dictionary. So one can get the list of all attributes with:

```
entity.entity['attributes'].keys()
```

3.3.3 Update entity by GUID

Suppose you want to change the description of the entity here above and send it to Atlas:

```
entity.entity['attributes']['description'] = 'my new description'
entity.update(attribute='description')
```

3.3.4 Delete entity by GUID

To delete our entity:

```
entity.delete()
```

3.3.5 Get classifications by GUID

To get all classification type names related to an entity GUID:

```
entity = client.entity(GUID)
for classification_info in entity.classifications:
    for classification_item in classification_info.list:
        print(classification_item.typeName)
```

3.3.6 Update classifications by GUID

To update classifications to an existing entity represented by a guid:

```
entity = client.entity(GUID)
for classification_info in entity.classifications:
    for classification_item in classification_info.list:
        if classification_item.typeName == 'Semi-Confidential'
            classification_item.typeName = 'Confidential'
entity.classifications.update()
```

The entity will now be tagged as 'Confidential' instead of 'Semi-Confidential'.

3.3.7 Create classifications by GUID

To add classifications to an existing GUID:

```
new_classifications = [{"typeName": "Confidential"},
                       {"typeName": "Customer"}
                      ]
entity = client.entity(GUID)
entity.classifications.create(data=new_classifications)
```

This will create 2 new classifications for the entity.

3.3.8 Get classification info by GUID and by classification type name

To get info about some specific classification for some entity:

```
entity = client.entity(GUID)
entity.classifications('Confidential').refresh()._data
```

The refresh() method is used to load data from the Atlas server, which is then stored in the _data attribute.

To get some specific info about the classification, say the 'totalCount':

```
entity.classifications('Confidential').totalCount
```

In that case, no need to use the refresh method since the client will see that the attribute totalCount is not yet available and will therefore send a request to the Atlas server.

3.3.9 Delete a classification by GUID

To delete a given classification from an existing entity represented by a GUID:

```
client.entity_guid(GUID).classifications('Confidential').delete()
```

This will delete the classification 'Confidential' for that specific entity only.

3.3.10 Get entities by bulk

To retrieve list of entities identified by its GUIDs:

```
bulk_collection = client.entity_bulk(guid=[GUID1, GUID2])
```

3.3.11 Get entities by bulk (with relationship attributes)

In some cases, you may want to need the details of relationship attributes along with entity, There is a helper function available for that:

```
bulk_collection = client.entity_bulk(guid=[GUID1, GUID2])
for collection in bulk_collection:
    entities = collection.entities_with_relationships()


# You can also specify the attributes as a list you want in particular to optimize␣
↪implementation
for collection in bulk_collection:
    entities = collection.entities_with_relationships(attributes=["database"])
```

3.3.12 Create entities by bulk

To create entities:

```
bulk = {"entities" : [ {
                "attributes": {"qualifiedName": "my_awesome_data", "name": "my_
↪awesome_data_name", "path": "/my-awesome-path"},
                "status" : "ACTIVE",
                "version" : 3,
                "classifications" : [ {"typeName" : "Customer"}, {"typeName" :
↪"Confidential"}],
                "typeName" : "hdfs_path"}],
         "referredEntities": {}
        }
client.entity_bulk.create(data=bulk)
```

This will create an hdfs_path entity with 2 classifications. Note that you can pass a list of entities (not limited to 1).

3.3.13 Delete multiple entities

To delete a list of entities:

```
client.entity_bulk.delete(guid=[GUID1, GUID2])
```

3.3.14 Associate a tag to multiple entities

To associate a tag to multiple entities:

```
entity_bulk_tag = {"classification": {"typeName": "Confidential"},
                   "entityGuids": [GUID1, GUID2]}
client.entity_bulk_classification.create(data=entity_bulk_tag)
```

This will create the tag 'Confidential' both GUIDs.

