RIPE Atlas Cousteau Documentation

Release 2.0.0

The RIPE Atlas Team

Feb 10, 2023

Contents

Welcome to the official Python wrapper around the RIPE Atlas API. This package can used by anyone to communicate with the RIPE Atlas API using Python. You will need to create an API key for many use cases.

This package is maintained by RIPE Atlas developers.

CHAPTER 1

Contents:

1.1 Installation

1.1.1 From PyPy with pip

The quickest and easiest way to install RIPE Atlas Cousteau is to use `pip`:

```
$ pip install ripe-atlas-cousteau
```

1.1.2 From GitHub

If you're feeling a little more daring and want to use whatever is on GitHub, you can have pip install right from there:

```
$ pip install git+https://github.com/RIPE-NCC/ripe-atlas-cousteau.git
```

1.1.3 From a Tarball

If for some reason you want to just download the source and install it manually, you can always do that too. Simply un-tar the file and run the following in the same directory as `setup.py`:

```
$ python setup.py install
```

1.2 Use & Examples

RIPE Atlas Cousteau wraps the majority of the RIPE Atlas v2 API calls, but not all of them. For some of these calls you will need to have a specific API key that you can create with the API key manager.

1.2.1 Creating Measurements

Important: An API key is needed for this function.

You can create multiple measurements with one API request that will share the same start/end times and allocated probes. This means that if you create a ping and a traceroute with one call they will start and finish at the same time and will use same probes.

Measurement Types

The first step is to create the measurement specification object. Currently you can use the following measurement types objects:

- Ping
- Traceroute
- Dns
- Sslcert
- Ntp
- Http

You can initialise any of these objects by passing any of arguments stated in the API docs. Keep in mind that this library is trying to comply with what is stated in these docs. This means that if you try to create a measurement that is missing a field stated as required in these docs, the library won't go ahead and do the HTTP query. On the contrary, it will raise an exception with some info in it. The required fields for each of the above type are:

Ping	Traceroute	Dns	Sslcert	Ntp	Http
af	af	af	af	af	af
description	description	description	description	description	description
target	target	query_class	target	target	target
	protocol	query_type			
		query_argument			

Examples:

```python
from ripe.atlas.cousteau import (
  Ping,
  Traceroute
)

ping = Ping(
    af=4,
    target="www.google.gr",
    description="Ping Test"
)

traceroute = Traceroute(
    af=4,
    target="www.ripe.net",
    description="Traceroute Test",
    protocol="ICMP",
)
```

Measurement Sources

The second step is to create the measurements source(s). In order to do that you have to create an AtlasSource object using the arguments type, value, requested, and optionally tags. Type as described in the API docs should be one of the "area", "country", "prefix", "asn", "probes", "msm". Value is the actual value of the type and requested is the number of probes that will be selected from this source. Optionally you can use tags argument, which has to be a dictionary like {"include": [], "exclude": []}. Examples:

```
from ripe.atlas.cousteau import AtlasSource

source = AtlasSource(
    type="area",
    value="WW",
    requested=5,
    tags={"include":["system-ipv4-works"]}
)
source1 = AtlasSource(
    type="country",
    value="NL",
    requested=50,
    tags={"exclude": ["system-anchor"]}
)
```

Create Request

The last step is to make the actual HTTP request. Before you do this you need at least to specify if you measurements will be oneoff and you API key. Additional you can have start and stop time defined.

Examples:

```
from datetime import datetime
from ripe.atlas.cousteau import (
  Ping,
  Traceroute,
  AtlasSource,
  AtlasCreateRequest
)

ATLAS_API_KEY = ""

ping = Ping(af=4, target="www.google.gr", description="testing new wrapper")

traceroute = Traceroute(
    af=4,
    target="www.ripe.net",
    description="testing",
    protocol="ICMP",
)

source = AtlasSource(
    type="area",
    value="WW",
    requested=5,
    tags={"include":["system-ipv4-works"]}
)
source1 = AtlasSource(
```

(continues on next page)

(continued from previous page)

```
    type="country",
    value="NL",
    requested=50,
    tags={"exclude": ["system-anchor"]}
)


atlas_request = AtlasCreateRequest(
    key=ATLAS_API_KEY,
    measurements=[ping, traceroute],
    sources=[source, source1],
    is_oneoff=True
)

(is_success, response) = atlas_request.create()
```

1.2.2 Changing Measurement Sources

Important: An API key is needed for this function.

