
Python Atlas client Documentation

Release 0.1.3

Atlas client

Jan 07, 2019

Contents

1	Credits	3
1.1	Development Lead	3
1.2	Contributors	3
2	History	5
2.1	0.1.2 (2018-03-27)	5
2.2	0.1.1 (2018-03-07)	5
2.3	0.1.0 (2018-01-09)	5
3	Installation	7
4	Apache Atlas Client in Python	9
4.1	Get started	9
4.2	Features	9
4.3	TODO features	9
4.4	Credits	10
5	Usage	11
5.1	DiscoveryREST	12
5.2	EntityREST	12
5.3	LineageREST	16
5.4	RelationshipREST	16
5.5	TypesREST	16
6	Indices and tables	21

Contents:

1.1 Development Lead

- Jean-Baptiste Poulet <jeanbaptistepoulet@gmail.com>

1.2 Contributors

None yet. Why not be the first?

2.1 0.1.2 (2018-03-27)

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

2.2 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)

2.3 0.1.0 (2018-01-09)

- First push.

CHAPTER 3

Installation

Install the package with pip:

```
$ pip install atlasclient
```

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>.

4.1 Get started

```
>>> from atlasclient.client import Atlas
>>> client = Atlas('<atlas.host>', port=21000, username='admin', password='admin')
>>> client.entityguid(<guid>).status
>>> params = {'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)
```

4.2 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.

4.3 TODO features

- allow multiprocessing

4.4 Credits

This package was created with [Cookiecutter](#) and the [audreyr/cookiecutter-pypackage](#) project template.

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

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.

5.1 DiscoveryREST

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

5.1.1 Search by attribute

To search for entities with a special attribute name:

```
params = {'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)
```

5.1.2 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)
```

5.1.3 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:
        print(e.classificationNames)
        print(e.attributes)
```

5.2 EntityREST

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

5.2.1 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)
```

5.2.2 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()
```

5.2.3 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')
```

5.2.4 Delete entity by GUID

To delete our entity:

```
entity.delete()
```

5.2.5 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)
```

5.2.6 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'.

5.2.7 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.

5.2.8 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.

5.2.9 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.

5.2.10 Get entities by bulk

To retrieve list of entities identified by its GUIDs:

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

5.2.11 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).

5.2.12 Delete multiple entities

To delete a list of entities:

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

5.2.13 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.

5.2.14 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')
```

5.2.15 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 ####
```

5.2.16 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()
```

5.3 LineageREST

5.3.1 Get lineage by GUID

To get lineage info about entity identified by GUID:

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

5.4 RelationshipREST

TO BE DONE...

5.5 TypesREST

5.5.1 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.

5.5.2 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": [],
  "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
        }
      ]
    }
  ]
}
```

5.5.3 Create typeDefs

To create typedefs:

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

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

5.5.4 Update typeDefs

To update typedefs:

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

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

5.5.5 Get typeDefs headers

To get typedefs headers:

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

5.5.6 Get classificationDefs by GUID

To get classificationdefs by GUID:

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

5.5.7 Get classificationDefs by name

To get classificationdefs by name:

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

5.5.8 Get entityDefs by GUID

To get entitydefs by GUID:

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

5.5.9 Get entityDefs by name

To get entitydefs by name:

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

5.5.10 Get enumDefs by GUID

To get enumdefs by GUID:

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

5.5.11 Get enumDefs by name

To get enumdefs by name:

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

5.5.12 Get relationshipDefs by GUID

To get relationshipdefs by GUID:

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

5.5.13 Get relationshipDefs by name

To get relationshipdefs by name:

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

5.5.14 Get structDefs by GUID

To get structdefs by GUID:

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

5.5.15 Get structDefs by name

To get structdefs by name:

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

5.5.16 Get typeDefs by GUID

To get typedefs by GUID:

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

5.5.17 Get typeDefs by name

To get typedefs by name:

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


CHAPTER 6

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)