3.3.15 Get entity by unique attribute

To fetch an entity given its type and unique attribute:

```
entity = client.entity_unique_attribute('hdfs_path', qualifiedName='/my/awesome/path')
```

3.3.16 Update entity for subset of attributes

To update a subset of attributes on an entity which is identified by its type and unique attribute:

```
####  TO BE IMPLEMENTED ####
```

3.3.17 To delete an entity by unique attribute

To delete an entity identified by its type and unique attributes:

```
entity = client.entity_unique_attribute('hdfs_path', qualifiedName='/my/awesome/path')
entity.delete()
```

3.4 LineageREST

3.4.1 Get lineage by GUID

To get lineage info about entity identified by GUID:

```
lineage = client.lineage_guid(GUID)
print(lineage.relations)
print(lineage.lineageDirection)
```

3.5 RelationshipREST

TO BE DONE...

3.6 TypesREST

3.6.1 Get typeDefs

Typedefs can be seen as a collection of type definitions in Atlas and can accessed with:

```
client.typedefs
```

This only creates an object is not actually requesting the Atlas server. Suppose we want to access all elements of type 'enumDefs':

```
for t in client.typedefs:
    for e in t.enumDefs:
        for el in e.elementDefs:
            print(el.value)
```

We can access the classification types in a similar way:

```
for t in client.typedefs:
    for classification_type in t.classificationDefs:
        print(classification_type.description)
```

Idem for entityDefs and structDefs.

3.6.2 Delete typeDefs

To delete typedefs:

```
client.typedefs.delete(data=typedef_dict)
```

Where *typedef_dict* is the body to pass. Here is an example as illustration:

```
typedef_dict = {
"enumDefs":[],
"structDefs":[],
"classificationDefs":[],
```

(continues on next page)

(continued from previous page)

```
"entityDefs":[
   {
      "superTypes":[
        "DataSet"
      ],
      "name":"test_entity_7",
      "description":"test_entity_7",
      "createdBy": "admin",
      "updatedBy": "admin",
      "attributeDefs":[
         {
            "name":"test_7_1",
            "isOptional": True,
            "isUnique": False,
            "isIndexable": False,
            "typeName":"string",
            "valuesMaxCount":1,
            "cardinality":"SINGLE",
            "valuesMinCount":0
         },
        {
            "name":"test_7_2",
            "isOptional": True,
            "isUnique": False,
            "isIndexable": False,
            "typeName":"string",
            "valuesMaxCount":1,
            "cardinality":"SINGLE",
            "valuesMinCount":0
         }
      ]

   }
]
}
```

3.6.3 Create typeDefs

To create typedefs:

```
client.typedefs.create(data=typedef_dict)
```

An example for *typedef_dict* is given at the subsection above.

3.6.4 Update typeDefs

To update typedefs:

```
client.typedefs.update(data=typedef_dict)
```

An example for *typedef_dict* is given at the subsection above.

3.6.5 Get typeDefs headers

To get typedefs headers:

```
for header in client.typedefs_headers:
    print(header.name)
    print(header.category)
```

3.6.6 Get classificationDefs by GUID

To get classificationdefs by GUID:

```
class_defs = client.classificationdef_guid(CLASSIFICATION_GUID)
class_defs.name
class_defs._data
```

3.6.7 Get classificationDefs by name

To get classificationdefs by name:

```
CLASSIFICATION_NAME = 'Confidential'
class_defs = client.classificationdef_name(CLASSIFICATION_NAME)
class_defs.description
```

3.6.8 Get entityDefs by GUID

To get entitydefs by GUID:

```
entity_defs = client.entitydef_guid(ENTITY_GUID)
entity_defs.description
```

3.6.9 Get entityDefs by name

To get entitydefs by name:

```
ENTITY_NAME = 'hdfs_path'
entity_defs = client.entitydef_name(ENTITY_NAME)
entity_defs.description
```

3.6.10 Get enumDefs by GUID

To get enumdefs by GUID:

```
enum_defs = client.enumdef_guid(ENUM_GUID)
enum_defs.elementDefs
```

3.6.11 Get enumDefs by name

To get enumdefs by name:

```
ENUM_NAME = 'file_action'
enum_defs = client.enumdef_name(ENUM_NAME)
enum_defs.elementDefs
```

3.6.12 Get relationshipDefs by GUID

To get relationshipdefs by GUID:

```
relationship_defs = client.relationshipdef_guid(RELATIONSHIP_GUID)
relationship_defs._data
```