If you want to add or remove probes from an existing measurement you have to use the AtlasChangeRequest. First step is to create an AtlasChangeSource objects which is similar to AtlasSource object for the creation of measurements. The difference is that here you have to specify an additional action argument. This parameter takes only two values "add" or "remove". In case of "remove" the type of the source can only be "probes". For more info check the API docs.

Example:

```
from ripe.atlas.cousteau import AtlasChangeSource, AtlasChangeRequest

ATLAS_MODIFY_API_KEY = ""

# Add probes
source = AtlasChangeSource(
    value="GR",
    requested=3,
    type="country",
    tags={"include":["system-ipv4-works"], "exclude": ["system-anchor"]},
    action="add"
)
source1 = AtlasChangeSource(
    value="4,5,6",
    requested=3,
    type="probes",
    action="add"
)

# Remove probes
source2 = AtlasChangeSource(
    value="1,2,3",
    type="probes",
    requested=3,
    action="remove"
```

(continues on next page)

(continued from previous page)

```
)

atlas_request = AtlasChangeRequest(
    key=ATLAS_MODIFY_API_KEY,
    msm_id=1000001,
    sources=[source, source1, source2]
)

(is_success, response) = atlas_request.create()
```

1.2.3 Stopping Measurements

Important: An API key is needed for this function.

You can stop a measurement by creating a AtlasStopRequest and specifying measurement ID as shown below:

```
from ripe.atlas.cousteau import AtlasStopRequest

ATLAS_STOP_API_KEY = ""

atlas_request = AtlasStopRequest(msm_id=1000001, key=ATLAS_STOP_API_KEY)

(is_success, response) = atlas_request.create()
```

1.2.4 Results

Fetching Results

Note: If measurement is not public you will need an API key with "download results of a measurement" permission.

You can fetch results for any measurements using AtlasResultsRequest. You can filter them by start/end time and probes. Times can be python datetime objects, Unix timestamps or string representations of dates.

Example:

```
from datetime import datetime
from ripe.atlas.cousteau import AtlasResultsRequest

kwargs = {
    "msm_id": 2016892,
    "start": datetime(2015, 05, 19),
    "stop": datetime(2015, 05, 20),
    "probe_ids": [1,2,3,4]
}

is_success, results = AtlasResultsRequest(**kwargs).create()

if is_success:
    print(results)
```

Fetching Latest Results

Note: If measurement is not public you will need an API key with "download results of a measurement" permission.

In case you want to download latest results of a measurement or your measurement is an oneoff measurements is easier and faster to use the API for the latest results. Fetching latest results is done by using AtlasLatestRequest and there is an option for filtering by probes.

Example:

```python
from ripe.atlas.cousteau import AtlasLatestRequest

kwargs = {
    "msm_id": 2016892,
    "probe_ids": [1,2,3,4]
}

is_success, results = AtlasLatestRequest(**kwargs).create()

if is_success:
    print(results)
```

Streaming API

Atlas supports getting results and other events through a WebSocket stream to get them close to real time. The AtlasStream class provides an interface to this stream, and supports both callback-based and iterator-based access.

Measurement Results

You have to create an AtlasStream object and subscribe to the "result" stream type. More details on the available parameters of the stream can be found on the streaming documentation.

Example using the callback-interface:

```python
from ripe.atlas.cousteau import AtlasStream

def on_result_response(*args):
    """
    Function that will be called every time we receive a new result.
    Args is a tuple, so you should use args[0] to access the real message.
    """
    print(args[0])

atlas_stream = AtlasStream()
atlas_stream.connect()

# Bind function we want to run with every result message received
atlas_stream.bind("atlas_result", on_result_response)

# Subscribe to new stream for 1001 measurement results
stream_parameters = {"msm": 1001}
atlas_stream.subscribe(stream_type="result", **stream_parameters)

```

(continues on next page)

(continued from previous page)

```
# Process incoming events for 5 seconds, calling the callback defined above.
# Make sure you have this line after you start *all* your streams
atlas_stream.timeout(seconds=5)

# Shut down everything
atlas_stream.disconnect()
```

Connection Events

Besides results, the streaming API also supports probe connect/disconnect events. Again you have to create an AtlasStream object, but this time you subscribe to the "probestatus" stream type. More info about additional parameters can be found on the streaming documentation.

Example using the iterator-interface:

```
from ripe.atlas.cousteau import AtlasStream

atlas_stream = AtlasStream()
atlas_stream.connect()

stream_parameters = {"enrichProbes": True}
atlas_stream.subscribe(stream_type="probestatus", **stream_parameters)

# Iterate over the incoming results for 5 seconds
for event_name, payload in atlas_stream.iter(seconds=5):
    print(event_name, payload)

# Shut down everything
atlas_stream.disconnect()
```

Using the Sagan Library

In case you need to do further processing with any of the results you can use our official RIPE Atlas results parsing library called Sagan. An example of how to combine two libraries is the below:

```
from ripe.atlas.cousteau import AtlasLatestRequest
from ripe.atlas.sagan import Result

kwargs = {
    "probe_ids": [1,2,3,4]
}

is_success, results = AtlasLatestRequest(**kwargs).create()

if is_success:
    for result in results:
        print(Result.get(result))
```

1.2.5 Metadata

RIPE Atlas API allows you to get metadata about probes and measurements in the system. You can get metadata for a single object or filter based on various criteria.

Single Objects

Every time you create a new instance of either Measurement/Probe objects it will fetch meta data from API and return an object with selected attributes.

Measurement

Using the Measurement object will allow you to have a python object with attributes populated from specific measurement's meta data.

Example:

```
from ripe.atlas.cousteau import Measurement

measurement = Measurement(id=1000002)
print(measurement.protocol)
print(measurement.description)
print(measurement.is_oneoff)
print(measurement.is_public)
print(measurement.target_ip)
print(measurement.target_asn)
print(measurement.type)
print(measurement.interval)
print(dir(measurement)) # Full list of properties
```

Probe

Using the Probe object will allow you to have a python object with attributes populated from specific probe's meta data.

```
from ripe.atlas.cousteau import Probe

probe = Probe(id=3)
print(probe.country_code)
print(probe.is_anchor)
print(probe.is_public)
print(probe.address_v4)
print(dir(probe)) # Full list of properties
```

Filtering

This feature queries API for probes/measurements based on specified filters. Available filters can be found in the API docs. Underneath it will follow all next urls until there are no more objects. It returns a python generator that you can use in a for loop to access each object.

Probe

The following example will fetch all measurements with Status equals to "Specified". More info on filters for this call can be found in the API docs.

```
from ripe.atlas.cousteau import ProbeRequest

filters = {"tags": "NAT", "country_code": "NL", "asn_v4": "3333"}
probes = ProbeRequest(**filters)

for probe in probes:
    print(probe["id"])

# Print total count of found probes
print(probes.total_count)
```

Measurement

The following example will fetch all probes from NL with asn_v4 3333 and with tag NAT. More info on filters for this call can be found in the API docs.

```
from ripe.atlas.cousteau import MeasurementRequest

filters = {"status": 1}
measurements = MeasurementRequest(**filters)

for msm in measurements:
    print(msm["id"])

# Print total count of found measurements
print(measurements.total_count)
```

1.2.6 General GET API Requests

Using the general AtlasRequest object you can do any GET request to the RIPE Atlas API considering you provide the url path.

Example:

```
url_path = "/api/v2/anchors"
request = AtlasRequest(**{"url_path": url_path})
result = namedtuple('Result', 'success response')
(is_success, response) = request.get()
if not is_success:
    return False

return result.response["participant_count"]
```

1.3 How To Contribute

We would love to have contributions from everyone and no contribution is too small. Please submit as many fixes for typos and grammar bloopers as you can!