3.6.13 Get relationshipDefs by name

To get relationshipdefs by name:

```
relationship_defs = client.relationshipdef_guid(RELATIONSHIP_NAME)
relationship_defs._data
```

3.6.14 Get structDefs by GUID

To get structdefs by GUID:

```
struct_defs = client.structdef_guid(STRUCT_GUID)
struct_defs._data
```

3.6.15 Get structDefs by name

To get structdefs by name:

```
struct_defs = client.structdef_guid(STRUCT_NAME)
struct_defs._data
```

3.6.16 Get typeDefs by GUID

To get typedefs by GUID:

```
type_defs = client.typedef_guid(TYPE_GUID)
type_defs._data
```

3.6.17 Get typeDefs by name

To get typedefs by name:

```
type_defs = client.typedef_guid(TYPE_NAME)
type_defs._data
```

3.7 AdminREST

3.7.1 Get Admin Metrics

This endpoint is not yet mentioned in the official atlas documentation, but gives the complete statistics available for Atlas >2.x only. Endpoint is *api/atlas/admin/metrics*:

```
for metrics in client.admin_metrics:
    # This gives the entities count for both active and deleted entities
    entity_stats = metrics.entity

    # Provides the general Atlas statistics, about the counts, and different␣
↪timestamps
    general_stats = metrics.general

    # Provides a list of tags, along with the count of entities using that tag
    tag_stats = metrics.tag
```

CHAPTER 4

Utility / Helpers

4.1 parse_table_qualified_name()

atlasclient provides helper function to parse the table qualified name and returns a dictionary containing *db_name*, *table_name* and *cluster_name* as keys:

```
from atlasclient.utils import parse_table_qualified_name

# Happy Scenario
qualified_name = 'database.table@cluster'
qn_dict = parse_table_qualified_name(qualified_name)
print(qn_dict["db_name"])
# Output: database

print(qn_dict["table_name"])
# Output: table

print(qn_dict["cluster_name"])
# Output: cluster
```

In case if the entity is created manually and somehow does not fully satisfies the atlas qualified name pattern, this helper function handles the edge cases:

```
qualified_name = 'table@cluster'
qn_dict = parse_table_qualified_name(qualified_name)
print(qn_dict["db_name"])
# Output: default

print(qn_dict["table_name"])
# Output: table

print(qn_dict["cluster_name"])
# Output: cluster
```

4.2 make_table_qualified_name()

There is also a function to make the table qualified name, back from the parsed result. It verifies if all three i.e., *table_name*, *cluster* and *db* parameters are there and not *default*. If the value is default or not available, then this helper handles the edge case accordingly:

```
from atlasclient.utils import make_table_qualified_name

# Happy Scenario
qualified_name = make_table_qualified_name('table', 'cluster', 'database')
print(qualified_name)
# Output: 'database.table@cluster'
```

CHAPTER 5

Credits

5.1 Development Lead

- Jean-Baptiste Poullet <jeanbaptistepoullet@gmail.com>

5.2 Contributors

- Verdan Mahmood <verdan.mahmood@gmail.com>

CHAPTER 6

History

6.1 1.0.0 (2019-08-10)

- Adds the helper functions to parse the qualified name
- Updates the version to 1.x to get some confidence from community as the module is pretty stable now

6.2 0.1.8 (2019-08-08)

- Add support for Atlas' Admin Metrics REST API

6.3 0.1.7 (2019-07-08)

- Add support for Atlas' DSL Saved Search (#81)
- Fixes list lookups for searching

6.4 0.1.6 (2019-04-26)

- Call of DependentClass inflate (#79)

6.5 0.1.5 (2019-04-24)

- Add support for Post type Basic Search (#76)

6.6 0.1.4 (2019-04-16)

- fixes (BasicSearch, when no result in _data, etc)

6.7 0.1.3 (2019-04-05)

- HTTP Auth
- Basic search inflate
- relationshipAttributes

6.8 0.1.2 (2018-03-27)

- Bug fixes
- Response is returned after entity creation (easier to figure out the guid)

6.9 0.1.1 (2018-03-07)

- Bug fixes
- Most of the resources have been implemented (except RelationshipREST)
- Basic authentication (only the Basic token is sent on the network)

6.10 0.1.0 (2018-01-09)

- First push.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search