To make participation in this project as pleasant as possible for everyone, we adhere to the Code of Conduct by the Python Software Foundation.

The following steps will help you get started:

Fork, then clone the repo:

```
$ git clone git@github.com:your-username/ripe-atlas-cousteau.git
```

Make sure the tests pass beforehand:

```
$ tox
```

Then:

- Make your changes.
- Include new tests/modify existing tests for your change.
- Make sure the tests still pass:
- Push to your fork and submit a pull request.

Here are a few guidelines that will increase the chances of a quick merge of your pull request:

- *Always* try to add tests and docs for your code. If a feature is tested and documented, it's easier for us to merge it.
- Follow flake8 and Black style/linting.
- Write good commit messages.
- If you change something that is noteworthy, don't forget to add an entry to the changes.

Note:

- If you think you have a great contribution but aren't sure whether it adheres – or even can adhere – to the rules: **please submit a pull request anyway**! In the best case, we can transform it into something usable, in the worst case the pull request gets politely closed. There's absolutely nothing to fear.
- If you have a great idea but you don't know how or don't have the time to implement it, please consider opening an issue and someone will pick it up as soon as possible.

Thank you for considering a contribution to this project! If you have any questions or concerns, feel free to reach out the RIPE Atlas team via the mailing list, GitHub Issue Queue, or messenger pigeon – if you must.

1.4 Releases History

1.4.1 2.0.0 (release 2023-01-20)

Changes:

- The AtlasStream class has been updated to use the new WebSocket interface
- AtlasStream objects can now be iterated as an alternative to using callbacks
- There used to be both start_stream() and subscribe() methods which did the same thing, except that start_stream() had extra validation. This extra validation has been added to subscribe(), and start_stream() is now an alias to it.
- bind_channel() was renamed to bind(), although it is maintained as an alias, and there is a corresponding unbind() to remove a callback

- Deprecated event aliases were dropped, you have to use full event names like “atlas_result” and “atlas_metadata” when binding

1.4.2 1.5.1 (release 2022-05-23)

Bug Fixes:

- Make it possible to instantiate AtlasMeasurement directly by passing in a “type” value in the spec

1.4.3 1.5.0 (released 2022-02-23)

Changes:

- Drop support for old Python versions (>= 3.6 supported)
- Documentation fixes
- Use python-socketio instead of socketIO-client for streaming
- Include an ‘unsubscribe’ method on AtlasStream, as well as constants for currently available event names
- Update testing framework to use pytest

1.4.4 1.4.2 (released 2018-03-22)

New features:

- Add support for tagging and untagging measurements

1.4.5 1.4.1 (released 2018-01-08)

Changes:

- Minor fixes

1.4.6 1.4 (released 2017-04-21)

New features:

- Expose *bill_to* option for measurement creation
- Enable User-Agent for stream connections

Changes:

- Update stream channel names to new naming scene in the background

Bug Fixes:

- Fix bug where stream was not reconnected after a disconnect
- Fix unicode cases on the stream spotted by @JonasGroeger

1.4.7 1.3 (released 2016-10-21)

Changes:

- Improved streaming support:
- Introduce error handling
- Channels errors binded by default
- Introduced debug mode
- Update features set. See all here https://atlas.ripe.net/docs/result-streaming/
- Deprecated short events name and local event name checking. See the event names here https://atlas.ripe.net/docs/result-streaming/
- Introduced support for proxies and additional headers
- Timezone aware objects for measurement meta data

1.4.8 1.2 (released 2016-03-02)

Changes:

- Backwards incompatible field changes on the Measurement object:
- destination_address -> target_ip
- destination_asn -> target_asn
- destination_name -> target

1.4.9 1.1 (released 2016-02-09)

New features:

- Start supporting Anchors listing API.
- Brand new documentation hosted on readthedocs.

Changes:

- Various refactorings to clean up codebase.

1.4.10 1.0.7 (released 2016-01-13)

Changes:

- Backwards compatible change of the format we expect for measurement type to handle upcoming change in the API.

Bug Fixes:

- Fix bug when creating stream for probes connection channel. Updating also wrong documentation.

1.4.11 1.0.6 (released 2015-12-15)

Changes:

- Add copyright text everywhere for debian packaging.

1.4.12 1.0.5 (released 2015-12-14)

Changes:

- Add tests to the package itself.
- Make user-agent changeable by the user.
- Various refactorings.

1.4.13 1.0.4 (released 2015-11-06)

Changes:

- Handle both string/dictionary as input for probe_ids filter for Result and LatestResult requests.

1.4.14 1.0.2 (released 2015-10-26)

Bug Fixes:

- Fix bug where key parameter was added to the url even if it was empty.
- Fix bug where we didn't try to unjson 4xx responses even if they could contain json structure.

1.4.15 1.0.1 (released 2015-10-23)

Changes:

- Now we conform to new API feature that allows for specifying tags when adding probes to existing measurements

Bug Fixes:

- Fix bug we didn't allow user to specify single tag include/exclude.

1.4.16 1.0 (released 2015-10-21)

New features:

- Add support for include/exclude tags in changing sources requests.
- Add support for latest results API call.
- Implement HTTP measurement creation.
- Support for python 3 (<=3.4).
- Support for pypy/pypy3.
- Support for wheels format.

Changes:

- Migrate all Atlas requests to use requests library and refactor a lot of code to have a cleaner version.
- Create an API v2 translator to address several option name changing. A deprecation warning will be given.

Bug Fixes:

- Fix bug where python representation of measurements without a stop time was exploding.
- Make sure start/stop timestamps in measurement create request are always in UTC.

1.4.17 0.10.1 (released 2015-10-06)

New features:

- Implement support for object return in the request generators for probe/measurement.

Changes:

- Probe/Measurement python representation takes meta data from v2 API as well. Now everything should point to v2 API.

1.4.18 0.10 (released 2015-10-01)

New features:

- add object representation of meta data for a probe or a measurement.

Changes:

- Abandon v1 RIPE ATLAS API and use only v2.

Bug Fixes:

- Fix bug that prevented users from specifying all possible source types when they tried to add more probes to existing measurements.
- Cover case where a user specified really long list of probes/measurements in the ProbeRequest/MeasurementRequest that was causing 'HTTP ERROR 414: Request-URI Too Long'. Additionally, now if API returns error raise an exception instead of stopping iteration.

1.4.19 0.9.2 (released 2015-09-21)

Changes:

- Small refactor of Stream class and manually enforce websockets in SocketIO client

1.4.20 0.9.1 (released 2015-09-03)

Bug Fixes:

- Fix bug related to binding result atlas stream.

1.4.21 0.9 (released 2015-09-01)

New features:

- add support for fetching real time results using RIPE Atlas stream server.
- this version and on will be available on PYPI.

1.4.22 0.8 (released 2015-08-31)

New features:

- add support for NTP measurements.

1.4.23 0.7 (released 2015-06-03)

New features:

- add support for fetching probes/measurements meta data using python generators.

1.4.24 0.6 (released 2014-06-17)

New features:

- add support for querying results based on start/end time, msm_id and probe id.

Changes:

- add http agent according to package version to all requests.

1.4.25 0.5 (released 2014-05-22)

Changes:

- change package structure to comply with the new structure of atlas packages
- add continuous integration support
- add tests
- enable travis
- enable code health checks
- add required files for uploading to github

1.4.26 0.4 (released 2014-03-31)

New features:

- add support for stopping a measurement.

1.4.27 0.3 (released 2014-02-25)

New features:

- add simple support for HTTP GET queries.

1.4.28 0.2 (released 2014-02-03)

New features:

- add support for adding/removing probes API request.

Changes:

- use AtlasCreateRequest instead of AtlasRequest for creating a new measurement.

1.4.29 0.1 (released 2014-01-21)

- Initial release.